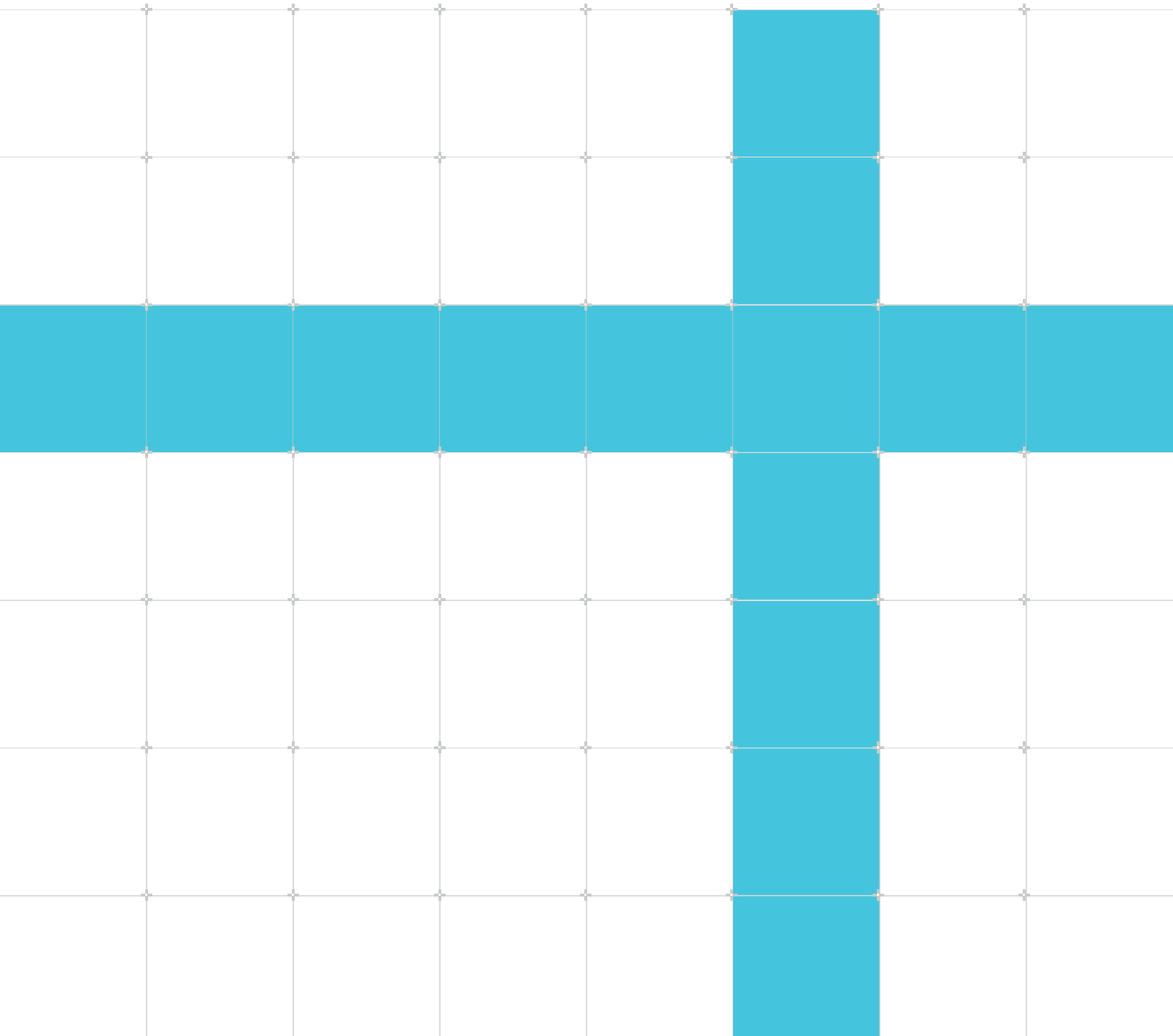
Release Notes for Arm Compiler for Embedded FuSa 6.22.2

Version 6.22.2

Issue 01
110601_062202_01_en



Release Notes for Arm Compiler for Embedded FuSa 6.22.2

This document (110601_062202_01_en) was issued on 2025-08-19. There might be a later issue at https://developer.arm.com/documentation/110601

The product version is 6.22.2.

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 FuSa to build a project with functional safety or long-term maintenance requirements. The document includes an overview of the Arm Compiler for Embedded FuSa 6.22.2 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 FuSa product and the Arm Arm Compiler for Embedded FuSa 6.22.2 release.

1.1 Product description

Arm Compiler for Embedded FuSa 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 with functional safety (FuSa) or long-term support requirements.

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 functional safety embedded system developers to meet challenging performance goals and memory constraints.

Arm Compiler for Embedded FuSa is used by leading companies in a wide variety of industries, including automotive (ISO 26262), consumer electronics, industrial (IEC 61508), medical (IEC 62304), networking, railway (EN 50716), storage, and telecommunications.

Key changes in Arm Compiler for Embedded FuSa 6.22LTS when compared to the first release of Arm Compiler for Embedded FuSa 6.16LTS include:

- Support for the following Automotive Enhanced processors:
  - Neoverse V3AE
  - Cortex-A720AE, Cortex-A520AE
  - Cortex-R82AE
- Support for the following processors:
  - Neoverse V2
  - Cortex-X4, Cortex-X3, Cortex-X2, Cortex-X1C
  - Cortex-A720, Cortex-A715, Cortex-A710, Cortex-A520, Cortex-A510
  - Cortex-R82
  - Cortex-M85, Cortex-M52
- Support for the following architectures:
  - Armv9-A up to Armv9.5-A
  - Armv8.8-A and Armv8.9-A
  - Armv8-R AArch64 implementations without hardware floating-point, except when compiling in a C++ source language mode

    - Armv8-R AArch64 implementations with hardware floating-point
- Support for the following architecture extensions:
    - 2023 extensions for A-profile architectures
    - The M-profile PACBTI Extension
- Support for User-based Licensing (UBL)
- Removal of FlexNet Publisher (FNP) and Keil licensing
- Support for C++17
- Support for execute-only memory (XOM) regions when building for an Armv6-M target
- Support for the following security features:
    - Undefined Behavior Sanitizer (UBSan)
    - Control Flow Integrity (CFI)
    - Shadow Call Stack for AArch64 state
    - Mitigations for the PACMAN security vulnerability
    - Mitigations for the Straight-Line Speculation (SLS) security vulnerability
    - Mitigations for the VLLDM instruction security vulnerability
- Support for automatic vectorization for SVE and SVE2
- Deprecation of the legacy assembler, armasm

For a summary of changes between Arm Compiler for Embedded FuSa 6.16LTS and Arm Compiler for Embedded FuSa 6.22LTS, see the *Summary of changes between Arm Compiler for Embedded FuSa 6.16LTS and Arm Compiler for Embedded FuSa 6.22LTS* section of *Arm Compiler for Embedded FuSa Migration and Compatibility Guide version 6.22.2*.

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

1.2 Release highlights

Arm Compiler for Embedded FuSa 6.22.2 is the latest release as of August 2025. It has been derived from the unqualified Arm Compiler for Embedded 6.22 release, includes additional defect fixes, and adds a Qualification Kit for functional safety purposes.

Subject to your license terms, Arm Compiler for Embedded FuSa 6.22.2 can be used to build for the following Arm Architectures and Processors:

Architecture	Processor Family	Standard Processors	Automotive Enhanced Processors
Armv9-A up to Armv9.5-A	Neoverse	V2 N2	V3AE

Architecture	Processor Family	Standard Processors	Automotive Enhanced Processors
	Cortex	X4, X3, X2 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	-
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 *Arm Compiler for Embedded FuSa User Guide version 6.22.2*.

1.3 Included components

This section lists the toolchain components and different types of documentation included in Arm Compiler for Embedded FuSa 6.22.2.

Category	Component	Description
Qualified 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
Unqualified toolchain components	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.
Qualification Kit	Safety Manual	Describes the scope of qualification, and how to use the toolchain for safety-related development
	Defect Report	Provides information about known safety-related defects at the time of release. For up-to-date information about known safety-related defects, see the Arm Compiler for Embedded FuSa 6.22LTS Defect Notification Report.
	Test Report	Contains results from language conformance tests
	Development Process	Contains an overview of the process used to develop the toolchain
	Release History	Identification information for all the releases to date of the Arm Compiler for Embedded FuSa 6.22LTS release series
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 FuSa
	Release Notes	These release notes

These components can be obtained from the following sources:

Component type	Source
Qualified toolchain components	Available via Product Download Hub (PDH)
Unqualified toolchain components	Available via PDH
Qualification Kit	Available via 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 the following:

- The *Features not recommended for use in safety-related development* section of *Arm Compiler for Embedded FuSa Qualification Kit Safety Manual version 6.22.2*.
- The *Support level definitions* section of *Arm Compiler for Embedded FuSa User Guide version 6.22.2*.

2. Download Arm Compiler for Embedded FuSa 6.22.2

This chapter provides information about how to download and install Arm Compiler for Embedded FuSa 6.22.2.

Arm Compiler for Embedded FuSa 6.22.2 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 ACOMP622 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 August 2025*:

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 Server 2019 Windows 11 Windows 10	x86_64 Windows	Standalone installation Integrated into Arm Development Studio Integrated into 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 change the environments without prior notice.

The Qualification Kit documents can be obtained from additional download packages in PDH under the same product code, ACOMP622.

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.

For more information, see the following:

- The *System requirements and installation* section of *Arm Compiler for Embedded FuSa User Guide version 6.22.2* 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).

3. Differences from previous release

This chapter describes differences from the previous release in the Arm Compiler for Embedded FuSa 6.22LTS release series, Arm Compiler for Embedded 6.22.1.

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 FuSa.

SDCOMP-68439

The `-fthreadsafe-statics` and `-fno-threadsafe-statics` options have been changed from Community features to Product features.

For more information, see the *-fthreadsafe-statics, -fno-threadsafe-statics* section of the *Reference Guide*.

SDCOMP-68073

The `-fstrict-flex-arrays=<value>` option to control what the compiler considers to be a flexible array has been changed from a Community feature to a Product feature.

For more information, see the *-fstrict-flex-arrays=<value>* section of the *Reference Guide*.

SDCOMP-67299

The `-mcpu=cortex-a510` option has been added to the compiler for AArch32 state. To target the Cortex-A510 processor in AArch32 state, select from the following `armclang` options:

Cryptographic Extension	Options
Included	`--target=arm-arm-none-eabi -mcpu=cortex-a510 -mfpu=crypto-neon-fp-armv8`
Not included	`--target=arm-arm-none-eabi -mcpu=cortex-a510`

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-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, see the *Alphabetical list of A64 base instructions* section of version L.a of the *Arm Architecture Reference Manual for the A-profile architecture*.

SDCOMP-65404

`--cpu=<name>` options for the Cortex-M52 processor have been added to the linker and the ELF processing utility.

For more information, see the *Combinations of architecture features supported for the Cortex-M52 processor* section of the *Reference Guide*.

3.2 Defect fixes

This section contains information about defect fixes made in this release of Arm Compiler for Embedded FuSa. 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-68535

When compiling for AArch64 state and at any optimization level except `-O0`, the compiler could incorrectly generate an instruction that performs a 32-bit multiplication operation instead of a 64-bit multiplication operation. This has been fixed.

SDCOMP-68410

When compiling for AArch64 state, the compiler could generate code that incorrectly performs unsigned extension instead of signed extension. This has been fixed.

SDCOMP-68363

The inline assembler and integrated assembler could incorrectly fail to ignore a string comparison assembler directive inside a conditional assembler directive. This has been fixed.

SDCOMP-68219

When compiling for AArch32 state, the inline assembler and integrated assembler incorrectly failed to report an error for an invalid `VCVT` instruction that specifies different destination and source registers, and converts between floating-point and fixed-point values. This has been fixed. The inline assembler and integrated assembler now report the following error:

- `source and destination registers must be the same`

SDCOMP-68081

The inline assembler and integrated assembler could incorrectly report one of the following errors for a valid `VSCCLRM` instruction:

- `invalid register in register list`
- `register expected`

This has been fixed.

SDCOMP-67974

When compiling with target options that enable the M-profile Vector Extension (MVE), and at any optimization level except `-O0`, the compiler could generate incorrect code for a memory access operation. This has been fixed.

SDCOMP-67921

When compiling for AArch64 state, and in a C++ source language mode, the compiler could generate code that incorrectly fails to ignore a function parameter of zero size. This has been fixed.

SDCOMP-67823

When compiling for AArch64 state, the compiler could generate incorrect code for a function with pointer authentication branch protection using the Program Counter as a second diversifier for return address signing. This has been fixed.

SDCOMP-67799

When compiling at any optimization level except `-O0`, the compiler could generate incorrect code for an M-profile Vector Extension (MVE) intrinsic of the form `*vsbcq*()` defined in the `<arm_mve.h>` system header. This has been fixed.

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

SDCOMP-67678

When compiling at any optimization level except `-O0`, the compiler could generate incorrect code for a function that contains a call to the following forms of M-profile Vector Extension (MVE) intrinsics defined in the `<arm_mve.h>` system header:

- `*vadcq_*()`
- `*vsbcq_*()`

This has been fixed.

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

SDCOMP-67666

The compiler could generate incorrect code for a Scalable Vector Extension version 2 (SVE2) intrinsic of the form `svwhilele_*()` defined in the `<arm_sve.h>` system header. This has been fixed.

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

SDCOMP-67662

The compiler could generate incorrect code for an M-profile Vector Extension (MVE) intrinsic of the form `*vcmlaq_*_f32()` defined in the `<arm_mve.h>` system header. This has been fixed.

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

SDCOMP-67658

When compiling for AArch32 state, the compiler could generate incorrect code for the `__builtin_trap()` built-in function. This has been fixed.

`__builtin_trap()` is a [COMMUNITY] feature.

SDCOMP-67650

When compiling at any optimization level except `-O0`, the compiler could generate incorrect code for the following forms of M-profile Vector Extension (MVE) intrinsics defined in the `<arm_mve.h>` system header:

- `*vfmaq_m*()`

- `*vfmsq_m*()`
- `*vfmasq_m*()`

This has been fixed.

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

SDCOMP-67638

The compiler could generate incorrect code for an M-profile Vector Extension (MVE) intrinsic `z` of the form `*vcmp*_f*()` defined in the `<arm_mve.h>` system header, where a parameter to `z` includes a NaN. This has been fixed.

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

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-67424

When assembling for AArch32 state, the inline assembler and integrated assembler incorrectly failed to report an error for a `MOVT` or `MOVW` instruction which specifies a source operand with an offset that is outside the range for the instruction. This has been fixed. The inline assembler and integrated assembler now report the following error:

- `Relocation Not In Range`

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-66894

The compiler incorrectly failed to report an error for an `-march` or `-mcpu` option that specifies an invalid feature modifier. This has been fixed. The compiler now reports one of the following errors:

- `unsupported argument '<name>+<feature>' to option '-march='`
- `unsupported argument '<name>+<feature>' to option '-mcpu='`

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-66632

When assembling for AArch64 state, the integrated assembler incorrectly failed to implicitly set the minimum alignment requirement for a user-defined executable section to 4 bytes. This has been fixed.

SDCOMP-66328

When compiling for AArch64 state, the compiler could generate incorrect C++ exception unwinding information and incorrect debug information for a function with pointer authentication branch protection using the Program Counter as a second diversifier for return address signing. This has been fixed.

SDCOMP-66256

When compiling for AArch64 state, the compiler could generate code that incorrectly split a Homogenous Floating-point Aggregate (HFA) parameter between a register and the stack. Such code does not conform to the *Procedure Call Standard for the Arm 64-bit Architecture*. This has been fixed.

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-65590

When compiling at any optimization level except `-O0`, the compiler could generate incorrect code for a function that contains a call to an M-profile Vector Extension (MVE) intrinsic of the form `*vsetq_lane_*32()` defined in the `<arm_mve.h>` system header. This has been fixed.

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

SDCOMP-65579

The compiler could generate incorrect code for a Scalable Vector Extension (SVE) intrinsic of the form `svuzp*()` defined in the `<arm_sve.h>` system header. This has been fixed.

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

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-65550

When compiling for AArch64 state with target options that enable the Advanced SIMD Extension (FEAT_AdvSIMD), the compiler could generate incorrect code. This has been fixed.

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-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-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-64255

When assembling for AArch32 state, the inline assembler and integrated assembler incorrectly failed to report an error for a `DMB`, `DSB`, or `ISB` instruction that has an invalid operand. This has been fixed. The inline assembler and integrated assembler now report the following error:

- `expected an immediate or barrier type`

SDCOMP-64194

When compiling for an Armv6-M target or an Armv8-M target without the Main Extension, and with `-mframe-chain=aapcs`, the compiler could generate incorrect code for the `__builtin_return_address(<N>)` built-in function. This has been fixed.

`__builtin_return_address(<N>)` is a [COMMUNITY] feature.

SDCOMP-63984

When compiling code that contains a thread-local variable with `-mexecute-only`, the compiler incorrectly failed to report the following error:

- `thread-local storage is not supported for the current target`

This has been fixed.

SDCOMP-63912

The compiler could generate incorrect code for an access to a bit-field with a size that is greater than the size of its type. Such incorrect code does not conform to the *Procedure Call Standard for the Arm Architecture*. This has been fixed.

SDCOMP-63911

When compiling for AArch32 state and in a C++ source language mode, the compiler incorrectly assumed that a register is not required for a function parameter which is a `class`, `struct`, or `union` without any members. Subsequently, the compiler generated code that incorrectly did not conform to the *Procedure Call Standard for the Arm Architecture*. This has been fixed.

SDCOMP-63205

When compiling for AArch32 state, a big-endian target that includes the Advanced SIMD Extension (FEAT_AdvSIMD), and without `-fno-vectorize`, the compiler could generate incorrect code for a loop. This has been fixed.

SDCOMP-63114

When compiling with `-fsanitize=memtag-stack`, the compiler could generate an incorrect `STG` instruction. This has been fixed.

SDCOMP-62378

When compiling for AArch32 state, the compiler could generate incorrect code for a Neon intrinsic of the form `vld*_x*()` or `vst*_x*()` defined in the `<arm_neon.h>` system header. This has been fixed.

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

SDCOMP-62176

When compiling for a big-endian target and AArch32 state, the compiler could generate incorrect code for a function which has a half-precision floating-point parameter that is passed using the stack. 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.

SDCOMP-55200

When compiling with target options that enable the M-profile Vector Extension (MVE) for floating-point types, and with `-mfpu=fpv5-d16` or `-mfpu=fpv5-sp-d16`, the compiler incorrectly set bit 1 of the Arm C Language Extensions (ACLE) feature macro `__ARM_FEATURE_MVE`. This has been fixed.

SDCOMP-55040

When compiling at any optimization level except `-O0`, with `-fsanitize=memtag-stack`, and for a target that supports the Memory Tagging Extension (FEAT_MTE), the compiler could generate incorrect code. This has been fixed.

SDCOMP-50408

When compiling for AArch32 state and a big-endian target that includes the Advanced SIMD Extension (FEAT_AdvSIMD), the compiler could generate incorrect code for a function that has a parameter of vector type. This has been fixed.

3.2.2 ELF processing utility, fromelf

This section contains a list of defect fixes made in the ELF processing utility, `fromelf`.

SDCOMP-66692

The fromelf utility could disassemble the `VSCCLRM` instruction incorrectly. This has been fixed.

3.2.3 Libraries and system headers

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

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

The Arm C library variant for Armv8-R AArch64 targets without hardware floating-point support could return an incorrect result or set the `FE_INEXACT` floating-point exception flag incorrectly for a double-precision floating-point operation. This has been fixed.

SDCOMP-63111

The Arm C Library implementation of BFloat16 intrinsics of the form `vcvt*_f32_bf16()` defined in the `<arm_neon.h>` system header incorrectly relied on C undefined behavior. This has been fixed.

For more information about BFloat16 intrinsics, see https://developer.arm.com/architectures/instruction-sets/intrinsics.

SDCOMP-45879

The Arm C library implementations of the `bsearch()` and `qsort()` functions could incorrectly corrupt the stack when used to process an array larger than 4GB. This has been fixed.

4. Support

This chapter includes guidance on how to obtain support for using Arm Compiler for Embedded FuSa 6.22.2.

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 output of running `armlm --inspect`.
- 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 *Arm Compiler for Embedded FuSa Reference Guide version 6.22.2*.

5. Release history

This chapter contains the history of Arm Compiler for Embedded FuSa 6.22LTS releases.

Version	Release Date
Arm Compiler for Embedded FuSa 6.22.2	19 Aug 2025
Arm Compiler for Embedded FuSa 6.22.1	21 Aug 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
062202-01	19 August 2025	Non-Confidential	Initial release

Change history

Change	Location
Initial release for Arm Compiler for Embedded FuSa 6.22.2	-

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**.

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

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.


Tip

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


Remember

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.

Arm documents are available on developer.arm.com/documentation.

Confidential documents are only available to licensees, when logged in. Each document link in the tables below provides direct access to the online version of the document.

Arm product resources	Document ID	Confidentiality
Arm Compiler for Embedded FuSa 6.22LTS Defect Notification Report	110099	Non-Confidential
Arm Compiler for Embedded FuSa 6.22LTS documentation index	KA006002	Non-Confidential
Arm Compiler for Embedded FuSa Migration and Compatibility Guide version 6.22.2	109444	Non-Confidential
Arm Compiler for Embedded FuSa Qualification Kit Safety Manual version 6.22.2	109409	Confidential
Arm Compiler for Embedded FuSa Reference Guide version 6.22.2	109443	Non-Confidential
Arm Compiler for Embedded FuSa User Guide version 6.22.2	109442	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