arm

Release notes for Arm Compiler for Embedded 6.18

1 Introduction

Arm Compiler for Embedded is the most advanced embedded C/C++ compilation toolchain from Arm for the development of bare-metal software, firmware, and Real-Time Operating System (RTOS) applications.

Arm Compiler for Embedded provides the earliest, most complete, and most accurate support for the latest architectural features and extensions of the Arm architecture. It supports all the latest Arm processors, including those in development. Through powerful optimization techniques and optimized libraries, Arm Compiler for Embedded enables embedded system developers to meet challenging performance goals and memory constraints.

Arm Compiler for Embedded is used by leading companies in a wide variety of industries, including automotive, consumer electronics, industrial, medical, networking, railway, storage, and telecommunications. If your project has long-term maintenance requirements or functional safety requirements (such as EN 50128, IEC 61508, IEC 62304, and ISO 26262), Arm strongly recommends that you consider a LTS and qualified version of Arm Compiler for Embedded FuSa instead of this release.

Arm Compiler for Embedded 6.18 is the latest release as of March 2022 and supersedes every previous release.

The key features of Arm Compiler for Embedded 6.18 include support for:

- Armv9.3-A
- Armv8.8-A
- The A-profile Hinted Conditional Branches Extension
- The A-profile Memory Operations Extension
- Features to enable or disable support for the optional Armv8.4-A additions to the AArch64 Performance Monitors Extension
- Automatic vectorization for SVE and SVE2
- The Cortex-X1C processor
- Using the toolchain on AArch64 Ubuntu Desktop Edition 20.04 LTS host platforms
- User-based licensing

1.1 Arm Compiler for Embedded 6.18 Configuration

Arm Compiler for Embedded 6.18 consists of:

- Toolchain components:
  - **armclang:** Compiler and integrated assembler based on LLVM and Clang technology
  - **armar:** Archiver which enables sets of ELF object files to be collected together

    - **armlink:** Linker that combines objects and libraries to produce an executable
    - **fromelf:** Image conversion utility and dissembler
    - **armasm:** Legacy assembler for armasm-syntax assembly code for older Arm architectures only. Use the armclang integrated assembler for all new assembly files.
    - **Arm C Libraries:** Runtime support libraries for embedded systems
    - **Arm C++ Libraries:** Libraries based on the LLVM libc++ project
- User documentation:
    - **User Guide:** Provides examples and guides to help you use the toolchain
    - **Reference Guide:** Provides information to help you configure the toolchain
    - **Arm C and C++ Libraries and Floating-Point Support User Guide:** Provides information about the Arm libraries and floating-point support
    - **Errors and Warnings Reference Guide:** Provides a list of the errors and warnings that tools in Arm Compiler for Embedded can report
    - **Migration and Compatibility Guide:** Provides information to help you migrate from Arm Compiler 5 to Arm Compiler for Embedded
- Release notes

The toolchain can be used:

- With an Arm Development Studio toolkit
- With a Keil MDK toolkit
- As a standalone installation

Contact your sales representative or visit https://developer.arm.com/buy-arm-products to enquire about any of these products. Visit https://developer.arm.com/support/licensing to manage your licenses or access troubleshooting resources.

1.2 What's Supported in Arm Compiler for Embedded 6.18?

Subject to your license terms, Arm Compiler for Embedded 6.18 can be used to build for the following Arm Architectures and Processors:

Architecture	Cortex		Neoverse	Other
	Standard	Automotive Enhanced		
Armv9-A up to Armv9.3-A	X2 A710, A510			
Armv8-A up to Armv8.8-A	X1C, X1 A78C, A78, A77, A76, A75, A73, A72 A65 A57, A55, A53 A35, A34, A32	A78AE, A76AE A65AE	V1 N2, N1 E1	
Armv7-A	A17, A15, A12 A9, A8, A7, A5			
Armv8-R AArch64 [BETA]	R82 [BETA]			
Armv8-R	R52+, R52			
Armv7-R	R8, R7, R5, R4F, R4			
Arvm8-M up to Armv8.1-M	M55 M35P, M33 M23			STAR-MC1
Armv7-M	M7, M4, M3			SecurCore SC300
Armv6-M	M1, M0, M0+			SecurCore SC000

For more information about the level of support for the Architectures and Processors, refer to the relevant IDE documentation:

- Arm Development Studio: https://developer.arm.com/tools-and-software/embedded/arm-development-studio
- Keil MDK: https://developer.arm.com/tools-and-software/embedded/keil-mdk

For more information about [COMMUNITY], [ALPHA], and [BETA] features, refer to the *Support level definitions* section of the *User Guide*.

Support for certain Architectures and Processors is available only as part of Arm Development Studio Platinum Edition. Arm Development Studio Platinum Edition is reserved for Arm partners developing the latest IP before it becomes available in devices. It includes all the features of Arm Development Studio Gold Edition, and additionally has support for the latest announced IP from Arm. Please contact Arm for more information.

2 Installation and integration into IDEs

Arm Compiler for Embedded 6.18 is supported on the following host architectures and host operating systems:

Host architecture	Host operating system	Toolchain download package	Supported use cases
x86_64	Red Hat Enterprise Linux 7 Red Hat Enterprise Linux 8 Ubuntu Desktop Edition 18.04 LTS Ubuntu Desktop Edition 20.04 LTS	x86_64 Linux	Standalone installation Integrated into Arm Development Studio
	Windows Server 2012 Windows Server 2016 Windows Server 2019 Windows 8.1 Windows 10	x86_64 Windows	Standalone installation Integrated into Arm Development Studio
		x86_64 Windows for Keil MDK	Integrated into Keil MDK
AArch64	Ubuntu Desktop Edition 20.04 LTS	AArch64 Linux	Standalone installation

Note:

- Arm Compiler for Embedded 6.18 is not expected to work on:
  - Host operating system platforms older than the ones listed above.
  - Linux host operating system platforms with a version of glibc older than 2.15.
- With a Keil MDK license, Arm Compiler for Embedded 6.18 is supported on Windows host platforms only.
- x86_32 host platforms are not supported.
- If you are using a FlexNet Publisher floating license, your license server must be running version 11.14.1.0 or later of both armlmd and lmgrd. Arm recommends that you always use the latest version of the license server software that is available from https://developer.arm.com/products/software-development-tools/license-management/downloads

If you received Arm Compiler for Embedded 6.18 as part of a toolkit, for example Arm Development Studio, the toolkit installer takes care of the installation process. Please refer to the installation instructions for the toolkit in such cases.

For all other cases, you must select an appropriate installation location depending on how you intend to use Arm Compiler for Embedded 6.18:

- Integrated into Arm Development Studio Bronze / Silver / Gold Edition 2021.2 or later
- Integrated into Arm Development Studio Platinum Edition 2021.c or later
- Integrated into Keil MDK 5.36 or later
- As a standalone installation

Please refer to the relevant instructions from the following:

- Installation instructions: The *Installing Arm Compiler* section of the *User Guide*
- FlexNet Publisher license and toolkit configuration instructions: https://developer.arm.com/tools-and-software/software-development-tools/license-management/resources/product-and-toolkit-configuration

- User-based licensing documentation: https://lm.arm.com
- Instructions for integration into Arm Development Studio:
    0. Install the toolchain into any directory except an Arm Development Studio installation directory.
    1. Configure your project to use the toolchain by following the instructions in the *Register a compiler toolchain* section of the *Arm Developer Studio User Guide*.
- Instructions for integration into Keil MDK:
    0. Install the toolchain into the `ARM` subdirectory of the Keil MDK installation directory. For example, if your Keil MDK installation directory is in `C:\Keil_v5`, the recommended installation path is `C:\Keil_v5\ARM\ARMCompiler6.18`
    1. Configure your project to use the toolchain by following the instructions at http://www.keil.com/support/man/docs/uv4/uv4_armcompilers.htm

3 Feedback and Support

Your feedback is important to us, and you are welcome to send us defect reports and suggestions for improvement on any aspect of the product. Defect fixes and enhancements will be considered for a future release according to the Arm Compiler for Embedded maintenance policy. For projects that have long-term maintenance or functional safety requirements, please consider using a LTS and qualified version of Arm Compiler for Embedded FuSa.

Please contact your supplier or visit https://developer.arm.com/support and open a case with feedback or support issues, using your work or academic email address if possible. Where appropriate, please provide the following information:

- `--vsn` output from the tool.
- The complete content of any error message that the tool produces.
- Preprocessed source code, other files, and command-line options necessary to reproduce the issue. For information on how to preprocess source code, refer to the *-E* section of the *Reference Guide*.

4 Changes in Arm Compiler for Embedded 6.18

Below is a summary of enhancements and defect fixes in Arm Compiler for Embedded 6.18.

Arm Compiler for Embedded 6.18 is not a Long-Term Support (LTS) release. Further defect fixes and enhancements will be considered for a future release according to the Arm Compiler for Embedded maintenance policy.

The information below may include technical inaccuracies or typographical errors. Each itemized change is accompanied by a unique SDCOMP-<n> identifier. If you need to contact Arm about a specific issue within these release notes, please quote the appropriate identifier.

Changes are listed since the previous feature release, Arm Compiler for Embedded 6.17.

General changes in Arm Compiler for Embedded 6.18

- [SDCOMP-60274] Support has been added for the Armv9.3-A architecture. To target Armv9.3-A, select from the following armclang options:

State	Options
AArch64	--target=aarch64-arm-none-eabi -march=armv9.3-a
AArch32	--target=arm-arm-none-eabi -march=armv9.3-a

- [SDCOMP-60273] Support has been added for the Armv8.8-A architecture. To target Armv8.8-A, select from the following armclang options:

State	Options
AArch64	--target=aarch64-arm-none-eabi -march=armv8.8-a
AArch32	--target=arm-arm-none-eabi -march=armv8.8-a

- [SDCOMP-60265] Previously, the tools used the name Star for the STAR-MC1 processor. This has been changed. To target STAR-MC1, select from the following options:

DSP	Floating-point	armclang	armasm, armlink and fromelf
Included	Included	--target=arm-arm-none-eabi -mcpu=star-mc1	--cpu=Star-MC1
Included	Not included	--target=arm-arm-none-eabi -mcpu=star-mc1 -mfloat-abi=soft	--cpu=Star-MC1.no_fp
Not included	Included	--target=arm-arm-none-eabi -mcpu=star-mc1+nodsp	--cpu=Star-MC1.no_dsp
Not included	Not included	--target=arm-arm-none-eabi -mcpu=star-mc1+nodsp -mfloat-abi=soft	--cpu=Star-MC1.no_dsp.no_fp

- [SDCOMP-60138] Previously, the compiler did not ignore __attribute__((packed)) for C++ non-POD types. This behavior has been changed. The compiler now ignores __attribute__((packed)) for C++ non-POD types.
- [SDCOMP-60121] Support for the following options has been removed from the Arm Compiler for Embedded installer for Linux host platforms:
  - --gunzip
  - --pager
  - --tar
- [SDCOMP-60066] Automatic vectorization for the following optional features is now supported:
  - The Armv8-A Scalable Vector Extension (SVE).
  - The Armv9-A Scalable Vector Extension version 2 (SVE2).

  For more information, refer to the *-fvectorize*, *-fno-vectorize* section of the *Reference Guide*.

- [SDCOMP-60044] Arm Compiler for Embedded is now supported on AArch64 Ubuntu Desktop Edition 20.04 LTS host operating systems. To use it on such systems, download the AArch64 Linux package.
- [SDCOMP-59863] Support has been added for the A-profile Hinted Conditional Branches Extension. To target a specific feature set configuration of the extension, select from the following options:

Base Architecture	Default
Armv9.3-A	Extension enabled
Armv9.2-A	Extension disabled
Armv8.8-A	Extension enabled
Armv8.7-A	Extension disabled

  - To enable the extension when it is disabled by default, compile with the +hbc -march or -mcpu feature.
  - To disable the extension when it is enabled by default, compile with the +nohbc -march or -mcpu feature.

For more information, refer to the *-march* and *-mcpu* sections of the *Reference Guide*.

- [SDCOMP-59736] Support has been removed from the Arm C++ libraries for the *Pointer safety [util.dynamic.safety]* feature of C++11 and later source language modes. The std::pointer_safety type and the following pointer safety functions are no longer supported:
  - std::declare_no_pointers()
  - std::declare_reachable()
  - std::get_pointer_safety()
  - std::undeclare_no_pointers()
  - std::undeclare_reachable()
- [SDCOMP-59400] Support has been added for the A-profile Memory Operations Extension. To target a specific feature set configuration of the extension, select from the following options:

Base Architecture	Default
Armv9.3-A	Extension enabled
Armv9.2-A	Extension disabled
Armv8.8-A	Extension enabled
Armv8.7-A	Extension disabled

  - To enable the extension when it is disabled by default, compile with the +mops -march or -mcpu feature.
  - To disable the extension when it is enabled by default, compile with the +nomops -march or -mcpu feature.

For more information, refer to the *-march* and *-mcpu* sections of the *Reference Guide*.

- [SDCOMP-59207] The following linker warning has been downgraded to a remark:
  - L6413W: Disabling merging for <object>(<section>), Section contains misaligned string(s)
- [SDCOMP-58875] The following linker warning has been downgraded to a remark:
  - L6440W: Disabling merging for <object>(<string_section>), unsupported relocation R_ARM_REL32 from <object>(<other_section>) to STT_SECTION type symbol
- [SDCOMP-58814] Support has been added for the +pmuv3 and +nopmuv3 features for the -march and -mcpu options. These features enable or disable support for the optional Armv8.4-A additions to the AArch64 Performance Monitors Extension. To configure support for these additions, select from the following options:

Target	Default	+pmuv3	+nopmuv3
Armv8-A, Armv9-A, Armv8-R architectures	Disabled	Enabled	Disabled
Armv8-A, Armv9-A, Armv8-R processors	Enabled	Enabled	Disabled

- The features only control code generation for calls to __builtin_readcyclecounter() for C/C++ source code.
- Note that the default behavior has been changed since Arm Compiler for Embedded 6.17 for certain architectures and processors.
- For more information, refer to the *-march* and *-mcpu* sections of the *Reference Guide*.

Enhancements in Arm Compiler for Embedded 6.18

armclang

- [SDCOMP-59790] Support has been added for the Cortex-X1C processor. To target Cortex-X1C, select from the following armclang options:

State	Cryptographic Extension	Options
AArch64	Included	--target=aarch64-arm-none-eabi -mcpu=cortex-x1c
AArch64	Not included	--target=aarch64-arm-none-eabi -mcpu=cortex-x1c+nocrypto
AArch32	Included	--target=arm-arm-none-eabi -mcpu=cortex-x1c -mfpu=crypto-neon-fp-armv8
AArch32	Not included	--target=arm-arm-none-eabi -mcpu=cortex-x1c -mfpu=neon-fp-armv8

- [SDCOMP-59314] Support has been added for the -ftrivial-auto-var-init option to control the initialization of trivial automatic variables.

  For more information, refer to the *-ftrivial-auto-var-init* section of the *Reference Guide*.

- [SDCOMP-57657] Previously, when compiling in a C++14 or later source language mode, the -fno-sized-deallocation option was selected by default. This behavior has been changed. In these circumstances, -fsized-deallocation is now selected by default.
- [SDCOMP-57610] Support has been added for the -faggressive-jump-threading and -fno-aggressive-jump-threading options to enable or disable the Aggressive Jump Threading (AJT) optimization.

  For more information, refer to the *-faggressive-jump-threading*, *-fno-aggressive-jump-threading* section of the *Reference Guide*.

- [SDCOMP-57588] Issue G.a of the *Arm Architecture Reference Manual Armv8, for Armv8-A architecture profile* revoked the partial deprecation of IT blocks. The compiler and integrated assembler have been updated to match this change.

  Previously, when assembling for an Armv8-A or Armv9-A target, and T32 state, the inline assembler and integrated assembler reported one of the following warnings for an IT block that contains a deprecated instruction:

  - applying IT instruction to more than one subsequent instruction is deprecated
  - deprecated instruction in IT block

  This behavior has been changed. The inline assembler and integrated assembler do not report warnings for an IT block that contains a formerly-deprecated instruction.

  Additionally, the compiler can now generate code that contains an IT block with a formerly-deprecated instruction. Support has been added for the -mrestrict-it and -mno-restrict-it options to control this behavior.

For more information, refer to the *-mrestrict-it, -mno-restrict-it* section of the *Reference Guide*.

armlink

- [SDCOMP-57665] Support has been added for the --require_bti and --info=bti options.

  Previously, when linking without --library_security=pacbti-m, and if at least one but not all input objects have been built with M-profile PACBTI Extension branch protection features, the linker reported the following error:

  - Cannot link object <non_BTI_object> as its attributes are incompatible with the image attributes. ... BTI co mpatible clashes with BTI incompatible.

  This behavior has been changed to the following:

--require_bti	Linker behavior
Included	Reports the following error: ○ L6111E: Composition of BTI and non-BTI objects detected. Use --info=bti to print out the list of objects with their corresponding BTI mark
Not included	Reports the following warning: ○ L6110W: Composition of BTI and non-BTI objects detected. The PACBTI-M library variant has been selected. Use --info=bti to print out the list of objects with their corresponding BTI mark

  For more information, refer to the following sections of the *Reference Guide*:

  - *--info=topic[,topic,...] (armlink)*
  - *--library_security=protection*
  - *--require_bti*

Libraries and system headers

- [SDCOMP-58721] The __ARM_TPL_condvar_monotonic_timedwait() function has been added to the [ALPHA] Arm C++ library Thread-Porting Layer (TPL).

  For more information, refer to the *Condition variables [ALPHA]* section of the *Arm C and C++ Libraries and Floating-Point Support User Guide*.

Defect fixes in Arm Compiler for Embedded 6.18

armclang

- [SDCOMP-60177] When compiling with -march=armv8-r and for AArch64 state, the compiler and integrated assembler incorrectly enabled the following features by default:
  - Armv8.4-A additions to the AArch64 Performance Monitors Extension.
  - Half-precision floating-point extensions.
  - Speculation Barrier instruction.

    - Speculation restriction instructions.
    - Speculative Store Bypass Safe.

  This has been fixed.

- [SDCOMP-60103] When assembling for an Armv8-R target and AArch64 state, the inline assembler and integrated assembler incorrectly failed to report an error for an MRS or MSR instruction that specifies one of the following as the system register to be accessed:
    - PRBAR0_EL1
    - PRBAR0_EL2
    - PRLAR0_EL1
    - PRLAR0_EL2

  Instead, the inline assembler and integrated assembler incorrectly treated these registers as aliases to the following registers respectively:

    - PRBAR_EL1
    - PRBAR_EL2
    - PRLAR_EL1
    - PRLAR_EL2

  This has been fixed. The inline assembler and integrated assembler now report one of the following errors:

    - expected readable system register
    - expected writable system register or pstate
- [SDCOMP-59879] The __VERSION__ predefined macro incorrectly contained extra characters after the underlying Clang version. This has been fixed.
- [SDCOMP-59809] The compiler incorrectly searched for certain unnecessary files within the directories specified by the PATH environment variable. Subsequently, this could result in slow compilation. This has been fixed.
- [SDCOMP-59788] When compiling for an Armv7-M or Armv8-M target and at any optimization level except -O0, the compiler could incorrectly generate one of the following instructions when accessing an element of a **char** or **short** array:
    - LDRBT
    - LDRHT
    - LDRSBT
    - LDRSHT
    - STRBT
    - STRHT

  This has been fixed.

- [SDCOMP-59656] When compiling for AArch64 state at -O0, the compiler could generate incorrect code for an integer literal that can be represented by fewer than 64 bits and is cast to a pointer type. This has been fixed.
- [SDCOMP-59654] When compiling for an Armv8.1-M target with a -mbranch-protection=*protection* option that enables Pointer Authentication Code (PAC) branch protection features, the compiler could generate incorrect code. This has been fixed.
- [SDCOMP-59605] When compiling with -mno-unaligned-access, the compiler could incorrectly fail to report a warning for a C++ class or class data member that is annotated with __attribute__((packed)) or #pragma pack. This has been fixed. The compiler now reports the following warning:

    - field <member> within '<class A>' is less aligned than '<class B>' and is usually due to '<class A>' being packed, which can lead to unaligned accesses
- [SDCOMP-59059] When compiling with -frwpi, and with -g or -gdwarf-*version*, the compiler could generate incorrect debug information for a global variable that is not **const**. This has been fixed.
- [SDCOMP-58523] The compiler, inline assembler, and integrated assembler incorrectly enabled cryptographic instructions by default for an Armv8-R AArch64 target. This has been fixed.
- [SDCOMP-57884] When compiling in a C++ source language mode with C++ exceptions enabled, the compiler generated code that incorrectly raises a std::bad_array_new_length exception for a **new** nothrow expression that specifies a negative array length, or specifies more initializers than the array length. This has been fixed.
- [SDCOMP-52680] When assembling for an Armv8-R AArch64 target, the inline assembler and integrated assembler incorrectly failed to report an error for an MRS or MSR instruction that specifies VSTTBR_EL2 or VTTBR_EL2 as the system register to be accessed. This has been fixed. The inline assembler and integrated assembler now report one of the following errors:
    - expected readable system register
    - expected writable system register or pstate

armlink

- [SDCOMP-59391] When linking for AArch64 state, and an input object contains a branch to a section symbol, the linker could incorrectly report the following error:
    - L6286E: Relocation #RELA:<relocation_number> in <object>(<section>) with respect to [Anonymous Symbol]. Value(<value>) out of range(<range>) for (R_AARCH64_CALL26)

  This has been fixed.

- [SDCOMP-57994] The linker could incorrectly select a different implementation of the cbrtf() or strcmp() Arm C library functions for two identical linker invocations. This has been fixed.

fromelf

- [SDCOMP-59890] When processing an ELF format input file that contains DWARF 4 debug information and a DWARF line table structure with a version lower than 4, the fromelf utility could incorrectly fail to generate debug line information. This has been fixed.

Libraries and system headers

- [SDCOMP-60224] The Arm C library variant of the cbrtf() function for an Armv8.1-M target with the PACBTI Extension could result in unexpected run-time behavior. This has been fixed.
- [SDCOMP-60157] The Arm C library implementation of the POSIX mbsnrtowcs() function could incorrectly update the source pointer when the destination pointer is a null pointer. This has been fixed.
- [SDCOMP-59569] The [ALPHA] Arm C++ library Thread-Porting Layer (TPL) implementations of the std::mutex::try_lock() and std::recursive_mutex::try_lock() functions returned an incorrect result. This has been fixed.

- [SDCOMP-59054] The Arm C++ library implementation of the std::allocator<T>::allocate() function incorrectly raised a std::length_error exception instead of std::bad_alloc or std::bad_array_new_length. This has been fixed.
- [SDCOMP-58926] The Arm C++ library implementation of std::reverse_iterator<Iter>::operator= was incorrect. Subsequently, this could result in one of the following:
    - The compiler making an unnecessary but valid call to a conversion constructor.
    - The compiler reporting the following error:
        - no matching member function for call to 'operator='

  This has been fixed.