



Arm Compiler for Embedded FuSa 6.22LTS

Defect Notification Report

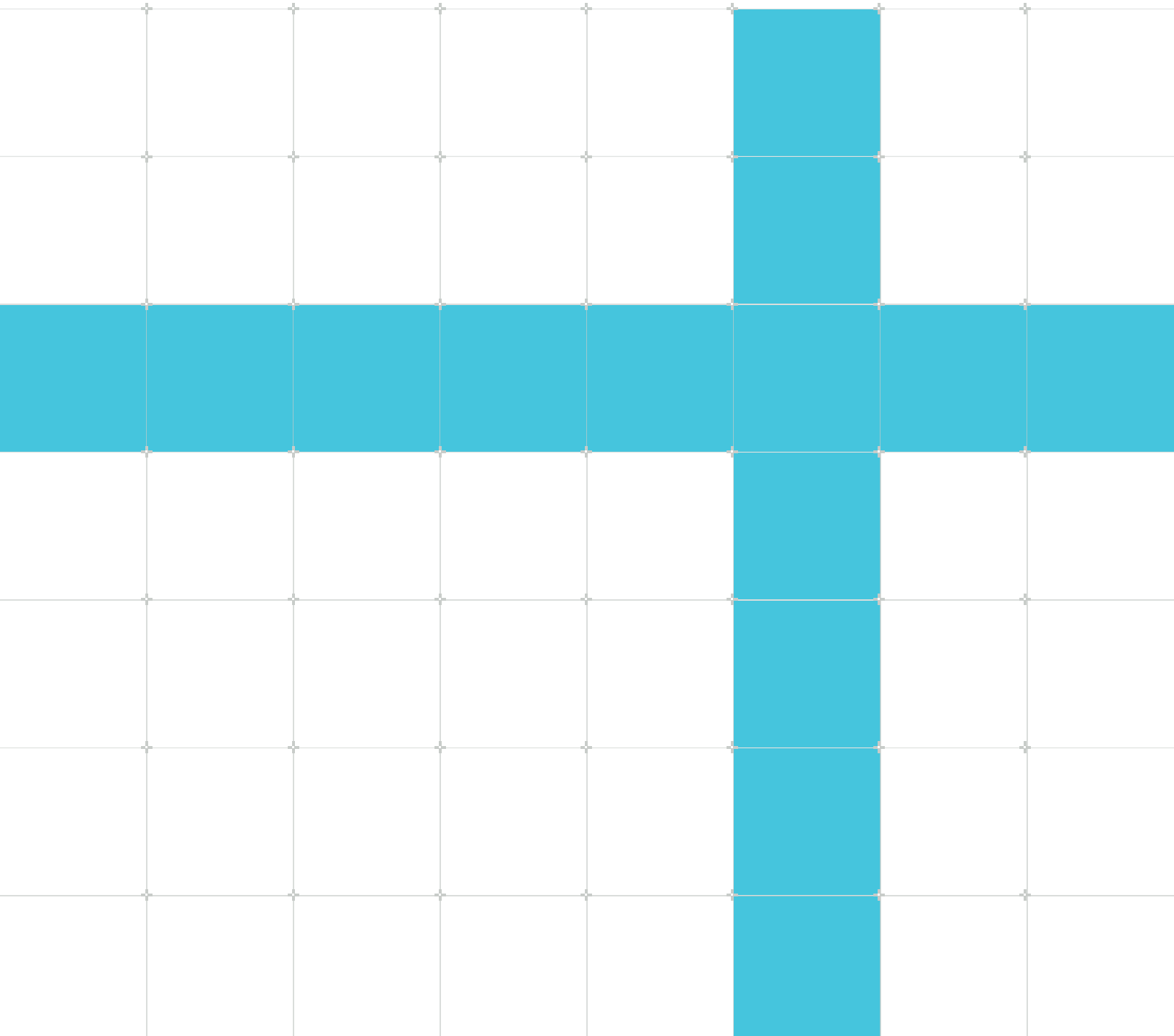
Version January 2025

Non-Confidential

Copyright © 2025 Arm Limited (or its affiliates).
All rights reserved.

Issue 00

110099_2025-01_00_en



Arm Compiler for Embedded FuSa 6.22LTS Defect Notification Report

This document is Non-Confidential.

Copyright © 2025 Arm Limited (or its affiliates). All rights reserved.

This document is protected by copyright and other intellectual property rights.

Arm only permits use of this document if you have reviewed and accepted [Arm's Proprietary Notice](#) found at the end of this document.

This document (110099_2025-01_00_en) was issued on 2025-01-24. There might be a later issue at <https://developer.arm.com/documentation/110099>

The product version is January 2025.

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 has a valid license for Arm Compiler for Embedded FuSa 6.22LTS, and is using an Arm Compiler for Embedded FuSa 6.22LTS release to build a project with functional safety or long-term maintenance requirements. The document includes descriptions of known safety-related defects that affect each release of Arm Compiler for Embedded FuSa 6.22LTS.

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.....	5
1.1 Scope of the Defect Lists.....	5
1.2 Derivation of the Defect Lists.....	5
1.3 Documentation releases for documentation synchronization faults.....	6
2. Defects.....	7
2.1 Format of a Defect Entry.....	7
2.1.1 Target environment.....	8
2.2 Machine-readable defects list.....	9
2.3 Defects affecting qualified components.....	10
2.3.1 Translation faults.....	12
2.3.2 Missing diagnostic faults.....	55
2.3.3 Determinism faults.....	77
2.3.4 Documentation synchronization faults.....	77
2.4 Defects affecting unqualified components.....	79
2.4.1 Translation faults.....	80
2.4.2 Missing diagnostic faults.....	90
2.4.3 Determinism faults.....	90
2.4.4 Documentation synchronization faults.....	91
2.5 Defects affecting both qualified and unqualified components.....	91
2.5.1 Translation faults.....	92
2.5.2 Missing diagnostic faults.....	93
2.5.3 Determinism faults.....	93
2.5.4 Documentation synchronization faults.....	93
A. Changes since the Arm Compiler for Embedded FuSa 6.22LTS Defect Notification Report for December 2024.....	94
A.1 Defects added.....	94
A.2 Defects updated.....	95
Proprietary notice.....	96
Product and document information.....	98

Product status..... 98

Revision history..... 98

Conventions..... 99

Useful resources..... 101

1. Introduction

This document is intended for functional safety managers and software developers using Arm Compiler for Embedded FuSa 6.22LTS for functional safety projects.

This document has been created based on information available to Arm as of 24 January 2025. It provides an updated list of known safety-related defects that affect a release of Arm Compiler for Embedded FuSa 6.22LTS, and has been published on a discretionary basis.

Functional safety managers can reference the known defects in Arm Compiler for Embedded FuSa to address requirement 11.4.4 in ISO 26262-8, *Planning of usage of a software tool*, and the equivalent requirement in IEC 61508-4 section 7.4.4.5.

Software developers can study the known defect list and apply appropriate safeguards and workarounds if they think they are at risk.

For information on the referenced documents, see the *Arm Compiler for Embedded FuSa 6.22LTS Qualification Kit Safety Manual*.

1.1 Scope of the Defect Lists

The defect lists within this document contain an entry for each known defect that is in a safety-related fault category and, at the time this document was generated, identified as affecting the following Arm Compiler for Embedded FuSa 6.22LTS releases: 6.22.1.

See *The role of Arm Compiler for Embedded FuSa in Safety-related Development* in the *Arm Compiler for Embedded FuSa 6.22LTS Qualification Kit Safety Manual* for an explanation of the safety-related fault categories.

See the *Arm Compiler for Embedded FuSa 6.22LTS Qualification Kit Development Process* document for an explanation of how the Arm Compiler for Embedded FuSa development process handles safety-related defects.

Defects are grouped according to whether they affect qualified or unqualified toolchain components, the fault category, and are listed in descending order of the defect identifier number.

1.2 Derivation of the Defect Lists

This section describes how the information in the defect lists within this document are derived.

The information in the defect lists in this document is derived directly from the Arm defect tracking system.

All incoming Arm Compiler for Embedded FuSa defects are assessed for their impact on functional safety. See the Arm Compiler for Embedded FuSa 6.22LTS Qualification Kit *Development Process* document for more information.

The provided information might change in future versions of this document. Such changes may include the removal of a defect from the document.

1.3 Documentation releases for documentation synchronization faults

This section explains the relationship between documentation releases and toolchain releases in the context of Documentation synchronization faults.

Documentation synchronization faults apply to specific releases of the documentation.

For each affected release specified in a documentation synchronization fault, use the *References* section of the matching *Arm Compiler for Embedded FuSa 6.22LTS Qualification Kit Safety Manual* to identify the specific release of the documentation to which the fault applies.

For example, if a documentation synchronization fault affects release 6.22.1 of the Arm Compiler for Embedded FuSa tools, see the *References* section of release 6.22.1 of the *Arm Compiler for Embedded FuSa 6.22LTS Qualification Kit Safety Manual* for the specific release of the documentation to which the fault applies.

2. Defects

This chapter contains information about all known safety-related defects that affect releases of Arm Compiler for Embedded FuSa 6.22LTS.

2.1 Format of a Defect Entry

This section describes the format of a defect entry in this document.

Each defect entry contains the following information:

Item	Description
Defect identifier	A unique identifier for the defect, of the form <code>SDCOMP-<N></code> . This identifier is used as the title of the section describing the defect. It should be used in all communication regarding the defect.
Components	<p>The Arm Compiler for Embedded FuSa components affected by the defect. The affected components might be one or more of:</p> <ul style="list-style-type: none"> Qualified toolchain components: <ul style="list-style-type: none"> Compiler and integrated assembler, <code>armclang</code> ELF processing utility, <code>fromelf</code> Librarian, <code>armar</code> Linker, <code>armlink</code> Unqualified toolchain components: <ul style="list-style-type: none"> Legacy assembler, <code>armasm</code> Libraries
Fault category	<p>Each defect in this document is listed in a section based on its safety-related fault category classification:</p> <ul style="list-style-type: none"> Translation fault Missing diagnostic fault Determinism fault Documentation synchronization fault <p>For more information about fault categories, see the <i>Arm Compiler for Embedded FuSa 6.22LTS Qualification Kit Safety Manual</i>.</p>
Target environment	Where feasible, describes the set of target Arm architectures or processor states that might be affected by the defect. The default value is "Any", which means the issue could affect any supported target Arm architecture or processor state. For more information, see the Target environment section.
Affected releases	A list of the releases in scope that the defect is observable in.
Unaffected releases	A list of the releases in scope that the defect is not observable in.
Description	A summary of the defect and its impact.
Conditions	A list of conditions that must hold to observe the defect.

The information describing the scope of a defect is included in a table in each defect entry in this document. You can use the information in this table to determine if a defect is relevant to your project without having to read the full details of the defect.

To avoid a known defect, manually inspect the source code and command-line options to ensure that at least one condition for the defect does not hold true. Arm Support might be able to help you identify other workarounds for known defects, if a generic workaround is not suitable.

2.1.1 Target environment

This section describes the purpose and meaning of the target environment associated with each defect entry included in this document.

Where feasible, the target environment is used to limit the scope of each defect.

The target environment specifies one of the following:

- One or more architectures
- One or more processor states
- A combination of architectures and processor states
- The value "Any"

It does not specify any of the following:

- Architecture revisions, such as Armv8.1-M
- Architecture extensions, such as the M-profile Vector Extension (MVE)

Instead, the conditions of a defect may include statements that further limit the scope of the defect. For example, for a defect with the target environment "Armv8-M with the Main Extension", the conditions may include the following statement to limit the scope of the defect to only targets that implement the M-profile Vector Extension (MVE):

- The program is compiled for a target with the M-profile Vector Extension (MVE).

The following target environments are included within the scope of this document:

Target environment	Description
Any	The scope defect is not limited to any specific target environments, and can affect any target Arm architecture or processor subject only to the conditions under which the defect can occur.
A32 state	An Arm architecture or processor in A32 state (formerly Arm state). Depending on the <code>-mcpu</code> or <code>-march</code> option used with the compiler, A32 state may be the default. For example, it is the default when compiling with <code>-mcpu=cortex-r52</code> . For more information, see the <code>-march</code> and <code>-mcpu</code> sections of the Arm Compiler for Embedded FuSa <i>Reference Guide</i> .
AArch32 state	An Arm architecture or processor in AArch32 state. This includes A32 state (formerly Arm state) and T32 state (formerly Thumb state).
AArch64 state	An Arm architecture or processor in AArch64 state.
Armv6-M	The Armv6-M architecture, or a processor based on the Armv6-M architecture. For example, Cortex-M0.
Armv7-A	The Armv7-A architecture, or a processor based on the Armv7-A architecture. For example, Cortex-A9.
Armv7-M	The Armv7-M architecture, or a processor based on the Armv7-M architecture. For example, Cortex-M3.
Armv7-R	The Armv7-R architecture, or a processor based on the Armv7-R architecture. For example, Cortex-R5.

Target environment	Description
Armv8-A	Any version of the Armv8-A architecture, or a processor based on any version of the Armv8-A architecture. Unless otherwise specified in the conditions of a defect, this includes both AArch64 state and AArch32 state. For example, Cortex-A53.
Armv8-M	Any version of the Armv8-M architecture, or a processor based on any version of the Armv8-M architecture. Unless otherwise specified in the conditions of a defect, this includes both Armv8-M with the Main Extension and Armv8-M without the Main Extension. For example, this includes projects that are compiled with <code>-march=armv8-m.base</code> , <code>-march=armv8.1-m.main</code> , or <code>-mcpu=cortex-m55</code> .
Armv8-M with the Main Extension	Any version of the Armv8-M architecture with the Main Extension, or a processor based on any version of the Armv8-M architecture with the Main Extension. For example, this includes projects that are compiled with <code>-march=armv8.1-m.main</code> or <code>-mcpu=cortex-m33</code> .
Armv8-M without the Main Extension	Any version of the Armv8-M architecture without the Main Extension, or a processor based on any version of the Armv8-M architecture without the Main Extension. For example, this includes projects that are compiled with <code>-march=armv8-m.base</code> or <code>-mcpu=cortex-m23</code> .
Armv8-R	The Armv8-R architecture, or a processor based on the Armv8-R architecture. This does not include the Armv8-R AArch64 architecture. For example, Cortex-R52.
Armv8-R AArch64	The Armv8-R AArch64 architecture, or a processor based on the Armv8-R AArch64 architecture. For example, Cortex-R82AE.
Armv9-A	Any version of the Armv9-A architecture, or a processor based on any version of the Armv9-A architecture. Unless otherwise specified in the conditions of a defect, this includes both AArch64 state and AArch32 state. For example, Armv9.2-A and Cortex-A710.
T32 state	An Arm architecture or processor in T32 state (formerly Thumb state). For example, this always applies when compiling for an M-profile target.

2.2 Machine-readable defects list

This section provides information about the JSON format defect lists included as an attachment with this document.

The contents of the defects lists in this document are available in a machine-readable JSON format. The file `defects_as_json.zip` attached to this document contains the following files that can be used to programmatically analyze the defects listed within this document:

defects.json

A JSON file containing a list of all defects from this document, and information about the scope of the list. The entry for each defect follows the same format as described in [Format of a Defect Entry](#). The defect description and conditions are provided as HTML markup.

schema.json

The JSON schema for the file `defects.json`. It includes descriptions of the contents of the `defects.json` file.

Arm does not provide tools to analyze the JSON format defects list.

2.3 Defects affecting qualified components

This section contains details about known safety-related defects that affect the qualified toolchain components of Arm Compiler for Embedded FuSa 6.22LTS.

The qualified toolchain components are:

- The compiler and integrated assembler, `armclang`.
- The ELF processing utility, `fromelf`.
- The librarian, `armar`.
- The linker, `armlink`.

The following defects are included in this section:

Identifier	Fault category	Affected components
SDCOMP-67799	Translation fault	armclang
SDCOMP-67678	Translation fault	armclang
SDCOMP-67666	Translation fault	armclang
SDCOMP-67662	Translation fault	armclang
SDCOMP-67650	Translation fault	armclang
SDCOMP-67544	Translation fault	armclang
SDCOMP-67448	Translation fault	armclang
SDCOMP-67446	Translation fault	armclang
SDCOMP-67194	Translation fault	armclang
SDCOMP-66895	Translation fault	armclang
SDCOMP-66787	Translation fault	armclang
SDCOMP-66692	Translation fault	fromelf
SDCOMP-66658	Translation fault	armclang
SDCOMP-66632	Translation fault	armclang
SDCOMP-66328	Translation fault	armclang
SDCOMP-66256	Translation fault	armclang
SDCOMP-65607	Translation fault	armclang
SDCOMP-65592	Translation fault	armclang
SDCOMP-65590	Translation fault	armclang
SDCOMP-65579	Translation fault	armclang
SDCOMP-65564	Translation fault	armclang
SDCOMP-65550	Translation fault	armclang
SDCOMP-65418	Translation fault	armclang
SDCOMP-64877	Translation fault	armclang
SDCOMP-64590	Translation fault	armlink
SDCOMP-64397	Translation fault	armclang
SDCOMP-64335	Translation fault	armclang
SDCOMP-63984	Translation fault	armclang

Identifier	Fault category	Affected components
SDCOMP-63912	Translation fault	armclang
SDCOMP-63911	Translation fault	armclang
SDCOMP-63205	Translation fault	armclang
SDCOMP-63114	Translation fault	armclang
SDCOMP-63088	Translation fault	armclang
SDCOMP-62378	Translation fault	armclang
SDCOMP-62176	Translation fault	armclang
SDCOMP-62133	Translation fault	armclang
SDCOMP-61486	Translation fault	armclang
SDCOMP-60117	Translation fault	armlink
SDCOMP-58780	Translation fault	armclang
SDCOMP-58354	Translation fault	armlink
SDCOMP-57725	Translation fault	armclang
SDCOMP-57255	Translation fault	armclang
SDCOMP-57229	Translation fault	armclang
SDCOMP-57213	Translation fault	armlink
SDCOMP-56435	Translation fault	armlink
SDCOMP-55460	Translation fault	armclang
SDCOMP-55200	Translation fault	armclang
SDCOMP-55184	Translation fault	fromelf
SDCOMP-55040	Translation fault	armclang
SDCOMP-50968	Translation fault	fromelf
SDCOMP-50408	Translation fault	armclang
SDCOMP-44980	Translation fault	fromelf
SDCOMP-28728	Translation fault	fromelf
SDCOMP-24899	Translation fault	fromelf
SDCOMP-11947	Translation fault	fromelf
SDCOMP-67984	Missing diagnostic fault	armclang
SDCOMP-67968	Missing diagnostic fault	armclang
SDCOMP-67424	Missing diagnostic fault	armclang
SDCOMP-67120	Missing diagnostic fault	armclang
SDCOMP-66894	Missing diagnostic fault	armclang
SDCOMP-65243	Missing diagnostic fault	armclang
SDCOMP-64683	Missing diagnostic fault	armclang
SDCOMP-64255	Missing diagnostic fault	armclang
SDCOMP-62201	Missing diagnostic fault	armclang
SDCOMP-61489	Missing diagnostic fault	fromelf
SDCOMP-61488	Missing diagnostic fault	armlink
SDCOMP-61461	Missing diagnostic fault	armclang
SDCOMP-59512	Missing diagnostic fault	armclang

Identifier	Fault category	Affected components
SDCOMP-58367	Missing diagnostic fault	armclang
SDCOMP-56812	Missing diagnostic fault	armclang
SDCOMP-56331	Missing diagnostic fault	armclang
SDCOMP-56220	Missing diagnostic fault	armclang
SDCOMP-56212	Missing diagnostic fault	armclang
SDCOMP-55983	Missing diagnostic fault	armclang
SDCOMP-53903	Missing diagnostic fault	armclang
SDCOMP-52627	Missing diagnostic fault	armclang
SDCOMP-50017	Missing diagnostic fault	armclang
SDCOMP-49961	Missing diagnostic fault	armclang
SDCOMP-49919	Missing diagnostic fault	armclang
SDCOMP-49763	Missing diagnostic fault	armclang
SDCOMP-25238	Missing diagnostic fault	armclang
SDCOMP-18689	Missing diagnostic fault	armlink
SDCOMP-17355	Missing diagnostic fault	armlink
SDCOMP-66862	Documentation synchronization fault	armclang
SDCOMP-61514	Documentation synchronization fault	armclang

2.3.1 Translation faults

This section contains details about safety-related defects that have been classified as a translation fault.

For more information about the definition of a translation fault, see the *Arm Compiler for Embedded FuSa tools and functional safety* section of the *Arm Compiler for Embedded FuSa 6.22LTS Qualification Kit Safety Manual*.

2.3.1.1 SDCOMP-67799

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-67799.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Armv8-M with the Main extension	6.22.1	-

Description

The compiler can generate incorrect code for an M-profile Vector Extension (MVE) intrinsic defined in the `<arm_mve.h>` system header.

For example, for the call to the `__arm_vsbcq_u32()` intrinsic in the following code:

```
#include <arm_mve.h>

uint32x4_t test1(uint32x4_t lhs, uint32x4_t rhs)
{
    unsigned carry = 0;
    return vsbcq_u32(lhs, rhs, &carry);
}
```

the compiler incorrectly generates a `vsbci.i32` instruction instead of a `vsbc.i32` instruction.

The `vsbci.i32` instruction assumes that the carry flag is always set. Subsequently, this can result in unexpected run-time behavior.

This defect is associated with the issue described in [SDCOMP-67678](#).

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

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled at any optimization level except `-O0`.
- The program is compiled for a target with the M-profile Vector Extension (MVE).
- The program contains a call `z` to an MVE intrinsic `i` defined in the `<arm_mve.h>` system header.
- `i` is of the form `vsbcq()`.
- The behavior of the program depends on `z`.

2.3.1.2 SDCOMP-67678

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-67678.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Armv8-M with the Main extension	6.22.1	-

Description

The compiler can generate incorrect code for a function that contains a call to an M-profile Vector Extension (MVE) intrinsic defined in the `<arm_mve.h>` system header.

This defect is associated with the issue described in [SDCOMP-67799](#).

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

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled for a target with the M-profile Vector Extension (MVE).
- The program is compiled at any optimization level except -o0.
- The program contains a function \mathbb{F} .
- \mathbb{F} contains a call to an MVE intrinsic \mathbb{I} defined in the `<arm_mve.h>` system header.
- \mathbb{I} has one of the following forms:
 - `vadcq_()`
 - `vsrcq_()`
- The behavior of the program depends on \mathbb{F} .

2.3.1.3 SDCOMP-67666

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-67666.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.22.1	-

Description

The compiler can generate incorrect code for Scalable Vector Extension version 2 (SVE2) intrinsics defined in the `<arm_sve.h>` system header.

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

Conditions

The safety-related system is at risk when all the following are true:

- The program is built with target options that enable the Scalable Vector Extension version 2 feature (FEAT_SVE2).
- The program is compiled at any optimization level except -o0.
- The program contains a call to an intrinsic \mathbb{I} that is defined in the `<arm_sve.h>` system header.
- \mathbb{I} is of the form `svwhilele_*()`.
- The behavior of the program depends on the vector returned by \mathbb{I} .

2.3.1.4 SDCOMP-67662

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-67662.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Armv8-M with the Main Extension	6.22.1	-

Description

The compiler can generate incorrect code for an M-profile Vector Extension (MVE) intrinsic defined in the <arm_mve.h> system header.

For example, for the call to the `__arm_vcmlaq_rot90_f32()` intrinsic in the following:

```
#include <arm_mve.h>

float32x4_t func(float32x4_t v)
{
    return __arm_vcmlaq_rot90_f32(v, v, v);
}
```

the compiler incorrectly generates a `vcmla.f32` instruction that uses the register `q0` for all operands:

```
vcmla.f32    q0, q0, q0, #90
bx          lr
```

A `vcmla.f32` instruction which specifies the same register as both the destination and a source operand is architecturally **CONSTRAINED UNPREDICTABLE**. Subsequently, this can result in unexpected run-time behavior.

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

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled for a target with the M-profile Vector Extension (MVE).
- The program contains a call to an MVE intrinsic `i` defined in the <arm_mve.h> system header.
- `i` is of the form `vcmlaq__f32()`.
- The behavior of the program depends on `i`.

2.3.1.5 SDCOMP-67650

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-67650.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Armv8-M with the Main Extension	6.22.1	-

Description

The compiler can generate incorrect code for an M-profile Vector Extension (MVE) intrinsic defined in the `<arm_mve.h>` system header.

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

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled at any optimization level except `-O0`.
- The program is compiled for a target with the M-profile Vector Extension (MVE).
- The program contains a call to an MVE intrinsic `ι` defined in the `<arm_mve.h>` system header.
- `ι` has one of the following forms:
 - `vfmag_m()`
 - `vfmagq_m()`
 - `vfmagq_m()`
- The behavior of the program depends on `ι`.

2.3.1.6 SDCOMP-67544

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-67544.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch32 state	6.22.1	-

Description

The compiler can generate incorrect code for a **volatile** variable of a single-precision floating-point type.

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled at any optimization level except `-O0`.
- The program contains a **volatile** variable `v` of type `T`.
- `T` is a single-precision floating-point type.
- The program contains an operation `A` that accesses `v`.
- The program contains an operation `B` that accesses `v`.
- The behavior of the program depends on the relative order of `A` and `B` being preserved.

2.3.1.7 SDCOMP-67448

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-67448.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch32 state	6.22.1	-

Description

The integrated assembler incorrectly fails to set the minimum alignment requirement for a user-defined executable section based on the default target instruction set. Instead, it incorrectly always sets the minimum alignment requirement to 1 byte.

For example, the integrated assembler incorrectly sets the alignment requirement to 1 byte for `.text.func` when assembling the following for an Armv8-A target:

```
.section .text.func, "ax"
nop
```

The default target instruction set for Armv8-A targets is A32. Therefore, `.text.func` must have a minimum alignment requirement of 4 bytes.

To avoid this issue, explicitly specify an alignment requirement of 4 bytes for each user-defined executable section as follows:

Default target instruction set	Alignment directive
A32	<code>.p2align 2</code>
T32	<code>.p2align 1</code>

This defect is associated with the issues described in [SDCOMP-67446](#) and [SDCOMP-66632](#).

Conditions

The safety-related system is at risk when all the following are true:

- The program contains a user-defined executable section `s`.
- `s` does not contain any of the following directives:
 - `.align`
 - `.arm`
 - `.balign`
 - `.code 16`
 - `.code 32`
 - `.p2align`
 - `.thumb`

2.3.1.8 SDCOMP-67446

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-67446.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch32 state	6.22.1	-

Description

The integrated assembler can incorrectly change the minimum alignment requirement of a user-defined executable section for a source file that contains an instruction set directive.

For example, the integrated assembler incorrectly sets the alignment requirement to 2 bytes for `.text.func` when assembling the following for an Armv8-A target:

```
.section .text.func, "ax"
nop
.thumb
```

The default target instruction set for Armv8-A targets is A32. Therefore, `.text.func` must have a minimum alignment requirement of 4 bytes.

To avoid this issue, explicitly specify an alignment requirement of 4 bytes for each user-defined executable section using an alignment directive. For example, using `.p2align 2`:

```
.section .text.func, "ax"
.p2align 2
nop
.thumb
```

This defect is associated with the issues described in [SDCOMP-67448](#) and [SDCOMP-66632](#).

Conditions

The safety-related system is at risk when all the following are true:

- The program contains a user-defined executable section `s`.
- `s` contains one of the following directives:
 - `.arm`
 - `.code 16`
 - `.code 32`
 - `.thumb`
- `s` does not contain any of the following directives:
 - `.align`
 - `.balign`
 - `.p2align`

2.3.1.9 SDCOMP-67194

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-67194.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.22.1	-

Description

The compiler can generate incorrect code for an outlined function.

Conditions

The safety-related system is at risk when one of the following is true:

- The program is compiled at `-oz` and without `-mno-outline`.
- The program is compiled with `-moutline`.

2.3.1.10 SDCOMP-66895

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-66895.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Armv8-M	6.22.1	-

Description

The compiler can generate incorrect debug information for a call from a secure function to a non-secure function.

Conditions

This defect can occur when all the following are true:

- The program is compiled for a target with the Security Extension.
- The program is compiled with `-mcmse`.
- The program is compiled with `-g` or `-gdwarf<version>`.
- The program contains a secure function `F`.
- `F` is annotated with `attribute((cmse_nonsecure_entry))`.
- `F` calls a non-secure function via a function pointer `P`.
- `P` is annotated with `attribute((cmse_nonsecure_call))`.

The safety-related system is only at risk when the incorrect debug information causes you to manually make an incorrect change to the safety-related system.

2.3.1.11 SDCOMP-66787

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-66787.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Armv8-M with the Main Extension	6.22.1	-

Description

The compiler can generate code which incorrectly fails to indicate that the floating-point unit may be used by a Non-secure function `N` that is called from a Secure function `S`. Subsequently, this can result in the floating-point registers being corrupted after returning from `N` to `S`.

For example, the floating-point registers may be corrupted after the return from `N()` to `S()` in the following:

```
typedef float __attribute__((cmse_nonsecure_call)) nsfunc(void);

float N(void) /* Non-secure function that returns a value of float type */
{
    return 1.0f;
}

float S(void) /* Secure function that calls a Non-secure function */
{
    nsfunc *P = (nsfunc *)N; /* Non-secure function pointer P for N() */
    if (cmse_is_nsfptr(P))
    {
```

```

float value = P(); /* Non-secure function call to N() via P */
/* Floating-point registers may be corrupted here */
return value;
}
else
{
    return 0.0f;
}
}

```

Conditions

The safety-related system is at risk when all the following are true:

- The program is not compiled for an Armv8.1-M target with the Main Extension.
- The program is compiled with `-mfloat-abi=hard`.
- The program contains a Secure function `s` in a source file `A`.
- `A` is compiled with `-mcmse`.
- The program contains a Non-secure function `N` in a source file `B`.
- `B` is not compiled with `-mcmse`.
- `N` returns a value of type `T`.
- `N` does not have any parameters of type `T`.
- `T` is one of the following:
 - A floating-point type.
 - A vector type.
- `s` calls `N` via a function pointer `P`.
- `P` is annotated with `attribute((cmse_nonsecure_call))`.
- The Secure Floating-point Context is not active within `s` before the call to `N` via `P`. To determine if the Secure Floating-point Context is active, check the value of the `FPCA` field of the `CONTROL_S` register.

2.3.1.12 SDCOMP-66692

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-66692.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
fromelf	Armv8-M with the Main Extension	6.22.1	-

Description

The fromelf utility disassembles the `VSCCLRM` instruction incorrectly.

For example, the `fromelf` utility incorrectly disassembles the following valid `vscclrm` instruction as an invalid `vldm` instruction:

```
vscclrm {d8, d9, d10, d11, d12, d13, d14, d15, vpr}
```

Conditions

This defect occurs when all the following are true:

- The `fromelf` utility is used to disassemble an ELF format input file `F`.
- `F` is disassembled for an Armv8.1-M target with the Main Extension. For example, `F` is disassembled with `--cpu=8.1-M.Main`.
- `F` contains a `vscclrm` instruction.

The safety-related system is only at risk when the incorrect disassembly output causes you to manually make an incorrect change to the safety-related system.

2.3.1.13 SDCOMP-66658

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-66658.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.22.1	-

Description

The compiler incorrectly always applies the rules for type auto-deduction for direct-list-initialization from C++17, regardless of which C++ source language mode a program is compiled for.

For example, the compiler incorrectly always deduces the type of `var` as `int` instead of `std::initializer_list<int>` in the following:

```
auto var{ 1 };
```

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled in a C++11 or C++14 source language mode.
- The program contains a variable `v`.
- The type of `v` is determined using type auto-deduction.
- `v` is initialized with direct-list-initialization using an initializer list `L`.
- `L` contains only one element `E`.
- `E` is of type `T`.

- The behavior of the program depends on `v` being of type `std::initializer_list<T>` instead of `T`.

2.3.1.14 SDCOMP-66632

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-66632.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.22.1	-

Description

The integrated assembler incorrectly fails to automatically set the minimum alignment requirement for a user-defined executable section to 4 bytes. Instead, it incorrectly always sets the minimum alignment requirement to 1 byte.

For example, the integrated assembler incorrectly sets the alignment to 1 byte `.text.func` in the following:

```
.section .text.func, "ax"
nop
```

To avoid this issue, explicitly specify an alignment for each user-defined executable section using the following directive:

```
.p2align 2
```

This defect is associated with the issues described in [SDCOMP-67448](#) and [SDCOMP-67446](#).

Conditions

The safety-related system is at risk when all the following are true:

- The program contains a user-defined executable section `s`.
- `s` does not contain any of the following directives:
 - `.align`
 - `.balign`
 - `.p2align`

2.3.1.15 SDCOMP-66328

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-66328.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.22.1	-

Description

The compiler can generate incorrect C++ exception unwinding information and incorrect debug information for a function.

Conditions

This defect can occur when all the following are true:

- The program is compiled in a C++ source language mode.
- One of the following is true:
 - The program is compiled with C++ exceptions enabled.
 - The program is compiled with `-g` or `-gdwarf-<version>`.
- The target options used to build the program enable the pointer authentication instructions that allow signing of LR using SP and PC as diversifiers feature (FEAT_PAuth_LR).
- The target options used to build the program disable the pointer authentication feature (FEAT_PAuth).
- The program is compiled with a `-mbranch-protection=<protection>` option that enables pointer authentication branch protection using the Program Counter as a second diversifier for return address signing. For example, `-mbranch-protection=pac-ret+pc`.

The safety-related system is at risk when one of the following is true:

- A C++ exception is thrown by, or propagated through, an affected function at run-time.
- The incorrect debug information causes you to manually make an incorrect change to the safety-related system.

2.3.1.16 SDCOMP-66256

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-66256.



Arm has not yet fully investigated this defect. Therefore, the information in this section is likely to change in future versions of this document.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.22.1	-

Description

The compiler can generate code that incorrectly does not conform to the *Procedure Call Standard for the Arm 64-bit Architecture*.

For example, when compiling the following code with `-march=armv8.2-a+bf16` and at `-O1`, the compiler can generate code that incorrectly splits the Homogenous Floating-point Aggregate (HFA) parameter `src` between the register `h7` and the stack:

```
#include <arm_neon.h>

volatile __bf16 dst;

typedef struct
{
    __bf16 x, y;
} hfa_t;

void func(double a, double b, double c, double d,
          double e, double f, double g, hfa_t src)
{
    dst = src.x;
    dst = src.y;
}

ldr    h0, [sp]
adrp   x8, dst
str     h7, [x8, :lo12:dst]
str     h0, [x8, :lo12:dst]
ret
```

The *Procedure Call Standard for the Arm 64-bit Architecture* does not permit a HFA parameter to be split between registers and the stack.

Conditions

The safety-related system is at risk when all the following are true:

- The program contains a function `F`.
- `F` has `M` consecutive floating-point parameters, followed by a parameter of `T` type.
- `T` is a Homogenous Floating-point Aggregate type, which consists of `N` members of `__bf16` type.
- `1 < N <= 4`.
- `M < 8`.
- `M + N > 8`.

2.3.1.17 SDCOMP-65607

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-65607.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.22.1	-

Description

The compiler can generate incorrect code for Scalable Vector Extension version 2 (SVE2) intrinsics defined in the `<arm_sve.h>` system header.

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

Conditions

The safety-related system is at risk when all the following are true:

- The program is built with target options that enable the Scalable Vector Extension version 2 feature (FEAT_SVE2).
- The program is compiled at any optimization level except `-O0`.
- The program contains a call to an intrinsic `⊔` that is defined in the `<arm_sve.h>` system header.
- `⊔` has one of the following forms:
 - `svwhilegt_*`
 - `svwhilege_*`
- The behavior of the program depends on the vector returned by `⊔`.

2.3.1.18 SDCOMP-65592

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-65592.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.22.1	-

Description

The compiler can incorrectly generate a Scalable Vector Extension (SVE) instruction for a function that is not annotated as being executed in streaming mode. Subsequently, this can result in unexpected run-time behavior.

For more information about streaming mode, see the [Controlling the use of streaming mode section of the Arm C Language Extensions \(ACLE\)](#).

Conditions

The safety-related system is at risk when all the following are true:

- The program is built with target options that enable the Scalable Matrix Extension feature (FEAT_SME).
- The program is built with target options that do not enable the Scalable Vector Extension feature (FEAT_SVE).
- The program contains a function F .
- F is not annotated as being executed in streaming mode.
- The behavior of the program depends on F being executed in non-streaming mode at run-time.

2.3.1.19 SDCOMP-65590

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-65590.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Armv8-M with the Main Extension	6.22.1	-

Description

The compiler can 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.

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

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled at any optimization level except `-O0`.
- The program is compiled for a target with the M-profile Vector Extension (MVE).
- The program contains a function F .
- F contains a call to an MVE intrinsic I defined in the `<arm_mve.h>` system header.
- I is of the form `vsetq_lane_32()`.
- The behavior of the program depends on F .

2.3.1.20 SDCOMP-65579

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-65579.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.22.1	-

Description

The compiler can generate incorrect code for a Scalable Vector Extension (SVE) intrinsic defined in the `<arm_sve.h>` system header.

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

Conditions

The safety-related system is at risk when all the following are true:

- The program is built with target options that enable one of the following features:
 - Scalable Matrix Extension (FEAT_SME)
 - Scalable Vector Extension (FEAT_SVE)
- The program is compiled at any optimization level except `-O0`.
- The program contains a call to an SVE intrinsic `⌈` defined in the `<arm_sve.h>` system header.
- `⌈` is of the form `svuzp*()`.
- The behavior of the program depends on `⌈`.

2.3.1.21 SDCOMP-65564

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-65564.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.22.1	-

Description

The compiler can generate incorrect code for Scalable Vector Extension (SVE) intrinsics defined in the `<arm_sve.h>` system header.

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

Conditions

The safety-related system is at risk when all the following are true:

- The program is built with target options that enable the Scalable Vector Extension feature (FEAT_SVE).
- The program contains a call to an intrinsic `__` that is defined in the `<arm_sve.h>` system header.
- One of the following is true:
 - `__` is `svqadd[_n_s8](<op1>, <op2>)`, and `<op2>` is a negative scalar value.
 - `__` is `svqadd[_s8](<op1>, <op2>)`, and either operand is a vector in which all elements are the same negative value.
 - `__` is `svqsub[_n_s8](<op1>, <op2>)`, and `<op2>` is a negative scalar value.
 - `__` is `svqsub[_s8](<op1>, <op2>)`, and `<op2>` is a vector in which all elements are the same negative value.
- The behavior of the program depends on the vector returned by `__`.

2.3.1.22 SDCOMP-65550

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-65550.



Arm has not yet fully investigated this defect. Therefore, the information in this section is likely to change in future versions of this document.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.22.1	-

Description

The compiler can generate incorrect code.

Conditions

The safety-related system is at risk when the program is built with target options that enable the use of Neon instructions. For example, `-march=armv8-a`.

This defect can occur when certain conditions hold for LLVM Machine Intermediate Representation (MIR).

Note: Arm is not aware of any conditions that must hold for C, C++, or assembly language source code to observe this defect.

2.3.1.23 SDCOMP-65418

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-65418.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch32 state	6.22.1	-

Description

The compiler can generate code that incorrectly corrupts the frame pointer register R11 . The frame pointer register must remain reserved throughout the execution of a program. Subsequently, this can result in a debugger displaying incorrect frame chain information.

This defect is associated with the issues described in [SDCOMP-67120](#) and [SDCOMP-64397](#).

Conditions

This defect occurs when all the following are true:

- The program is compiled with one of the following:
 - `-mframe-chain=aapcs`
 - `-mframe-chain=aapcs+leaf`
- The program is compiled without `-fno-omit-frame-pointer`.

The safety-related system is only at risk when the incorrect frame chain information causes you to manually make an incorrect change to the safety-related system.

2.3.1.24 SDCOMP-64877

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-64877.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state, Armv8-M with the Main Extension	6.22.1	-

Description

The compiler incorrectly fails to enable branch protection using pointer authentication.

Conditions

The safety-related system is at risk when all the following are true:

- The program is built Link-Time Optimization (LTO) enabled.
- The program contains two compilation units A and B .

- Both `A` and `B` are compiled with LTO enabled.
- `A` is compiled with a `-mbranch-protection=<protection>` option that enables branch protection using pointer authentication. For example, `-mbranch-protection=standard`.
- `A` contains a function `F`.
- `F` is not annotated with a `attribute((target("branch-protection=<protection>")))` attribute that enables branch protection using pointer authentication.
- One of the following is true:
 - `B` is compiled without a `-mbranch-protection=<protection>` option.
 - `B` is compiled with a `-mbranch-protection=<protection>` option that does not enable branch protection using pointer authentication. For example, `-mbranch-protection=bti`.
- The behavior of the program depends on branch protection using pointer authentication to be enabled for `F`.

2.3.1.25 SDCOMP-64590

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-64590.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armlink	Any	6.22.1	-

Description

The linker can generate incorrect stack usage information.

Irrespective of this defect, you must treat any maximum stack size usage reported by the linker as a lower limit. For more information, see the *Linker maximum stack size calculation* section of the *Safety Manual*.

Conditions

This defect can occur when all the following are true:

- The program is compiled with `-fno-omit-frame-pointer`.
- The program is linked using one of the following options:
 - `--callgraph`
 - `--info=stack`
 - `--info=summarystack`

The safety-related system is only at risk when the incorrect stack usage information causes you to manually make an incorrect change to the safety-related system.

2.3.1.26 SDCOMP-64397

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-64397.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Armv8-M with the Main Extension	6.22.1	-

Description

The compiler can generate code that incorrectly fails to preserve the frame pointer register `R11`. Subsequently, this can result in a debugger displaying incorrect frame chain information.

This defect is related with the issues described in [SDCOMP-67120](#) and [SDCOMP-65418](#).

Conditions

The defect can occur when all the following are true:

- The program is compiled with an `-mbranch-protection=<protection>` option that enables branch protection using pointer authentication. For example, `-mbranch-protection=standard`.
- The program is compiled with one of the following:
 - `-mframe-chain=aapcs`
 - `-mframe-chain=aapcs+leaf`

The safety-related system is only at risk when the incorrect frame chain information causes you to manually make an incorrect change to the safety-related system.

2.3.1.27 SDCOMP-64335

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-64335.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.22.1	-

Description

The compiler can generate code that incorrectly fails to acquire a lock for an atomic exchange operation. Such incorrect code specifies the `wzr` or `xzr` register as the second operand for a `SWP` or `SWPL` instruction.

For example, the compiler incorrectly generates a `SWPL` instruction that specifies `WZR` as the second operand for the following:

```
#include <stdatomic.h>

void func(atomic_int *var)
{
    atomic_exchange_explicit(var, 1, memory_order_release);
}
```

Conditions


The safety-related system is at risk when all the following are true:

- The target options used to build the program enable the Large System Extensions feature (FEAT_LSE). For example, `-march=armv8.2-a` OR `-march=armv8.1-a+lse`.
- The program contains a call to one of the following:
 - An `__atomic_exchange*()` built-in.
 - An `atomic_exchange*()` function defined in the `<stdatomic.h>` Arm C library header.
 - An `exchange*()` member function of the class template `std::atomic<T>`.
 - A `std::atomic_exchange*()` function defined in the `<atomic>` Arm C++ library header.

To detect if the safety-related system is at risk, disassemble the program with `fromelf --text -c` and manually inspect the output. If the output contains a `SWP` or `SWPL` instruction that specifies `WZR` or `XZR` as the second operand, then the safety-related system is at risk.

2.3.1.28 SDCOMP-63984

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-63984.



Note

Arm has not yet fully investigated this defect. Therefore, the information in this section is likely to change in future versions of this document.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	T32 state	6.22.1	-

Description

The compiler can generate a code section that incorrectly contains a literal pool.

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled with `-mexecute-only`.
- The program contains a thread-local variable `v`.
- The behavior of the program depends on an access to `v`.

2.3.1.29 SDCOMP-63912

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-63912.



Arm has not yet fully investigated this defect. Therefore, the information in this section is likely to change in future versions of this document.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch32 state	6.22.1	-

Description

The compiler can generate incorrect code for an access to a bit-field. Such incorrect code does not conform to the *Procedure Call Standard for the Arm Architecture*.

For example, the compiler generates code that incorrectly uses the register `R1` for the parameter `b` in the following:

```
struct S
{
    int x : 64;
};

int func(int a, struct S b)
{
    return b.x;
}
```

To avoid this issue, compile with `-Werror=bitfield-width` to make the compiler report the following error for potentially affected code:

- width of bit-field '`<bit-field>`' (`<width_of_bit-field>` bits) exceeds the width of its type; value will be truncated to `<width_of_type>` bits.

Conditions

The compiler generates code that incorrectly does not conform to the *Procedure Call Standard for the Arm Architecture* when all the following are true:

- The program is compiled in a C++ source language mode.
- The program contains a **class**, **struct**, or **union** type `A`.

- \mathbf{A} has a bit-field member \mathbf{M} of size \mathbf{s} and type \mathbf{B} .
- \mathbf{s} is greater than the size of \mathbf{B} .
- The program contains a function \mathbf{F} .
- \mathbf{F} has a parameter of type \mathbf{A} .
- \mathbf{F} accesses \mathbf{M} .

The safety-related system is not at risk when \mathbf{F} is compiled using the same toolchain that is used to compile all code that calls \mathbf{F} .

2.3.1.30 SDCOMP-63911

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-63911.



Arm has not yet fully investigated this defect. Therefore, the information in this section is likely to change in future versions of this document.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch32 state	6.22.1	-

Description

The compiler generates code that incorrectly does not conform to the *Procedure Call Standard for the Arm Architecture*.

For example, the compiler generates code that incorrectly assumes that the register $\mathbf{r0}$ is used for the parameter \mathbf{b} in the following:

```
struct S
{
};

int func(struct S a, int b)
{
    return b;
}
```

Conditions

The compiler generates code that incorrectly does not conform to the *Procedure Call Standard for the Arm Architecture* when all the following are true:

- The program is compiled in a C++ source language mode.
- The program contains a **class**, **struct**, or **union** type \mathbf{T} .

- The program contains a function F with a parameter A of type T .
- T does not have any members.
- F has a second parameter B with non-empty type, which appears after A in the argument list.

The safety-related system is not at risk when F is compiled using the same toolchain that is used to compile all code that calls F .

2.3.1.31 SDCOMP-63205

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-63205.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch32 state	6.22.1	-

Description

The compiler can generate incorrect code for a loop.

To avoid this issue, compile with `-fno-vectorize`.

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled at any optimization level except `-O0`.
- The program is compiled for a big-endian target.
- The program is compiled for a target that includes the Advanced SIMD Extension.
- The program contains a loop.

2.3.1.32 SDCOMP-63114

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-63114.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.22.1	-

Description

The compiler can generate an incorrect `strg` instruction. Subsequently, this can result in unexpected run-time behavior.

Conditions

This defect can occur when all the following are true:

- The program is built with target options that enable the Memory Tagging Extension feature (FEAT_MTE).
- The program is compiled with `-fsanitize=mementag-stack`.

To detect if the safety-related system is at risk, compile with `-s` and manually inspect the output. The safety-related system is only at risk when the output contains an `stg` instruction with an immediate offset of 4096, for example:

```
stg    sp, [sp], #4096
```

The `stg` instruction supports an immediate offset that is a multiple of 16 and in the range [-4096, 4080].

2.3.1.33 SDCOMP-63088

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-63088.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.22.1	-

Description

The compiler can generate incorrect code for an access to an `_Atomic` 128-bit variable.

For example, the compiler can generate incorrect code for the following when compiling with `-mbig-endian`:

```
_Atomic __uint128_t v;
__uint128_t func(void)
{
    return v;
}
```

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled for a big-endian target.
- The program contains an `_Atomic` 128-bit variable `v`.
- The program contains an access to `v`.

2.3.1.34 SDCOMP-62378

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-62378.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch32 state	6.22.1	-

Description

The compiler generates an incorrect Neon load or store instruction for a Neon intrinsic defined in the `<arm_neon.h>` system header. The incorrect instruction has an alignment specifier. Subsequently, at run-time, this can result in a Data Abort when the address being accessed is not aligned to the alignment specified by the alignment specifier.

For example, the compiler incorrectly generates:

```
vld1.8 {d16, d17, d18, d19}, [r0:256]
```

instead of:

```
vld1.8 {d16, d17, d18, d19}, [r0]
```

for the following:

```
uint8x16x2_t vector = vld1q_u8_x2(address);
```

The alignment specifier `:256` means that the `vld1.8` instruction will result in a Data Abort if `address` is not aligned to a 256-byte boundary.

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

Conditions

The safety-related system is at risk when all the following are true:

- The program contains a call to a Neon intrinsic `I` that is defined in the `<arm_neon.h>` system header.
- `I` has one of the following forms:
 - `vld*_x2()`
 - `vld*_x3()`
 - `vld*_x4()`
 - `vst*_x2()`
 - `vst*_x3()`

- `vst*_x4()`
- `i` is used to access an address `x`.
- `x` is not aligned to a 256-byte boundary.

To avoid this issue, manually inspect the source code, and ensure each address that is accessed using an affected Neon intrinsic is aligned to a 256-byte boundary.

2.3.1.35 SDCOMP-62176

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-62176.



Arm has not yet fully investigated this defect. Therefore, the information in this section is likely to change in future versions of this document.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch32 state	6.22.1	-

Description

The compiler can generate incorrect code for a function which has a half-precision floating-point parameter that is passed using the stack.

For example, the compiler can generate incorrect code for the following:

```
__fp16 func(int a, int b, int c, int d, __fp16 e)
{
    return e + 1.0;
}
```

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled for a big-endian target.
- The program is compiled for a target with half-precision floating-point support.
- The program contains a function `F`.
- `F` has a parameter `P` that is one of the following types:
 - `_Float16`
 - `__fp16`
- `P` is passed using the stack.

- The behavior of the program depends on p .

2.3.1.36 SDCOMP-62133

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-62133.



Arm has not yet fully investigated this defect. Therefore, the information in this section is likely to change in future versions of this document.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.22.1	-

Description

The compiler can generate incorrect code for a function that contains an inline assembly statement with a $+&r$ constraint code.

For example, the compiler can generate incorrect code for the following:

```
int func(void)
{
    register int V __asm("x0") = 1;
    __asm volatile("add %w0, %w0, #1" : "+&r" (V));
    return V;
}
```

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled at $-oo$.
- The program contains a function F .
- The program contains a named register variable v that uses a 64-bit register.
- F contains an inline assembly statement s .
- s specifies an output operand κ .
- κ is associated with v .
- κ has the constraint code $+&r$.

2.3.1.37 SDCOMP-61486

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-61486.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.22.1	-

Description

The compiler generates incorrect code for the expression `noexcept (typeid(V))`.

For example, the compiler generates code that incorrectly evaluates `noexcept (typeid(obj))` to **false** in the following:

```
class C { virtual void func(void) {} };
C obj;
noexcept(typeid(obj)) ? puts("OK") : puts("Not OK");
```

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled in a C++11 or later source language mode.
- The program is not compiled with `-fno-rtti`.
- The program contains a polymorphic class `c`.
- The program contains an expression `E`.
- `E` is of the form `noexcept (typeid(V))`.
- `v` is one of the following:
 - A glvalue of type `c`.
 - A reference to `c`.
- The behavior of the program depends on the result of `E`.

2.3.1.38 SDCOMP-60117

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-60117.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armlink	Any	6.22.1	-

Description

The linker can generate incorrect stack usage information.

To avoid this issue, compile the input objects with `-gdwarf-3`.

Irrespective of this defect, you must treat any maximum stack size usage reported by the linker as a lower limit. For more information, see the *Linker maximum stack size calculation* section of the *Safety Manual*.

Conditions

This defect can occur when all the following are true:

- The input objects are compiled with `-g` or `-gdwarf-<version>`, where `<version>` is not 3.
- Stack usage information is obtained from the linker using any of the following options:
 - `--callgraph`
 - `--info=stack`

The safety-related system is only at risk when the incorrect stack usage information causes you to manually make an incorrect change to the safety-related system.

2.3.1.39 SDCOMP-58780

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-58780.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.22.1	-

Description

The compiler generates code that incorrectly fails to raise a `std::bad_array_new_length` exception for a **new** expression. Subsequently, this can result in unexpected run-time behavior.

For example, the compiler generates code that incorrectly fails to raise a `std::bad_array_new_length` exception for the following:

```
void no_init_array(int len)
{
    (void)new char[len];
}

void test(void)
{
    try
    {
        no_init_array(-1);
    }
    catch (const std::bad_array_new_length &)
    {
    }
```

```
        std::cout << "Exception caught" << std::endl;
    }
}
```

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled in a C++14 or later source language mode.
- The program is compiled with C++ exceptions enabled.
- The program contains a **new** expression E .
- E is not a constant expression.
- E specifies a negative array length.
- E allocates an array with elements of type T .
- The size of T is 1 byte.
- The behavior of the program depends on a `std::bad_array_new_length` exception being raised for E .

2.3.1.40 SDCOMP-58354

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-58354.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armlink	Any	6.22.1	-

Description

The linker incorrectly reports an ELF section that is not Zero Initialized (ZI) as ZI in the `--map` output.

For example, the linker incorrectly reports the ELF section for the execution region `EXEC` as `zero` in the `--map` output for the following:

```
LOAD 0x8000
{
    EXEC +0x0 FILL 0xFFFFFFFF 0x100 {}
}
```

Conditions

The safety-related system is at risk when all the following are true:

- The program is linked with `--map`.
- The program is linked with a scatter file that contains an execution region E .
- E has one of the following execution region attributes:

- PADVALUE
- ZEROPAD
- FILL
- The `--map` output causes you to manually make an incorrect change to the safety-related system.

2.3.1.41 SDCOMP-57725

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-57725.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.22.1	-

Description

The compiler can generate incorrect debug location information for a **static** variable.

Conditions

This defect can occur when all the following are true:

The safety-related system is only at risk when the incorrect debug information causes you to manually make an incorrect change to the safety-related system.

2.3.1.42 SDCOMP-57255

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-57255.



Arm has not yet fully investigated this defect. Therefore, the information in this section is likely to change in future versions of this document.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.22.1	-

Description

The compiler generates incorrect debug information for a C++ tuple-like binding `b`. The incorrect debug information associates an identifier `i` in `b` with the source code line on which `b` is declared instead of the source code line on which `i` is used.

For example, the compiler generates incorrect debug information that associates the identifier `a` with the line on which `a` is bound to `src` in the following:

```
std::tuple<int,short> src(x,y);

void func(void)
{
    const auto [a,b] = src;

    if (a == 0)
    {
        std::cout << "a is zero" << std::endl;
    }
}
```

Conditions

This defect occurs when all the following are true:

- The program is compiled in a C++17 source language mode.
- The program is compiled with `-g` or `-gdwarf-<version>`.
- The program contains a tuple-like binding.

The safety-related system is only at risk if the incorrect debug information causes you to manually make an incorrect change to the safety-related system.

2.3.1.43 SDCOMP-57229

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-57229.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.22.1	-

Description

The compiler incorrectly generates debug information at the function scope for a `static` variable defined in a lexical block within a function.

Subsequently, this can result in an affected variable incorrectly being displayed by a debugger as being in scope at function scope.

Conditions

This defect occurs when all the following are true:

- The program contains a function `F`.
- `F` defines a **static** variable in a lexical block within `F`.

The safety-related system is only at risk when the incorrect debug information causes you to manually make an incorrect change to the safety-related system.

2.3.1.44 SDCOMP-57213

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-57213.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armlink	Any	6.22.1	-

Description

The linker can generate an incorrect address for a local symbol that is associated with an unused merged string.

For example, the linker can generate an incorrect address for the symbol `str2` in the following:

```
.section strings, "aMS", %progbits, 1
str1:
.asciz "Hello, world!"
str2:
.asciz "Hello, world!"
```

Conditions

This defect can occur when all the following are true:

- The program is linked without `--no_merge`.
- The program contains an assembly language source file with an ELF section `s`.
- `s` has the `SHF_MERGE` and `SHF_STRING` flags set.
- `s` contains a null-terminated string `A`.
- `s` contains a null-terminated string `B` associated with the symbol `x`.
- `B` is identical to `A`, or is a suffix of `A`.
- `x` is a local symbol.
- `x` is not used.

The safety-related system is only at risk if the incorrect address for `x` causes you to manually make an incorrect change to the safety-related system.

2.3.1.45 SDCOMP-56435

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-56435.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armlink	Any	6.22.1	-

Description

The linker incorrectly includes the size of uninitialized data in the `p_memsz` field of the ELF program header for the execution region containing the uninitialized data.

Subsequently, an ELF processing tool that creates the execution view directly from the ELF program headers can zero-initialize memory that was not intended to be zero-initialized. This can result in unexpected run-time behavior.

To avoid this issue, use one of the following workarounds:

- Do not use the program headers to derive the execution view when loading the image onto the target device. Instead, use the `fromelf` utility to generate a binary file for the image, and then load that binary file.
- Use `--elf-output-format=gnu` for better compatibility with program loaders. This may require you to modify your scatter file. For more details, see the `--elf-output-format` section of the *Reference Guide*.
- Do not use the `EMPTY` or `UNINIT` execution region attributes.

Conditions

The safety-related system is at risk when all the following are true:

- An ELF processing tool is used to directly create the execution view of the program from the ELF program headers.
- The program is linked with a scatter file that contains an execution region `E`.
- One of the following is true:
 - `E` has the `EMPTY` attribute.
 - `E` has the `UNINIT` attribute and contains ZI data.
- The behavior of the program depends on the ELF processing tool not zero-initializing memory that was not intended to be zero-initialized.

2.3.1.46 SDCOMP-55460

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-55460.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.22.1	-

Description

The compiler can generate code that incorrectly fails to ignore the partial specializations of a member template when the primary member template is subsequently explicitly specialized.

For example, the compiler incorrectly uses the partial specialization of `B` for the variable `obj` in the following:

```
#include <stdio.h>

// Template class A
template<class T> struct A
{
    // Member template B
    template<class T2> struct B
    {
        void f(void)
        {
            printf("Incorrect: Default\n");
        }
    };
    // Partial specialization of B
    template<class T2> struct B<T2*>
    {
        void f(void)
        {
            printf("Incorrect: Partial specialization\n");
        }
    };
};

// Explicit specialization of A::B
template<> template<class T2> struct A<short>::B
{
    void f(void)
    {
        printf("Correct: Explicit specialization\n");
    }
};

int main(void)
{
    A<short>::B<int*> obj;

    obj.f();

    return 0;
}
```

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled in a C++ source language mode.
- The program defines a template class `A`.
- `A` contains a member template `B`.
- The program defines a partial specialization of `B`.

- `A::B` is subsequently explicitly specialized.

2.3.1.47 SDCOMP-55200

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-55200.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Armv8-M with the Main Extension	6.22.1	-

Description

The compiler incorrectly sets bit 1 of the Arm C Language Extensions (ACLE) feature macro `__ARM_FEATURE_MVE`.

For example, the compiler incorrectly sets bit 1 of `__ARM_FEATURE_MVE` when compiling with `-march=armv8.1-m.main+mve.fp -mfpv5-d16`.

Conditions

The safety-related system is at risk when all the following are true:

- The program is built with an `-march=<name>` or `-mcpu=<name>` option that enables the floating-point M-profile Vector Extension (MVE-FP). For example, `-march=armv8.1-m.main+mve.fp` or `-mcpu=cortex-m55`.
- The program is built with `-mfpv5=<name>`, where `<name>` is one of the following:
 - `fpv5-d16`
 - `fpv5-sp-d16`
- The behavior of the program depends on bit 1 of the Arm C Language Extensions (ACLE) feature macro `__ARM_FEATURE_MVE`.

2.3.1.48 SDCOMP-55184

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-55184.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
fromelf	Any	6.22.1	-

Description

When the linker removes unused sections from an image that contains debug information, the auxiliary debug information sections in the resulting image can contain unused but valid padding bytes between DWARF records. When processing such an image with `-g` or `--text -g`, the

`fromelf` utility can incorrectly fail to decode DWARF records that follow such padding bytes, and subsequently report incorrect information about the correlation between the image and the original source code.

Conditions

The `fromelf` utility can report incorrect information when all the following are true:

- An executable ELF format input file `F` contains debug information.
- The debugging information in `F` contains one the following:
 - Line table entries for a function `x` in the `.debug_line` section.
 - Variable location entries for a variable in a function `x` in the `.debug_loc` section.
- The code for `x` is not present in `F`.
- `F` is processed using the `-g` or `--text -g` options.

The output from the `-g` or `--text -g` options is not directly used by debugging tools.

The safety-related system is only at risk if incorrect information in the output from the `-g` or `--text -g` options causes you to manually make an incorrect change to the safety-related system.

2.3.1.49 SDCOMP-55040

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-55040.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.22.1	-

Description

The compiler can generate incorrect code when compiling with `-fsanitize=memtag-stack`. Subsequently, this can result in a memory tagging exception against an address in the stack.

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled at any optimization level except `-O0`.
- The program is built with target options that enable the Memory Tagging Extension feature (FEAT_MTE).
- The program is compiled with `-fsanitize=memtag-stack`.

2.3.1.50 SDCOMP-50968

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-50968.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
fromelf	Any	6.22.1	-

Description

When processing an image with `-g` or `--text -g`, the `fromelf` utility can report incorrect debug information.

Conditions

The `fromelf` utility can report incorrect debug information when all the following are true:

- An executable ELF format input file `F` contains all the following:
 - Debug information.
 - A `RELA` relocation entry.
- `F` is processed using the `-g` or `--text -g` options.

The output from the `-g` or `--text -g` options is not directly used by debugging tools.

The safety-related system is only at risk if incorrect debug information in the output from the `-g` or `--text -g` options causes you to manually make an incorrect change to the safety-related system.

2.3.1.51 SDCOMP-50408

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-50408.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch32 state	6.22.1	-

Description

The compiler can generate incorrect code for a function that has a parameter of vector type.

Such code does not conform to the *Procedure Call Standard for the Arm Architecture*.

Conditions

This defect can occur when all the following are true:

- The program is compiled for a big-endian target.

- The program is compiled for a target that includes the Advanced SIMD Extension.
- The program contains a function `F`.
- `F` has a parameter of vector type `T`.
- `T` is defined in the `<arm_neon.h>` system header.

The safety-related system is only at risk when the entire program is not built using the same toolchain.

2.3.1.52 SDCOMP-44980

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-44980.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
fromelf	Any	6.22.1	-

Description

The `fromelf` utility can report incorrect bit-field offsets when processing an ELF file that contains bit-fields with `-a` or `--text -a`.

Conditions

This defect occurs when all the following are true:

- An ELF format input file `F` contains debug information.
- `F` contains a global or `static` variable `v`.
- `v` contains a bit-field.
- The global and static data addresses in `F` are printed using any of the following options:
 - `-a`
 - `--text -a`

The safety-related system is only at risk if the incorrect bit-field offset information causes you to manually make an incorrect change to the safety-related system.

2.3.1.53 SDCOMP-28728

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-28728.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
fromelf	AArch64 state	6.22.1	-

Description

The `fromelf` utility incorrectly decodes certain **UNDEFINED** instructions as `MRS` or `MSR` instructions.

For example, the `fromelf` utility incorrectly disassembles the following **UNDEFINED** instruction as an `MRS` instruction:

```
.inst 0xd5200000
```

Conditions

This defect can occur when all the following are true:

- An ELF format input file contains an instruction with a bit pattern `p`.
- `p` is within the A64 System Instruction Class encoding space.
- `p` is architecturally **UNDEFINED**.

The safety-related system is only at risk if the incorrect disassembly output causes you to manually make an incorrect change to the safety-related system.

2.3.1.54 SDCOMP-24899

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-24899.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
fromelf	T32 state	6.22.1	-

Description

The `fromelf` utility can disassemble certain instructions incorrectly and associate symbols with an incorrect address.

For example, for the following code:

```
.section .text
.p2align 2
.type func1, %function
.type func2, %function
.global func1
.global func2
func1:
    bx lr
    .inst.n 0xffff
func2:
    bx lr
```

the `fromelf` utility incorrectly disassembles the output as:

```
func1
  0x00000000:    4770      pG      BX      lr
func2
  0x00000002:   ffff4770    ..pG    VQSHL.U32  q10,q8,#31
```

Conditions

This defect can occur when all the following are true:

- An ELF format input file contains two consecutive 16-bit opcodes `A` and `B`.
- `A` is not a valid 16-bit instruction.
- A symbol is associated with the address of `B`.

The safety-related system is only at risk when the incorrect output from the `fromelf` utility causes you to manually make an incorrect change to the safety-related system.

2.3.1.55 SDCOMP-11947

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-11947.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
fromelf	Any	6.22.1	-

Description

The `fromelf` utility incorrectly disassembles a 16-bit addend as an 8-bit addend.

Conditions

This defect can occur when all the following are true:

- An ELF format input file `F` contains a 16-bit data word `D`.
- `F` is processed using `--disassemble`.
- `D` is associated with a relocation `R` of type `T`.
- `T` is one of the following:
 - `R_AARCH64_ABS16`
 - `R_ARM_ABS16`
- `R` has an addend greater than 255.

The safety-related system is only at risk when the incorrect disassembly output causes you to manually make an incorrect change to the safety-related system.

2.3.2 Missing diagnostic faults

This section contains details about safety-related defects that have been classified as a missing diagnostic fault.

For more information about the definition of a missing diagnostic fault, see the *Arm Compiler for Embedded FuSa tools and functional safety* section of the *Arm Compiler for Embedded FuSa 6.22LTS Qualification Kit Safety Manual*.

2.3.2.1 SDCOMP-67984

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-67984.



Arm has not yet fully investigated this defect. Therefore, the information in this section is likely to change in future versions of this document.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Armv8-M	6.22.1	-

Description

The compiler can incorrectly set bit 3 of the Arm C Language Extensions (ACLE) feature macro `__ARM_FEATURE_LDREX`.

For example, the compiler incorrectly sets bit 3 of `__ARM_FEATURE_LDREX` when compiling with `-march=armv8-m.main`.

Conditions

The safety-related system is at risk when the behavior of the program depends on bit 3 of the Arm C Language Extensions (ACLE) feature macro `__ARM_FEATURE_LDREX`.

2.3.2.2 SDCOMP-67968

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-67968.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.22.1	-

Description

The inline assembler and integrated assembler incorrectly ignore a `.cpu` target selection directive that does not explicitly disable an extension `E` using `+no<extension>`. Subsequently, the inline assembler and integrated assembler incorrectly fail to report an error for an instruction that requires `E`.

For example, when assembling the following with `-mcpu=cortex-a53+ras`, the integrated assembler incorrectly ignores the `.cpu cortex-a53` target selection directive, and subsequently fails to report an error for the `esb` instruction:

```
.cpu cortex-a53
esb    // invalid without the RAS extension
```

Conditions

The safety-related system is at risk when all the following are true:

- One of the following is true:
 - The program is assembled with an `-mcpu` option `A`.
 - The program contains a `.cpu` target selection directive `A`.
- `A` enables an extension `E`.
- The program contains a subsequent `.cpu` target selection directive `B`.
- `B` does not explicitly disable `E` using `+no<extension>`.
- The program contains an instruction `I`.
- `I` follows `B`.
- `I` requires `E`.

2.3.2.3 SDCOMP-67424

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-67424.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch32 state	6.22.1	-

Description

The inline assembler and integrated assembler incorrectly fail to report an error for a `movt` or `movw` instruction with an offset that is outside the range for the instruction.

For example, the integrated assembler incorrectly fails to report an error for the following instruction:

```
movt r0, #:lower16:.. + 32768 // out of range offset
```

Conditions

The safety-related system is at risk when all the following are true:

- The program contains an instruction `ι`.
- `ι` is one of the following:
 - `MOVT`
 - `MOVW`
- `ι` specifies a source operand `s`.
- `s` has one of the following forms:
 - `#:lower16:.. + <offset>`
 - `#:lower16:<label> + <offset>`
 - `#:upper16:.. + <offset>`
 - `#:upper16:<label> + <offset>`
- `<offset>` is an immediate value.
- `<offset>` is outside the range `[-32768, 32767]`.
- The behavior of the program depends on `ι`.

2.3.2.4 SDCOMP-67120

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-67120.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch32 state	6.22.1	-

Description

The inline assembler incorrectly fails 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`

The frame pointer register must remain reserved throughout the execution of a program. Subsequently, if the inline assembly statement corrupts `R11`, this can result in a debugger displaying incorrect frame chain information.

For example, the inline assembler incorrectly fails to report the warning for the inline assembly statement in the following:

```
int func(int input)
{
    int output;

    __asm volatile(
        "add r11, %[input]\n\t"
        "mov %[output], r11\n\t"
        : [output] "=&r" (output)
        : [input] "r" (input)
        : "r11"
    );

    return output;
}
```

This defect is associated with the issues described in [SDCOMP-65418](#) and [SDCOMP-64397](#).

Conditions

This defect occurs when all the following are true:

- The program is compiled with one of the following:
 - `-mframe-chain=aapcs`
 - `-mframe-chain=aapcs+leaf`
- The program is compiled without `-fno-omit-frame-pointer`.
- The program contains an inline assembly statement `s`.
- The clobber list of `s` contains `R11`.

The safety-related system is only at risk when all the following are true:

- `s` corrupts `R11`.
- The incorrect frame chain information causes you to manually make an incorrect change to the safety-related system.

2.3.2.5 SDCOMP-66894

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-66894.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.22.1	-

Description

The compiler incorrectly fails to report an error for an `-march` or `-mcpu` option that specifies an invalid feature modifier.

For example, when compiling with `-march=armv8.4-a+no1se2`, the compiler incorrectly fails to report the following error:

- unsupported argument 'armv8.4-a+no1se2' to option '-march='

+no1se2 is an invalid feature modifier.

For more information about feature modifiers, see the following sections of the *Reference Guide*:

- `-march`
- `-mcpu`

Conditions

The safety-related system is at risk when all the following are true:

- The program is built with an `-march=<name>` Or `-mcpu=<name>` option x.
- x specifies an invalid feature modifier.

2.3.2.6 SDCOMP-65243

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-65243.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.22.1	-

Description

The inline assembler and integrated assembler incorrectly fail to report the following error for a Scalable Vector Extension (SVE) instruction that specifies an invalid predication pattern:

- invalid operand for instruction

Instead, the inline assembler and integrated assembler incorrectly generate code that does not contain the instruction. Subsequently, this can result in unexpected run-time behavior.

For example, the inline assembler and integrated assembler incorrectly fail to report an error for each of the following instructions:

```
ptrue    p0.d, #ALL
cntb     x0, #ALL, mul #1
```

Conditions

The safety-related system is at risk when all the following are true:

- The program contains an SVE instruction `⌈`.
- `⌈` has an invalid predication pattern specifier operand.

- The behavior of the system depends on `1` being executed.

2.3.2.7 SDCOMP-64683

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-64683.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Armv7-A, Armv7-M, Armv7-R, Armv8-A, Armv8-M with the Main Extension, Armv8-R, Armv9-A	6.22.1	-

Description

The inline assembler and integrated assembler incorrectly fail to report an error for a PC-relative load (literal) instruction. Subsequently, this can result in one or more of the following unexpected run-time behaviors:

- A load from an incorrect address.
- An alignment fault.

For example, the inline assembler and integrated assembler incorrectly fail to report an error for the `LDRD` instruction in the following:

```
.thumb
.section .text.func, "ax"
.balign 4
.global func
.type func, %function
func:
    ldrd r0, r1, src
    .byte 0xff
src:
    .word 0x11223344, 0x55667788
```

where the address of `src` is not aligned to a 4-byte boundary.

Conditions

The safety-related system is at risk when all the following are true:

- The program is assembled for AArch32 state.
- The program contains an instruction `1`.
- `1` is one of the following:
 - `LDRD` (literal)
 - `VLDR` (literal)
- `1` specifies a label `x` as the label of the literal data item to be loaded.

- The address of `x` is not aligned to a 4-byte boundary.
- The behavior of the program depends on `1`.

2.3.2.8 SDCOMP-64255

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-64255.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch32 state	6.22.1	-

Description

The inline assembler and integrated assembler incorrectly fail to report the following error for a `DMB`, `DSB`, or `ISB` instruction that has an invalid operand:

- expected an immediate or barrier type

Instead, the inline assembler and integrated assembler incorrectly generate code that does not contain the instruction. Subsequently, this can result in unexpected run-time behavior.

For example, the inline assembler and integrated assembler incorrectly fail to report an error for the following:

```
dmb [r0]
```

Conditions

The safety-related system is at risk when all the following are true:

- The program contains a `DMB`, `DSB`, or `ISB` instruction `1`.
- `1` has an invalid operand.
- The behavior of the system depends on `1` being executed.

2.3.2.9 SDCOMP-62201

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-62201.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Armv8-A	6.22.1	-

Description

The compiler incorrectly fails to report an error for an atomic read of a **const** 128-bit variable. Instead, the compiler can generate code that incorrectly performs a write access to the variable.

For example, when compiling with `-march=armv8-a`, the compiler incorrectly fails to report an error, and subsequently generates an `LDAXP` / `STLXP` instruction pair to access `src` for the following:

```
volatile const __int128 _Atomic src = 1;

__int128 func(void)
{
    return src + 1;
}
```

The `STLXP` instruction performs a write operation to `src`.

Conditions

- The safety-related system is at risk when all the following are true:
- The program is compiled for AArch64 state.
 - The program is compiled with target options that do not enable the Large System Extensions version 2 feature (FEAT_LSE2). For example, `-march=armv8-a`.
 - The program contains a **const** 128-bit variable `v`.
 - The program accesses `v`.
 - The behavior of the safety-related system depends on `v` not being written to.

2.3.2.10 SDCOMP-61489

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-61489.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
fromelf	Any	6.22.1	-

Description

The `fromelf` utility can incorrectly fail to report an error for an invalid combination of the `--cpu=name` and `--fpu=name` options.

Conditions

- This defect can occur when all the following are true:
- The `fromelf` utility is used to disassemble an ELF format input file `F` with the `--cpu=A` and `--fpu=B` options.
 - `A` and `B` are incompatible.

- \mathbb{F} contains an instruction \mathbb{I} .
- \mathbb{I} is not compatible with \mathbb{A} .

The safety-related system is only at risk when the output from the `fromelf` utility prevents you from detecting the presence of \mathbb{I} .

2.3.2.11 SDCOMP-61488

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-61488.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armlink	Any	6.22.1	-

Description

The linker can incorrectly fail to report an error for an invalid combination of the `--cpu=name` and `--fpu=name` options.

For example, the linker incorrectly fails to report an error when linking with `--cpu=Cortex-M3` and `--fpu=FPv5-D16`.

Conditions

The safety-related system is at risk when all the following are true:

- The program is linked with `--cpu=A` and `--fpu=B`.
- \mathbb{A} and \mathbb{B} are incompatible.

2.3.2.12 SDCOMP-61461

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-61461.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	A32 state	6.22.1	-

Description

The inline assembler and integrated assembler incorrectly fail to report an error for a conditional Advanced SIMD element or structure load/store instruction. Advanced SIMD element or structure load/store instructions must be unconditional.

For example, the inline assembler and integrated assembler incorrectly fail to report an error for the following instructions:

```
vld1eq.32 {d0}, [r0]
vst1eq.32 {d0}, [r0]
```

Conditions

The safety-related system is at risk when all the following are true:

- The program contains an instruction `ι`.
- `ι` is one of the following:
 - VLD1
 - VLD2
 - VLD3
 - VLD4
 - VST1
 - VST2
 - VST3
 - VST4
- `ι` is conditional.
- The behavior of the program depends on `ι` being executed conditionally.

2.3.2.13 SDCOMP-59512

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-59512.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.22.1	-

Description

The compiler incorrectly fails to report an error for an explicit template instantiation that is not in the same namespace as the template definition.

For example, the compiler incorrectly fails to report an error for the explicit template instantiation in the following:

```
// Template definition
template<class T>
int func(T x)
{
    return x;
}
```



```
}

namespace N
{
    // Explicit template instantiation
    template int func<int>(int);

    // An unrelated definition of a function named func()
    int func(double x)
    {
        return 2 * x;
    }
}
```

To avoid this issue, compile with `-Werror=c++11-compat`.

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled in a C++98 or C++03 source language mode.
- The program is compiled without `-Werror=c++11-compat`.
- The program contains a template `T` in a namespace `A`.
- The program contains an explicit template instantiation of `T` in a namespace `B`.
- `A` and `B` are not the same.
- The behavior of the program depends on `T` being used.

2.3.2.14 SDCOMP-58367

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-58367.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	T32 state	6.22.1	-

Description

The inline assembler and integrated assembler incorrectly fail to report an error for a T32 instruction with an invalid `.n` width specifier. Instead, the inline assembler and integrated assembler assemble the instruction as a 32-bit instruction.

For example, the integrated assembler incorrectly fails to report an error for the following:

```
adc.n r0, r1, #1
```

Conditions

The safety-related system is at risk when all the following are true:

- The program contains an assembly instruction `I` with the `.n` width specifier.

- `__i` does not have a 16-bit instruction encoding.
- The behavior of the program depends on `__i` being assembled as a 16-bit instruction.

2.3.2.15 SDCOMP-56812

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-56812.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.22.1	-

Description

The compiler incorrectly fails to report an error for an invalid `#define` or `#undef` preprocessor macro.

For example, the compiler incorrectly fails to report an error for both invalid preprocessor macros in the following:

```
#undef noreturn
#define noreturn 1
```

This defect is associated with the issue described in [SDCOMP-56212](#).

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled in a C++11 or later source language mode.
- The program is compiled with one of the following options:
 - `-pedantic`
 - `-Weverything`
 - `-Wkeyword-macro`
 - `-Wpedantic`
- One of the following is true:
 - The program contains a `#define` preprocessor macro which specifies a name `N` that is lexically identical to an attribute token.
 - The program contains an `#undef` preprocessor macro which specifies a name `N` that is lexically identical to one of the following:
 - A keyword which is not an alternative operator representation.
 - An identifier with a special meaning.
 - An attribute token.

- The behavior of the program depends on n .

2.3.2.16 SDCOMP-56331

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-56331.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.22.1	-

Description

The inline assembler and integrated assembler incorrectly fail to report an error for an instruction that specifies `w31` or `x31` as a general-purpose register operand.

For example, the integrated assembler incorrectly fails to report an error for the following:

```
mov w0, w31
```

Conditions

The safety-related system is at risk when all the following are true:

- The program contains an instruction \mathfrak{I} .
- \mathfrak{I} specifies one of the following as a general-purpose register operand \mathfrak{R} :
 - `w31`
 - `x31`
- The behavior of the program depends on \mathfrak{I} accessing \mathfrak{R} as a general-purpose register instead of as the zero register.

2.3.2.17 SDCOMP-56220

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-56220.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.22.1	-

Description

The compiler incorrectly fails to report an error for the redefinition of a variable originally declared in the controlling expression of a range-based **for** statement.

For example, the compiler incorrectly fails to report an error for the redeclaration of `var` in the following:

```
void func(void)
{
    for (int var : {1, 2, 3})
    {
        extern int var();
    }
}
```

This defect is associated with the issue described in [SDCOMP-50017](#).

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled in a C++ source language mode.
- The program contains a range-based `for` statement `s`.
- `s` has a controlling expression that defines a variable `v`.
- The outermost block of `s` contains a redeclaration of `v` as a function.
- The behavior of the program depends on `v` not being redeclared as a function.

2.3.2.18 SDCOMP-56212

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-56212.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.22.1	-

Description

The compiler incorrectly fails to report an error for an invalid `#define` or `#undef` preprocessor macro that redefines or undefines a name that is used in an Arm C++ standard library header.

For example, the compiler incorrectly fails to report an error for the invalid `#define` preprocessor macro in the following:

```
#include <iostream>
#define cout cerr

int main(void)
{
    std::cout << "Hello, world!" << std::endl;

    return 0;
}
```

This defect is associated with the issue described in [SDCOMP-56812](#).

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled in a C++ source language mode.
- The program includes an Arm C++ standard library header `h`.
- The program contains a `#define` or `#undef` preprocessor macro `m`.
- `m` specifies a name that is lexically identical to a name `n` that is used in `h`.
- The behavior of the program depends on `m` not changing `n`.

2.3.2.19 SDCOMP-55983

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-55983.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	A32 state	6.22.1	-

Description

The inline assembler and integrated assembler can incorrectly fail to report an error for a branch instruction with an offset that is outside the range for the instruction.

For example, the integrated assembler incorrectly fails to report an error for the following:

```
b . + 33554440 // An A32 B instruction has the range
                // -33554432 to 33554428
```

Instead, the integrated assembler incorrectly encodes the instruction as:

```
b . - 33554424
```

Conditions

The safety-related system is at risk when all the following are true:

- The program contains a branch instruction `ι` with a destination `δ`.
- `δ` has one of the following forms:
 - `. + <offset>`
 - `<label> + <offset>`
- `<offset>` is an immediate value.
- The behavior of the program depends on `ι` branching to `δ`.

2.3.2.20 SDCOMP-53903

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-53903.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.22.1	-

Description

The compiler can incorrectly fail to report one of the following warnings:

- inline namespace reopened as a non-inline namespace
- non-inline namespace reopened as an inline namespace

For example, the compiler incorrectly fails to report a warning for the inline namespace being re-opened as a non-inline namespace for the following:

```
namespace A {
    inline namespace {}
}
namespace {}
```

and incorrectly fails to report a warning for the non-inline namespace being re-opened as an inline namespace for the following:

```
namespace A {
    namespace {}
}
inline namespace {}
```

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled in a C++ source language mode.
- The program contains a namespace **A**.
- **A** contains a namespace **B**.
- One of the following is true:
 - **B** is inline, and is re-opened as non-inline outside **A**.
 - **B** is non-inline, and is re-opened as inline outside **A**.
- The behavior of the program depends on the visibility of the members of **B** remaining unchanged.

2.3.2.21 SDCOMP-52627

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-52627.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.22.1	-

Description

The compiler incorrectly fails to report an error when a **constexpr** constructor of a class template fails to initialize an anonymous union member.

For example, the compiler incorrectly fails to report an error for the invalid **constexpr** constructor of `z`, which does not initialize `var`, in the following:

```
template < class > struct Z {
    union {
        int var;
    };
    constexpr Z() {}
};

constexpr Z<int> z;
```

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled in a C++11 or later source language mode.
- The program contains a class template `T`.
- `T` contains an anonymous member `m` of **union** type.
- `T` contains a **constexpr** constructor `z`.
- `z` does not initialize any member of `m`.
- The program contains a **constexpr** variable instantiation `i` of `T`.
- The behavior of the program depends on `i` initializing `m`.

2.3.2.22 SDCOMP-50017

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-50017.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.22.1	-

Description

The compiler incorrectly fails to report an error for the redefinition of a variable originally declared in the controlling expression of an **if**, **for**, **switch**, or **while** statement.

For example, the compiler incorrectly fails to report an error for the redeclaration of `var` in the following:

```
void func(void)
{
    if (int var = 0)
    {
        extern int var();
    }
}
```

This defect is associated with the issue is described in [SDCOMP-56220](#).

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled in a C++ source language mode.
- The program contains an **if**, **for**, **switch**, or **while** statement `s`.
- `s` has a controlling expression that defines a variable `v`.
- The outermost block of `s` contains a redeclaration of `v` as a function.
- The behavior of the program depends on `v` not being redeclared as a function.

2.3.2.23 SDCOMP-49961

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-49961.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.22.1	-

Description

The compiler incorrectly fails to report a warning for a variadic function arguments list that contains an argument of `__fp16` or `_Float16` type. Use of these types in a variadic function arguments list has undefined behavior.

For example, the compiler incorrectly fails to report a warning for `var` in the following:

```
#include <stdarg.h>

void func(int a, ...)
{
    va_list vl;
```



```
    va_start(vl, a);
    __fp16 var = va_arg(vl, __fp16);
}
```

Conditions

The safety-related system is at risk when all the following are true:

- The program uses the `va_arg` macro with a variadic function arguments list `L`.
- The next parameter in `L` is `P`.
- `P` is of `__fp16` or `_Float16` type.
- The behavior of the program depends on `P` not being promoted to a different type.

2.3.2.24 SDCOMP-49919

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-49919.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.22.1	-

Description

The compiler incorrectly fails to report an error for an ambiguous call to an **extern "C"** function using a default argument.

For example, the compiler incorrectly fails to report an error for the ambiguous call to `func1()` in the following:

```
namespace A
{
    extern "C" int func1 (int var = 1);
}

namespace B
{
    extern "C" int func1 (int var = 2);
}

using A::func1;
using B::func1;

int func2(void)
{
    return func1();
}
```

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled in a C++ source language mode.

- The program contains two namespaces `A` and `B`.
- Both `A` and `B` declare an `extern "C"` function `F`.
- `F` has the same name in both `A` and `B`.
- `F` has a default argument.
- The program contains using-declarations or using-directives that make both `A::F` and `B::F` accessible in a block `x`.
- `x` contains a call to `F` using a default argument.

2.3.2.25 SDCOMP-49763

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-49763.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.22.1	-

Description

The compiler incorrectly fails to report an error for a base class destructor that is called using the type name of a derived class.

For example, the compiler incorrectly fails to report an error for the call to `Derived::~~Base()` in the following:

```
struct Base
{
    ~Base() { }
};

struct Derived : Base {};

void func(void)
{
    Derived *ptr = new Derived;
    ptr-> Derived::~~Base();
}
```

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled in a C++ source language mode.
- The program contains a class `z`.
- The program contains a class `D` that is derived from `z`.
- The program contains an instance `I` of `D`.
- The program contains an expression that has one of the following forms:

- `I.D::~~Z()`
- `I->D::~~Z()`

2.3.2.26 SDCOMP-25238

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-25238.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.22.1	-

Description

The compiler incorrectly fails to report an error for an uninitialized variable of **union** type that contains a member of **const** type.

For example, the compiler incorrectly fails to report an error for the variable `u` in the following:

```
union U
{
    const short a;
    const int b;
} f;
```

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled in a C++ source language mode.
- The program contains a **union** `u`.
- `u` has a member of **const** type.
- `u` does not have a user-defined default constructor.
- The program contains an uninitialized variable `v` of type `u`.
- The behavior of the program depends on `v`.

2.3.2.27 SDCOMP-18689

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-18689.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armlink	Any	6.22.1	-

Description

The linker incorrectly fails to report an error for a call to a linker execution address or load address built-in function that uses an ambiguous execution region or load region name.

Conditions

The safety-related system is at risk when all the following are true:

- The program is linked with a scatter file `F`.
- `F` contains an execution region with the name `A`.
- `F` contains a load region with the name `B`.
- `A` and `B` are the same name `N`.
- `F` contains a call `z` to one of the following linker execution address or load address built-in functions:
 - `ImageBase()`
 - `ImageLength()`
 - `ImageLimit()`
 - `LoadBase()`
 - `LoadLength()`
 - `LoadLimit()`
- `z` uses `N`.
- The memory layout of the program depends on `z`.

2.3.2.28 SDCOMP-17355

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-17355.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armlink	Any	6.22.1	-

Description

The linker incorrectly fails to report an error for an `ARM_LIB_STACK` or `ARM_LIB_STACKHEAP` execution region that does not end at one of the following:

- A 16-byte boundary for AArch64 state.
- An 8-byte boundary for AArch32 state.

For example, the linker incorrectly fails to report an error for the following:

```
ARM_LIB_STACKHEAP 0xF000 EMPTY 0x1004 { }
```

Conditions

The safety-related system is at risk when all the following are true:

- The program is linked with a scatter file \mathbb{F} .
- \mathbb{F} contains an execution region \mathbb{E} that has one of the following names:
 - `ARM_LIB_STACK`
 - `ARM_LIB_STACKHEAP`
- One of the following is true:
 - The program is built for AArch64 state and \mathbb{E} does not end at a 16-byte boundary.
 - The program is built for AArch32 state and \mathbb{E} does not end at an 8-byte boundary.

2.3.3 Determinism faults

There are no known determinism faults that affect qualified components.

For more information about the definition of a determinism fault, see the *Arm Compiler for Embedded FuSa tools and functional safety* section of the *Arm Compiler for Embedded FuSa 6.22LTS Qualification Kit Safety Manual*.

2.3.4 Documentation synchronization faults

This section contains details about safety-related defects that have been classified as a documentation synchronization fault.

For more information about the definition of a documentation synchronization fault, see the *Arm Compiler for Embedded FuSa tools and functional safety* section of the *Arm Compiler for Embedded FuSa 6.22LTS Qualification Kit Safety Manual*.

2.3.4.1 SDCOMP-66862

This section describes the scope of the documentation synchronization fault defect with the unique identifier SDCOMP-66862.



Arm has not yet fully investigated this defect. Therefore, the information in this section is likely to change in future versions of this document.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.22.1	-

Description

The *Restrictions with Link-Time Optimization* section of the *User Guide* incorrectly does not state that the compiler is not guaranteed to report an error for invalid instructions in file-scope inline assembly when compiling with Link-Time Optimization (LTO) enabled.

For example, the compiler is not guaranteed to report an error for the following file-scope inline assembly statement when compiling for AArch32 state with `-march=armv7-a+nofp`:

```
__asm("vmov s0, s1");
```

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled with Link-Time Optimization (LTO) enabled.
- The target options specified to the compiler disable an architectural feature \mathbb{F} .
- The program contains an instruction \mathbb{I} .
- \mathbb{I} requires \mathbb{F} .

2.3.4.2 SDCOMP-61514

This section describes the scope of the documentation synchronization fault defect with the unique identifier SDCOMP-61514.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch32 state	6.22.1	-

Description

The *ACLE support* section of the *Reference Guide* incorrectly does not state that the `poly8_t`, `poly16_t`, and `poly64_t` types are defined as **signed** in the `<arm_neon.h>` system header. Subsequently, this can result in unexpected run-time behavior if the program depends on `poly8_t`, `poly16_t`, or `poly64_t` being **unsigned**.

Conditions

The safety-related system is at risk when all the following are true:

- The program contains a variable v of type \mathbb{A} or type \mathbb{B} , where:
 - \mathbb{A} is one of the following types defined in the `<arm_neon.h>` system header:
 - `poly8_t`
 - `poly16_t`
 - `poly64_t`
 - \mathbb{B} is derived from \mathbb{A} .

- The behavior of the program depends on `v` being **unsigned**.

2.4 Defects affecting unqualified components

This section contains details about known safety-related defects that affect the unqualified toolchain components of Arm Compiler for Embedded FuSa 6.22LTS.

The unqualified toolchain components are:

- The legacy assembler, `armasm`.
- The libraries supplied with the toolchain.



Unqualified toolchain components are outside the scope of the Qualification Kit. Defects related to unqualified toolchain components are provided in this document for information only.

The following defects are included in this section:

Identifier	Fault category	Affected components
SDCOMP-66090	Translation fault	Libraries
SDCOMP-65871	Translation fault	Libraries
SDCOMP-63111	Translation fault	Libraries
SDCOMP-62801	Translation fault	Libraries
SDCOMP-60784	Translation fault	Libraries
SDCOMP-60162	Translation fault	Libraries
SDCOMP-53422	Translation fault	Libraries
SDCOMP-50751	Translation fault	Libraries
SDCOMP-50064	Translation fault	Libraries
SDCOMP-45879	Translation fault	Libraries
SDCOMP-30903	Translation fault	Libraries
SDCOMP-30359	Translation fault	Libraries
SDCOMP-29077	Translation fault	Libraries
SDCOMP-18016	Translation fault	Libraries
SDCOMP-13831	Translation fault	Libraries

2.4.1 Translation faults

This section contains details about safety-related defects that have been classified as a translation fault.

For more information about the definition of a translation fault, see the *Arm Compiler for Embedded FuSa tools and functional safety* section of the *Arm Compiler for Embedded FuSa 6.22LTS Qualification Kit Safety Manual*.

2.4.1.1 SDCOMP-66090

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-66090.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	AArch64 state	6.22.1	-

Description

The Arm C library implementation of the `calloc(num, size)` function can incorrectly fail to return a null pointer. This can result in unexpected run-time behavior.

To avoid this issue, manually inspect the source code and ensure that the program explicitly checks that `num*size` does not overflow $(1 << 64) - 1$ before each call to `calloc()`.

Conditions

The safety-related system is at risk when all the following are true:

- The program is linked with the unqualified C libraries supplied with the toolchain.
- The program calls `calloc(num, size)`.
- The value of `num*size` is greater than or equal to $(1 << 64)$.

2.4.1.2 SDCOMP-65871

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-65871.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	Armv8-R AArch64	6.22.1	-

Description

A floating-point arithmetic operation or a C library function call involving a value or variable of **double** type can return an incorrect result or set the `FE_INEXACT` floating-point exception flag incorrectly.

Conditions

The safety-related system is at risk when all the following are true:

- The program is linked with the C libraries supplied with Arm Compiler.
- The program is built with target options that disable hardware floating-point support.
- One of the following is true:
 - The program contains an arithmetic operation `x` involving a value or variable of **double** type.
 - The program contains a call `x` to a C library function with an argument of **double** type. For example, the `nearbyint()` function defined in the `<math.h>` system header.
- The behavior of the program depends on one of the following:
 - The `FE_INEXACT` floating-point exception flag being set or cleared by `x`.
 - The result of `x`.

2.4.1.3 SDCOMP-63111

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-63111.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	Any	6.22.1	-

Description

The Arm C Library implementation of certain BFloat16 intrinsics defined in the `<arm_neon.h>` system header incorrectly relies on C undefined behavior. Subsequently, this can result in unexpected run-time behavior.

For more information about C undefined behavior, see the article [What is "undefined behavior" in terms of compiling C/C++ code, and what implications can it have on a project?](#)

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

Conditions

The safety-related system is at risk when all the following are true:

- The program is built with target options that enable the BFloat16 Floating-point Extension feature (FEAT_BF16).

- The program contains a call to a BFloat16 intrinsic `1` defined in the `<arm_neon.h>` system header.
- `1` is of the form `vcvt*_f32_bf16()`.

2.4.1.4 SDCOMP-62801

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-62801.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	Armv8-M with the Main Extension	6.22.1	-

Description

The M-profile PACBTI variant of the Arm C library implementation of the `strcmp()` function incorrectly assumes that the function parameters are always aligned to a 4-byte boundary. Subsequently, this can result in unexpected run-time behavior.

Conditions

The safety-related system is at risk when all the following are true:

- The program is linked with the C libraries supplied with Arm Compiler.
- The program is not linked with `--library_security=none`.
- The program is compiled with `-mno-unaligned-access`.
- The program is compiled with an `-mbranch-protection=<protection>` option that enables Branch Target Identification (BTI) branch protection. For example, `-mbranch-protection=standard`.
- The program contains call `z` to the `strcmp()` function.
- A parameter of `z` is not aligned to a 4-byte boundary.

2.4.1.5 SDCOMP-60784

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-60784.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	Any	6.22.1	-

Description

The Arm C library implementations of the `fma()` and `fmaf()` functions incorrectly fail to set `errno` to `ERANGE` upon an overflow or underflow. Additionally, upon an underflow, these functions can return an incorrect sign of zero.

Conditions

The safety-related system is at risk when all the following are true:

- The program is built for one of the following:
 - AArch32 state.
 - An Armv8-R AArch64 target without hardware floating-point support.
- The program is linked with the C libraries supplied with Arm Compiler.
- The program contains a call `z` to one of the following functions:
 - `fma()`
 - `fmaf()`
- The third parameter of `z` is zero.
- The product of the the first two parameters of `z` results in an overflow or underflow.
- The behavior of the program depends on one of the following:
 - `z` setting `errno` to `ERANGE` upon an overflow or underflow.
 - `z` returning the correct sign of zero upon an underflow.

2.4.1.6 SDCOMP-60162

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-60162.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	AArch64 state	6.22.1	-

Description

The Arm C library implementations of functions that convert between multibyte characters and wide characters can result in an alignment fault at run-time.

For example, the call to the `printf()` function in the following code results in an alignment fault when run on an Armv8-A target with unaligned memory accesses disabled:

```
#include <stdio.h>
#include <wchar.h>

__asm(".global __use_utf8_ctype\n");

int main(void)
{
```

```
const wchar_t *wstr = L"wide string";
printf("%ls\n", wstr);

return 0;
}
```

Conditions

The safety-related system is at risk when all the following are true:

- The program uses one of the following `LC_TYPE` locales:
 - A user-defined locale that uses the `LC_CTYPE_multibyte` legacy assembler macro.
 - Shift-JIS
 - UTF-8
- The program contains a call to an Arm C library function that converts between multibyte characters and wide characters. For example:
 - `mbtowc()`
 - `printf()` with a `%ls` format specifier.
 - `wctomb()`
 - `wprintf()` with a `%s` format specifier.
- The program is run on a target that has unaligned memory accesses disabled.

2.4.1.7 SDCOMP-53422

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-53422.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	Any	6.22.1	-

Description

The Arm C library implementation of the `pow()` function can incorrectly fail to set `errno` to `ERANGE` when the return value overflows to `HUGE_VAL`.

Conditions

The safety-related system is at risk when all the following are true:

- The program is linked with the C libraries supplied with Arm Compiler.
- The program contains a call `z` to the `pow()` function.
- `z` has arguments that result in an overflow to `HUGE_VAL`.
- The behavior of the program depends on `z` setting `errno` to `ERANGE`.

2.4.1.8 SDCOMP-50751

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-50751.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	Any	6.22.1	-

Description

The Arm C library implementation of the `setlocale()` function incorrectly fails to return a null pointer for a locale selection that cannot be honored at run-time.

For example, the Arm C library implementation of the `setlocale()` function incorrectly fails to return a null pointer for the following:

```
const char *retstr = setlocale(LC_ALL, "invalid");
```

Conditions

The safety-related system is at risk when all the following are true:

- The program is linked with the C libraries supplied with Arm Compiler.
- The program contains a call `z` to the `setlocale()` function with a locale string `s`.
- `s` specifies a locale selection that cannot be honored at run-time.
- The behavior of the program depends on `z` returning a null pointer.

2.4.1.9 SDCOMP-50064

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-50064.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	Any	6.22.1	-

Description

The Arm implementation of the C++ regular expressions library can behave incorrectly for an invalid regular expression, resulting in one of the following:

- A failure to call the `abort()` function.
- A failure to throw a `std::regex_error` exception.
- Throwing an incorrect `std::regex_error` exception.

For example, the Arm implementation of the `std::regex` constructor incorrectly fails to call the `abort()` function or throw a `std::regex_error` exception for the invalid regular expression `[c-a]` in the following:

```
std::regex re("[c-a]", std::regex_constants::basic);
```

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled in a C++ source language mode.
- The program uses the C++ regular expressions library with a regular expression `R`.
- `R` is invalid.
- The behavior of the program depends on the use of `R` causing one of the following:
 - A call to the `abort()` function
 - A `std::regex_error` exception

2.4.1.10 SDCOMP-45879

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-45879.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	AArch64 state	6.22.1	-

Description

The Arm C library implementations of the `bsearch()` and `qsort()` functions can incorrectly corrupt the stack when used to process an array larger than 4GB. Subsequently, this can result in unexpected run-time behavior.

Conditions

The safety-related system is at risk when all the following are true:

- The program is linked with the C libraries supplied with Arm Compiler.
- The program contains a call `z` to one of the following functions:
 - `bsearch()`
 - `qsort()`
- `z` specifies an array containing `M` members of size `N` each.
- `M * N` is larger than 4GB.

2.4.1.11 SDCOMP-30903

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-30903.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	Any	6.22.1	-

Description

The Arm C++ library implementations of the assignment operators of the following classes can return an incorrect result:

- `std::gslice_array`
- `std::indirect_array`
- `std::mask_array`
- `std::slice_array`

For example, the assignment expression `A = B` returns an incorrect result in the following:

```
std::valarray<int> V = { 0, 1, 2, 3, 4, 5, 6 };
const std::slice_array<int> A = V[std::slice(1, 3, 2)];
const std::slice_array<int> B = V[std::slice(0, 3, 1)];
A = B;
```

Conditions

The safety-related system is at risk when all the following are true:

- The program is compiled in a C++ source language mode.
- The program contains a variable `v` of `std::valarray<T>` type.
- The program contains two variables `A` and `B`.
- `A` and `B` both have one of the following types:
 - `std::gslice_array`
 - `std::indirect_array`
 - `std::mask_array`
 - `std::slice_array`
- `A` and `B` are each initialized with an expression of the form `v[<index>]`.
- `<index>` is an expression that has one of the following types:
 - `std::gslice`
 - `std::slice`
 - `std::valarray<bool>`
 - `std::valarray<size_t>`

- The program contains an assignment expression $A = B$.

2.4.1.12 SDCOMP-30359

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-30359.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	Any	6.22.1	-

Description

The constructors of the Arm C++ library implementation of `std::locale` incorrectly either call the `abort()` function or throw a `std::runtime_error` exception.

For example, when compiling with `-fno-exceptions`, the constructor incorrectly calls the `abort()` function for the following:

```
std::locale obj("C");
```

Conditions

The safety-related system is at risk when all the following are true:

- The program is linked with the C++ libraries supplied with Arm Compiler.
- The program contains a call to a `std::locale` constructor with a locale name `N`.
- `N` is a valid standard C locale name.
- The behavior of the program depends on the locale being successfully set to `N`.

2.4.1.13 SDCOMP-29077

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-29077.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	Any	6.22.1	-

Description

The constructors of the Arm C++ library implementations of certain `std::<facet_category>_byname` locale-specific facet categories incorrectly always either call the `abort()` function or throw a `std::runtime_error` exception.

Conditions

The safety-related system is at risk when all the following are true:

- The program is linked with the C++ libraries supplied with Arm Compiler.
- The program contains a call `z` to a constructor of one of the following locale-specific facet categories:
 - `ctype_byname`
 - `codecvt_byname`
 - `collate_byname`
 - `moneypunct_byname`
 - `time_get_byname`
 - `time_put_byname`
- The behavior of the program depends on `z` returning successfully.

2.4.1.14 SDCOMP-18016

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-18016.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	Any	6.22.1	-

Description

The Arm C library `__heapstats()` and `__heapvalid()` functions can result in unexpected run-time behavior for a program that does not use the heap.

To avoid this issue, include the following file-scope inline assembly statement in an affected program:

```
__asm(".global __use_no_heap\n\t");
```

Conditions

The safety-related system is at risk when all the following are true:

- The program is linked with the C libraries supplied with Arm Compiler.
- The program contains a call to one of the following functions:
 - `__heapstats()`
 - `__heapvalid()`
- The program does not use the heap.

2.4.1.15 SDCOMP-13831

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-13831.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	Armv7-M	6.22.1	-

Description

The Arm C library implementation of `strcmp()` can incorrectly read up to 3 bytes past the end of a string being compared. This can result in unexpected run-time behavior.

For example, for a string placed at the end of accessible memory, this can result in a memory access fault.

Conditions

The safety-related system is at risk when all the following are true:

- The program is linked with the C libraries supplied with Arm Compiler.
- The program is not linked with `microlib`.
- The program contains a call `z` to the `strcmp()` function.
- An argument to `z` is a pointer `p`.
- `p` is not a multiple of 4 bytes.
- `p` points to a string `s`.
- The behavior of the program depends on `strcmp()` not accessing memory beyond the end of `s`.

2.4.2 Missing diagnostic faults

There are no known missing diagnostic faults that affect unqualified components.

For more information about the definition of a missing diagnostic fault, see the *Arm Compiler for Embedded FuSa tools and functional safety* section of the *Arm Compiler for Embedded FuSa 6.22LTS Qualification Kit Safety Manual*.

2.4.3 Determinism faults

There are no known determinism faults that affect unqualified components.

For more information about the definition of a determinism fault, see the *Arm Compiler for Embedded FuSa tools and functional safety* section of the *Arm Compiler for Embedded FuSa 6.22LTS Qualification Kit Safety Manual*.

2.4.4 Documentation synchronization faults

There are no known documentation synchronization faults that affect unqualified components.

For more information about the definition of a documentation synchronization fault, see the *Arm Compiler for Embedded FuSa tools and functional safety* section of the *Arm Compiler for Embedded FuSa 6.22LTS Qualification Kit Safety Manual*.

2.5 Defects affecting both qualified and unqualified components

This section contains details about known safety-related defects that affect both the qualified and unqualified toolchain components of Arm Compiler for Embedded FuSa 6.22LTS.

The qualified toolchain components are:

- The compiler and integrated assembler, `armclang`.
- The ELF processing utility, `fromelf`.
- The librarian, `armar`.
- The linker, `armlink`.

The unqualified toolchain components are:

- The legacy assembler, `armasm`.
- The libraries supplied with the toolchain.



Note

Unqualified toolchain components are outside the scope of the Qualification Kit. Defects related to unqualified toolchain components are provided in this document for information only.

The following defects are included in this section:

Identifier	Fault category	Affected components
SDCOMP-63948	Translation fault	armclang, Libraries

2.5.1 Translation faults

This section contains details about safety-related defects that have been classified as a translation fault.

For more information about the definition of a translation fault, see the *Arm Compiler for Embedded FuSa tools and functional safety* section of the *Arm Compiler for Embedded FuSa 6.22LTS Qualification Kit Safety Manual*.

2.5.1.1 SDCOMP-63948

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-63948.



Arm has not yet fully investigated this defect. Therefore, the information in this section is likely to change in future versions of this document.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang, Libraries	AArch64 state	6.22.1	-

Description

The compiler can generate incorrect C++ exception-handling code. Subsequently, this can result in unexpected run-time behavior.

Conditions

The safety-related system is at risk when all the following are true:

- The program is built with target options that enable the Scalable Vector Extension feature (FEAT_SVE).
- The program is compiled with C++ exceptions enabled.
- The program uses a type that is defined in the `<arm_sve.h>` system header.
- The program throws a C++ exception.

2.5.2 Missing diagnostic faults

There are no known missing diagnostic faults that affect both qualified and unqualified components.

For more information about the definition of a missing diagnostic fault, see the *Arm Compiler for Embedded FuSa tools and functional safety* section of the *Arm Compiler for Embedded FuSa 6.22LTS Qualification Kit Safety Manual*.

2.5.3 Determinism faults

There are no known determinism faults that affect both qualified and unqualified components.

For more information about the definition of a determinism fault, see the *Arm Compiler for Embedded FuSa tools and functional safety* section of the *Arm Compiler for Embedded FuSa 6.22LTS Qualification Kit Safety Manual*.

2.5.4 Documentation synchronization faults

There are no known documentation synchronization faults that affect both qualified and unqualified components.

For more information about the definition of a documentation synchronization fault, see the *Arm Compiler for Embedded FuSa tools and functional safety* section of the *Arm Compiler for Embedded FuSa 6.22LTS Qualification Kit Safety Manual*.

Appendix A Changes since the Arm Compiler for Embedded FuSa 6.22LTS Defect Notification Report for December 2024

This appendix provides information about changes made to the defect lists compared to the [Arm Compiler for Embedded FuSa 6.22LTS Defect Notification Report for December 2024](#).

A.1 Defects added

This section contains a list of defects that have been added to this document compared to the [Arm Compiler for Embedded FuSa 6.22LTS Defect Notification Report for December 2024](#).

Identifier	Fault category	Affected components	Target environment
SDCOMP-67984	Missing diagnostic fault	armclang	Armv8-M
SDCOMP-67968	Missing diagnostic fault	armclang	AArch64 state
SDCOMP-67799	Translation fault	armclang	Armv8-M with the Main extension
SDCOMP-67678	Translation fault	armclang	Armv8-M with the Main extension
SDCOMP-67650	Translation fault	armclang	Armv8-M with the Main Extension
SDCOMP-66895	Translation fault	armclang	Armv8-M
SDCOMP-66894	Missing diagnostic fault	armclang	AArch64 state
SDCOMP-66692	Translation fault	fromelf	Armv8-M with the Main Extension
SDCOMP-65590	Translation fault	armclang	Armv8-M with the Main Extension
SDCOMP-65579	Translation fault	armclang	AArch64 state
SDCOMP-65550	Translation fault	armclang	AArch64 state
SDCOMP-64397	Translation fault	armclang	Armv8-M with the Main Extension
SDCOMP-63205	Translation fault	armclang	AArch32 state
SDCOMP-63114	Translation fault	armclang	AArch64 state
SDCOMP-63088	Translation fault	armclang	AArch64 state
SDCOMP-62801	Translation fault	Libraries	Armv8-M with the Main Extension
SDCOMP-55200	Translation fault	armclang	Armv8-M with the Main Extension
SDCOMP-55040	Translation fault	armclang	AArch64 state

A.2 Defects updated

This section contains a list of defects that have been updated in this document compared to the [Arm Compiler for Embedded FuSa 6.22LTS Defect Notification Report for December 2024](#).

Identifier	Fault category	Affected components	Target environment
SDCOMP-67662	Translation fault	armclang	Armv8-M with the Main Extension
SDCOMP-67424	Missing diagnostic fault	armclang	AArch32 state
SDCOMP-67120	Missing diagnostic fault	armclang	AArch32 state
SDCOMP-65592	Translation fault	armclang	AArch64 state
SDCOMP-65418	Translation fault	armclang	AArch32 state
SDCOMP-64590	Translation fault	armlink	Any
SDCOMP-63911	Translation fault	armclang	AArch32 state
SDCOMP-63111	Translation fault	Libraries	Any
SDCOMP-62378	Translation fault	armclang	AArch32 state
SDCOMP-62201	Missing diagnostic fault	armclang	Armv8-A
SDCOMP-60117	Translation fault	armlink	Any
SDCOMP-50408	Translation fault	armclang	AArch32 state
SDCOMP-45879	Translation fault	Libraries	AArch64 state

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.

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

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
202501-00	24 January 2025	Non-Confidential	Initial release

Change history

For a list of technical changes since the last release listed in the release history of this document, see [Changes since the Arm Compiler for Embedded FuSa 6.22LTS Defect Notification Report for December 2024](#).

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
<i>italic</i>	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 <u>underline</u>	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: <div>MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2></div>
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.



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



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

Useful resources

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

Access to Arm documents depends on their confidentiality:

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

Arm product resources	Document ID	Confidentiality
<i>Arm Compiler for Embedded FuSa 6.22LTS Qualification Kit Safety Manual</i>	109409	Confidential
Arm Compiler for Embedded FuSa 6.22LTS documentation index	KA006002	Non-Confidential
Does Arm document all known issues that affect each Arm Compiler release?	KA005052	Non-Confidential