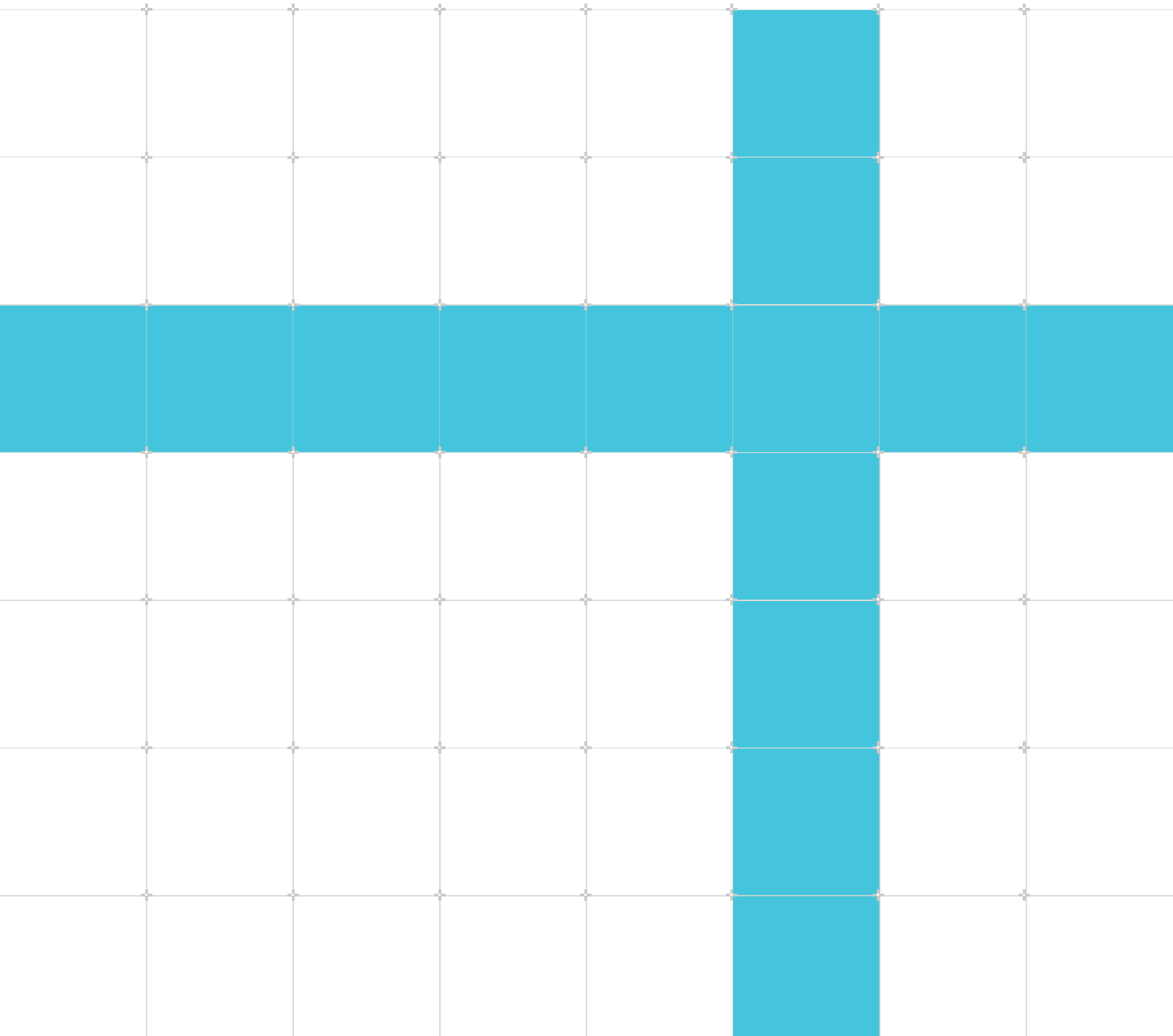
Release Notes for Arm Compiler for Embedded 6.23

Version 6.23

Issue 00
109995_062300_00_en



Release Notes for Arm Compiler for Embedded 6.23

This document (109995_062300_00_en) was issued on 2024-10-16. There might be a later issue at https://developer.arm.com/documentation/109995

The product version is 6.23.

See also: Proprietary notice | Product and document information | Useful resources

Start reading

If you prefer, you can skip to the start of the content.

Intended audience

This document is intended for use by a software developer who is using Arm Compiler for Embedded 6.23. The document includes an overview of the Arm Compiler for Embedded 6.23 release, changes and enhancements, and defects fixed.

Inclusive language commitment

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

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com.

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Contents

1. Release overview

This chapter provides an overview of the Arm Compiler for Embedded product and the Arm Compiler for Embedded 6.23 release.

1.1 Product description

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.

The toolchain is compatible with the following Arm Integrated Development Environments (IDEs):

- Arm Development Studio
- Arm Keil MDK v6

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 consumer electronics, industrial, medical, networking, railway, storage, and telecommunications. Arm strongly recommends that you consider a LTS and qualified version of Arm Compiler for Embedded FuSa instead of this release if your project:

- Has long-term maintenance requirements
- Has functional safety requirements such as EN 50128, IEC 61508, IEC 62304, or ISO 26262
- Uses an Automotive Enhanced processor

Contact your sales representative or submit an inquiry online to find out more about licensing Arm Compiler for Embedded or an Arm IDE.

1.2 Release highlights

Arm Compiler for Embedded 6.23 is the latest release as of October 2024.

The key highlights of this release include:

- Beta support for the Armv9.6-A architecture
- Beta support for 2024 extensions for A-profile architectures
- Support for the following processors:
  - Neoverse V3
  - Neoverse N3

    - Cortex-X925
    - Cortex-A725
- Support for the automatic generation of Checked Pointer Arithmetic instructions
- Support for an option to print the architecture features enabled by the target options specified to the compiler

Subject to your license terms, Arm Compiler for Embedded 6.23 can be used to build for the following Arm Architectures and Processors:

Architecture	Processor Family	Standard Processors	Automotive Enhanced Processors
Armv9.6-A [BETA]	-	-	-
Armv9-A up to Armv9.5-A	Neoverse	V3, V2 N3, N2	V3AE
	Cortex	X925 X4, X3, X2 A725, A720, A715, A710 A520, A510	A720AE, A520AE
Armv8-A up to Armv8.9-A	Neoverse	V1 N1 E1	-
	Cortex	X1C, X1 A78C, A78, A77, A76, A75, A73, A72 A65 A57, A55, A53 A35, A34, A32	A78AE, A76AE A65AE
Armv7-A	Cortex	A17, A15, A12 A9, A8, A7, A5	-
Armv8-R AArch64	Cortex	R82	R82AE
Armv8-R	Cortex	R52+, R52	-
Armv7-R	Cortex	R8, R7, R5, R4F, R4	-
Armv8-M up to Armv8.1-M	Cortex	M85 M55, M52 M35P, M33 M23	-
	STAR	STAR-MC1	-

Architecture	Processor Family	Standard Processors	Automotive Enhanced Processors
Armv7-M	Cortex	M7, M4, M3	-
	SecurCore	SC300	-
Armv6-M	Cortex	M1, M0, M0+	-
	SecurCore	SC000	-

For more information, see the following:

- The *Arm Development Studio* product page.
- The *Arm Keil MDK v6* product page.
- The *Support level definitions* section of the *Arm Compiler for Embedded User Guide* document for this release.

1.3 Included components

This section lists the toolchain components and different types of documentation included in Arm Compiler for Embedded 6.23.

Category	Component	Description
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`	ELF image conversion utility and dissembler
	Arm C libraries	Runtime support libraries for embedded systems
	Arm C++ libraries	Libraries based on the LLVM libc++ project
	`armasm`	Deprecated legacy assembler for armasm-syntax assembly code for older Arm architectures only. Use the `armclang` integrated assembler for all new assembly files.
User documentation	User Guide	Provides instructions and examples 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 can be reported by `armar`, `armasm`, `armlink`, and `fromelf`
	Migration and Compatibility Guide	Provides information to help you migrate from Arm Compiler 5 to Arm Compiler for Embedded
	Release Notes	These release notes

These components can be obtained from the following sources:

Component type	Source
Toolchain components	Available via Product Download Hub (PDH)
Documentation	Available via Arm Developer

1.4 Product quality

This product is a Final release quality product which is suitable for use in a production environment.

Certain features within this product are not Final release quality features or are unsupported. The status of features is explicitly stated within the documentation where applicable. For more information about such features, see the *Support level definitions* section of the *Arm Compiler for Embedded User Guide*.

2. Download Arm Compiler for Embedded 6.23

This chapter provides information about how to download and install Arm Compiler for Embedded 6.23.

Arm Compiler for Embedded 6.23 might be available to download standalone, as part of an Arm Integrated Development Environment (IDE), or as part of a Success Kit depending on your license and entitlements.

To download this release standalone, use the `ACOMPE` code on Arm Product Download Hub (PDH). The toolchain download packages within this product code are intended to be used in the following environments as of October 2024*:

Host architecture	Host operating system	Toolchain download package	Host environment
x86_64	• Red Hat Enterprise Linux 9 • Red Hat Enterprise Linux 8 • Red Hat Enterprise Linux 7 • Ubuntu 24.04 LTS • Ubuntu 22.04 LTS • Ubuntu 20.04 LTS	x86_64 Linux	• Standalone installation • Integrated into Arm Development Studio • Integrated into Keil MDK version 6
	• Windows Server 2022 • Windows 11 • Windows 10	x86_64 Windows	• Standalone installation • Integrated into Arm Development Studio • Integrated into Keil MDK version 6
		for Keil® MDK	• Integrated into µVision within Keil MDK version 6
AArch64	• Ubuntu 24.04 LTS • Ubuntu 22.04 LTS • Ubuntu 20.04 LTS	AArch64 Linux	• Standalone installation • Integrated into Keil MDK version 6

*Arm reserves the right to add additional environments for this release. The environments may differ between releases.

The following restrictions apply:

- The minimum required version of glibc for Linux host platforms is as follows:
  - 2.15 for x86_64 host platforms.
  - 2.17 for AArch64 host platforms.
- The toolchain must not be installed directly into an Arm Development Studio installation directory.
- µVision within Keil MDK version 6 requires that the toolchain is installed into the `ARM` sub-directory of the µVision installation directory. For example, `C:\Keil_v5\ARM\ARMCompiler6.23`.

- Use with a legacy Keil license is only permitted on x86_64 Windows host platforms.

For more information, see the following:

- The *System requirements and installation* section of the *Arm Compiler for Embedded User Guide* for toolchain installation instructions.
- The *Register a compiler toolchain* section of the *Arm Development Studio Getting Started Guide* for instructions to integrate the toolchain into Arm Development Studio.
- The *Arm Keil Studio Visual Studio Code Extensions User Guide* for instructions to integrate the toolchain into *Arm Keil MDK v6*.
- The *User-based Licensing User Guide* for instructions to configure the toolchain to use a User-based License (UBL).
- The *Manage Arm Compiler Versions* section of the *uVision User's Guide*.

3. Differences from previous release

This chapter describes differences from the previous release, Arm Compiler for Embedded 6.22.

For more information about the scope of this section, see the article *Does Arm document all known issues that affect each Arm Compiler release?*.

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.

3.1 General changes

This section contains a list of general changes made in this release of Arm Compiler for Embedded.

- [SDCOMP-67423] Support has been added for the `--print-enabled-extensions` option to print the architecture features enabled by the target options specified to the compiler.

  For more information, refer to the *--print-enabled-extensions* section of the *Reference Guide*.
- [SDCOMP-67398] Previously, when compiling with `-march=<name>`, where `<name>` specifies an Armv9-A or later target, the compiler enabled the Realm Management Extension (RME) Memory Encryption Contexts feature (FEAT_MEC). This behavior has been changed. The compiler now enables FEAT_MEC only where `<name>` specifies an Armv9.2-A or later target.
- [SDCOMP-67153] Previously, when compiling for AArch64 state with an `-march=<name>` or `-mcpu=<name>` option, and without the `+fpmr` architecture feature modifier, the compiler and integrated assembler would report an error for an `MRS` or `MSR` instruction that specifies the Floating-point Mode Register (FPMR) as the special register to be accessed.

  This has been changed to the following:

  - The compiler and integrated assembler no longer report an error for a `MRS` or `MSR` instruction that specifies FPMR as the special register to be accessed.
  - The compiler and integrated assembler now report one of the following errors when compiling with an `-march=<name>` or `-mcpu=<name>` option that specifies the `+fpmr` architecture feature modifier:
    - `unsupported argument '<name>+fpmr' to option '-march='`
    - `unsupported argument '<name>+fpmr' to option '-mcpu='`

  FPMR is available on targets that implement the Floating-point Mode Register controls feature (FEAT_FPMR).
- [SDCOMP-66891] Support has been added for the automatic generation of Checked Pointer Arithmetic instructions when compiling with target options that enable the Instruction-only Checked Pointer Arithmetic feature (FEAT_CPA).
- [SDCOMP-66170] Support for the following A-profile architectures has been changed:

Architecture	State	-march=<name> option	Previous support level	New support level
Armv9.6-A	AArch64	`armv9.6-a`	Unsupported	Beta
Armv9.6-A	AArch32	`armv9.6-a`	Unsupported	Beta

For more information, refer to the *-march* section of the *Reference Guide*.

- [SDCOMP-66018] Support for the following A-profile architecture features for AArch64 state has been changed:

Feature identifier	Feature description	-march / -mcpu +<feature> option(s)	Previous support level	New support level
FEAT_CMPBR	Compare and Branch instruction	`cmpbr`	Unsupported	Beta
FEAT_F8F16MM	FP8 to Half-Precision Matrix Multiplication	`f8f16mm`	Unsupported	Beta
FEAT_F8F32MM	FP8 to Single-Precision Matrix Multiplication	`f8f32mm`	Unsupported	Beta
FEAT_FPRCVT	Floating-Point to/from Integer in Scalar FP register	`fprcvt`	Unsupported	Beta
FEAT_LSFE	Large System Float Extension	`lsfe`	Unsupported	Beta
FEAT_LSUI	Unprivileged Load Store	`lsui`	Unsupported	Beta
FEAT_OCCMO	Outer Cacheable Cache Maintenance Operation	`occmo`	Unsupported	Beta
FEAT_PCDPHINT	Producer-Consumer Data Placement Hints	`pcdphint`	Unsupported	Beta
FEAT_PoPs	Point of Physical Storage	`pops`	Unsupported	Beta
FEAT_RME_GPC3	Realm Management Extension (RME) Granule Protection Check 3 Extension	`rme-gpc3`	Unsupported	Beta
FEAT_SME2	Scalable Matrix Extension version 2	`sme2`	Alpha	Supported
FEAT_SME2p1	Scalable Matrix Extension version 2.1	`sme2p1`	Alpha	Supported
FEAT_SME2p2	Scalable Matrix Extension version 2.2	`sme2p2`	Unsupported	Beta
FEAT_SME_B16B16	Non-widening BFloat16 to BFloat16 SME ZA-targeting arithmetic	`sme-b16b16`	Unsupported	Supported
FEAT_SSVE_AES	Streaming Scalable Vector Extensions (SVE) Mode Advanced Encryption Standard and 128-bit polynomial multiply long instructions	`ssve-aes`	Unsupported	Beta
FEAT_SSVE_F8F16MM	Streaming SVE FP8 to Half-Precision Matrix Multiplication	`ssve-f8f16mm`	Unsupported	Beta
FEAT_SSVE_F8F32MM	Streaming SVE FP8 to Single-Precision Matrix Multiplication	`ssve-f8f32mm`	Unsupported	Beta
FEAT_SVE2p1	Scalable Vector Extensions version 2.1	`sve2p1`	Beta	Supported
FEAT_SVE2p2	Scalable Vector Extensions version 2.2	`sve2p2`	Unsupported	Beta
FEAT_SVE_AES2	SVE multi-vector Advanced Encryption Standard and 128-bit polynomial multiply long instructions	`sve-aes2`	Unsupported	Beta
FEAT_SVE_B16B16	Non-widening BFloat16 to BFloat16 arithmetic for SVE2 and SME2	`sve-b16b16`	Unsupported	Supported
FEAT_SVE_BFSCALE	BFloat16 Floating-Point Adjust Exponent	`sve-bfscale`	Unsupported	Beta

Feature identifier	Feature description	-march / -mcpu +<feature> option(s)	Previous support level	New support level
FEAT_SVE_F16F32MM	SVE Half-Precision to Single-Precision Matrix Multiplication	`sve-f16f32mm`	Unsupported	Beta

When compiling with `-march=armv9.6-a`, the compiler enables the following subset of these features by default:

- FEAT_CMPBR
- FEAT_FPRCVT
- FEAT_LSUI
- FEAT_OCCMO
- FEAT_SME2
- FEAT_SME2p1
- FEAT_SME2p2
- FEAT_SVE2p1
- FEAT_SVE2p2

For more information, refer to:

- The *-march* section of the *Reference Guide*.
- The *-mcpu* section of the *Reference Guide*.
- The relevant *Arm Architecture Reference Manual* or *Reference Manual Supplement* for each feature.

- [SDCOMP-65970] Previously, the Arm C++ library permitted parsing strings to generate floating-point infinities and NaNs. For example, the Arm C++ library permitted parsing the following strings:

String	Parsed as
`Inf`	Floating-point infinity
`NaN`	Floating-point NaN

This behavior has been changed. The Arm C++ library no longer permits parsing strings to generate floating-point infinites and NaNs.

To generate floating-point infinities and NaNs, you may instead compile with `-ffp-mode=full` and use the appropriate member functions of the `std::numeric_limits` class template.

- [SDCOMP-65848] Previously, when compiling for AArch32 state with `-march=armv8-r`, the compiler generated code that was optimized for a Cortex-R52 target. Additionally, the default `-mfpu=<name>` option was set to `-mfpu=neon-fp-armv8`. This behavior has been changed to the following:
  - The compiler no longer generates code that is optimized for a Cortex-R52 target.
  - The default `-mfpu=<name>` option is now set to `-mfpu=fpv5-sp-d16`.

To generate code that is optimized for a Cortex-R52 target, compile with `-mcpu=cortex-r52` instead of `-march=armv8-r`.

- [SDCOMP-65776] The compiler could incorrectly fail to report an error for a translation unit that is larger than the maximum acceptable size of approximately 2GB. This has been fixed. The compiler now reports one of the following errors:
  - `file '<filename>' is too large for Clang to process`
  - `translation unit is too large for Clang to process: ran out of source locations`
- [SDCOMP-65746] Version 2024-06 of the *Arm A-profile A64 Instruction Set Architecture* has changed the mnemonics of the register variants of the following FEAT_PAuth_LR instructions:

Previous mnemonic	New mnemonic
AUTIASPPC	AUTIASPPCR
AUTIBSPPC	AUTIBSPPCR
RETAASPPC	RETAASPPCR
RETABSPPC	RETABSPPCR

The compiler and integrated assembler have been updated to match this change.

For more information, refer to the *Base instructions* section of version 2024-06 of the *Arm A-profile A64 Instruction Set Architecture*.

- [SDCOMP-65617] Previously, when compiling for AArch64 state with target options that enable the Pointer authentication instructions that allow signing of LR using SP and PC as diversifiers feature (FEAT_PAuth_LR), the `-mbranch-protection=standard` option was equivalent to `-mbranch-protection=bti+pac-ret`. This behavior has been changed. The `-mbranch-protection=standard` option is now equivalent to `-mbranch-protection=bti+pac-ret+pc`.

  To restore the previous behavior, compile with `-mbranch-protection=bti+pac-ret` instead of `-mbranch-protection=standard` when compiling with target options that enable FEAT_PAuth_LR.

  For more information, refer to the *-mbranch-protection* section of the *Reference Guide*.
- [SDCOMP-65457] Support has been added for the Cortex-X925 processor. To target Cortex-X925, select from the following `armclang` options:

Cryptographic Extension	Options
Included	`--target=aarch64-arm-none-eabi -mcpu=cortex-x925+crypto+sve2-aes+sve2-sha3+sve2-sm4`
Not included	`--target=aarch64-arm-none-eabi -mcpu=cortex-x925`

- [SDCOMP-65456] Support has been added for the Cortex-A725 processor. To target Cortex-A725, select from the following `armclang` options:

Cryptographic Extension	Options
Included	`--target=aarch64-arm-none-eabi -mcpu=cortex-a725+crypto+sve2-aes+sve2-sha3+sve2-sm4`
Not included	`--target=aarch64-arm-none-eabi -mcpu=cortex-a725`

- [SDCOMP-65350] Support has been added for the Neoverse N3 processor. To target Neoverse N3, select from the following `armclang` options:

Cryptographic Extension	Options
Included	`--target=aarch64-arm-none-eabi -mcpu=neoverse-n3+crypto+sve2-aes+sve2-sha3+sve2-sm4`
Not included	`--target=aarch64-arm-none-eabi -mcpu=neoverse-n3`

- [SDCOMP-63482] Previously, the compiler only generated debug information for a C++ structured binding to a bit-field member `B` of a **`struct`** when `B` is aligned to an 8-bit boundary, and the size of `B` is 8, 16, 32, or 64 bits. This behavior has been changed. The compiler now generates debug information for a C++ structured binding to all bit-field members of a **`struct`**.
- [SDCOMP-58939] Support has been added for the Neoverse V3 processor. To target Neoverse V3, select from the following `armclang` options:

Cryptographic Extension	Options
Included	`--target=aarch64-arm-none-eabi -mcpu=neoverse-v3+crypto+sve2-aes+sve2-sha3+sve2-sm4`
Not included	`--target=aarch64-arm-none-eabi -mcpu=neoverse-v3`

3.2 Defect fixes

This section contains information about defect fixes made in this release of Arm Compiler for Embedded. It contains sub-sections that each focus on the defect fixes in a specific component of the toolchain.

3.2.1 Compiler and integrated assembler, armclang

This section contains a list of defect fixes made in the compiler and integrated assembler, `armclang`.

- [SDCOMP-67544] When compiling for AArch32 state, the compiler could generate incorrect code for a **`volatile`** variable of a single-precision floating-point type. This has been fixed.
- [SDCOMP-67194] When compiling with `-moutline` or at `-Oz` without `-mno-outline`, the compiler could generate incorrect code for an outlined function. This has been fixed.

- [SDCOMP-67150] When compiling for an Armv9.5-A target, the compiler and integrated assembler incorrectly failed to enable the following architecture features by default:

Feature identifier	Feature description	-march=<name> / -mcpu=<name> +<feature> option
FEAT_FAMINMAX	Maximum and minimum absolute value instructions	`faminmax`
FEAT_LUT	Lookup table instructions	`lut`

  This has been fixed.
- [SDCOMP-67120] When compiling for AArch32 state, without `-fno-omit-frame-pointer`, and with `-mframe-chain=aapcs` or `-mframe-chain=aapcs+leaf`, the inline assembler incorrectly failed to report the following warning for an inline assembly statement that contains the frame pointer register `R11` in its clobber list:
  - `inline asm clobber list contains reserved registers: R11`

  This has been fixed.
- [SDCOMP-66787] When compiling with `-mfloat-abi=hard` and for an Armv8-M target with the Main Extension, the compiler could generate code which incorrectly failed to indicate that the floating-point unit may be used by a Non-secure function that is called from a Secure function. This has been fixed.
- [SDCOMP-66328] When compiling for AArch64 state, and with a `-mbranch-protection=<protection>` option that enables pointer authentication branch protection using the Program Counter as a second diversifier for return address signing, the compiler could generate incorrect C++ exception unwinding information and incorrect debug information for a function. This has been fixed.
- [SDCOMP-65997] When assembling for AArch64 state, the inline assembler and integrated assembler incorrectly reported one of the following errors for an `MRS` or `MSR` instruction that specifies `TRBMPAM_EL1` as the special register to be accessed:
  - `expected readable system register`
  - `expected writable system register or pstate`

  This has been fixed.

  `TRBMPAM_EL1` is available on targets that implement the Trace Buffer MPAM extensions feature (FEAT_TRBE_MPAM).
- [SDCOMP-65607] The compiler could generate incorrect code for the following forms of Scalable Vector Extension version 2 (SVE2) intrinsics defined in the `<arm_sve.h>` system header:
  - `svwhilegt_*()`
  - `svwhilege_*()`

  This has been fixed.

  For more information about SVE2 intrinsics, see https://developer.arm.com/architectures/instruction-sets/intrinsics.

- [SDCOMP-65592] When compiling with target options that enable the Scalable Matrix Extension feature (FEAT_SME) and do not enable the Scalable Vector Extension feature (FEAT_SVE), the compiler could incorrectly generate a Scalable Vector Extension (SVE) instruction for a function that is not annotated as being executed in streaming mode. This has been fixed.

  For more information about streaming mode, refer to the Controlling the use of streaming mode section of the Arm C Language Extensions (ACLE).
- [SDCOMP-65564] The compiler could generate incorrect code for the following Scalable Vector Extension (SVE) intrinsics defined in the `<arm_sve.h>` system header:
  - `svqadd[_n_s8]()`
  - `svqadd[_s8]()`
  - `svqsub[_n_s8]()`
  - `svqsub[_s8]()`

  This has been fixed.

  For more information about SVE intrinsics, see https://developer.arm.com/architectures/instruction-sets/intrinsics.
- [SDCOMP-65460] The compiler and integrated assembler incorrectly failed to implicitly enable all the required dependencies for the following architecture features:

Feature identifier	Feature description	-march=<name> / -mcpu=<name> +<feature> option
FEAT_FP8	Arm FP8 formats E5M2 and E4M3 for FP8 instructions	`fp8`
FEAT_FP8DOT2	FP8 to half-precision 2-way dot product instructions	`fp8dot2`
FEAT_FP8DOT4	FP8 to single-precision 4-way dot product instructions	`fp8dot4`
FEAT_FP8FMA	FP8 to half-precision and single-precision multiply-accumulate instructions	`fp8fma`
FEAT_SME_F8F16	SME2 FP8 to half-precision multiply-accumulate, dot product and outer product instructions	`sme-f8f16`
FEAT_SSVE_FP8DOT2	SVE2 FP8 to half-precision 2-way dot product instructions in Streaming SVE mode	`ssve-fp8dot2`
FEAT_SSVE_FP8DOT4	SVE2 FP8 to single-precision 4-way dot product instructions in Streaming SVE mode	`ssve-fp8dot4`

  This has been fixed.

  For example, when compiling with `-march=armv9.4-a+fp8`, the compiler now implicitly enables the following dependencies of FEAT_FP8:
  - The Maximum and minimum absolute value instructions feature (FEAT_FAMINAX).
  - The Lookup table instructions feature (FEAT_LUT).

- [SDCOMP-65418] When compiling for AArch32 state, without `-fno-omit-frame-pointer`, and with `-mframe-chain=aapcs` or `-mframe-chain=aapcs+leaf`, the compiler could generate code that incorrectly corrupts the frame pointer register `R11`. This has been fixed.
- [SDCOMP-65243] The inline assembler and integrated assembler incorrectly failed to report an error for a Scalable Vector Extension (SVE) instruction that specifies an invalid predication pattern. This has been fixed. The inline assembler and integrated assembler now report the following error:
  - `invalid operand for instruction`
- [SDCOMP-65172] When compiling with `-gdwarf-2` or `-gdwarf-3`, the compiler could generate incorrect debug information for a bit-field. This has been fixed.
- [SDCOMP-65141] When assembling for a big-endian target, the inline assembler and integrated assembler generated incorrect code for a sequence of `LDR` instructions that loads the same literal value into a combination of 32-bit and 64-bit registers. This has been fixed.
- [SDCOMP-65113] When assembling for AArch64 state, with an `-march=<name>` or `-mcpu=<name>` option that enables one of the following architectural features `F` by default, and with the `+no<feature>` modifier to disable `F`, the inline assembler and integrated assembler incorrectly failed to disable `F`:

Feature identifier	Feature description	-march=<name> / -mcpu=<name> +no<feature> option
FEAT_FlagM	Condition flag manipulation instructions	`+noflagm`
FEAT_SB	Speculation Barrier	`+nosb`
FEAT_SPECRES	Speculation restriction instructions	`+nopredres`
FEAT_SSBS	Speculative Store Bypass Safe	`+nossbs`

  This has been fixed.
- [SDCOMP-65094] When compiling without `-fno-optimize-sibling-calls`, with an `-mbranch-protection=<protection>` option that enables Pointer Authentication Code (PAC) protection, and for an Armv8.1-M target with the Main Extension, the compiler could generate incorrect code for a function that uses `R12` for a named register variable. This has been fixed.
- [SDCOMP-64877] When applying Link-Time Optimization (LTO) to objects that have been compiled with different `-mbranch-protection=<protection>` options, the compiler incorrectly failed to enable branch protection using pointer authentication for objects that have been compiled with a `-mbranch-protection=<protection>` option that enables branch protection using pointer authentication. This has been fixed.
- [SDCOMP-64829] When compiling with target options that enable the Standardization of memory operations feature (FEAT_MOPS), the compiler could incorrectly generate an architecturally `CONSTRAINED UNPREDICTABLE` instruction. This has been fixed.
- [SDCOMP-64335] When compiling for AArch64 state, and with target options that enable the Large System Extensions feature (FEAT_LSE), the compiler could generate code that incorrectly failed to acquire a lock for an atomic exchange operation. This has been fixed.
- [SDCOMP-64209] When assembling for AArch32 state, the integrated assembler could incorrectly report the following fatal error for an assembly language source file that contains a `.fpu <fpu_name>` directive:
  - `error in backend: Unknown FPU: <number>`

This has been fixed.

- [SDCOMP-63913] When compiling with a `-mharden-sls=<option>` option that enables the mitigation against Straight-Line Speculation (SLS) for `RET` and `BR` instructions, and with `-moutline` or at `-Oz` without `-mno-outline`, the compiler could generate incorrect code. This has been fixed.
- [SDCOMP-61486] When compiling in a C++11 or later source language mode, the compiler generated incorrect code for the expression `noexcept(typeid(V))`. This has been fixed.

3.2.2 Linker, armlink

This section contains a list of defect fixes made in the linker, `armlink`.

- [SDCOMP-65517] For two execution regions `A` and `B`, where `A` is at a lower address than `B`, the linker incorrectly assumed that the name of `A` is always lexically earlier than the name of `B`. Subsequently, this could result in the linker generating incorrect callgraph or stack usage information. This has been fixed. The linker no longer assumes that the name of `A` is always lexically earlier than the name of `B`.
- [SDCOMP-64999] When linking an input object that has been compiled with C++ exceptions enabled and for AArch32 state, the linker could generate an incorrect C++ exception-handling table. This has been fixed.

3.2.3 Libraries and system headers

This section contains a list of defect fixes made in the C and C++ libraries and system headers supplied with the toolchain.

- [SDCOMP-67219] The microlib implementation of the `qsort()` function could result in high stack usage at run-time. This has been fixed.
- [SDCOMP-66090] The Arm C library implementation of the `calloc(num, size)` function for AArch64 state was incorrect. Subsequently, this could result in one of the following:
  - `calloc()` incorrectly failing to return a null pointer when the value of `num*size` is greater than or equal to `(1<<64)`. This could result in unexpected run-time behavior.
  - `calloc()` incorrectly always returning a null pointer when the value of `num*size` is greater than or equal to `(1<<32)` and less than `(1<<64)`.

  This has been fixed.
- [SDCOMP-65871] When compiling for an Armv8-R AArch64 target without hardware floating-point support, a floating-point arithmetic operation or C library function call involving a value or variable of **`double`** type could return an incorrect result or set the `FE_INEXACT` floating-point exception flag incorrectly. This has been fixed.
- [SDCOMP-65388] The Arm C library was not thread-safe for a multithreaded program that contains C++ objects with static storage duration. This has been fixed.

4. Support

This chapter includes guidance on how to obtain support for using Arm Compiler for Embedded 6.23.

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. Please contact your supplier or 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, see the *-E* section of the *Arm Compiler for Embedded Reference Guide*.

5. Release history

This chapter contains the history of Arm Compiler for Embedded releases.

Version	Release Date
Arm Compiler 6.00	10 Apr 2014
Arm Compiler 6.00 update 1	29 May 2014
Arm Compiler 6.00 update 2	30 Oct 2014
Arm Compiler 6.01	18 Dec 2014
Arm Compiler 6.01 update 1	9 Feb 2015
Arm Compiler 6.01 update 2	20 Mar 2015
Arm Compiler 6.02	26 Jun 2015
Arm Compiler 6.3	17 Nov 2015
Arm Compiler 6.4	1 Mar 2016
Arm Compiler 6.5	30 Jun 2016
Arm Compiler 6.6	10 Nov 2016
Arm Compiler 6.7	31 Mar 2017
Arm Compiler 6.7.1	31 May 2017
Arm Compiler 6.8	27 Jul 2017
Arm Compiler 6.9	25 Oct 2017
Arm Compiler 6.10	15 Mar 2018
Arm Compiler 6.10.1	13 Jun 2018
Arm Compiler 6.11	25 Oct 2018
Arm Compiler 6.12	28 Feb 2019
Arm Compiler 6.13	10 Oct 2019
Arm Compiler 6.13.1	18 Nov 2019
Arm Compiler 6.14	28 Feb 2020
Arm Compiler 6.14.1	9 Jun 2020
Arm Compiler 6.15	9 Oct 2020
Arm Compiler 6.16	9 Mar 2021
Arm Compiler for Embedded 6.17	20 Oct 2021
Arm Compiler for Embedded 6.18	31 Mar 2022
Arm Compiler for Embedded 6.19	12 Oct 2022
Arm Compiler for Embedded 6.20	15 Mar 2023
Arm Compiler for Embedded 6.20.1	25 Apr 2023
Arm Compiler for Embedded 6.21	11 Oct 2023
Arm Compiler for Embedded 6.22	19 Mar 2024
Arm Compiler for Embedded 6.23	16 Oct 2024

Proprietary Notice

This document is protected by copyright and other related rights and the use 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 Limited ("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 the subject matter of this document infringes any third party patents.

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm's view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

This document may include technical inaccuracies or typographical errors. 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, any patents, copyrights, trade secrets, trademarks, or other rights.

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.

Reference by Arm to any third party's products or services within this document is not an express or implied approval or endorsement of the use thereof.

This document consists solely of commercial items. You shall be responsible for ensuring that any permitted 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 this document shall prevail.

The validity, construction and performance of this notice shall be governed by English Law.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-1121-V1.0

Product and document information

Read the information in these sections to understand the release status of the product and documentation, and the conventions used in Arm documents.

Product status

All products and services provided by Arm require deliverables to be prepared and made available at different levels of completeness. The information in this document indicates the appropriate level of completeness for the associated deliverables.

Product completeness status

The information in this document is Final, that is for a developed product.

Revision history

These sections can help you understand how the document has changed over time.

Document release information

The Document history table gives the issue number and the released date for each released issue of this document.

Document history

Issue	Date	Confidentiality	Change
062300-00	16 October 2024	Non-Confidential	Initial release

Change history

The Change history tables describe the technical changes between released issues of this document in reverse order. Issue numbers match the revision history in Document release information on page 23.

For a list of technical changes in Arm Compiler for Embedded 6.23, see the *Differences from previous release* section.

Conventions

The following subsections describe conventions used in Arm documents.

Glossary

The Arm Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the Arm Glossary for more information: developer.arm.com/glossary.

Typographic conventions

Arm documentation uses typographical conventions to convey specific meaning.

Convention	Use
italic	Citations.
bold	Interface elements, such as menu names. Terms in descriptive lists, where appropriate.
`monospace`	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace underline`	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`<and>`	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Terms that have specific technical meanings as defined in the *Arm® Glossary*. For example, **IMPLEMENTATION DEFINED**, **IMPLEMENTATION SPECIFIC**, **UNKNOWN**, and **UNPREDICTABLE**.

Caution

We recommend the following. If you do not follow these recommendations your system might not work.

Warning

Your system requires the following. If you do not follow these requirements your system will not work.

You are at risk of causing permanent damage to your system or your equipment, or harming yourself.

This information is important and needs your attention.

A useful tip that might make it easier, better or faster to perform a task.

A reminder of something important that relates to the information you are reading.

Useful resources

This document contains information that is specific to this product. See the following resources for other useful information.

Access to Arm documents depends on their confidentiality:

- Non-Confidential documents are available at developer.arm.com/documentation. Each document link in the following tables goes to the online version of the document.
- Confidential documents are available to licensees only through the product package.

Arm product resources	Document ID	Confidentiality
Arm Compiler for Embedded Migration and Compatibility Guide	100068	Non-Confidential
Arm Compiler for Embedded Reference Guide	101754	Non-Confidential
Arm Compiler for Embedded User Guide	100748	Non-Confidential
Arm Compiler for Embedded documentation index	KA005061	Non-Confidential
Arm Development Studio	–	Non-Confidential
Arm Development Studio Getting Started Guide	101469	Non-Confidential
Arm Keil MDK v6	–	Non-Confidential
Arm Keil Studio Visual Studio Code Extensions User Guide	108029	Non-Confidential
Does Arm document all known issues that affect each Arm Compiler release?	KA005052	Non-Confidential
User-based Licensing User Guide	102516	Non-Confidential
uVision User's Guide	101407	Non-Confidential