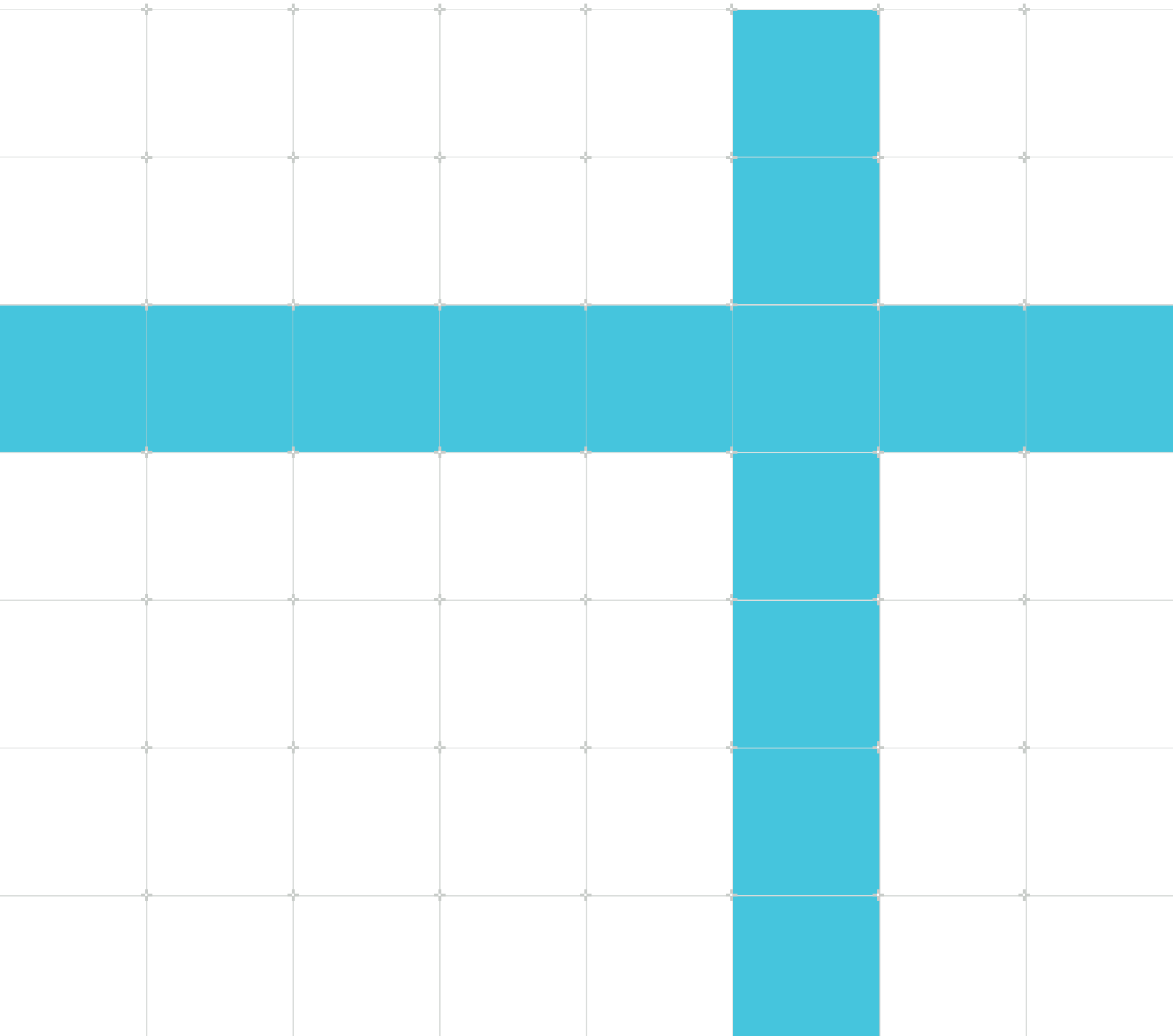
Arm Compiler for Embedded FuSa 6.16.3 User Documentation and Qualification Kit Addendum

Version January 2026

Issue 01
111217_2026-01_01_en



Arm Compiler for Embedded FuSa 6.16.3 User Documentation and Qualification Kit Addendum

This document (111217_2026-01_01_en) was issued on 2026-01-29. There might be a later issue at https://developer.arm.com/documentation/111217

The product version is January 2026.

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 6.16.3.

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. Introduction

This document serves as an addendum for the User Documentation and the Qualification Kit for the Arm Compiler for Embedded FuSa 6.16.3 release. It summarizes the User Documentation and Qualification Kit deliverables available for this release.

- The User Documentation issued with Arm Compiler for Embedded FuSa 6.16.2 remains valid for Arm Compiler for Embedded FuSa 6.16.3 subject to the following:
  - The Changes applicable to the Arm Compiler for Embedded FuSa 6.16.2 Reference Guide for use with Arm Compiler for Embedded FuSa 6.16.3 section of this document.
  - The Changes applicable to the Arm Compiler for Embedded FuSa 6.16.2 User Guide for use with Arm Compiler for Embedded FuSa 6.16.3 section of this document.
  - The Documentation for new Product Features added in Arm Compiler for Embedded FuSa 6.16.3 chapter of this document.
  - Information about the Documentation synchronization faults described in the following sections of the latest *Arm Compiler for Embedded FuSa 6.16LTS Defect Notification Report*:
    - Documentation synchronization faults that affect qualified components of the toolchain
    - Documentation synchronization faults that affect unqualified components of the toolchain
    - Documentation synchronization faults that affect both qualified and unqualified components of the toolchain
- The Qualification Kit documents issued with Arm Compiler for Embedded FuSa 6.16.2 remain valid for Arm Compiler for Embedded FuSa 6.16.3 as per the Qualification Kit documents for Arm Compiler for Embedded FuSa 6.16.3 section of this document. This section explains that:
  - A new version of the *Qualification Kit Defect Report* is provided with Arm Compiler for Embedded FuSa 6.16.3.
  - A new version of the *Qualification Kit Release History* is is provided with Arm Compiler for Embedded FuSa 6.16.3.
  - The *Qualification Kit Safety Manual* document issued with Arm Compiler for Embedded FuSa 6.16.2 can be used with Arm Compiler for Embedded FuSa 6.16.3 subject to the changes described in Changes applicable to the Arm Compiler for Embedded FuSa 6.16LTS Qualification Kit Safety Manual version 6.16.2 for use with Arm Compiler for Embedded FuSa 6.16.3.
  - The *Qualification Kit Development Process* document issued with Arm Compiler for Embedded FuSa 6.16.2 can be used with Arm Compiler for Embedded FuSa 6.16.3 subject to the changes described in Changes applicable to the Arm Compiler for Embedded FuSa 6.16LTS Qualification Kit Development Process version 6.16.2 for use with Arm Compiler for Embedded FuSa 6.16.3.
  - The remaining Qualification Kit documents issued with Arm Compiler for Embedded FuSa 6.16.2 can be used with Arm Compiler for Embedded FuSa 6.16.3 without changes.

The latest *Arm Compiler for Embedded FuSa 6.16LTS Defect Notification Report* can be used for an up-to-date list of known safety-related defects that affect each release of Arm Compiler for Embedded FuSa 6.16LTS.

2. Applying the user documentation and Qualification Kit for Arm Compiler for Embedded FuSa 6.16.2 to Arm Compiler for Embedded FuSa 6.16.3

The Arm Compiler for Embedded FuSa 6.16.3 release does not include updates to all the User Documentation and Qualification Kit documents. This chapter explains which existing documents for Arm Compiler for Embedded FuSa 6.16.2 can be used with Arm Compiler for Embedded FuSa 6.16.3.

2.1 User Documentation for Arm Compiler for Embedded FuSa 6.16.3

For existing features, the user documentation for the Arm Compiler for Embedded FuSa 6.16.2 release is also applicable to the Arm Compiler for Embedded FuSa 6.16.3 release, except as described in this Addendum document, the *Qualification Kit Defect Report*, or the *Arm Compiler for Embedded FuSa 6.16LTS Defect Notification Report*.

The documents listed in the following table can be used as User Documentation for Arm Compiler for Embedded FuSa 6.16.3. These documents are available online on Arm Developer.

Table 2-1: User Documentation applicable to Arm Compiler for Embedded FuSa 6.16.3

User documentation document	Document ID	Document version	Justification
Release Notes	110644	6.16.3	New document for Arm Compiler for Embedded FuSa 6.16.3.
Arm C and C++ Libraries and Floating-Point Support User Guide	102286	6.16.2LTS	The scope of the document is identical between Arm Compiler for Embedded FuSa 6.16.2 and 6.16.3.
Errors and Warnings Reference Guide	102287	6.16.2LTS	The scope of the document is identical between Arm Compiler for Embedded FuSa 6.16.2 and 6.16.3.
Migration and Compatibility Guide	102285	6.16.2LTS	The scope of the document is identical between Arm Compiler for Embedded FuSa 6.16.2 and 6.16.3.
Reference Guide	102284	6.16.2LTS	The scope of the document is identical between Arm Compiler for Embedded FuSa 6.16.2 and 6.16.3, except as described in Changes applicable to the Arm Compiler for Embedded FuSa 6.16.2 Reference Guide for use with Arm Compiler for Embedded FuSa 6.16.3. Additional Product Features added in Arm Compiler for Embedded FuSa 6.16.3 are documented in Documentation for new Product Features added in Arm Compiler for Embedded FuSa 6.16.3 instead of the *Arm Compiler for Embedded FuSa 6.16.2 Reference Guide*.

User documentation document	Document ID	Document version	Justification
User Documentation and Qualification Kit Addendum	111217	January 2026	New document for Arm Compiler for Embedded FuSa 6.16.3.
User Guide	102283	6.16.2LTS	The scope of the document is identical between Arm Compiler for Embedded FuSa 6.16.2 and 6.16.3, except as described in Changes applicable to the Arm Compiler for Embedded FuSa 6.16.2 User Guide for use with Arm Compiler for Embedded FuSa 6.16.3.

2.1.1 Changes applicable to the Arm Compiler for Embedded FuSa 6.16.2 Reference Guide for use with Arm Compiler for Embedded FuSa 6.16.3

This section includes details about changes applicable to the *Arm Compiler for Embedded FuSa 6.16.2 Reference Guide* when using it with Arm Compiler for Embedded FuSa 6.16.3.

2.1.1.1 Changes to the __attribute__((target("options"))) function attribute section

The *__attribute__((target("options"))) function attribute* section of the *Arm Compiler for Embedded FuSa 6.16.2 Reference Guide* does not apply to Arm Compiler for Embedded FuSa 6.16.3.

In Arm Compiler for Embedded FuSa 6.16.2, only `__attribute__((target("branch-protection=none")))` is available as a Product Feature. Instead, in Arm Compiler for Embedded FuSa 6.16.3, the following additional attributes have been added as new Product Features:

- The `__attribute__((target("branch-protection=<protection>")))` function attribute, where `<protection>` includes `pac-ret`.
- The `__attribute__((target("harden-pac-ret=<value>")))` function attribute.

2.1.1.2 Changes to the --interleave=option section

Support has been removed for the `fromelf --interleave=source` option.

Therefore, the documentation for `--interleave=source` from the *--interleave=option* section of the *Arm Compiler for Embedded FuSa 6.16.2 Reference Guide* does not apply to Arm Compiler for Embedded FuSa 6.16.3. To view the mapping between disassembly and source code, compile with `-g` and use an external debugger tool such as Arm Development Studio.

2.1.1.3 Changes to the --emit=option[,option,...] section

Support has been removed for the `fromelf --emit=got` option to print the contents of the Global Offset Table (GOT) section.

Therefore, the documentation for `--emit=got` from the *--emit=option[,option,...]* section of the *Arm Compiler for Embedded FuSa 6.16.2 Reference Guide* does not apply to Arm Compiler for Embedded FuSa 6.16.3.

2.1.2 Changes applicable to the Arm Compiler for Embedded FuSa 6.16.2 User Guide for use with Arm Compiler for Embedded FuSa 6.16.3

This section includes details about changes applicable to the *Arm Compiler for Embedded FuSa 6.16.2 User Guide* when using it with Arm Compiler for Embedded FuSa 6.16.3.

2.1.2.1 Changes to the Installing a standalone Arm Compiler for Embedded FuSa on Windows platforms section

The content of the *Installing a standalone Arm Compiler for Embedded FuSa on Windows platforms* section of the *Arm Compiler for Embedded FuSa 6.16.2 User Guide* is superseded by the contents of this section.

To install Arm Compiler for Embedded FuSa 6.16.3 as a standalone product on Windows x86_64 host platforms:

1. Run the `Arm Compiler for Embedded FuSa 6.16.3.msi` installer on your machine.
2. Follow the on-screen instructions.
3. Some license types require you to complete further configuration steps. To check if your license requires further configuration, and to learn how to configure that license, see:
   - Arm Compiler Licensing Configuration for FlexNet Publisher (FNP) licenses.
   - The User-based Licensing (UBL) User Guide for UBL licenses.

If you have an older version of Arm Compiler for Embedded FuSa 6.16LTS and you want to upgrade, Arm recommends that you uninstall the older version of Arm Compiler for Embedded FuSa 6.16LTS before installing Arm Compiler for Embedded FuSa 6.16.3.

Arm Compiler for Embedded FuSa 6.16.3 requires the Universal C Runtime in Windows to be installed. For more information, see Update for Universal C Runtime in Windows.

For use with Keil MDK version 5, the toolchain must be installed into the `ARM` sub-directory of the Keil MDK version 5 installation directory. For example, if your Keil MDK version 5 installation directory is `C:\Keil_v5`, the recommended installation path is `C:\Keil_v5\ARM\ARMCompiler6.16.3`.

2.2 Qualification Kit documents for Arm Compiler for Embedded FuSa 6.16.3

The Qualification Kit documents for Arm Compiler for Embedded FuSa 6.16.2 are also applicable to the Arm Compiler for Embedded FuSa 6.16.3 release, except as described in this Addendum. New *Qualification Kit Defect Report* and *Qualification Kit Release History* are provided with Arm Compiler for Embedded FuSa 6.16.3.

The documents listed in the following table can be used as Qualification Kit documents for Arm Compiler for Embedded FuSa 6.16.3. These documents are available via the Qualification Kit download packages for Arm Compiler for Embedded FuSa 6.16.3 on Arm Product Download Hub (PDH).

Table 2-2: Qualification Kit documents applicable to Arm Compiler for Embedded FuSa 6.16.3

Qualification Kit document	Document ID	Document version	Justification
Safety Manual	102288	6.16.2	The scope of the *Qualification Kit Safety Manual* is identical Compiler for Embedded FuSa 6.16.2 and 6.16.3, except as described in Changes applicable to the Arm Compiler for Embedded FuSa 6.16LTS Qualification Kit Safety Manual version 6.16.2 for use with Arm Compiler for Embedded FuSa 6.16.3.
Defect Report	102291	6.16.3	New document for Arm Compiler for Embedded FuSa 6.16.3. For an up-to-date list of known safety-related defects that affect each release of Arm Compiler for Embedded FuSa 6.16LTS, see the latest *Arm Compiler for Embedded FuSa 6.16LTS Defect Notification Report* on Arm Developer.
Development Process	102289	6.16.2	The scope of the document is identical between Arm Compiler for Compiler for Embedded FuSa 6.16.2 and 6.16.3, except as described in Changes applicable to the Arm Compiler for Embedded FuSa 6.16LTS Qualification Kit Development Process version 6.16.2 for use with Arm Compiler for Embedded FuSa 6.16.3.
Release History	102290	6.16.3	New document for Arm Compiler for Embedded FuSa 6.16.3.
Test Report	102292	6.16.2	Language conformance testing of Arm Compiler for Embedded FuSa 6.16.3 has been completed using the methodology described in the *Perennial test methodology* and *SuperTest test methodology* sections of the *Qualification Kit Test Report*. The results for Arm Compiler for Embedded FuSa 6.16.3 are identical to the results for Arm Compiler for Embedded FuSa 6.16.2.

2.2.1 Changes applicable to the Arm Compiler for Embedded FuSa 6.16LTS Qualification Kit Safety Manual version 6.16.2 for use with Arm Compiler for Embedded FuSa 6.16.3

This section includes details about changes applicable to the *Arm Compiler for Embedded FuSa 6.16LTS Qualification Kit Safety Manual* version 6.16.2 when using it with Arm Compiler for Embedded FuSa 6.16.3.

2.2.1.1 General changes

This section contains a list of general changes that are applicable to the *Arm Compiler for Embedded FuSa 6.16LTS Qualification Kit Safety Manual* version 6.16.2 for use with Arm Compiler for Embedded FuSa 6.16.3.

The following changes are applicable:

- For all instances where the toolchain version is listed as 6.16.2, you can infer that the content is applicable as if the version was listed as 6.16.3.
- For all instances where the document provides a reference to the *Arm Compiler for Embedded FuSa 6.16LTS Qualification Kit Defect Report*, you can also see the latest *Arm Compiler for Embedded FuSa 6.16LTS Defect Notification Report* for an up-to-date list of known safety-related defects that affect each release of Arm Compiler for Embedded FuSa 6.16LTS.

2.2.1.2 Changes to the Qualification kit file naming conventions section

This section provides information about changes to the *Qualification kit file naming conventions* section of the *Arm Compiler for Embedded FuSa 6.16LTS Qualification Kit Safety Manual* version 6.16.2 for use with Arm Compiler for Embedded FuSa 6.16.3.

The *Qualification kit file naming conventions* section explains that the *Qualification Kit Defect Report* is provided in HTML format. This is not applicable to Arm Compiler for Embedded FuSa 6.16.3.

The *Qualification Kit Defect Report* for Arm Compiler for Embedded FuSa 6.16.3 is provided in PDF and JSON formats only.

2.2.1.3 Changes to the Assumptions about the reader section

This section provides information about changes to the *Assumptions about the reader* section of the *Arm Compiler for Embedded FuSa 6.16LTS Qualification Kit Safety Manual* version 6.16.2 for use with Arm Compiler for Embedded FuSa 6.16.3.

The following additional assumption about the reader applies when using Arm Compiler for Embedded FuSa 6.16.3:

AC_UR_AU_5

It is assumed that you have access to the *Arm Compiler for Embedded FuSa 6.16.3 User Documentation and Qualification Kit Addendum*, and refer to it together with the applicable User Documentation and Qualification Kit documents when using Arm Compiler for Embedded FuSa 6.16.3.

2.2.1.4 Changes to the Legal notice section

The content of the *Legal notice* section of the *Arm Compiler for Embedded FuSa 6.16LTS Qualification Kit Safety Manual* version 6.16.2 is superseded by the contents of this section.

Use of Arm Compiler for Embedded FuSa 6.16.3 and the applicable User Documentation and Qualification Kit is subject to the following:

- The End-User License Agreement (EULA) included in the `license_terms/license_agreement.txt` file from the Qualification Kit download package for Arm Compiler for Embedded FuSa 6.16.3.
- The proprietary notice at the front of this each document applicable to Arm Compiler for Embedded FuSa 6.16.3.

2.2.1.5 Changes to the Product Configuration section

This section provides information about changes to the *Product Configuration* section of the *Arm Compiler for Embedded FuSa 6.16LTS Qualification Kit Safety Manual* version 6.16.2 for use with Arm Compiler for Embedded FuSa 6.16.3.

The *Product Configuration* section contains a list of product configuration files used for license management by Arm Compiler for Embedded FuSa 6.16.2.

The `<install directory>\sw\mappings\ult.elmap` license management file has been added in Arm Compiler for Embedded FuSa 6.16.3. This file is not a user-modifiable file.

2.2.1.6 Changes to the User Requirements section

This section provides information about changes to the *User Requirements* section of the *Arm Compiler for Embedded FuSa 6.16LTS Qualification Kit Safety Manual* version 6.16.2 for use with Arm Compiler for Embedded FuSa 6.16.3.

The following additional User Requirements are applicable when using Arm Compiler for Embedded FuSa 6.16.3:

AC_UR_CL_14

The `-mharden-pac-ret=<option>` command-line option must only be used when compiling for AArch64 state. This requirement is about the -mharden-pac-ret command-line option for the compiler and integrated assembler, armclang.

AC_UR_CL_15

The `__attribute__((target("harden-pac-ret=<value>")))` function attribute must only be used when compiling for AArch64 state. This requirement is about the __attribute__((target("harden-pac-ret=<value>"))) function attribute.

AC_UR_CL_16

If you annotate a function `F` with the `__attribute__((target("harden-pac-ret=load-return-address")))` function attribute, you must manually inspect your source code and ensure that you use also annotate `F` with a `__attribute__((target("branch-protection=<protection>")))` function attribute, where `<protection>` includes `pac-ret`.

This requirement is about the following function attributes:

- __attribute__((target("branch-protection=<protection>")))
- __attribute__((target("harden-pac-ret=<value>")))

AC_UR_CL_17

The `__attribute__((target("branch-protection=<protection>")))` function attribute must only be used when compiling for AArch64 state. This requirement is about the __attribute__((target("branch-protection=<protection>"))) function attribute.

AC_UR_CL_18

The `__attribute__((target("branch-protection=<protection>")))` function attribute must only be used for the following purposes:

- To disable branch protection by specifying `<protection>` as `none`.
- To enable return address signing hardening as specified by the documentation for the `__attribute__((target("harden-pac-ret=load-return-address")))` function attribute.

This requirement is about the following function attributes:

- __attribute__((target("branch-protection=<protection>")))
- __attribute__((target("harden-pac-ret=<value>")))

AC_UR_CL_19

The `-isystem <directory>` command-line option must only be used when using both the Arm Certified C Library and the Arm Certified C++ Library.

Use of the `-isystem <directory>` command-line option for any other purpose is considered a [COMMUNITY] feature. In such a scenario, you must ensure that the requirements for the functional safety specification that you are using are met.

This requirement is about the -isystem <directory> command-line option for the compiler and integrated assembler, armclang.

AC_UR_CL_20

The `-nobuiltininc` command-line option must only be used when using both the Arm Certified C Library and the Arm Certified C++ Library.

Use of the `-nobuiltininc` command-line option for any other purpose is considered a [COMMUNITY] feature. In such a scenario, you must ensure that the requirements for the functional safety specification that you are using are met.

This requirement is about the -nobuiltininc command-line option for the compiler and integrated assembler, armclang.

AC_UR_CL_21

The `-mharden-sls=<option>` command-line option must only be used when compiling for AArch64 state. This requirement is about the -mharden-sls command-line option for the compiler and integrated assembler, armclang.

AC_UR_AU_5

It is assumed that you have access to the *Arm Compiler for Embedded FuSa 6.16.3 User Documentation and Qualification Kit Addendum*, and refer to it together with the applicable User Documentation and Qualification Kit documents when using Arm Compiler for Embedded FuSa 6.16.3.

2.2.2 Changes applicable to the Arm Compiler for Embedded FuSa 6.16LTS Qualification Kit Development Process version 6.16.2 for use with Arm Compiler for Embedded FuSa 6.16.3

This section includes details about changes applicable to the *Arm Compiler for Embedded FuSa 6.16LTS Qualification Kit Development Process* version 6.16.2 when using it with Arm Compiler for Embedded FuSa 6.16.3.

2.2.2.1 Changes to the Customer case management section

The content of the *Customer case management* section of the *Arm Compiler for Embedded FuSa 6.16LTS Qualification Kit Development Process* version 6.16.2 is superseded by the contents of this section.

All customer questions and reports are logged as a customer case in the customer case database. Each customer case receives a unique Case Number which remains constant throughout the lifetime of the case.

Raising a customer case

A customer can raise a case with the support group through Arm Developer.

Customers can register for an Arm account on the Arm Developer website at https://developer.arm.com/support-hub and raise a case directly in the customer support portal.

Customer case information

The following table shows an extract of some of the information that a customer case contains:

Information	Description
Case Number	The unique identifier of the case.
Account	The company account associated with the primary customer for the case.
Contact	The primary customer for the case.
Product	The product that the case is about.
Case Owner	The individual or queue within Arm that the case is assigned to.
Priority	The current priority of the case.
Status Reason	The current state of the case.
Case Resolution	Summary of the case after it has been resolved.
External Note	Log of communication between the support group and the customer.
Internal Note	Log of communication private to Arm. For example, conversations with the development team.
Attachments	Files that are attached to the case. For example, reproducible examples.
Case Relationships	Links to other related cases.
External Cross Reference	Links to the Arm internal issue database.


Note

The customer support portal only displays a subset of this information.

Customer case assignment

All customer cases are assigned to individuals or queues within Arm.

The following table shows the properties of the two states:

Owner	Properties
Individual	• An individual is actively working on the case. • An individual is responsible for progressing the case. • If the individual is unable to work on the case, the individual can reassign the case to a queue or another individual. For example, they can reassign a case while they are out of the office. • Cases waiting for an update from the development team remain assigned to an individual.
Queue	• The case is not being actively worked on. • The case can be assigned to an individual.

2.2.2.2 Changes to the Defect Notification Reports section

The content of the *Defect Notification Reports* section of the *Arm Compiler for Embedded FuSa 6.16LTS Qualification Kit Development Process* version 6.16.2 is superseded by the contents of this section.

The Arm Compiler Support team co-ordinates with the development team to publish a regularly updated list of known safety-related defects that affected each release of Arm Compiler for Embedded FuSa 6.16LTS.

This list is published as the non-confidential document *Arm Compiler for Embedded FuSa 6.16LTS Defect Notification Report*. The document is provided in the following formats:

- A version that can be browsed online on Arm Developer.
- A PDF file that can be downloaded for offline access.
- A JavaScript Object Notation (JSON) file for automated consumption.

For more information about the Defect Notification Report, see the *Introduction* section of the *Arm Compiler for Embedded FuSa 6.16LTS Defect Notification Report*.

3. Documentation for new Product Features added in Arm Compiler for Embedded FuSa 6.16.3

This chapter provides User Documentation for new Product Features added to Arm Compiler for Embedded FuSa 6.16.3. It must be read in conjunction with the *Arm Compiler for Embedded FuSa 6.16.2 Reference Guide*.

The following Product Features have been added to Arm Compiler for Embedded FuSa 6.16.3:

- The `__attribute__((target("branch-protection=<protection>")))` function attribute.
- The `__attribute__((target("harden-pac-ret=<value>")))` function attribute.
- The `-isystem <directory>` command-line option for the compiler and integrated assembler, `armclang`.
- The `-mharden-pac-ret` command-line option for the compiler and integrated assembler, `armclang`.
- The `-mharden-sls` command-line option for the compiler and integrated assembler, `armclang`.
- The `-nobuiltininc` command-line option for the compiler and integrated assembler, `armclang`.

3.1 __attribute__((target("harden-pac-ret=<value>"))) function attribute

The `__attribute__((target("harden-pac-ret=<value>")))` function attribute can be used to control return address signing hardening on a per-function basis.

The function attribute takes precedence over the `-mharden-pac-ret=<option>` command-line option.

Syntax

```
__attribute__((target("harden-pac-ret=<value>")))
```

Parameters

`<value>`

One of:

`none`

No return address signing hardening.

`load-return-address`

Enables return address signing hardening.

This function attribute must be used with a `__attribute((target("branch-protection=<protection>")))` function attribute where `<protection>` includes `pac-ret`.

Otherwise, `__attribute__((target("harden-pac-ret=load-return-address")))` has no effect, and the compiler reports `warning: 'harden-pac-ret' attribute requires 'branch-protection=pac-ret'; 'target' attribute ignored`.

Restrictions

This function attribute is only a Product Feature when compiling for AArch64 state.

Use of this function attribute is subject to the following User Requirements described in Changes to the User Requirements section of the Arm Compiler for Embedded FuSa 6.16LTS Qualification Kit Safety Manual version 6.16.2:

- AC_UR_CL_15
- AC_UR_CL_16

Justification for addition as a new Product Feature

The function attribute provides a mitigation for the PACMAN security vulnerability.

Arm has completed validation testing of this attribute.

Impact on a project migrated from Arm Compiler for Embedded FuSa 6.16.2 to Arm Compiler for Embedded FuSa 6.16.3

None.

Arm Compiler for Embedded FuSa 6.16.2 does not implement the `__attribute__(target("harden-pac-ret=<value>"))` function attribute.

Compiling with Arm Compiler for Embedded FuSa 6.16.3 does not implicitly add the `__attribute__(target("harden-pac-ret=load-return-address"))` function attribute to source code.

Related information

-mharden-pac-ret command-line option for the compiler and integrated assembler, armclang on page 21

3.2 __attribute__((target("branch-protection=<protection>"))) function attribute

The `__attribute__((target("branch-protection=<protection>")))` function attribute can be used to control branch protection on a per-function basis.

The function attribute takes precedence over the `-mbranch-protection=<protection>` command-line option, and is required when using `__attribute__((target("harden-pac-ret=load-`

`return-address"))`. For more information, see the `__attribute__((target("harden-pac-ret=<value>")))` function attribute section of this document.

Syntax

```
__attribute__((target("branch-protection=<protection>")))
```

Parameters

`<protection>`

One of:

`none`

This disables all types of branch protection.

`pac-ret`

This enables branch protection using Pointer Authentication using key A. This protects functions that save the Link Register (LR) on the stack. This does not generate branch protection code for leaf functions that do not save the LR on the stack.

If you specify a value for `<protection>` which includes `pac-ret`, you can also specify one or more of the following additional parameters to modify the pointer authentication protection using the `+` separator:

`leaf`

This enables pointer authentication on all leaf functions, including the leaf functions that do not save the Link Register on the stack.

`b-key`

This enables pointer authentication with Key B, rather than Key A.

Key A and Key B refer to secret values that are used for generating a signature for authenticating the return addresses.

Restrictions

This function attribute is only a Product Feature when all the following are true:

- The program is compiled for AArch64 state.
- One of the following is true:
  - `<protection>` is `none`.
  - `<protection>` includes `pac-ret` and the function attribute is used in combination with `__attribute__((target("harden-pac-ret=load-return-address")))`.

Use of this function attribute is subject to the following User Requirements described in Changes to the User Requirements section of the Arm Compiler for Embedded FuSa 6.16LTS Qualification Kit Safety Manual version 6.16.2:

- AC_UR_CL_17
- AC_UR_CL_18

Operation

The `__attribute__((target("branch-protection=<protection>")))` function attribute overrides the behavior specified using the `-mbranch-protection=<protection>` command-line option on a per-function basis.

For example, when compiling with `-mbranch-protection=pac-ret`, annotating a function `F` with `__attribute__((target("branch-protection=none")))` disables `pac-ret` branch protection for `F`.

For more information about how each type of branch protection works, see the `-mbranch-protection` section of the *Arm Compiler for Embedded FuSa 6.16.2 Reference Guide*.

Note

The `__attribute__((target("branch-protection=<protection>")))` function attribute is typically used in combination with the `__attribute__((target("harden-pac-ret=<value>")))` attribute. For more information, see the `__attribute__((target("harden-pac-ret=<value>")))` function attribute section of this document.

Justification for addition as a new Product Feature

The `__attribute__((target("branch-protection=<protection>")))` function attribute is required when using the `__attribute__((target("harden-pac-ret=<value>")))` function attribute.

Arm has completed validation testing for the combination of `__attribute__((target("harden-pac-ret=load-return-address")))` and `__attribute__((target("branch-protection=<protection>")))`, where `<protection>` includes `pac-ret`.

Impact on a project migrated from Arm Compiler for Embedded FuSa 6.16.2 to Arm Compiler for Embedded FuSa 6.16.3

None.

Compiling with Arm Compiler for Embedded FuSa 6.16.3 does not implicitly add the combination of the `__attribute__(target("harden-pac-ret=load-return-address")))` and `__attribute__((target("branch-protection=<protection>")))` function attributes to source code.

Related information

--library_security=protection
-mbranch-protection
Pointer authentication in AArch64 state

3.3 -isystem <directory> command-line option for the compiler and integrated assembler, armclang

The `-isystem <directory>` command-line option allows you to specify an additional directory as a system header directory. This means that headers from the `<directory>` included with `-isystem <directory>` are treated as system headers by the compiler.

The option must be used when building a project using Arm Compiler for Embedded FuSa 6.16.3 with both the Arm Certified C Library and the Arm Certified C++ Library as per the following user requirements:

- FCL_UR_ENV_3 from the *Safety Manual* for versions 6.6.B and 6.16.A of the *Arm Certified C Library*.
- FCPL_UR_ENV_9 from the *Safety Manual* for versions 6.16LTS Revision 0 and 6.16LTS Revision 1 of the *Arm Certified C++ Library*.

Syntax

```
-isystem <directory>
```

Parameters

`<directory>`

The directory that you want to add as an additional system header directory.

Restrictions

This option is only considered a Product Feature when it is used to add a directory containing the following:

- The Arm Certified C Library header files.
- The header files in the `include/libcxx` directory within the installation directory of Arm Compiler for Embedded FuSa 6.16.3.

Use of this command-line option is subject to the User Requirements AC_UR_CL_19 described in Changes to the User Requirements section of the Arm Compiler for Embedded FuSa 6.16LTS Qualification Kit Safety Manual version 6.16.2.

Operation

Using this option causes the compiler to search in the specified `<directory>` to resolve all `#include` directives.

For example, when compiling with `-isystem /path/to/certified/c/library/headers`, the compiler searches `/path/to/certified/c/library/headers` to find the header file `stdio.h` for all of the following types of `#include` directives:

- A `#include` directive which uses system paths, such as `#include <stdio.h>`
- A `#include` directive which uses regular paths, such as `#include "stdio.h"`.

`-isystem <directory>` is typically used in conjunction with the `-nobuiltininc` command-line option, which prevents the compiler from searching the default system headers directory.

For more information about using `-isystem <directory>` when building a project with both the Arm Certified C Library and the Arm Certified C++ Library, see the *Safety Manual* documents for both the Arm Certified C Library and the Arm Certified C++ Library you are using.

Justification for addition as a new Product Feature

The `-isystem <directory>` command-line option is required when using both the Arm Certified C Library and the Arm Certified C++ Library.

Arm has completed validation testing of both the Arm Certified C Library and the Arm Certified C++ Library using the `-isystem <directory>` command-line option.

Impact on a project migrated from Arm Compiler for Embedded FuSa 6.16.2 to Arm Compiler for Embedded FuSa 6.16.3

None.

The behavior of the `-isystem <directory>` command-line option is identical between Arm Compiler for Embedded FuSa 6.16.2 and Arm Compiler for Embedded FuSa 6.16.3.

Compiling with Arm Compiler for Embedded FuSa 6.16.3 does not add the `-isystem <directory>` command-line option by default.

Related information

-nobuiltininc on page 29

3.4 -mharden-pac-ret command-line option for the compiler and integrated assembler, armclang

The `-mharden-pac-ret=<option>` command-line option can be used to harden return address signing to protect against a PACMAN brute-force attack that tries to determine valid Pointer Authentication Codes (PAC).

Syntax

```
-mharden-pac-ret=<option>
```

Parameters

`<option>`

One of:

`none`

Disables return address signing hardening.

`load-return-address`

Enables return address signing hardening.

Must be used with a `__attribute((target("branch-protection=<protection>")))` function attribute or `-mbranch-protection=<protection>` command-line option where `<protection>` specifies a branch protection mode that includes `pac-ret` branch protection. For example, the `-mbranch-protection=standard` command-line option.


Note

This command-line option applies to all functions. However, you can override it for specific functions with the `__attribute__((target("harden-pac-ret=<value>")))` function attribute.

Restrictions

This option is only a Product Feature when compiling for AArch64 state.

Use of this command-line option is subject to the User Requirements AC_UR_CL_14 described in Changes to the User Requirements section of the Arm Compiler for Embedded FuSa 6.16LTS Qualification Kit Safety Manual version 6.16.2.

Operation

Use `-mharden-pac-ret` with a `-mbranch-protection=<protection>` option which includes `pac-ret` branch protection to protect against a PACMAN attack. PACMAN uses speculation to determine PAC, so reduces the protection that pointer authentication provides against Return Oriented Programming (ROP) and Jump Oriented Programming (JOP) attacks.

The `-mharden-pac-ret` command-line option enables code generation for return address signing hardening. The compiler uses one of the following instructions to implement return address signing hardening:

- The `XPACI` instruction when compiling for a target with the Pointer authentication feature (FEAT_PAuth). FEAT_PAuth is mandatory for Armv8.3-A and later targets.
- The `XPACLRI` instruction for compiling for a target without the Pointer authentication feature. The `XPACLRI` instruction is a hint-space instruction which is treated as a **NOP** by a target without FEAT_PAuth.

Example: -mharden-pac-ret without the use of attributes

Create the file `address.c` containing the following C code:

```
extern void func2();

int func1(int cond)
{
    func2();
    if (cond)
    {
        return 0;
    }
    return 1;
}
```

Compile `address.c` with the options `-mharden-pac-ret=load-return-address` and `-mbranch-protection=pac-ret+leaf`:

```
armclang --target=aarch64-arm-none-eabi -march=armv8.3-a -O2 -mharden-pac-ret=load-
return-address -mbranch-protection=pac-ret+leaf -S -o address.s address.c
```

The compiler generates code with return address signing hardening enabled, and generates an instruction sequence containing the XPACI instruction:

```
    ...
func1:
    paciasp
    stp x30, x19, [sp, #-16]!           // 16-byte Folded Spill
    mov w19, w0
    bl  func2
    cmp w19, #0
    cset    w0, eq
    ldp x30, x19, [sp], #16             // 16-byte Folded Reload
    autiasp
    mov x16, x30
    xpaci   x16
    ldr w16, [x16]
    ret
    ...
```

To compare against code generated without return address signing hardening enabled, compile `address.c` again without the option `-mharden-pac-ret=load-return-address`:

```
armclang --target=aarch64-arm-none-eabi -march=armv8.3-a -O2 -mbranch-
protection=pac-ret+leaf -S -o address.s address.c
```

The generated assembly now contains:

```
    ...
func1:
    paciasp
    stp x30, x19, [sp, #-16]!           // 16-byte Folded Spill
    mov w19, w0
    bl  func2
    cmp w19, #0
    cset    w0, eq
    ldp x30, x19, [sp], #16             // 16-byte Folded Reload
    retaa
    ...
```

The presence of the RETAA instruction makes this instruction sequence vulnerable to the PACMAN security vulnerability.

RETAA is a contraction of the AUTIASP and RET code sequence. Therefore, this code sequence and RETAA are indications that return address signing hardening is required. That is, hardening is required if a return address is authenticated, then returned to, with no mitigation code sequence in between.

Example: Return address signing hardening in Armv8.3-A

Create the file `address1.c` containing the following C code:

```
extern void func2();

#ifdef ADD_ATTRIBUTE
__attribute__((target("harden-pac-ret=load-return-address,branch-protection=pac-
ret")))
#endif
int func1(int cond)
{
    func2();
    if (cond)
    {
        return 0;
    }
    return 1;
}
```

Compile `address1.c` with the return address signing option `-mbranch-protection=pac-ret`:

```
armclang --target=aarch64-arm-none-eabi -march=armv8.3-a -O2 -mbranch-
protection=pac-ret -S -o address1.s address1.c
```

The generated file `address1.s` contains:

```
    ...
func1:
    paciasp
    stp x30, x19, [sp, #-16]!           // 16-byte Folded Spill
    mov w19, w0
    bl  func2
    cmp w19, #0
    cset    w0, eq
    ldp x30, x19, [sp], #16             // 16-byte Folded Reload
    retaa
    ...
```

The function `func1()` might constitute a PACMAN gadget. An attacker might leverage the speculative execution of the code leading up to the `RETAA` instruction. For more information, see the PACMAN: Attacking ARM Pointer Authentication with Speculative Execution paper.

Compile again without the `-mbranch-protection=pac-ret` option but with the `__attribute__((target("harden-pac-ret=load-return-address,branch-protection=pac-ret")))` function attribute as follows:

```
armclang --target=aarch64-arm-none-eabi -march=armv8.3-a -O2 -DADD_ATTRIBUTE -S -o
 address_attr.s address1.c
```

This command defines the preprocessor macro `ADD_ATTRIBUTE` and adds a function attribute to enable return address signing hardening for the function `func1()`.

The generated assembly now contains:

```
    ...
func1:
    paciasp
    stp x30, x19, [sp, #-16]!           // 16-byte Folded Spill
    mov w19, w0
    bl  func2
    cmp w19, #0
    cset    w0, eq
    ldp x30, x19, [sp], #16             // 16-byte Folded Reload
    autiasp
    mov x16, x30
    xpaci   x16
    ldr w16, [x16]
    ret
    ...
```

Justification for addition as a new Product Feature

The `-mharden-pac-ret=<value>` command-line option provides a mitigation for the PACMAN security vulnerability.

Arm has completed validation testing of the `-mharden-pac-ret=<value>` command-line option.

Impact on a project migrated from Arm Compiler for Embedded FuSa 6.16.2 to Arm Compiler for Embedded FuSa 6.16.3

None.

Arm Compiler for Embedded FuSa 6.16.2 does not implement the `-mharden-pac-ret=<value>` command-line option.

The default for Arm Compiler for Embedded FuSa 6.16.3 is `-mharden-pac-ret=none`, which does not enable any mitigations for the PACMAN security vulnerability.

Related information

-mbranch-protection
__attribute__((target("harden-pac-ret=<value>"))) on page 16

3.5 -mharden-sls command-line option for the compiler and integrated assembler, armclang

The `-mharden-sls` option can be used to enable mitigations for the Straight-Line Speculation (SLS) security vulnerability.

When compiling with `-mharden-sls=<opt>`, the compiler generates code that helps prevent the processor from speculating past an indirect branch which is vulnerable to SLS.

Syntax

```
-mharden-sls=<option>
```

Parameters

<option>

One of:

all

Enable all mitigations against SLS that are implemented.

blr

Enable the mitigation against SLS for `BLR` instructions.

`armclang` creates a thunk `__llvm_slsblr_thunk_x<N>` for every `X<N>` register. Each thunk is placed in a separate section named `.text.__llvm_slsblr_thunk_x<N>`.

In Arm Compiler for Embedded FuSa 6.16.3, the separate thunk code is globally visible and might be called from a location where the `SB` instruction is locally disabled. Therefore, `armclang` always uses the `DSB` and `ISB` speculation barrier instructions after the `BR` instruction.

none

Disable all mitigations against SLS.

retbr

Enable the mitigation against SLS for `RET` and `BR` instructions.

`armclang` uses the `SB` speculation barrier instruction after `RET` and `BR` instructions if that instruction is supported by the target. Otherwise, it uses the `DSB` and `ISB` instructions.

Restrictions

This command-line option is only a Product Feature when compiling for AArch64 state.

Use of this command-line option is subject to the User Requirements AC_UR_CL_21 described in Changes to the User Requirements section of the Arm Compiler for Embedded FuSa 6.16LTS Qualification Kit Safety Manual version 6.16.2.

Example: Mitigation for the BLR instruction

Create the file `function.c` containing the following C code:

```
typedef int functype(int);

int call(functype *funcptr) {
    return funcptr(42) + funcptr(24);
}
```

Compile with the command:

```
armclang --target=aarch64-arm-none-eabi -mcpu=cortex-a57 -S function.c
```

The generated file `function.s` contains:

```
...
    mov w0, #42
    blr x8
    str w0, [sp, #4]                    // 4-byte Folded Spill
    ldr x8, [sp, #8]
    mov w0, #24
    blr x8
    mov w8, w0
...
```

Compile again with the command:

```
armclang --target=aarch64-arm-none-eabi -mcpu=cortex-a57 -mharden-sls=blr -S
 function.c
```

The generated file `function.s` now contains:

```
...
    mov w0, #42
    bl  __llvm_slsblr_thunk_x1
    str w0, [sp, #4]                    // 4-byte Folded Spill
    ldr x1, [sp, #8]
    mov w0, #24
    bl  __llvm_slsblr_thunk_x1
    mov w8, w0
...
    .section
  .text.__llvm_slsblr_thunk_x1,"axG",@progbits,__llvm_slsblr_thunk_x1,comdat
    .hidden __llvm_slsblr_thunk_x1
    .weak   __llvm_slsblr_thunk_x1
    .p2align    4
    .type   __llvm_slsblr_thunk_x1,@function
__llvm_slsblr_thunk_x1:
    mov x16, x1
    br  x16
    dsb sy
    isb
...
```

Example: Mitigation for the RET instruction

Compile the C file from the previous example with the command:

```
armclang --target=aarch64-arm-none-eabi -mcpu=cortex-a57 -S function.c
```

The generated file `function.s` contains:

```
...
    mov w8, w0
    ldr w0, [sp, #4]                    // 4-byte Folded Reload
    add w0, w0, w8
    ldr x30, [sp, #16]                  // 8-byte Folded Reload
    add sp, sp, #32
    ret
...
```

Compile again with the command:

```
armclang --target=aarch64-arm-none-eabi -mcpu=cortex-a57 -mharden-sls=retbr -S
 function.c
```

The generated file `function.s` now contains:

```
...
    mov w8, w0
    ldr w0, [sp, #4]                    // 4-byte Folded Reload
    add w0, w0, w8
    ldr x30, [sp, #16]                  // 8-byte Folded Reload
    add sp, sp, #32
    ret
    dsb sy
    isb
...
```

Compile again for a target that supports the `SB` instruction:

```
armclang --target=aarch64-arm-none-eabi -march=armv8.5-a -mharden-sls=retbr -S
 function.c
```

The generated file `function.s` now has the `SB` instruction after `RET`:

```
...
    mov w8, w0
    ldr w0, [sp, #4]                    // 4-byte Folded Reload
    add w0, w0, w8
    ldr x30, [sp, #16]                  // 8-byte Folded Reload
    add sp, sp, #32
    ret
    sb
...
```

Justification for addition as a new Product Feature

The `-mharden-sls` command-line option provides a mitigation for the SLS security vulnerability.

Arm has completed validation testing of the `-mharden-sls` command-line option.

Impact on a project migrated from Arm Compiler for Embedded FuSa 6.16.2 to Arm Compiler for Embedded FuSa 6.16.3

During validation testing, Arm found that Arm Compiler for Embedded FuSa 6.16.2 is affected by the Translation fault defect with the identifier SDCOMP-63913.

SDCOMP-63913 has been fixed for Arm Compiler for Embedded FuSa 6.16.3. For a project that is affected by SDCOMP-63913 as per the *Conditions* section of SDCOMP-63913, Arm Compiler for Embedded FuSa 6.16.3 generates different code compared to Arm Compiler for Embedded FuSa 6.16.2.

The default for Arm Compiler for Embedded FuSa 6.16.3 is `-mharden-sls=none`, which does not enable any mitigations for the SLS security vulnerability.

Related information

Straight-line speculation whitepaper
Straight-Line Speculation (SLS) hardening supplement for Arm Compiler for Embedded FuSa 6.16LTS

3.6 -nobuiltininc command-line option for the compiler and integrated assembler, armclang

The `-nobuiltininc` command-line option forces the compiler to exclude its built-in header files when resolving references to a header file mentioned in a `#include` directive.

The option must be used when building a project using Arm Compiler for Embedded FuSa 6.16.3 with both the Arm Certified C Library and the Arm Certified C++ Library as per the following user requirements:

- FCL_UR_ENV_3 from the *Safety Manual* for versions 6.6.B and 6.16.A of the *Arm Certified C Library*.
- FCPL_UR_ENV_7 from the *Safety Manual* for versions 6.16LTS Revision 0 and 6.16LTS Revision 1 of the *Arm Certified C++ Library*.

Syntax

```
-nobuiltininc
```

Parameters

None.

Restrictions

This option is only a Product Feature when it is used to prevent the compiler from searching the default system headers directory for the purpose of using both the Arm Certified C Library and the Arm Certified C++ Library.

Use of this command-line option is subject to the User Requirements AC_UR_CL_20 described in Changes to the User Requirements section of the Arm Compiler for Embedded FuSa 6.16LTS Qualification Kit Safety Manual version 6.16.2.

Operation

Using `-nobuiltininc` means that you can use the `-isystem <directory>` option to include different system header files from the directory `<directory>`.

For more information about using `-nobuiltininc` with `-isystem <directory>` when building a project with both the Arm Certified C Library and the Arm Certified C++ Library, see the *Safety Manual* documents for both the Arm Certified C Library and the Arm Certified C++ Library you are using.

Justification for addition as a new Product Feature

The `-nobuiltininc` command-line option is required when using both the Arm Certified C Library and the Arm Certified C++ Library.

Arm has completed validation testing of both the Arm Certified C Library and the Arm Certified C++ Library using the `-nobuiltininc` command-line option.

Impact on a project migrated from Arm Compiler for Embedded FuSa 6.16.2 to Arm Compiler for Embedded FuSa 6.16.3

None.

The behavior of the `-nobuiltininc` command-line option is identical between Arm Compiler for Embedded FuSa 6.16.2 and Arm Compiler for Embedded FuSa 6.16.3.

Related information

-isystem <directory> on page 19
-nostdlibinc

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
202601-01	29 January 2026	Non-Confidential	Initial release

Change history

Arm does not provide a detailed list of changes between different revisions of this document. For the release history of Arm Compiler for Embedded FuSa 6.16LTS, see the *Release history* section of the *Arm Compiler for Embedded FuSa 6.16.3 Release Notes*.

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.

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 Certified C Library version 6.16.A Safety Manual	101611	Confidential
Arm Certified C Library version 6.6.B Safety Manual	101611	Confidential
Arm Certified C++ Library 6.16LTS Revision 0 Safety Manual	109566	Confidential
Arm Certified C++ Library 6.16LTS Revision 1 Safety Manual	109566	Confidential
Arm Compiler for Embedded FuSa 6.16.2 Reference Guide	102284	Non-Confidential
Arm Compiler for Embedded FuSa 6.16.2 User Guide	102283	Non-Confidential
Arm Compiler for Embedded FuSa 6.16.3 Release Notes	110644	Non-Confidential
Arm Compiler for Embedded FuSa 6.16.3 User Documentation and Qualification Kit Addendum	111217	Non-Confidential
Arm Compiler for Embedded FuSa 6.16LTS Defect Notification Report	107987	Non-Confidential
Arm Compiler for Embedded FuSa 6.16LTS Qualification Kit Safety Manual	102288	Confidential
Arm Compiler for Embedded FuSa 6.16LTS documentation index	KA005062	Non-Confidential
Does Arm document all known issues that affect each Arm Compiler release?	KA005052	Non-Confidential