# Reduced Virtual Interrupt Controller specification

| | |
|---|---|
| Document number | ARM-DEN-0103 |
| Document quality | ALP |
| Document version | 00alp1 |
| Document confidentiality | Non-confidential |
| Document build information | a7fcf39 doctool 0.49.0 |

*Copyright © 2020 Arm Limited or its affiliates. All rights reserved.*

DRAFT

**Reduced Virtual Interrupt Controller (RVIC) specification**

**Quality Status: Alpha (ALP)**

Alpha quality status has a particular meaning to Arm of which the recipient must be aware. This document is for review purposes only and should not be used for any implementation, because significant changes are likely.

# Reduced Virtual Interrupt Controller specification

## Release information

| Date | Version | Changes |
|---|---|---|
| 2020/Sep/02 | 00alp1 | • Change to non-confidential license<br>• Clarify distinction between Trusted and Untrusted interrupts<br>• Correct rules regarding guarantee of interrupt delivery<br>• Clarify sequence of events required to change the target of an interrupt<br>• Add rules on notification of pending interrupts<br>• Clarify that PSTATE.I is PE architectural state<br>• Align error codes between RVIC and RVID |
| 2020/Jul/31 | 00alp0 | • Initial version |

DRAFT

## Non-Confidential Proprietary Notice

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

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

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

This document may include technical inaccuracies or typographical errors.

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

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

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

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

Copyright © 2020 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349 version 21.0


DRAFT

# Contents

# Reduced Virtual Interrupt Controller specification


Reduced Virtual Interrupt Controller specification . . . ii
- Release information . . . ii
- Non-Confidential Proprietary Notice . . . iii

**Preface**

Conventions . . . vi
- Typographical conventions . . . vi
- Numbers . . . vi

Rules-based writing . . . vii
- Content item classes . . . vii
- Identifiers . . . viii
- Examples . . . viii

Additional reading . . . ix
Feedback . . . ix
- Feedback on this book . . . ix

Open issues . . . x

**Chapter 1 Introduction**

1.1 Components . . . 12
1.2 Versioning . . . 12

**Chapter 2 Reduced Virtual Interrupt Controller (RVIC)**

2.1 RVIC overview . . . 14
2.2 RVIC architecture version . . . 14
2.3 RVIC functional description . . . 15
- 2.3.1 Instance enablement status . . . 15
- 2.3.2 Trusted and untrusted interrupts . . . 15
- 2.3.3 Interrupt mask status . . . 16
- 2.3.4 Interrupt life-cycle . . . 16
- 2.3.5 Trigger modes . . . 18
- 2.3.6 Signaling . . . 18
- 2.3.7 Multiple pending interrupts . . . 19
- 2.3.8 INTID assignments . . . 19
- 2.3.9 Interrupt routing . . . 20
- 2.3.10 Notification of pending interrupts . . . 20

2.4 RVIC programming interface . . . 22
- 2.4.1 AArch64 hypercall interface . . . 22
- 2.4.2 RVIC types . . . 23
- 2.4.3 RVIC commands . . . 25
  - 2.4.3.1 RVIC.Version . . . 25
  - 2.4.3.2 RVIC.Info . . . 26
  - 2.4.3.3 RVIC.Enable . . . 27
  - 2.4.3.4 RVIC.Disable . . . 28
  - 2.4.3.5 RVIC.SetMasked . . . 29
  - 2.4.3.6 RVIC.ClearMasked . . . 30
  - 2.4.3.7 RVIC.IsPending . . . 31
  - 2.4.3.8 RVIC.Signal . . . 32
  - 2.4.3.9 RVIC.ClearPending . . . 33
  - 2.4.3.10 RVIC.Acknowledge . . . 34


DRAFT


*Copyright © 2020 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

*Contents*

2.4.3.11 RVIC.Resample . . . . . . . . . . . . . . . . . . . . . . . . . . . 35

**Chapter 3** **Reduced Virtual Interrupt Distributor (RVID)**

3.1 RVID introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
3.2 RVID architecture version . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
3.3 RVID functional description . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
3.4 RVID programming interface . . . . . . . . . . . . . . . . . . . . . . . . . . . 39
3.4.1 AArch64 hypercall interface . . . . . . . . . . . . . . . . . . . . . . . . 39
3.4.2 RVID types . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
3.4.3 RVID commands . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 42
3.4.3.1 RVID.Version . . . . . . . . . . . . . . . . . . . . . . . . . . . . 42
3.4.3.2 RVID.Map . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43
3.4.3.3 RVID.Unmap . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44

**Glossary**

DRAFT


*Copyright © 2020 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

# Preface

# Conventions

## Typographical conventions

The typographical conventions are:

*italic*

Introduces special terminology, and denotes citations.

`monospace`

Used for command names and numerical values.

SMALL CAPITALS

Used for some common terms such as IMPLEMENTATION DEFINED.

Red text

Indicates an open issue.

Blue text

Indicates a link. This can be

- A cross-reference to another location within the document
- A URL, for example http://developer.arm.com

DRAFT

## Numbers

Numbers are normally written in decimal. Hexadecimal numbers are prefixed by by `0x`. The prefix and the associated value are written in a monospace font, for example `0xFFFF0000`.

# Rules-based writing

This specification consists of a set of individual *content items*. Content items are classified into the following types:

- Declaration
- Rule
- Information
- Rationale
- Software usage
- Implementation note

Declarations and Rules are normative statements. An implementation which is compliant with this specification must conform to all of the Declarations and Rules in this specification.

Declarations and Rules must not be read in isolation. Where a particular feature is specified by multiple Declarations and Rules, these are grouped into sections and subsections to provide context. Where appropriate, these sections begin with a short introduction to aid the reader.

Arm strongly recommends that implementers read *all* chapters and sections of this document to ensure that an implementation is compliant.

Content items other than Declarations and Rules are informative statements. These are provided purely as an aid to understanding this specification.

DRAFT

## Content item classes

### Declaration

A Declaration is a statement which either

- introduces the meaning of a concept or term, or
- describes the structure or encoding of data.

A Declaration does not describe behaviour.

A Declaration is identified by the letter D.

### Rule

A Rule is a statement which describes the behaviour of a compliant implementation.

A Rule does not define concepts or terminology.

A Rule is identified by the letter R.

### Information

An Information statement provides additional information and guidance as an aid to understanding the specification.

An Information statement is identified by the letter I.

### Rationale

A Rationale statement explains why the specification was specified as it was.

A Rationale statement is identified by the letter X.

### Implementation note

An Implementation note provides guidance on implementation of the specification.

An Implementation note is identified by the letter U.

### Software usage

A Software usage statement provides guidance on how software can make use of the features defined by the specification.

A Software usage statement is identified by the letter S.

## Identifiers

Each content item may have an associated identifier which is unique within the context of this specification.

When the document is prior to beta status:

- Content items are assigned numerical identifiers, in ascending order through the document (*0001*, *0002*, . . . ).
- Identifiers are volatile: the identifier for a given content item may change between versions of the document.

After the document reaches beta status:

- Content items are assigned random alphabetical identifiers (*HJQS*, *PZWL*, . . . ).
- Identifiers are preserved: a given content item has the same identifier across versions of the document.

## Examples

Below are examples showing the appearance of each type of content item.

D This is a Declaration.

R This is a Rule.

$R_{X001}$ This is a Rule with an identifier.

X This is a Rationale statement.

I This is an Information statement.

U This is an Implementation note.

S This is a Software usage statement.

DRAFT


Copyright © 2020 Arm Limited or its affiliates. All rights reserved.
Non-confidential

# Additional reading

This section lists publications by Arm and by third parties.

See Arm Developer (http://developer.arm.com) for access to Arm documentation.

[1] *Arm Generic Interrupt Controller Architecture Specification v3 and v4*. (Arm IHI 0069 F) Arm Ltd.

[2] *Arm Architecture Reference Manual for Armv8-A architecture profile*. (ARM DDI 0487 B) Arm Ltd.

[3] *Arm Server Base System Architecture version 6.0*. (ARM DEN 0029 C) Arm Ltd.

[4] *Arm SMC Calling Convention*. (ARM DEN 0028 B) Arm Ltd.

[5] *Arm Power State Coordination Interface*. (ARM DEN 0022 D) Arm Ltd.

# Feedback

Arm welcomes feedback on its documentation.

## Feedback on this book

If you have comments on the content of this book, send an e-mail to errata@arm.com. Give:

- The title (Reduced Virtual Interrupt Controller specification).
- The number (ARM-DEN-0103 00alp1).
- The page numbers to which your comments apply.
- The rule identifiers to which your comments apply, if applicable.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

---

**Note**

Arm tests PDFs only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the appearance or behavior of any document when viewed with any other PDF reader.

---

DRAFT

# Open issues

The following table lists known open issues in this version of the document.

| Key | Description |
|---|---|
| - | Consider whether prioritization of interrupts should be supported. |

DRAFT

# Chapter 1
# **Introduction**

DRAFT

This document is the specification of the Reduced Virtual Interrupt Controller architecture.


*Copyright © 2020 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 1.1 Components

This specification consists of the following components:

- **Reduced Virtual Interrupt Controller (RVIC)**

  A *Para-virtualized* (PV) interrupt controller architecture which provides basic interrupt support for *virtual machines* (VMs).

- **Reduced Virtual Interrupt Distributor (RVID)**

  A PV interrupt distributor which allows software in a VM to map virtual interrupts generated by virtual sources, such as an emulated I/O device, to specific *Virtual Processing Elements* (VPEs).

See also:

- Chapter 2 *Reduced Virtual Interrupt Controller (RVIC)*
- Chapter 3 *Reduced Virtual Interrupt Distributor (RVID)*

## 1.2 Versioning

The version of this document is 00alp1.

Each component of this specification has its own version, which is reported via the appropriate `[Component].Version` command. This allows the ABI of each component to be developed independently.

See also:

- 2.2 *RVIC architecture version*
- 3.2 *RVID architecture version*

DRAFT


Copyright © 2020 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Chapter 2
# Reduced Virtual Interrupt Controller (RVIC)

DRAFT

The *Reduced Virtual Interrupt Controller* (RVIC) is a PV interrupt controller architecture which provides basic interrupt support for VMs.


*Copyright © 2020 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 2.1 RVIC overview

The RVIC architecture supports a variety of hypervisor designs, including both monolithic hypervisors and split-mode hypervisors. Split-mode hypervisors are implemented across two or more levels of privilege (different Exception levels in the same security state, or different security states), resulting in the hypervisor functionality being divided into trusted and untrusted parts. This specification uses the term *Trusted hypervisor* to refer to the trusted part of the hypervisor and the term *Untrusted hypervisor* to refer to the untrusted part of the hypervisor. For monolithic hypervisor designs, these two terms refer to the same software component.

The RVIC architecture aims to provide the minimum required functionality to support virtual interrupts in VMs. A key design goal of the RVIC architecture is to allow the RVIC implementation to be small and simple, and therefore suitable to be part of the Trusted hypervisor. Functionality which does not need to be in the Trusted hypervisor is moved into either the VM or the Untrusted hypervisor.

There is one *RVIC instance* per VPE.

An RVIC instance delivers interrupts by signaling its VPE, which generates a virtual IRQ exception. When handling the virtual exception, the VPE can query its RVIC instance to determine which interrupt caused the exception. Each interrupt in each RVIC instance is represented by a unique number, the *Interrupt ID* (INTID).

The RVIC architecture supports two classes of interrupts:

- *Trusted interrupts*

  Can only be generated by the Trusted hypervisor

- *Untrusted interrupts*

  Can be generated by the Untrusted hypervisor

Each RVIC instance has its own separate state. Therefore, the RVIC architecture provides no mechanism allowing in-VM software to request a desired target VPE for an Untrusted interrupt (known as *interrupt routing*). Routing of Untrusted interrupts can be performed using a separate component such as the Reduced Virtual Interrupt Distributor (RVID), which is implemented outside the Trusted hypervisor.

$X_{0001}$ Avoiding shared state across all VPEs reduces the complexity of an RVIC implementation and allows the number of supported interrupts in a VM to scale naturally with the number of VPEs.

See also:

- 2.3.2 *Trusted and untrusted interrupts*
- 2.3.8 *INTID assignments*
- 2.3.9 *Interrupt routing*
- Chapter 3 *Reduced Virtual Interrupt Distributor (RVID)*

## 2.2 RVIC architecture version

$I_{0002}$ The version of the RVIC architecture is 0.3.

$I_{0003}$ The ABI is binary compatible across minor version number changes but not necessarily across major version number changes.

See also:

- 2.4.3.1 *RVIC.Version*

DRAFT


Copyright © 2020 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 2.3 RVIC functional description

### 2.3.1 Instance enablement status

D0004 The *enablement status* of an RVIC instance is either *Enabled* or *Disabled*.

R0005 If an RVIC instance is Disabled:

- No interrupts are signaled to the VPE.
- No interrupts become Pending on the VPE.
- Interrupts cannot be acknowledged.
- Interrupts can still be masked / unmasked.
- The Pending state of an interrupt can be cleared.

X0006 In-VM software must have the ability to reset and quiesce the interrupt controller state by disabling the interrupt controller, masking interrupts, and clearing their Pending state.

R0007 When an RVIC instance is reset, it becomes Disabled.

See also:

- 2.3.3 *Interrupt mask status*
- 2.3.4 *Interrupt life-cycle*
- 2.4.3.3 *RVIC.Enable*
- 2.4.3.4 *RVIC.Disable*

### 2.3.2 Trusted and untrusted interrupts

D0008 A Trusted interrupt is generated by the Trusted hypervisor.

I0009 There are two categories of Trusted interrupts:

- Software-generated interrupts (SGIs) are sent from a VPE to itself, or to another VPE managed by the Trusted hypervisor.
- Peripherals managed by the Trusted hypervisor can generate Trusted interrupts. For example, per-VPE Arm Generic Timer instances managed by the Trusted hypervisor can generate Trusted interrupts to ensure timer signals are delivered to the VM when it is executed after the timer condition is met.

D0010 An Untrusted interrupt is generated on request of the Untrusted hypervisor.

I0011 For example, a virtual peripheral such as a PV network device generates Untrusted interrupts.

I0012 Trusted and untrusted interrupts have separate INTID spaces.

X0013 Trusted interrupts cannot be generated by software outside the VM's *Root of Trust* (RoT). In-VM software can therefore trust that an incoming interrupt was actually generated by a source VPE. Similarly, in-VM software can trust that an incoming interrupt from a VPE-local peripheral is the result of that peripheral asserting an output signal.

I0014 The number of implemented Trusted and Untrusted interrupts is IMPLEMENTATION DEFINED.

I0015 The number of implemented Trusted and Untrusted interrupts can be discovered via the `RVIC.Info` command.

I0016 Both Trusted and Untrusted interrupts are guaranteed to be delivered when they are Pending and Unmasked at the time the VPE is entered.

I0017 The RVIC architecture does not guarantee delivery of interrupts within any limit in real time.

X0018 Scheduling of VPEs is not handled by the RVIC implementation and the RVIC architecture cannot assume that scheduling of CPU resources is performed in the Trusted hypervisor.

See also:

- 2.3.6 *Signaling*

DRAFT


*Copyright © 2020 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

- 2.3.8 *INTID assignments*
- 2.4.3.2 *RVIC.Info*

### 2.3.3 Interrupt mask status

$D_{0019}$ The *mask status* of an interrupt is either *Masked* or *Unmasked*.

$I_{0020}$ Mask status is a per-VPE, per-interrupt control.

$I_{0021}$ Mask status is distinct from the PE architectural `PSTATE.I` bit.

$R_{0022}$ If an interrupt is Masked, it is not signaled to the VPE.

$I_{0023}$ A Masked interrupt is not signaled to the VPE even if it is Pending.

$R_{0024}$ When the status of an interrupt changes from Masked to Unmasked, if it is Pending then it is signaled to the VPE.

$R_{0025}$ When an RVIC instance is reset, all interrupts become Masked.

See also:

- 2.3.4 *Interrupt life-cycle*
- 2.4.3.5 *RVIC.SetMasked*
- 2.4.3.6 *RVIC.ClearMasked*

### 2.3.4 Interrupt life-cycle

$D_{0026}$ The *life-cycle state* of an interrupt is either *Idle* or *Pending*.

$R_{0027}$ When the source of an interrupt signals an event, the interrupt becomes Pending.

$I_{0028}$ When the VPE acknowledges an interrupt, the interrupt becomes Idle.

$I_{0029}$ When the VPE acknowledges an interrupt, the interrupt becomes Masked.

$I_{0030}$ Following acknowledgement, an interrupt must be Unmasked before it can signal the VPE again.

DRAFT


*Copyright © 2020 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

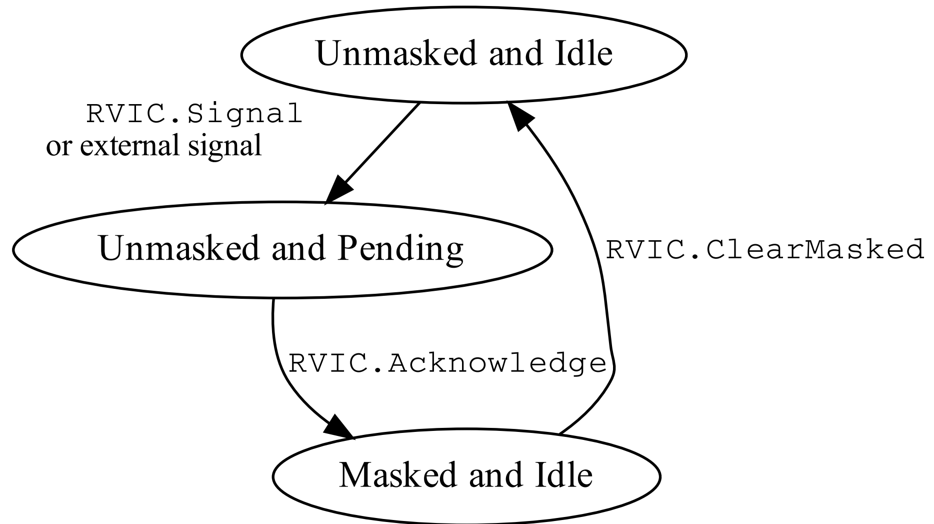



**Figure 2.1: Typical interrupt life cycle**

DRAFT

I0031 A typical cycle is shown in Figure 2.1:

1. Initial state: Interrupt is Unmasked and Idle
2. Interrupt signal (`RVIC.Signal`, or external signal): Interrupt is Unmasked and Pending
3. Interrupt acknowledge (`RVIC.Acknowledge`): Interrupt is Masked and Idle
4. Interrupt unmask (`RVIC.ClearMasked`): Interrupt is Unmasked and Idle

X0032 Automatically masking an interrupt on acknowledgement guarantees forward progress if an OS wants to consume all Pending interrupts by calling `RVIC.Acknowledge` in a loop. It also simplifies threaded interrupt handling, because in-VM software running on the handling VPE (which may be interrupted in a critical operation) does not have to explicitly mask the interrupt before assigning the work of processing the interrupt to a separate thread.

I0033 The RVIC architecture does not define an active state for interrupts like the Arm GIC architecture does.

X0034 Handling an active state complicates both the specification and implementation of the interrupt controller, especially when considering semantics for re-targeting interrupts from one VPE to another.

X0035 The active state in the GIC architecture provides support for threaded interrupt handling in that it allows the PE which takes an interrupt to clear `PSTATE.I` and take additional interrupts while a thread (running on any PE) finishes processing the first interrupt and eventually clears the active state. The same is achieved in the RVIC architecture by automatically masking the interrupt, but without two separate hardware states.

S0036 In-VM software must manage additional metadata to distinguish between:

- interrupts which are Masked because they were acknowledged, and
- interrupts which are Masked because software has disabled the interrupt.

See also:

- GIC [1]
- 2.4.3.6 *RVIC.ClearMasked*
- 2.4.3.8 *RVIC.Signal*
- 2.4.3.10 *RVIC.Acknowledge*


Copyright © 2020 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 2.3.5 Trigger modes

$D_{0037}$ Interrupt signals from interrupt sources are either *edge-triggered* or *level-triggered*.

$I_{0038}$ An edge-triggered signal is generated instantaneously when an event occurs.

$I_{0039}$ A level-triggered signal represents the internal state of the source.

$I_{0040}$ All interrupts in the RVIC architecture are edge-triggered.

$I_{0041}$ The RVIC architecture supports level-triggered signals by providing a re-sample operation that the interrupt controller driver can call after processing an interrupt from a source which uses level-triggered semantics for its interrupt signal.

$S_{0042}$ Software should always re-sample interrupt signals for level-triggered sources before completing the interrupt service routine and after re-programming the source. For example, the Arm Generic Timer specifies that the interrupt signal is asserted while the timer condition is met, and de-asserted otherwise. After processing a timer interrupt, the in-VM timer driver should re-sample the interrupt signal to generate an additional interrupt signal if the timer condition is still met.

$I_{0043}$ In rare situations, an RVIC instance might generate spurious interrupt signals from sources with level-triggered signals. This can occur if the source asserted and subsequently de-asserted its signal and the VPE did not acknowledge the interrupt before the source de-asserted the signal.

$S_{0044}$ Device drivers in VMs must be able to tolerate spurious interrupt signals from sources with level-triggered signals. For example, if a timer's condition is met and the interrupt signal is asserted, but the in-VM timer driver re-programs the timer and causes its condition to no longer be met without acknowledging the original interrupt, then the RVIC instance associated with the timer still signals an interrupt even though the timer condition is no longer met.

$X_{0045}$ It is much simpler to implement the RVIC architecture when all interrupts are edge-triggered. The complexity of implementing a re-sample operation is trivial, compared with supporting multiple interrupt life-cycle flows. Generation of spurious interrupts from level-triggered sources is expected to be extremely rare, and device drivers are commonly written to tolerate spurious interrupt signals. Existing VMs and virtual device semantics use almost exclusively edge-triggered semantics, with the exception of timers and PMUs.

DRAFT

$I_{0046}$ There is no need for the RVIC to provide interface for configuring which Trusted interrupts are level-triggered, because interrupt assignments are already described to the OS via firmware tables (such as ACPI or device tree) and device drivers are written with an understanding of the semantics of the signals generated by sources.

$R_{0047}$ Untrusted interrupts can only be used for sources which generate signals with edge-triggered semantics.

$X_{0048}$ While it would be possible to support level-triggered Untrusted interrupts, this would require additional complexity to re-sample the signal level with the Untrusted hypervisor, which is undesirable. Arm is not aware of a requirement for level-triggered interrupts from the Untrusted hypervisor, or for level-triggered interrupts from the Trusted hypervisor, which is not satisfied by edge-triggered interrupts with a resample operation.

$I_{0049}$ Resampling of Untrusted interrupts is not supported.

See also:

- 2.4.3.11 *RVIC.Resample*

### 2.3.6 Signaling

$R_{0050}$ If an RVIC instance is Enabled and has one or more interrupts which are both Unmasked and Pending, it signals an interrupt to the VPE.

$U_{0051}$ On AArch64 systems the RVIC implementation can set `HCR_EL2.VI` to generate a virtual IRQ exception. The VPE will observe the virtual IRQ exception according to the rules in [2]. Correspondingly, the RVIC implementation clears `HCR_EL2.VI` when either the RVIC instance is disabled or when there are no further Unmasked and Pending interrupts.


Copyright © 2020 Arm Limited or its affiliates. All rights reserved.
Non-confidential

X0052 An alternative would be to design the RVIC architecture such that interrupts could be presented using the GIC List Registers (LRs) and the GIC virtual CPU interface. However, that approach would have the following drawbacks:

- It would imply either the use of an active state, or using only INTIDs in the GIC LPI number space.
- Managing the LRs has shown to be error-prone and complex in some hypervisor implementations due to the duplicated and disconnected state in both the LRs and the virtual interrupt controller memory data structures.
- Supporting virtual level-triggered interrupts with the GIC LRs relies on receiving EOI maintenance interrupts which cannot necessarily be processed by the Trusted hypervisor unless it has built-in support for physical interrupt handling with the GIC, or can bear the performance cost of switching to the Untrusted hypervisor before re-sampling a level-triggered signal.

See also:

- GIC [1]
- 2.3.4 *Interrupt life-cycle*
- 2.3.5 *Trigger modes*

### 2.3.7 Multiple pending interrupts

I0053 A single interrupt can be acknowledged at a time.

I0054 If more than one interrupt is Unmasked and Pending, the order in which interrupts are presented upon acknowledgement is IMPLEMENTATION DEFINED.

Issue Consider whether prioritization of interrupts should be supported.

DRAFT

### 2.3.8 INTID assignments

I0055 INTIDs must be assigned such that the entity generating an interrupt uses the same ID as expected by in-VM software and without multiple sources using the same ID for different events.

R0056 The INTID number space is defined as follows:

- Trusted INTIDs: 0 to `NR_TRUSTED_INTERRUPTS - 1` (inclusive)
- Untrusted INTIDs: `NR_TRUSTED_INTERRUPTS` to `NR_TRUSTED_INTERRUPTS + NR_UNTRUSTED_INTERRUPTS - 1` (inclusive)

R0057 `NR_TRUSTED_INTERRUPTS` is a non-zero multiple of 32.

R0058 `NR_UNTRUSTED_INTERRUPTS` is a non-zero multiple of 32.

R0059 `NR_TRUSTED_INTERRUPTS + NR_UNTRUSTED_INTERRUPTS` is less than or equal to 2048.

R0060 All RVIC instances in a VM return the same value for `NR_TRUSTED_INTERRUPTS` via the `RVIC.Info` command.

R0061 All RVIC instances in a VM return the same value for `NR_UNTRUSTED_INTERRUPTS` via the `RVIC.Info` command.

R0062 Trusted interrupts generated by the Trusted hypervisor use Trusted INTIDs between 16 and 31 (inclusive).

R0063 Trusted interrupts generated by per-VPE peripherals use statically assigned INTIDs compliant with the PPI assignments in the Arm SBSA.

R0064 Trusted interrupts generated by VPEs (SGIs) use Trusted INTIDs allocated by in-VM software by choosing IDs that are not used by per-VPE peripherals.

S0065 Arm recommends that Trusted INTIDs are used to signal SGIs.

S0066 Arm recommends that `RVIC.Signal` is only used for Untrusted INTIDs when moving interrupt state between RVIC instances, for example when changing the target of an Untrusted interrupt.

I0067 INTIDs between 0 and 15 (inclusive) are reserved for software use for SGIs.

R0068 Untrusted interrupts use Untrusted INTIDs allocated by the Untrusted hypervisor.


Copyright © 2020 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I0069 The Untrusted hypervisor can inform in-VM software of its INTID assignments via virtual firmware tables such as ACPI or device tree.

R0070 Interrupt signals generated by the Untrusted hypervisor which use INTIDs outside the supported Untrusted INTID range do not change the state of any interrupts.

X0071 Arm is not aware of a reason to allow dynamic assignment of INTIDs for specific peripherals.

See also:

- SBSA [3]
- 2.3.9 *Interrupt routing*
- 2.4.3.2 *RVIC.Info*

### 2.3.9 Interrupt routing

In-VM software may need to change the destination RVIC for selected interrupts, for example to balance interrupt processing across several VPEs or to move interrupts away from a VPE when powering down that VPE. Selecting the target VPE of an interrupt is referred to as *interrupt routing*.

I0072 The RVIC architecture does not define any mechanism to configure interrupt routing. The mechanism used to re-program interrupt sources to specify a target VPE and target INTID is IMPLEMENTATION DEFINED. The Reduced Virtual Interrupt Distributor (RVID) can be used for this purpose.

I0073 The RVIC architecture allows in-VM software to use interrupt masking to enforce the requested routing, when the interrupt source configuration mechanism is implemented in the Untrusted hypervisor.

S0074 When in-VM software changes the target of an Untrusted interrupt from VPE A to VPE B, it is expected to follow a sequence of actions similar to the following:

1. Mask the interrupt on VPE A's RVIC instance (`RVIC.SetMasked`).
2. Configure the source to signal the interrupt to VPE B (IMPLEMENTATION DEFINED).
3. Read the interrupt Pending status on VPE A (`RVIC.IsPending`).
4. If the interrupt is Pending, re-trigger the interrupt on VPE B by signaling the interrupt to VPE B (`RVIC.Signal`).
5. Unmask the interrupt on VPE B's RVIC instance (`RVIC.ClearMasked`).

I0075 Masking an interrupt on a VPE that should not receive the interrupt, and unmasking the interrupt on the VPE that should receive the interrupt, ensures that software can trust the configured interrupt routing because the interrupt masking mechanism is implemented in the Trusted hypervisor.

X0076 Software outside the Trusted hypervisor can choose to ignore the desired routing and signal RVIC instances that are not configured to receive an interrupt, but these interrupts will never be signaled to the VM because they are Masked, and the end result is no different from the case where the Untrusted hypervisor never signals the interrupt. The RVIC architecture does not aim to provide mitigations against missing interrupt signals from untrusted sources.

See also:

- 2.4.3.5 *RVIC.SetMasked*
- 2.4.3.6 *RVIC.ClearMasked*
- 2.4.3.7 *RVIC.IsPending*
- 2.4.3.8 *RVIC.Signal*
- Chapter 3 *Reduced Virtual Interrupt Distributor (RVID)*

### 2.3.10 Notification of pending interrupts

There are situations where executing a command on VPE A's RVIC instance causes interrupts to become Pending and Unmasked on VPE B's RVIC instance. In such situations, the Trusted hypervisor may have to notify the Untrusted hypervisor that there are Pending and Unmasked interrupts on VPE B's RVIC instance. For example, scheduling of VPEs may be implemented only in the Untrusted hypervisor, and whether or not a VPE has any Pending and Unmasked interrupts may affect scheduling decisions.


*Copyright © 2020 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

DRAFT

$I_{0077}$ Interrupts can become Pending and Unmasked on VPE B's RVIC instance when executing one of the following commands on VPE A's RVIC instance:

- `RVIC.ClearMasked`
- `RVIC.Signal`

$R_{0078}$ An RVIC implementation in split-mode hypervisors must provide a notification mechanism which allows the Trusted hypervisor to inform the Unstrusted hypervisor that there are Pending and Unmasked interrupts on a VPE's enabled RVIC instance.

$R_{0079}$ The notification mechanism must uniquely identify which VPE has Pending and Unmasked interrupts.

$I_{0080}$ The notification mechanism is not required to identify the INTID of Pending and Unmasked interrupts.

$I_{0081}$ The RVIC architecture does not specify the details of a notification interface from the Trusted to the Untrusted hypervisor.

See also:

- 2.4.3.6 *RVIC.ClearMasked*
- 2.4.3.8 *RVIC.Signal*

DRAFT


*Copyright © 2020 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 2.4 RVIC programming interface

I0082 The RVIC is programmed from within the VM using a hypercall interface.

I0083 The RVIC exposes an external signaling interface via which the Untrusted hypervisor can signal Untrusted interrupts.

I0084 The external signaling interface provides a mechanism to raise an Untrusted interrupt using a specified INTID.

I0085 Other aspects of the external signaling interface are IMPLEMENTATION DEFINED.

### 2.4.1 AArch64 hypercall interface

I0086 The RVIC hypercall interface is compliant with the SMC Calling Convention (SMCCC) and uses the SMC64 or HVC64 calling convention.

R0087 A call to `SMCCC_ARCH_FEATURES` with `RVIC.Version` returns:

- `NOT_SUPPORTED` (-1) if RVIC is not implemented.
- `SUCCESS` (0) if RVIC is implemented.

S0088 If RVIC is implemented, a VM can call `RVIC.Version` to establish the implemented RVIC architecture version and make use of other commands defined in this specification.

I0089 On return from a command, the value in X0 is a return code which indicates command success or failure.

I0090 Conditions which can cause a command to fail are listed under “Failure conditions”.

R0091 Failure conditions are observed to be checked in the order in which they are listed.

R0092 If a failure condition is violated, the corresponding error code is returned in X0.

R0093 If a command succeeds, the values listed under “Output value” are returned.

R0094 If a command succeeds, the conditions listed under “Success conditions” are observable.

I0095 Command Function IDs are in the SMC64 Standard Hypervisor Service Calls range.

See also:

- SMCCC [4]
- 2.4.2.1 *CommandReturnCode*
- 2.4.3.1 *RVIC.Version*

DRAFT


*Copyright © 2020 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 2.4.2 RVIC types

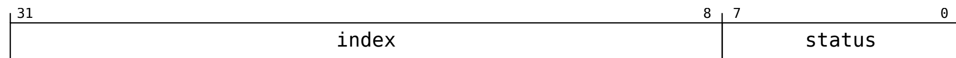
This section defines data types which are used by RVIC commands.

See also:

- 2.4.3 *RVIC commands*

### 2.4.2.1 CommandReturnCode

I0096 The *CommandReturnCode* type is used to report the result of an RVIC command.




D0097 The fields of the *CommandReturnCode* structure are shown in the following table.

| Name | Range | Description |
|---|---|---|
| `index` | [31:8] | Index which identifies the reason for a command failure |
| `status` | [7:0] | StatusCode of the command |

### 2.4.2.2 InfoKey

I0098 The *InfoKey* enumeration identifies a value which can be queried from the RVIC implementation.

D0099 The values of the *InfoKey* enumeration are shown in the following table.

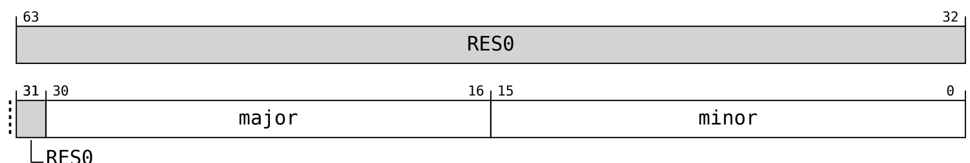
| Value | Name | Description |
|---|---|---|
| 0 | `NR_TRUSTED_INTERRUPTS` | Number of Trusted interrupts |
| 1 | `NR_UNTRUSTED_INTERRUPTS` | Number of Untrusted interrupts |

See also:

- 2.3.2 *Trusted and untrusted interrupts*
- 2.3.8 *INTID assignments*
- 2.4.3.2 *RVIC.Info*

### 2.4.2.3 InterfaceVersion

I0100 The *InterfaceVersion* type is used to report the version of an RVIC implementation.




D0101 The fields of the *InterfaceVersion* structure are shown in the following table.

| Name | Range | Description |
|---|---|---|
| `major` | [30:16] | Major version |


Copyright © 2020 Arm Limited or its affiliates. All rights reserved.
Non-confidential


DRAFT

| Name | Range | Description |
|---|---|---|
| `minor` | [15:0] | Minor version |

See also:

- 2.4.3.1 *RVIC.Version*

### 2.4.2.4 StatusCode

I0102 The *StatusCode* enumeration is used to report the success or failure of an RVIC command.

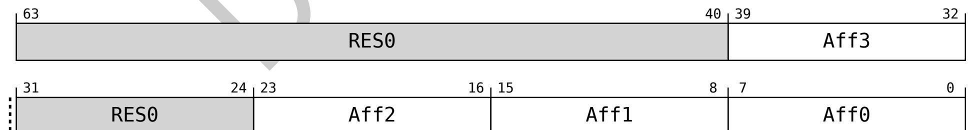
D0103 The values of the *StatusCode* enumeration are shown in the following table.

| Value | Name | Description |
|---|---|---|
| 0 | `STATUS_SUCCESS` | The command succeeded |
| 1 | `STATUS_ERROR_PARAMETER` | An argument is invalid |
| 2 | `STATUS_INVALID_VPE` | A target VPE argument does not match the MPIDR of any VPE |
| 3 | `STATUS_DISABLED` | The target RVIC instance is Disabled |
| 4 | `STATUS_NO_INTERRUPT` | No interrupt is pending on the target RVIC instance |

DRAFT

### 2.4.2.5 VPEId

I0104 The *VPEId* type is used to identify the VPE targeted by an RVIC command.




D0105 The fields of the *VPEId* structure are shown in the following table.

| Name | Range | Description |
|---|---|---|
| | [63:40] | Must be zero |
| `Aff3` | [39:32] | Aff3 of target core MPIDR |
| | [31:24] | Must be zero |
| `Aff2` | [23:16] | Aff2 of target core MPIDR |
| `Aff1` | [15:8] | Aff1 of target core MPIDR |
| `Aff0` | [7:0] | Aff0 of target core MPIDR |

X0106 The *VPEId* structure matches the `target_cpu` parameter used in PSCI.

See also:

- PSCI [5]


*Copyright © 2020 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 2.4.3 RVIC commands

### 2.4.3.1 RVIC.Version

Returns the version of the RVIC implementation.

***FID***

**Provisional**

Encodings will be added when the specification reaches Beta.

***Input arguments***

None

***Failure conditions***

None

***Output arguments***

| Name | Register | Type | Description |
|---|---|---|---|
| `version` | `X1` | *InterfaceVersion* | Version of the the RVIC implementation |

***Success conditions***

None

DRAFT


*Copyright © 2020 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

#### 2.4.3.2 RVIC.Info

Returns information about the RVIC implementation.

Given a key, this command returns the value for that key.

See also:

- 2.3.2 *Trusted and untrusted interrupts*
- 2.3.8 *INTID assignments*

***FID***

**Provisional**

Encodings will be added when the specification reaches Beta.

***Input arguments***

| Name | Register | Type | Description |
|---|---|---|---|
| `key` | `X1` | *InfoKey* | Selects the value to be queried |

***Failure conditions***

| Priority | Condition | Return code |
|---|---|---|
| 1 | `key` is invalid | `{ STATUS_ERROR_PARAMETER, 0 }` |

***Output arguments***

| Name | Register | Type | Description |
|---|---|---|---|
| `value` | `X1` | Unsigned integer | Value selected by `key` |

***Success conditions***

None

DRAFT


Copyright © 2020 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### 2.4.3.3 RVIC.Enable

Enables the RVIC instance for the current VPE.

See also:

- 2.3.1 *Instance enablement status*
- 2.4.3.4 *RVIC.Disable*

***FID***

**Provisional**

Encodings will be added when the specification reaches Beta.

***Input arguments***

None

***Failure conditions***

None

***Output arguments***

None

***Success conditions***

- The RVIC instance for the current VPE is Enabled.

DRAFT


Copyright © 2020 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### 2.4.3.4 RVIC.Disable

Disables the RVIC instance for the current VPE.

See also:

- 2.3.1 *Instance enablement status*
- 2.4.3.3 *RVIC.Enable*

***FID***

> **Provisional**
>
> Encodings will be added when the specification reaches Beta.

***Input arguments***

None

***Failure conditions***

None

***Output arguments***

None

***Success conditions***

- The RVIC instance for the current VPE is Disabled.

DRAFT


*Copyright © 2020 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

#### 2.4.3.5 RVIC.SetMasked

Masks an interrupt on an RVIC instance.

See also:

- 2.3.3 *Interrupt mask status*
- 2.4.3.6 *RVIC.ClearMasked*

***FID***

**Provisional**

Encodings will be added when the specification reaches Beta.

***Input arguments***

| Name | Register | Type | Description |
|---|---|---|---|
| `vpe` | `X1` | *VPEId* | Target VPE |
| `intid` | `X2` | Unsigned integer | Target INTID |

***Failure conditions***

| Priority | Condition | Return code |
|---|---|---|
| 1 | `vpe` is not a valid MPIDR encoding | `{ STATUS_ERROR_PARAMETER, 0 }` |
| 2 | `intid` is not a valid INTID | `{ STATUS_ERROR_PARAMETER, 1 }` |
| 3 | `vpe` does not match a VPE | `{ STATUS_INVALID_VPE, 0 }` |

DRAFT

***Output arguments***

None

***Success conditions***

- The target interrupt on the target RVIC instance is Masked.


*Copyright © 2020 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

#### 2.4.3.6 RVIC.ClearMasked

Unmasks an interrupt on an RVIC instance.

See also:

- 2.3.3 *Interrupt mask status*
- 2.4.3.5 *RVIC.SetMasked*

***FID***

**Provisional**

Encodings will be added when the specification reaches Beta.

***Input arguments***

| Name | Register | Type | Description |
|---|---|---|---|
| `vpe` | `X1` | *VPEId* | Target VPE |
| `intid` | `X2` | Unsigned integer | Target INTID |

***Failure conditions***

| Priority | Condition | Return code |
|---|---|---|
| 1 | `vpe` is not a valid MPIDR encoding | `{ STATUS_ERROR_PARAMETER, 0 }` |
| 2 | `intid` is not a valid INTID | `{ STATUS_ERROR_PARAMETER, 1 }` |
| 3 | `vpe` does not match a VPE | `{ STATUS_INVALID_VPE, 0 }` |

DRAFT

***Output arguments***

None

***Success conditions***

- The target interrupt on the target RVIC instance is Unmasked.


Copyright © 2020 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### 2.4.3.7 RVIC.IsPending

Queries whether an interrupt is Pending on an RVIC instance.

I0107 The value returned is independent of whether the interrupt is Masked or Unmasked.

See also:

- 2.3.4 *Interrupt life-cycle*

***FID***

**Provisional**

Encodings will be added when the specification reaches Beta.

***Input arguments***

| Name | Register | Type | Description |
|---|---|---|---|
| vpe | X1 | *VPEId* | Target VPE |
| intid | X2 | Unsigned integer | Target INTID |

***Failure conditions***

| Priority | Condition | Return code |
|---|---|---|
| 1 | vpe is not a valid MPIDR encoding | { STATUS_ERROR_PARAMETER, 0 } |
| 2 | intid is not a valid INTID | { STATUS_ERROR_PARAMETER, 1 } |
| 3 | vpe does not match a VPE | { STATUS_INVALID_VPE, 0 } |

***Output arguments***

| Name | Register | Type | Description |
|---|---|---|---|
| status | X1 | Bit | Activity status of the target interrupt on the target RVIC instance<br>• 0 : Idle<br>• 1 : Pending |

***Success conditions***

None

DRAFT


Copyright © 2020 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### 2.4.3.8 RVIC.Signal

Signal an interrupt to an RVIC instance.

I0108 This command can be used with any valid INTID.

I0109 This command can be used to implement SGIs.

S0110 Arm strongly recommends that software does not use INTIDs used for any other purpose (such as virtual peripherals) because the RVIC architecture does not provide a mechanism to disambiguate which source generated a signal.

See also:

- 2.3.4 *Interrupt life-cycle*
- 2.3.6 *Signaling*

***FID***

**Provisional**

Encodings will be added when the specification reaches Beta.

***Input arguments***

| Name | Register | Type | Description |
|---|---|---|---|
| `vpe` | `X1` | *VPEId* | Target VPE |
| `intid` | `X2` | Unsigned integer | Target INTID |

***Failure conditions***

| Priority | Condition | Return code |
|---|---|---|
| 1 | `vpe` is not a valid MPIDR encoding | `{ STATUS_ERROR_PARAMETER, 0 }` |
| 2 | `intid` is not a valid INTID | `{ STATUS_ERROR_PARAMETER, 1 }` |
| 3 | `vpe` does not match a VPE | `{ STATUS_INVALID_VPE, 0 }` |
| 4 | Target RVIC instance is Disabled | `{ STATUS_DISABLED, 0 }` |

***Output arguments***

None

***Success conditions***

- The target interrupt on the target RVIC instance is Pending.

DRAFT


Copyright © 2020 Arm Limited or its affiliates. All rights reserved.
Non-confidential

#### 2.4.3.9 RVIC.ClearPending

Clears the Pending state of an interrupt on an RVIC instance.

See also:

- 2.3.4 *Interrupt life-cycle*

***FID***

**Provisional**

Encodings will be added when the specification reaches Beta.

***Input arguments***

| Name | Register | Type | Description |
|---|---|---|---|
| `vpe` | `X1` | *VPEId* | Target VPE |
| `intid` | `X2` | Unsigned integer | Target INTID |

***Failure conditions***

| Priority | Condition | Return code |
|---|---|---|
| 1 | `vpe` is not a valid MPIDR encoding | `{ STATUS_ERROR_PARAMETER, 0 }` |
| 2 | `intid` is not a valid INTID | `{ STATUS_ERROR_PARAMETER, 1 }` |
| 3 | `vpe` does not match a VPE | `{ STATUS_INVALID_VPE, 0 }` |

DRAFT

***Output arguments***

None

***Success conditions***

- The target interrupt on the target RVIC instance is Idle.

$I_{0111}$ Observability of the Idle status is not guaranteed.

$X_{0112}$ The Idle status may not be observable if the target interrupt is signaled concurrently with execution of this command.


*Copyright © 2020 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

#### 2.4.3.10 RVIC.Acknowledge

Acknowledge a pending interrupt.

See also:

- 2.3.4 *Interrupt life-cycle*
- 2.3.7 *Multiple pending interrupts*

***FID***

**Provisional**

Encodings will be added when the specification reaches Beta.

***Input arguments***

None

***Failure conditions***

| Priority | Condition | Return code |
|---|---|---|
| 1 | There are no interrupts which are both Unmasked and Pending on the RVIC instance for the current VPE | `{ STATUS_NO_INTERRUPTS, 0 }` |
| 2 | RVIC instance for the current VPE is Disabled | `{ STATUS_DISABLED, 0 }` |

$I_{0113}$ `STATUS_NO_INTERRUPTS` may result from either:

- A Pending interrupt was Masked after the RVIC instance signaled an interrupt to the VPE.
- A VPE processes multiple interrupts on a single IRQ exception by calling this command in a loop until there are no further Unmasked and Pending interrupts.

***Output arguments***

| Name | Register | Type | Description |
|---|---|---|---|
| `intid` | `X1` | Unsigned integer | INTID of an Unmasked and Pending interrupt |

***Success conditions***

- The interrupt identified by `intid` is Idle.
- The interrupt identified by `intid` is Masked.

DRAFT


*Copyright © 2020 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

#### 2.4.3.11 RVIC.Resample

Resample the signal of an interrupt.

S0114 A VPE processing a Pending level-triggered interrupt should perform this call after programming the source of the interrupt to determine whether the interrupt signal is still asserted.

See also:

- 2.3.2 *Trusted and untrusted interrupts*
- 2.3.5 *Trigger modes*
- 2.3.8 *INTID assignments*

***FID***

**Provisional**

Encodings will be added when the specification reaches Beta.

***Input arguments***

| Name | Register | Type | Description |
|---|---|---|---|
| `intid` | `X1` | Unsigned integer | Target INTID |

***Failure conditions***

| Priority | Condition | Return code |
|---|---|---|
| 1 | `intid` is not a Trusted INTID | `{ STATUS_ERROR_PARAMETER, 0 }` |

DRAFT

***Output arguments***

None

***Success conditions***

- If the source interrupt signal is asserted, the interrupt on the RVIC instance of the calling VPE is Pending.


Copyright © 2020 Arm Limited or its affiliates. All rights reserved.
Non-confidential

# Chapter 3
# Reduced Virtual Interrupt Distributor (RVID)

DRAFT

The *Reduced Virtual Interrupt Distributor* (RVID) is a PV interrupt distributor which allows software in a VM to map virtual interrupts generated by virtual sources (such as an emulated I/O device) to specific VPEs.


*Copyright © 2020 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 3.1 RVID introduction

The RVID architecture supports a variety of hypervisor designs, including both monolithic hypervisors and split-mode hypervisors. Split-mode hypervisors are implemented across two or more levels of privilege (different Exception levels in the same security state, or different security states), resulting in the hypervisor functionality being divided into trusted and untrusted parts. This specification uses the term *Trusted hypervisor* to refer to the trusted part of the hypervisor and the term *Untrusted hypervisor* to refer to the untrusted part of the hypervisor. For monolithic hypervisor designs, these two terms refer to the same software component.

The RVID is designed to be implemented in the Untrusted hypervisor when using a split-mode hypervisor design, and relies on additional controls in the Trusted hypervisor to further enforce the routing requested by in-VM software.

The RVID is designed to work in conjunction with the Reduced Virtual Interrupt Controller (RVIC), but can work with any virtual interrupt controller architecture which allows injection of interrupts to a specific VPE.

The RVID allows in-VM software to control the mapping from a signal produced by an interrupt source (described to the VM via firmwware description tables such as ACPI or device tree) to a destination VPE and VPE-local *Interrupt ID* (INTID) value.

See also:

- Chapter 2 *Reduced Virtual Interrupt Controller (RVIC)*

## 3.2 RVID architecture version

I0115 The version of the RVID architecture is 0.3.

I0116 The ABI is binary compatible across minor version number changes but not necessarily across major version number changes.

See also:

- 3.4.3.1 *RVID.Version*

DRAFT


Copyright © 2020 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 3.3 RVID functional description

$D_{0117}$ An *Input* is a signal which can be produced by an interrupt source.

$D_{0118}$ A *Target* is the tuple of a VPE and an INTID value.

$I_{0119}$ The RVID exposes a hypercall interface which allows in-VM software to map an Input to a Target, or to unmap an Input from a Target.

$I_{0120}$ An Input is either:

- unmapped, or
- mapped to exactly one Target

$I_{0121}$ When an Input signal arrives at the the RVID, if the Input is unmapped, the event is ignored and neither recorded nor signaled to any entity.

$R_{0122}$ When an Input signal arrives at the the RVID, if the Input is mapped, the RVID signals the Target.

$I_{0123}$ An Input which is already mapped to a Target can be mapped to a different Target without losing interrupt signals.

$I_{0124}$ When an Input is signaled concurrently with changing the Target, it is either signaled to the old Target or to the new Target.

$I_{0125}$ The RVID is a separate component from the per-VPE virtual interrupt controller and does not maintain states of interrupts.

$I_{0126}$ Changing the Target of an interrupt which has already been signaled does not cause the interrupt to be signaled to the new Target.

$X_{0127}$ It is complicated to support automatic re-signaling of interrupts and in-VM software cannot rely on this functionality when the RVID implementation is in the Untrusted hypervisor. Instead, in-VM software can detect that the interrupt is in the Pending state on the old Target after changing the Target, and re-trigger the interrupt on the new Target. This paradigm is used by other (hardware) interrupt controllers.

$R_{0128}$ When an Input signal arrives at the RVID after changing the Target, the RVID signals the new Target irrespective of the state of the interrupt on the previous Target. This means that an interrupt can be in the Pending state on two VPEs at the same time.

$S_{0129}$ In-VM software must handle the possibility of an interrupt being in the Pending state on two VPEs at the same time through the use of synchronization mechanisms and the features of the virtual interrupt controller.

$R_{0130}$ When a VM is created or reset, all Inputs become unmapped from it.

$S_{0131}$ Arm strongly recommends that software never creates a mapping from two separate Inputs to the same Target.

DRAFT


Copyright © 2020 Arm Limited or its affiliates. All rights reserved.
Non-confidential

## 3.4 RVID programming interface

$I_{0132}$ The RVIC is programmed from within the VM using a hypercall interface.

$I_{0133}$ The RVIC exposes an external signaling interface via which devices can generate interrupts.

$I_{0134}$ The signaling interface is IMPLEMENTATION DEFINED and can be tightly integrated with the hypervisor component where the RVID is implemented.

$I_{0135}$ The signaling interface provides a mechanism for other software components, such as emulated peripherals, to raise interrupt signals to the RVID.

$I_{0136}$ The presence of the RVID is described in firmware tables (such as ACPI or device tree).

### 3.4.1 AArch64 hypercall interface

$I_{0137}$ The RVID hypercall interface is compliant with the SMC Calling Convention (SMCCC) and uses the SMC64 or HVC64 calling convention.

$I_{0138}$ On return from a command, the value in X0 is a status code which indicates command success or failure.

$I_{0139}$ Conditions which can cause a command to fail are listed under “Failure conditions”.

$R_{0140}$ Failure conditions are observed to be checked in the order in which they are listed.

$R_{0141}$ If a failure condition is violated, the corresponding error code is returned in X0.

$R_{0142}$ If a command succeeds, the values listed under “Output value” are returned.

$R_{0143}$ If a command succeeds, the conditions listed under “Success conditions” are observable.

$I_{0144}$ Command Function IDs are in the SMC64 Standard Hypervisor Service Calls range.

See also:

- SMCCC [4]
- 3.4.2.1 *CommandReturnCode*

DRAFT


*Copyright © 2020 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

## 3.4.2 RVID types

This section defines data types which are used by RVID commands.

See also:

- 3.4.3 *RVID commands*

### 3.4.2.1 CommandReturnCode

I0145 The *CommandReturnCode* type is used to report the result of an RVID command.

| 31 – 8 | 7 – 0 |
|---|---|
| index | status |

D0146 The fields of the *CommandReturnCode* structure are shown in the following table.

| Name | Range | Description |
|---|---|---|
| `index` | [31:8] | Index which identifies the reason for a command failure |
| `status` | [7:0] | StatusCode of the command |

### 3.4.2.2 InterfaceVersion

I0147 The *InterfaceVersion* type is used to report the version of an RVID implementation.

| 63 – 32 |
|---|
| RES0 |

| 31 | 30 – 16 | 15 – 0 |
|---|---|---|
| RES0 | major | minor |

D0148 The fields of the *InterfaceVersion* structure are shown in the following table.

| Name | Range | Description |
|---|---|---|
| `major` | [30:16] | Major version |
| `minor` | [15:0] | Minor version |

See also:

- 3.4.3.1 *RVID.Version*

### 3.4.2.3 StatusCode

I0149 The *StatusCode* enumeration is used to report the success or failure of an RVID command.

D0150 The values of the *StatusCode* enumeration are shown in the following table.

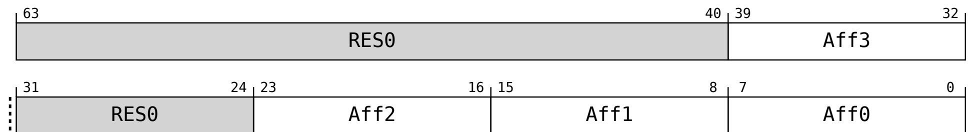
| Value | Name | Description |
|---|---|---|
| 0 | `STATUS_SUCCESS` | The command succeeded |
| 1 | `STATUS_ERROR_PARAMETER` | An argument is invalid |


*Copyright © 2020 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

| Value | Name | Description |
|---|---|---|
| 2 | STATUS_INVALID_VPE | A target VPE argument does not match the MPIDR of any VPE |

#### 3.4.2.4 VPEId

I0151 The *VPEId* type is used to identify the VPE targeted by an RVID command.




D0152 The fields of the *VPEId* structure are shown in the following table.

| Name | Range | Description |
|---|---|---|
| | [63:40] | Must be zero |
| Aff3 | [39:32] | Aff3 of target core MPIDR |
| | [31:24] | Must be zero |
| Aff2 | [23:16] | Aff2 of target core MPIDR |
| Aff1 | [15:8] | Aff1 of target core MPIDR |
| Aff0 | [7:0] | Aff0 of target core MPIDR |

DRAFT

X0153 The *VPEId* structure matches the `target_cpu` parameter used in PSCI.

See also:

- PSCI [5]


Copyright © 2020 Arm Limited or its affiliates. All rights reserved.
Non-confidential

### 3.4.3 RVID commands

#### 3.4.3.1 RVID.Version

Returns the version of the RVID implementation.

***FID***

**Provisional**

Encodings will be added when the specification reaches Beta.

***Input arguments***

None

***Output arguments***

| Name | Register | Type | Description |
|---|---|---|---|
| `version` | `X1` | *InterfaceVersion* | Version of the the RVID implementation |

***Failure conditions***

None

***Success conditions***

None

DRAFT


*Copyright © 2020 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

#### 3.4.3.2 RVID.Map

Maps an Input to a Target.

See also:

- 3.4.3.3 *RVID.Unmap*

***FID***

**Provisional**

Encodings will be added when the specification reaches Beta.

***Input arguments***

| Name | Register | Type | Description |
|---|---|---|---|
| input_intid | X1 | Unsigned integer | Input INTID, as specified to in-VM software via firmware tables |
| target_vpe | X2 | *VPEId* | Target VPE |
| target_intid | X3 | Unsigned integer | Target INTID |

DRAFT

***Failure conditions***

| Priority | Condition | Return code |
|---|---|---|
| 1 | input_intid is not a valid RVID input INTID | { STATUS_ERROR_PARAMETER, 0 } |
| 2 | target_vpe is not a valid MPIDR encoding | { STATUS_ERROR_PARAMETER, 1 } |
| 3 | target_vpe does not match a VPE | { STATUS_INVALID_VPE, 0 } |
| 4 | target_intid is not a valid INTID on the interrupt controller belonging to target_vpe | { STATUS_ERROR_PARAMETER, 2 } |

***Output arguments***

None

***Success conditions***

- If the specified Input was previously mapped, the mapping to the old Target is removed.
- The specified Input is mapped to the specified Target.


*Copyright © 2020 Arm Limited or its affiliates. All rights reserved.*
*Non-confidential*

#### 3.4.3.3 RVID.Unmap

Unmaps an Input.

See also:

- 3.4.3.2 *RVID.Map*

***FID***

**Provisional**

Encodings will be added when the specification reaches Beta.

***Input arguments***

| Name | Register | Type | Description |
|---|---|---|---|
| `input_intid` | `X1` | Unsigned integer | Input INTID, as specified to in-VM software via firmware tables |

***Failure conditions***

| Priority | Condition | Return code |
|---|---|---|
| 1 | `input_intid` is not a valid RVID input INTID | `{ STATUS_ERROR_PARAMETER, 0 }` |

DRAFT

***Output arguments***

None

***Success conditions***

- The specified Input is not mapped to any Target.


Copyright © 2020 Arm Limited or its affiliates. All rights reserved.
Non-confidential

# Glossary

**ABI**
: Application Binary Interface

**ACPI**
: Advanced Configuration and Power Interface

**EOI**
: End Of Interrupt

**FID**
: Function Identifier

**GIC**
: Generic Interrupt Controller [1]

**INTID**
: Interrupt Identifier

**IRQ**
: Interrupt ReQuest

**LPI**
: Locality-specific Peripheral Interrupt

DRAFT

**OS**
: Operating System

**PMU**
: Performance Management Unit

**PPI**
: Private Peripheral Interrupt

**PSCI**
: Power State Coordination Interface [5]

**PV**
: Para-virtualized

**RoT**
: Root of Trust

**RVIC**
: Reduced Virtual Interrupt Controller

**RVID**
: Reduced Virtual Interrupt Distributor

**SBSA**

Server Base System Architecture [3]

**SGI**

Software-generated Interrupt

**SMCCC**

Secure Monitor Call (SMC) Calling Convention [4]

**VM**

Virtual Machine

**VPE**

Virtual Processing Element.

DRAFT


Copyright © 2020 Arm Limited or its affiliates. All rights reserved.
Non-confidential