



Arm Compiler for Embedded FuSa 6.16LTS

Defect Notification Report

Version July 2024

Non-Confidential

Copyright © 2024 Arm Limited (or its affiliates).
All rights reserved.

Issue

107987_2024-07_en



Arm Compiler for Embedded FuSa 6.16LTS Defect Notification Report

This document is Non-Confidential.

Copyright © 2024 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 (107987_2024-07_en) was issued on 2024-07-26. There might be a later issue at <http://developer.arm.com/documentation/107987>

The product version is July 2024.

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 6.16LTS, and is using an Arm Compiler for Embedded FuSa 6.16LTS 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 6.16LTS.

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..... 13
 - 2.3.2 Missing diagnostic faults..... 80
 - 2.3.3 Determinism faults..... 109
 - 2.3.4 Documentation synchronization faults..... 110
 - 2.4 Defects affecting unqualified components..... 114
 - 2.4.1 Translation faults..... 115
 - 2.4.2 Missing diagnostic faults..... 140
 - 2.5 Defects affecting both qualified and unqualified components..... 146
 - 2.5.1 Translation faults..... 147
 - 2.5.2 Documentation synchronization faults..... 148
- A. Changes since the Arm Compiler for Embedded FuSa 6.16.2 Qualification Kit Defect Report..... 150**
 - A.1 Defects added..... 150
 - A.2 Defects removed..... 152
 - A.3 Defects updated..... 153
- Proprietary notice..... 156**
- Product and document information..... 158**
 - Product status..... 158
 - Revision history..... 158
 - Conventions..... 158

Useful resources..... 161

1. Introduction

This document is intended for functional safety managers and software developers using Arm Compiler for Embedded FuSa 6.16LTS for functional safety projects.

This document has been created based on information available to Arm as of 26 July 2024. It provides an updated list of known safety-related defects that affect a release of Arm Compiler for Embedded FuSa 6.16LTS, 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 6.16LTS 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 6.16LTS releases: 6.16.1 or 6.16.2.

See *The role of Arm Compiler for Embedded FuSa in Safety-related Development* in the *Arm Compiler for Embedded FuSa 6.16LTS Qualification Kit Safety Manual* for an explanation of the safety-related fault categories.

See the *Arm Compiler for Embedded FuSa 6.16LTS 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 6.16LTS 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 6.16LTS 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 6.16.2 of the Arm Compiler for Embedded FuSa tools, see the *References* section of release 6.16.2 of the *Arm Compiler for Embedded FuSa 6.16LTS 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 6.16LTS.

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 6.16LTS 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
- 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.
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 6.16LTS.

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-66632	Translation fault	armclang
SDCOMP-66256	Translation fault	armclang
SDCOMP-65517	Translation fault	armlink
SDCOMP-65172	Translation fault	armclang
SDCOMP-64999	Translation fault	armlink
SDCOMP-64595	Translation fault	armlink
SDCOMP-64591	Translation fault	armclang
SDCOMP-64590	Translation fault	armlink
SDCOMP-64165	Translation fault	armclang
SDCOMP-64066	Translation fault	armclang
SDCOMP-64059	Translation fault	armclang
SDCOMP-63984	Translation fault	armclang
SDCOMP-63952	Translation fault	armclang
SDCOMP-63946	Translation fault	armclang
SDCOMP-63913	Translation fault	armclang
SDCOMP-63912	Translation fault	armclang
SDCOMP-63911	Translation fault	armclang
SDCOMP-63894	Translation fault	armclang
SDCOMP-63761	Translation fault	armclang
SDCOMP-63752	Translation fault	armclang
SDCOMP-63738	Translation fault	fromelf
SDCOMP-63688	Translation fault	armclang
SDCOMP-63454	Translation fault	armclang
SDCOMP-62791	Translation fault	armclang
SDCOMP-62769	Translation fault	armclang
SDCOMP-62725	Translation fault	armclang
SDCOMP-62692	Translation fault	armclang
SDCOMP-62661	Translation fault	armclang

Identifier	Fault category	Affected components
SDCOMP-62378	Translation fault	armclang
SDCOMP-62352	Translation fault	armclang
SDCOMP-62330	Translation fault	armclang
SDCOMP-62251	Translation fault	armlink
SDCOMP-62221	Translation fault	armclang
SDCOMP-62217	Translation fault	fromelf
SDCOMP-62176	Translation fault	armclang
SDCOMP-62133	Translation fault	armclang
SDCOMP-62123	Translation fault	armclang
SDCOMP-62028	Translation fault	armclang
SDCOMP-61514	Translation fault	armclang
SDCOMP-61486	Translation fault	armclang
SDCOMP-61299	Translation fault	armclang
SDCOMP-61298	Translation fault	armclang
SDCOMP-61150	Translation fault	armlink
SDCOMP-61080	Translation fault	armclang
SDCOMP-60897	Translation fault	armclang
SDCOMP-60725	Translation fault	fromelf
SDCOMP-60659	Translation fault	armlink
SDCOMP-60632	Translation fault	armclang
SDCOMP-60589	Translation fault	armclang
SDCOMP-60443	Translation fault	armclang
SDCOMP-60342	Translation fault	armclang
SDCOMP-60326	Translation fault	fromelf
SDCOMP-60117	Translation fault	armlink
SDCOMP-59974	Translation fault	armclang
SDCOMP-59938	Translation fault	armlink
SDCOMP-59788	Translation fault	armclang
SDCOMP-59656	Translation fault	armclang
SDCOMP-59521	Translation fault	armclang
SDCOMP-59074	Translation fault	armclang
SDCOMP-59059	Translation fault	armclang
SDCOMP-58780	Translation fault	armclang
SDCOMP-58773	Translation fault	armclang
SDCOMP-58738	Translation fault	armclang
SDCOMP-58354	Translation fault	armlink
SDCOMP-57884	Translation fault	armclang
SDCOMP-57725	Translation fault	armclang
SDCOMP-57674	Translation fault	armclang
SDCOMP-57456	Translation fault	fromelf

Identifier	Fault category	Affected components
SDCOMP-57449	Translation fault	fromelf
SDCOMP-57255	Translation fault	armclang
SDCOMP-57229	Translation fault	armclang
SDCOMP-57213	Translation fault	armlink
SDCOMP-57200	Translation fault	armclang
SDCOMP-56435	Translation fault	armlink
SDCOMP-55460	Translation fault	armclang
SDCOMP-55184	Translation fault	fromelf
SDCOMP-54546	Translation fault	fromelf
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-65264	Missing diagnostic fault	armclang
SDCOMP-65243	Missing diagnostic fault	armclang
SDCOMP-64683	Missing diagnostic fault	armclang
SDCOMP-64255	Missing diagnostic fault	armclang
SDCOMP-63917	Missing diagnostic fault	armclang
SDCOMP-63697	Missing diagnostic fault	armclang
SDCOMP-62234	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-61089	Missing diagnostic fault	armclang
SDCOMP-59605	Missing diagnostic fault	armclang
SDCOMP-59512	Missing diagnostic fault	armclang
SDCOMP-59190	Missing diagnostic fault	armclang
SDCOMP-58367	Missing diagnostic fault	armclang
SDCOMP-57912	Missing diagnostic fault	armclang
SDCOMP-57528	Missing diagnostic fault	armclang
SDCOMP-57199	Missing diagnostic fault	armlink
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-55580	Missing diagnostic fault	armclang

Identifier	Fault category	Affected components
SDCOMP-55267	Missing diagnostic fault	armclang
SDCOMP-53903	Missing diagnostic fault	armclang
SDCOMP-52627	Missing diagnostic fault	armclang
SDCOMP-51180	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-46790	Missing diagnostic fault	armclang
SDCOMP-25238	Missing diagnostic fault	armclang
SDCOMP-18689	Missing diagnostic fault	armlink
SDCOMP-17355	Missing diagnostic fault	armlink
SDCOMP-57994	Determinism fault	armlink
SDCOMP-65669	Documentation synchronization fault	armclang
SDCOMP-61633	Documentation synchronization fault	armclang
SDCOMP-61465	Documentation synchronization fault	armclang
SDCOMP-61054	Documentation synchronization fault	armlink
SDCOMP-60826	Documentation synchronization fault	armlink

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 6.16LTS Qualification Kit Safety Manual*.

2.3.1.1 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	6.16.1, 6.16.2	-

Description

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

For example, the integrated assembly 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 issue described in [SDCOMP-57200](#).

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 explicitly contain one of the following directives:
 - `.align`
 - `.balign`
 - `.p2align`

2.3.1.2 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	6.16.1, 6.16.2	-

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`.

2.3.1.3 SDCOMP-65517

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-65517.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armlink	Any	6.16.1, 6.16.2	-

Description

For two execution regions **A** and **B**, where **A** is at a lower address than **B**, the linker incorrectly assumes that the name of **A** is always lexically earlier than the name of **B**. Subsequently, this can result in the linker generating incorrect callgraph or stack usage information. The order in which **A** and **B** are specified in the scatter file does not affect this behavior.

For example, when a program is linked for AArch32 state with a scatter file that contains the following:

```
LR1 0x02000000
{
    ER_A +0
    {
        ...
    }
}

LR2 0x00004000
{
    ER_B +0
    {
        ...
    }
}
```

ER_B is at the lower address 0x00004000 and **ER_A** is at the higher address 0x02000000. However, the name **ER_B** is not lexically earlier than the name **ER_A**. This can result in the linker generating incorrect callgraph or stack usage information. The fact that **ER_A** is specified before **ER_B** does not affect this behavior.

To avoid this issue, manually inspect the scatter file and ensure that all execution regions have the same lexical order as their address order. For the execution regions in the above example, the execution regions can be renamed as follows:

Original name	New name
ER_A	Y_ER_A
ER_B	X_ER_B

Conditions

This defect can occur when all the following are true:

- The program is linked with a scatter file **F**.
- **F** contains two execution regions **A** and **B**.
- The address of **A** is lower than the address of **B**.
- The name of **A** is not lexically earlier than the name of **B**.
- 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 output causes you to manually make an incorrect change to the safety-related system.

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

2.3.1.4 SDCOMP-65172

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-65172.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.16.1, 6.16.2	-

Description

The compiler can generate incorrect debug information for a bit-field.

For example, the compiler generates incorrect debug information for `b` in the following:

```
struct __attribute__((packed)) S
{
    unsigned char a: 7;
    unsigned char b: 4; /* Packing places bit0 of b
                        in a different byte to bits1-3 */
};
```

This defect is associated with the issue described in [SDCOMP-62330](#).

Conditions

This defect can occur when all the following are true:

- The program is compiled with `-gdwarf-2` OR `-gdwarf-3`.
- The program contains a bit-field `b`.
- One of the following is annotated with `attribute (packed) :`
 - `b`.
 - The **struct** containing `b`.
- Packing affects the number of bytes that `b` spans.

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.

2.3.1.5 SDCOMP-64999

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-64999.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armlink	AArch32 state	6.16.1, 6.16.2	-

Description

The linker can generate an incorrect C++ exception-handling table. This can result in unexpected run-time behavior.

For example, building a program containing the following code:

```
void __attribute__((naked)) F1()
{
}

void F2()
{
    throw 123;
}
```

can result in an output image with the following:

- F1, a zero-sized symbol of code type
- F2, a non-zero-sized symbol of code type
- F1 and F2 being placed at the same address, CODE_ADDR
- F1 and F2 being placed in the order F1 then F2

The C++ exception-handling table entries corresponding to these symbols must have the same order as the symbols themselves. For example, when the symbols are placed in the order F1 then F2, the C++ exception-handling table entries must also be placed in the order F1 then F2.

However, the output image can contain C++ exception-handling table entries in the incorrect order F2 then F1. This can be observed by manually inspecting the output image using fromelf --text -e:

```
<start of exception-handling table>
...
  <entry for F2>: ... [<CODE ADDR>]
                  EHT Inline Personality Routine #0 (Sul6)
                   01      vsp = vsp + 0x8
                   84 08      pop r7,r14 (0x8 bytes)
  <entry for F1>: ... [<CODE ADDR>]
                  EXIDX_CANTUNWIND
...
<end of exception-handling table>
```

The C++ exception unwinder uses the order of symbols to find the corresponding exception-handling table entries. Subsequently, this can result in unexpected run-time behavior when the entry for `F1` is used to unwind `F2`.

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 is compiled with C++ exceptions enabled.
- The program contains `code` symbols `F1` and `F2`.
- `F1` is zero bytes in size.
- `F2` is not zero bytes in size.
- `F1` and `F2` have the same address.
- `F1` and `F2` are placed in the order `F1` then `F2`.
- An exception is thrown by, or is propagated through, `F2`.

2.3.1.6 SDCOMP-64595

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-64595.



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
armlink	AArch64 state	6.16.1, 6.16.2	-

Description

The linker can generate incorrect stack usage information.

Conditions

This defect can occur when all the following are true:

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

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

2.3.1.7 SDCOMP-64591

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-64591.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch32 state	6.16.1, 6.16.2	-

Description

The compiler can generate incorrect code for Neon intrinsics defined in the `<arm_neon.h>` system header.

Conditions

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

- The program contains a call `z` to a Neon intrinsic `I` defined in the `<arm_neon.h>` system header.
- `I` is one of the following:
 - `vld2q_dup_bf16()`
 - `vld2q_dup_f16()`
 - `vld2q_dup_f32()`
 - `vld2q_dup_p16()`
 - `vld2q_dup_p8()`
 - `vld2q_dup_s16()`
 - `vld2q_dup_s32()`
 - `vld2q_dup_s8()`
 - `vld2q_dup_u16()`
 - `vld2q_dup_u32()`
 - `vld2q_dup_u8()`
- The behavior of the program depends on `z`.

2.3.1.8 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	6.16.1, 6.16.2	-

Description

The linker can generate incorrect stack usage information.

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.

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

2.3.1.9 SDCOMP-64165

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-64165.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	A32 state	6.16.1, 6.16.2	-

Description

The inline assembler and integrated assembler can generate incorrect code for a PC-relative `ADR` instruction or a load literal instruction.

For example, the integrated assembler generates incorrect code for the PC-relative `ADR` instruction in the following:

```
.arm

.section .data.src, "a", %progbits
.balign 4
.global src
src:
.word 0x11223344

.section .text.func, "ax"
.balign 4
.global func
.type func, %function
func:
    adr r0, src
    bx lr
```

This defect is associated with the issues described in [SDCOMP-63454](#) and [SDCOMP-55580](#).

Conditions

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

- The program contains an instruction `I`.
- One of the following is true:
 - `I` is `LDRD`.
 - The program is assembled for a big-endian target, and `I` is one of the following:
 - `ADR`
 - `LDR`
 - `LDRB`
 - `LDRH`
 - `LDRSB`
 - `LDRSH`
- `I` specifies a label operand `L`.
- `I` is in a section `A`.
- `L` is in a section `B`.
- `A` and `B` are different.

2.3.1.10 SDCOMP-64066

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-64066.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Armv8-M with the Main Extension	6.16.1, 6.16.2	-

Description

The compiler can generate code that incorrectly uses the same register for both operands of the `__arm_sqrshr()` and `__arm_uqrshl()` M-profile Vector Extension (MVE) intrinsics. 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 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 one of the following:
 - `__arm_sqrshr()`
 - `__arm_uqrshl()`
- The behavior of the program depends on `I`.

2.3.1.11 SDCOMP-64059

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-64059.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Armv8-M with the Main Extension	6.16.1, 6.16.2	-

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_vhcaddq_rot270_s32()` intrinsic in the following code:

```
#include <arm_mve.h>

int32x4_t func(void)
{
    int32x4_t a = __arm_vuninitializedq_s32();
    return __arm_vhcaddq_rot270_s32(a, a);
}
```

the compiler incorrectly generates a `VHCADD.S32` instruction that uses the register `q0` for all operands:

```
vhcadd.s32 q0, q0, q0, #270
```

```
vmov    r0, r1, d0
vmov    r2, r3, d1
bx      lr
```

A `vhcadd.s32` instruction which specifies the same register as both the destination and second source operand is architecturally `CONSTRAINED UNPREDICTABLE`. 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 contains a call `a` to an MVE intrinsic of the form `vuninitializedq()` defined in the `<arm_mve.h>` system header.
- `a` creates an uninitialized vector `v`.
- The program contains a call `b` to an MVE intrinsic defined in the `<arm_mve.h>` system header.
- `b` specifies `v` as an argument.
- The behavior of the program depends on `b`.

2.3.1.12 SDCOMP-63984

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



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	6.16.1, 6.16.2	-

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`.

2.3.1.13 SDCOMP-63952

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-63952.



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	6.16.1, 6.16.2	-

Description

The compiler can generate incorrect code for an access to a **union** with a member of half-precision floating-point type.

Conditions

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

- The program is built with target options that do not enable the half-precision floating-point feature (FEAT_FP16).
- The program contains a **union** *u*.
- *u* has a member of one of the following types:
 - `__fp16`
 - `_Float16`
- *u* has a member that is not of one of the following types:
 - `__fp16`
 - `_Float16`
- The behavior of the program depends on an access to *u*.

2.3.1.14 SDCOMP-63946

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-63946.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.16.1, 6.16.2	-

Description

The compiler can generate incorrect code for a call to a function.

Conditions

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

- The program is built for a target with the Scalable Vector Extension (SVE).
- The program contains a call to a function F .
- F has a parameter P of type T .
- T is defined in the `<arm_sve.h>` system header.
- P must be passed using the stack.
- The behavior of the program depends on F .

2.3.1.15 SDCOMP-63913

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-63913.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.16.1, 6.16.2	-

Description

The compiler can generate incorrect code when compiling with a `-mharden-sls=<option>` option that enables the mitigation against Straight-Line Speculation (SLS) for `RET` and `BR` instructions.

Note: `-mharden-sls=<option>` is considered a [COMMUNITY] feature in Arm Compiler for Embedded FuSa 6.16.1 and 6.16.2. For more information about this option, refer to the [Straight-Line Speculation \(SLS\) hardening supplement for Arm Compiler for Embedded FuSa 6.16LTS](#)


Conditions

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

- The program is compiled with `-mharden-sls=all` OR `-mharden-sls=retbr`.
- One of the following is true:
 - The program is compiled at `-Oz` and without `-mno-outline`.
 - The program is compiled with `-moutline`.

2.3.1.16 SDCOMP-63912

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



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	AArch32 state	6.16.1, 6.16.2	-

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`.
- `A` has a bit-field member `m` of size `s` and type `B`.
- `s` is greater than the size of `B`.
- The program contains a function `F`.
- `F` has a parameter of type `A`.

- F accesses M .

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 .

2.3.1.17 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	6.16.1, 6.16.2	-

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 r0 is used for the parameter 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 T .
- The program contains a function F with a parameter of type T .
- T does not have any members.

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 .

2.3.1.18 SDCOMP-63894

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-63894.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.16.1, 6.16.2	-

Description

The compiler can generate incorrect code for a program that contains a **volatile** variable *v*. Such incorrect code can result in any of the following unexpected run-time behaviors:

- Initializing *v* incorrectly.
- Loading a byte from *v* from an incorrect offset.
- Storing a byte to *v* at an incorrect offset.

For example, if a variable *v* is expected to consist of the following sequence of bytes:

Address offset	Byte value
+0x0	0x00
+0x1	0x11
+0x2	0x22
+0x3	0x33
+0x4	0x44
+0x5	0x55
+0x6	0x66
+0x7	0x77
+0x8	0x88
+0x9	0x99
+0xa	0xaa
+0xb	0xbb
+0xc	0xcc
+0xd	0xdd
+0xe	0xee
+0xf	0xff

the compiler can instead generate incorrect code that treats *v* as consisting of the following incorrect sequence of bytes:

Address offset	Byte value
+0x0	0x88
+0x1	0x99

Address offset	Byte value
+0x2	0xaa
+0x3	0xbb
+0x4	0xcc
+0x5	0xdd
+0x6	0xee
+0x7	0xff
+0x8	0x00
+0x9	0x11
+0xa	0x22
+0xb	0x33
+0xc	0x44
+0xd	0x55
+0xe	0x66
+0xf	0x77

Subsequently, at run-time, a load of `v[0]` would incorrectly return `0x88` instead of `0x00`.

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 a **volatile** variable `v`.
- The behavior of the program depends on `v`.

2.3.1.19 SDCOMP-63761

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-63761.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.16.1, 6.16.2	-

Description

The compiler can generate incorrect code for a function that accesses a `__thread` or `thread_local` variable.

Conditions

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

- The program contains a `__thread` or `thread_local` variable `v`.
- The program contains a function `f`.

- `F` contains an access to `v`.
- One of the following is true:
 - `F` is in a compilation unit that is compiled with `-ftls-model=global-dynamic`.
 - `v` is annotated with `attribute((tls_model("global-dynamic")))`.

2.3.1.20 SDCOMP-63752

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-63752.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.16.1, 6.16.2	-

Description

The compiler can generate incorrect code for a `vfmlalq_laneq_high_f16()` Or `vfmlalq_laneq_low_f16()` Neon intrinsic.

This defect is associated with the issue described in [SDCOMP-63917](#).

Conditions

This defect can occur when all the following are true:

- The program is compiled for a target that supports the floating-point half-precision multiplication instructions feature (FEAT_FHM).
- The program contains a call to a Neon intrinsic `I` defined in the `<arm_neon.h>` system header.
- `I` is one of the following:
 - `vfmlalq_laneq_high_f16()`
 - `vfmlalq_laneq_low_f16()`

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 one of the following is true:

- The output contains a `FMLAL` (by element) instruction with a second source register of the form `<vd>.H[<index>]`, where `<vd>` is outside the range `v0-v15`.
- The output contains a `FMLAL2` (by element) instruction with a second source register of the form `<vd>.H[<index>]`, where `<vd>` is outside the range `v0-v15`.

2.3.1.21 SDCOMP-63738

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-63738.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
fromelf	Armv8-M with the Main Extension	6.16.1, 6.16.2	-

Description

The `fromelf` utility incorrectly disassembles the label operand of a `WLS` or `WLSTP` instruction as an immediate offset instead of a PC-relative offset.

For example, the `fromelf` utility incorrectly disassembles the following `WLSTP` instruction:

```
wlstp.8 lr, r0, <label>
```

as:

```
wlstp.8 lr, r0, #<offset>
```

instead of:

```
wlstp.8 lr, r0, {pc}+<offset>
```

This defect is associated with the issue described in [SDCOMP-62217](#).

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 `WLS` or `WLSTP` 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.

2.3.1.22 SDCOMP-63688

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-63688.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Armv6-M, Armv8-M without the Main Extension	6.16.1, 6.16.2	-

Description

The compiler can generate incorrect code that corrupts register `®4`. This can result in unexpected run-time behavior.

Conditions

The safety-related system is at risk when the program is compiled with one of the following options:

- `-fstack-protector`
- `-fstack-protector-all`
- `-fstack-protector-strong`

2.3.1.23 SDCOMP-63454

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-63454.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	T32 state	6.16.1, 6.16.2	-

Description

The inline assembler and integrated assembler can generate incorrect code for a PC-relative `ADR` instruction or a load literal instruction.

For example, the integrated assembler generates incorrect code for the PC-relative `ADR` instruction in the following:

```
.thumb

.section .data.src, "a", %progbits
.balign 4
.global src
src:
.word 0x11223344

.section .text.func, "ax"
.balign 4
.global func
.type func, %function
func:
    adr r0, src
    bx lr
```

This defect is associated with the issues described in [SDCOMP-64165](#) and [SDCOMP-55580](#).

Conditions

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

- The program is assembled for a big-endian target.
- The program contains an instruction `ℐ`.
- `ℐ` is one of the following:
 - `ADR`
 - `LDR`
 - `LDRB`
 - `LDRH`
 - `LDRSB`
 - `LDRSH`
- `ℐ` specifies a label operand `ℒ`.
- `ℐ` is in a section `ℳ`.
- `ℒ` is in a section `ℴ`.
- `ℳ` and `ℴ` are different.

2.3.1.24 SDCOMP-62791

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-62791.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch32 state	6.16.1, 6.16.2	-

Description

The compiler can generate code that incorrectly does not preserve the half-precision floating-point value `v` returned by one function when making a call to another function `ℱ`. This can result in unexpected run-time behavior if `ℱ` corrupts the floating-point register containing `v`.

For example, when the following code is compiled with `-mfloat-abi=hard -O1` for an Armv8.2-A target with half-precision floating-point support:

```
extern __fp16 func1(void);
extern void func2(void);

__fp16 func3(void)
{
    __fp16 return_value = func1();
    func2();

    return return_value;
}
```

the compiler incorrectly generates the following code where `func2()` could potentially corrupt the value returned by `func1()` and cause `func3()` to return an invalid value:

```
func3:
    push    {r11, lr}
    bl      func1
    bl      func2
    pop     {r11, pc}
```

Instead, the compiler should preserve `s0` before the call to `func2()` and restore `s0` after the call to `func2()`. For example:

```
func3:
    push    {r11, lr}
    vpush   {d8}
    bl      func1
    vmov.f32    s16, s0
    bl      func2
    vmov.f32    s0, s16
    vpop      {d8}
    pop       {r11, pc}
```

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 half-precision floating-point support.
- The program contains a function that returns a value of type `T`.
- `T` is one of the following:
 - `__bf16`
 - `_Float16`
 - `__fp16`

2.3.1.25 SDCOMP-62769

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-62769.



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	6.16.1, 6.16.2	-

Description

The compiler can incorrectly convert a half-precision floating-point signalling NaN value to a quiet NaN value.

For example, the compiler incorrectly converts the compile-time constant 0x7c01, which represents a signalling NaN value, to the quiet NaN value 0x7e01 for the following:

```
void func(__fp16 *ptr)
{
    union {
        unsigned short s;
        __fp16 f;
    } u;
    u.s = 0x7c01;
    *ptr = u.f;
}
```

Conditions

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

- The program is compiled for a target without half-precision floating-point support.
- The program is compiled with -ffp-mode=full.
- The program contains a variable `A` of `__fp16` type.
- A signalling NaN value `B` is stored in `A`.
- `B` is a compile-time constant.

2.3.1.26 SDCOMP-62725

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-62725.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Armv8-M with the Main Extension	6.16.1, 6.16.2	-

Description

The compiler generates code that incorrectly does not conform to the *Procedure Call Standard for the Arm Architecture*, when compiling for a target that supports integer MVE only.

For example, the compiler generates code that incorrectly uses the general-purpose register `R0` for the return value of `func()` in the following:

```
float func(float *ptr)
{
    return *ptr;
}
```

The compiler should use the floating-point register `s0` instead.

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 for a target with the M-profile Vector Extension (MVE).
- The program is compiled for a target that supports integer MVE.
- The program is compiled for a target that does not support floating-point MVE.
- The program is compiled for a target without hardware floating-point support.
- The program is compiled with `-mfloat-abi=hard`.
- The program contains a variable of floating-point type.

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

2.3.1.27 SDCOMP-62692

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-62692.



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	Armv7-A, Armv7-M, Armv7-R, Armv8-A, Armv8-M with the Main Extension, Armv8-R	6.16.1, 6.16.2	-

Description

The compiler can generate code that incorrectly performs an unaligned access using a `LDRD` or `STRD` instruction for an access to a **struct**.

For example, the compiler incorrectly uses an `STRD` instruction to access `dst.y` in the following:

```
typedef struct __attribute__((packed)) {
    char x;
    volatile long long y;
} T;

T dst;

void func(long long y)
```

```
{
    dst.y = y;
}
```

Conditions

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

- The program is compiled for AArch32 state.
- The program contains `x`, where `x` is one of the following:
 - A member of a **volatile struct** `s`.
 - A **volatile** member of a **struct** `s`.
- `s` is greater than 8 bytes in size.
- The program accesses `x`.

2.3.1.28 SDCOMP-62661

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-62661.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.16.1, 6.16.2	-

Description

The compiler can generate incorrect code for a call to a variadic function. 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 contains a call `x` to a function `F`.
- `F` accepts a variadic arguments list `L`.
- `x` specifies an argument `A` as part of `L`.
- `A` is a **struct** `s`.
- `s` is annotated with `attribute((packed))`.
- `s` has a member that is an aggregate type. For example, `uint16x8_t`.

2.3.1.29 SDCOMP-62378

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



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	6.16.1, 6.16.2	-

Description

The compiler incorrectly generates a Neon load or store instruction that incorrectly 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.

Conditions

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

- The program contains a call to a Neon intrinsic `1` that is defined in the `<arm_neon.h>` system header.
- `1` has one of the following forms:
 - `vld*_x2()`
 - `vld*_x3()`
 - `vld*_x4()`

- `vst*_x2()`
- `vst*_x3()`
- `vst*_x4()`
- `1` 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.

2.3.1.30 SDCOMP-62352

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-62352.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Armv8-M with the Main Extension	6.16.1, 6.16.2	-

Description

The compiler and integrated assembler can incorrectly generate a `VMLA.U32` or `VMLAS.U32` instruction in accordance with revision B.q of the Armv8-M Architecture. Instead, the compiler and integrated assembler should generate a `VMLA.I32` or `VMLAS.I32` instruction in accordance with revision B.r of the Armv8-M Architecture.

To detect this issue, disassemble the output ELF file with `--cpu=8.1-M.Main.mve --text -c` and manually inspect the output for the following instructions:

- `VMLA.U32`
- `VMLAS.U32`

Conditions

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

- The program is compiled for a target that supports the M-profile Vector Extension (MVE).
- The program is run on a target that implements revision B.r or later of the Armv8-M Architecture.

2.3.1.31 SDCOMP-62330

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-62330.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.16.1, 6.16.2	-

Description

The compiler can generate incorrect debug information for a bit-field.

For example, the compiler generates incorrect debug information for `b` in the following:

```
struct __attribute__((packed)) S
{
    unsigned char a: 7;
    unsigned char b: 4; /* Packing places bit0 of b
                        in a different byte to bits1-3 */
};
```

This defect is associated with the issue described in [SDCOMP-65172](#).

Conditions

This defect can occur when all the following are true:

- The program is compiled with `-g` or `-gdwarf-4`.
- The program contains a bit-field `b`.
- One of the following is annotated with `attribute((packed))`:
 - `B`.
 - The **struct** containing `b`.
- Packing affects the number of bytes that `b` spans.

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.

2.3.1.32 SDCOMP-62251

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-62251.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armlink	AArch64 state	6.16.1, 6.16.2	-

Description

The linker can incorrectly calculate too small a value for the `p_memsz` field of the ELF program header describing a load region.

Subsequently, an ELF processing tool that creates the execution view directly from the ELF program headers might not be able to process the ELF file correctly. For example, an ELF processing tool might:

- Reject the ELF file as invalid.
- Not allocate enough memory for the program segment. This can result in unexpected run-time behavior.

Conditions

This defect occurs when one of the following is true:

- All the following are true:
 - The program is not linked with a scatter file.
 - The program contains a section that has an alignment requirement greater than 4 bytes.
 - The output image contains a non-contiguous execution region.
- All the following are true:
 - The program is linked with a scatter file F .
 - F contains a load region L .
 - L contains two execution regions A and B .
 - A and B are non-contiguous.
 - B has an alignment requirement greater than 4 bytes.

To detect if the safety-related system is at risk, link with `--map` and manually inspect the output. The safety-related system is only 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 memory map shows a non-contiguous execution region.

2.3.1.33 SDCOMP-62221

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-62221.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Armv6-M	6.16.1, 6.16.2	-

Description

The compiler can incorrectly generate a `B.W` instruction.

To detect this issue, disassemble the output ELF file without `--cpu=<name>`, with `--text -c`, and manually inspect the output for the following instruction:

- B.W

Conditions

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

- The program is compiled for an Armv6-M target.
- The program is run on an Armv6-M target.

Note: Arm has not observed this defect being triggered in a program built from C or C++ source code with any of the affected releases. The underlying defect is fixed as of Arm Compiler for Embedded FuSa 6.16.3.

2.3.1.34 SDCOMP-62217

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-62217.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
fromelf	Armv8-M with the Main Extension	6.16.1, 6.16.2	-

Description

The `fromelf` utility incorrectly disassembles the label operand of a `LE` or `LETP` instruction as an immediate offset instead of a PC-relative offset.

For example, the `fromelf` utility incorrectly disassembles the following `LETP` instruction:

```
letp lr, <label>
```

as:

```
letp lr, #<offset>
```

instead of:

```
letp lr, {pc}-<offset>
```

This defect is associated with the issue described in [SDCOMP-63738](#).

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 `LE` or `LETP` 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.

2.3.1.35 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	6.16.1, 6.16.2	-

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.



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	AArch64 state	6.16.1, 6.16.2	-

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 `-O0`.
- 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-62123

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-62123.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch32 state	6.16.1, 6.16.2	-

Description

The compiler can generate incorrect code for a `vdup_*f16()` Neon intrinsic.

Conditions

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

- The program is compiled for a target with half-precision floating-point support.
- The program contains a call `z` to a Neon intrinsic `i` that is defined in the `<arm_neon.h>` system header.
- `i` is of the form `vdup_*f16()`.
- The behavior of the program depends on `z`.

2.3.1.38 SDCOMP-62028

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-62028.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Armv8-M without the Main Extension	6.16.1, 6.16.2	-

Description

The compiler can generate incorrect code for an atomic compare exchange built-in or an atomic compare exchange function.

Conditions

The safety-related system is at risk when the program contains a call to one of the following:

- An `__atomic_compare_exchange*()` built-in.
- An `atomic_compare_exchange_*()` function defined in the `<stdatomic.h>` Arm C library header.
- A `compare_exchange_*()` member function of the class template `std::atomic<T>`.
- A `std::atomic_compare_exchange_*()` function defined in the `<atomic>` Arm C++ library header.

2.3.1.39 SDCOMP-61514

This section describes the scope of the translation 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	6.16.1, 6.16.2	-

Description

The `poly8_t`, `poly16_t`, and `poly64_t` types are incorrectly defined as **signed** instead of **unsigned** in the `<arm_neon.h>` system header. This can result in unexpected run-time behavior.

Conditions

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

- The program contains a variable `v` of type `A` or type `B`, where:
 - `A` is one of the following types defined in the `<arm_neon.h>` system header:
 - `poly8_t`
 - `poly16_t`
 - `poly64_t`
 - `B` is derived from `A`.
- The behavior of the program depends on `v` being **unsigned**.

2.3.1.40 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	6.16.1, 6.16.2	-

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 .

2.3.1.41 SDCOMP-61299

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-61299.



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	6.16.1, 6.16.2	-

Description

The compiler can incorrectly set bit 3 of the `__ARM_FP` predefined macro when compiling with an `-march` or `-mcpu` option that specifies `+nofp.dp`.

For example, the compiler incorrectly sets bit 3 of the `__ARM_FP` predefined macro when compiling with `-mcpu=cortex-m7+nofp.dp`.

Conditions

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

- The program is compiled with an `-march` or `-mcpu` option x .
- x specifies `+nofp.dp`.
- The behavior of the program depends on bit 3 of `__ARM_FP`.

2.3.1.42 SDCOMP-61298

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-61298.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch32 state	6.16.1, 6.16.2	-

Description

The compiler incorrectly fails to define the `SOFTFP` predefined macro.

This defect is associated with the issue described in [SDCOMP-61465](#).

Conditions

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

- The program is compiled for a target without hardware floating-point support.
- One of the following is true:
 - The program is compiled without `-mfloat-abi=<value>`.
 - The program is compiled with `-mfloat-abi=softfp`.
- The behavior of the program depends on `SOFTFP`.

2.3.1.43 SDCOMP-61150

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-61150.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armlink	AArch32 state	6.16.1, 6.16.2	-

Description

The linker can generate an image which decompresses an execution region that contains RW data incorrectly.

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

- Link with `--datacompressor=off`.
- Add the `NOCOMPRESS` attribute to each affected execution region.
- Add the `NOCOMPRESS` attribute to each load region that contains an affected execution region.

Conditions

This defect can occur when all the following are true:

- The program is not linked with `--datacompressor=off`.
- The program contains an execution region `E`.
- `E` contains RW data.
- `E` does not have any of the following attributes:

- NOCOMPRESS
- PI
- RELOC

To detect if the safety-related system is at risk, link with `--diag_warning=6703`, `--map`, and `--load_addr_map_info`, and manually inspect the output. The safety-related system is only at risk when all the following are true:

- The linker reports `L6703W: Section <irrelevant> implicitly marked as non-compressible`.
- The memory map shows an execution region for which all the following are true:
 - The `Load addr` value is `COMPRESSED`.
 - The `Exec base` and `Load base` values are the same.

2.3.1.44 SDCOMP-61080

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-61080.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.16.1, 6.16.2	-

Description

The compiler can generate incorrect code for an expression that involves both the `>>` and `<<` operators.

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

```
(var >> 8 | var << 8)
```

Conditions

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

- The program is not compiled for AArch64 state at `-O0`.
- The program contains an expression `E`.
- `E` involves both the `>>` and `<<` operators.
- The result of `E` is cast to a 16-bit type.

2.3.1.45 SDCOMP-60897

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-60897.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Armv8-M	6.16.2	6.16.1

Description

The compiler incorrectly fails to define certain predefined macros when the toolchain is used with user-based licensing.

Conditions

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

- The toolchain is used with user-based licensing.
- The program is compiled with `-mcmse`.
- The behavior of the program depends on one of the following predefined macros:
 - `ARM_ARCH_EXT_IDIV`
 - `__ARM_FEATURE_DSP`
 - `__ARM_FEATURE_FMA`
 - `__ARM_FEATURE_FP16_SCALAR_ARITHMETIC`
 - `__ARM_FEATURE_IDIV`
 - `__ARM_FEATURE_LDREX`
 - `__ARM_FEATURE_MVE`
 - `__ARM_FEATURE_SIMD32`
 - `__ARM_FP`
 - `ARM_FPV5`
 - `ARM_VFPV2`
 - `ARM_VFPV3`
 - `ARM_VFPV4`
 - `VFP_FP`

For more information about user-based licensing, refer to <https://lm.arm.com>.

2.3.1.46 SDCOMP-60725

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-60725.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
fromelf	Armv8-M with the Main Extension	6.16.1, 6.16.2	-

Description

The `fromelf` utility disassembles `CLRM` instructions incorrectly.

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

```
clrm {r0-r12}
```

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 `CLRM` 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.

2.3.1.47 SDCOMP-60659

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-60659.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armlink	Any	6.16.1, 6.16.2	-

Description

The linker incorrectly places all thread-local variables at the same offset in a dynamic symbol table.

For example, the linker incorrectly places both `var1` and `var2` at offset 0 when creating a shared library containing the following thread-local variables:

```
__thread int var1;
```

```
__thread int var2;
```

The incorrect offset value can be seen in the `.dynsym` section of the shared library:

Symbol table .dynsym (3 symbols, 0 local)							
#	Symbol Name	Value	Bind	Sec	Type	Vis	Size
=====							
1	var1	0x00000000	Gb	5	TLS	De	0x4
2	var2	0x00000000	Gb	5	TLS	De	0x4

Conditions

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

- The program is compiled with `-fpic`.
- The program contains multiple thread-local variables.
- The program is linked with `--shared --sysv`.

2.3.1.48 SDCOMP-60632

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-60632.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.16.1, 6.16.2	-

Description

The inline assembler and integrated assembler can incorrectly generate a `FNMADD` instruction instead of a `NOP` instruction for a `.align`, `.balign`, or `.p2align` assembly directive.

Conditions

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

- The program is compiled with `-mbig-endian`.
- The program contains a `.align`, `.balign`, or `.p2align` directive `D`.
- `D` does not have a `w` or `l` width suffix.
- `D` does not specify a fill value.

2.3.1.49 SDCOMP-60589

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-60589.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.16.1, 6.16.2	-

Description

The compiler generates code that incorrectly does not conform to the *Procedure Call Standard for the Arm Architecture* for a function with a prototype that has a **struct** containing a zero-length bit-field.

For example, the compiler generates code that incorrectly does not conform to the *Procedure Call Standard for the Arm Architecture* for the following:

```
struct S {
    float a;
    int : 0;
    float b;
};

struct S func(struct S x)
{
    x.b += 1.0f;

    return x;
}
```

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 is compiled for one of the following:
 - AArch64 state.
 - AArch32 state with `-mfloat-abi=hard`.
- The program contains a **struct** `s`.
- `s` contains a zero-length bit-field.
- All other members of `s` are of the same floating-point type and size.
- The program contains a function `F`.
- One of the following is true:
 - `F` has a parameter of `s` type.
 - `F` returns a value of `s` type.

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

- `F` is defined in a source file `A`.
- `F` is called from a source file `B`.
- One of the following is true:
 - `A` is compiled in a C source language mode and `B` is not.
 - `B` is compiled in a C source language mode and `A` is not.
 - `A` and `B` are not built using the same toolchain.

2.3.1.50 SDCOMP-60443

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-60443.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch32 state	6.16.2	6.16.1

Description

The compiler can generate incorrect debug information for a global variable that is `const`.

For example, when compiling with `-frwpi`, the compiler can generate incorrect debug information for `var` in the following:

```
const int var = 5;
```

This defect is associated with the issue described in [SDCOMP-59059](#).

Conditions

This defect can occur when all the following are true:

- The program is compiled with `-frwpi`.
- The program is compiled with `-g` OR `-gdwarf-version`.
- The program contains a global variable that is `const`.

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.

2.3.1.51 SDCOMP-60342

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-60342.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.16.1, 6.16.2	-

Description

The compiler can generate incorrect code for a call to a formatted input/output function declared in the `<stdio.h>` header or a formatted wide character input/output function declared in the `<wchar.h>` header, when the format string is a global variable.

For example, the compiler generates incorrect code for the call the `printf()` function in the following:

```
#include <stdio.h>
char format_string[] = "%d\n";
int main(void)
{
    format_string[1] = 'f';
    printf(format_string, 1.5);
    return 0;
}
```

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 without `-nostdlib`.
- The program contains a call `z` to one of the following:
 - A formatted input/output function declared in the `<stdio.h>` header.
 - A formatted wide character input/output function declared in the `<wchar.h>` header.
- The format string argument of `z` is a global variable `v`.
- The value of `v` is modified at run-time.
- The behavior of the program depends on `z`.

2.3.1.52 SDCOMP-60326

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-60326.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
fromelf	Any	6.16.1, 6.16.2	-

Description

The `fromelf` utility reports incorrect `DW_CFA_def_cfa` and `DW_CFA_def_cfa_offset` entries for stack frame unwinding debug information.

Conditions

The `fromelf` utility reports 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 of the following sections:
 - `.debug_frame`
 - `.ehframe`
- `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.

2.3.1.53 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	6.16.1, 6.16.2	-

Description

The linker can generate incorrect stack usage information.

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

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.

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

2.3.1.54 SDCOMP-59974

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-59974.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.16.1	6.16.2

Description

The compiler can generate incorrect code for a nested loop.

Conditions

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

- The program is compiled at any optimization level except `-o0` and `-o1`.
- The program is compiled with `-fwrapv`.
- The program contains a nested loop `A`.
- `A` contains a nested loop `B`.
- The behavior of the program depends on `A` or `B`.

2.3.1.55 SDCOMP-59938

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-59938.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armlink	AArch64 state	6.16.1, 6.16.2	-

Description

The linker can incorrectly fail to account for calls to **static** functions when generating a callgraph or stack usage information.

Conditions

This defect can occur when all the following are true:

- The program contains a **static** function.
- 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 output causes you to manually make an incorrect change to the safety-related system.

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

2.3.1.56 SDCOMP-59788

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-59788.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Armv7-M, Armv8-M	6.16.1, 6.16.2	-

Description

The compiler can incorrectly generate one of the following instructions when accessing an element of a **char** or **short** array:

- `LDRBT`
- `LDRHT`
- `LDRSBT`
- `LDRSHT`
- `STRBT`
- `STRHT`

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 an access `A` to an element of a `char` or `short` array `B`.
- `B` is in a memory region that does not permit unprivileged access.
- One of the following is true when `A` is performed:
 - The core is in Handler mode.
 - The core is in Thread mode, and `CONTROL.NPRIV` is set to zero.
- The behavior of the program depends on `A`.

2.3.1.57 SDCOMP-59656

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-59656.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.16.1	6.16.2

Description

The compiler can generate incorrect code for an integer literal that can be represented by fewer than 64 bits and is cast to a pointer type.

To avoid this issue, ensure that all integer literals which are cast to a pointer type have a suffix that ends with `L` or `UL`. For example, `0x8000L`.

Conditions

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

- The program is compiled at `-O0`.
- The program contains a cast `c` from an integer literal `i` to a pointer type.
- `i` can be represented by fewer than 64 bits.
- `i` does not have a `L`, `UL`, `l`, or `ll` suffix.
- The behavior of the program depends on `c`.

2.3.1.58 SDCOMP-59521

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-59521.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.16.1, 6.16.2	-

Description

The compiler can generate incorrect code for a call to an empty user-defined implementation of **operator delete**.

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 is compiled at any optimization level except `-O0`.
- The program contains a user-defined implementation \mathbb{I} of **operator delete**.
- \mathbb{I} is an empty function \mathbb{F} .
- The behavior of the program depends on \mathbb{F} returning.

2.3.1.59 SDCOMP-59074

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-59074.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.16.1	6.16.2

Description

The compiler can generate incorrect code for a comparison operation that always evaluates to **true**.

Additionally, the compiler correctly reports the following warning:

- `comparison of constant <constant> with expression of type <type> is always true`

This warning can be upgraded to an error by compiling with `-Werror=tautological-constant-out-of-range-compare`.

Conditions

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

- The program is compiled at `-O0`.
- The program is not compiled with `-Werror=tautological-constant-out-of-range-compare`.
- The program contains a comparison operation c .
- c compares \mathbb{A} and \mathbb{B} .

- `a` is a constant of type `x`.
- `b` is a variable of an integer type `y`.
- The size of `x` is larger than the size of `y`.
- `c` always evaluates to `true`.
- The behavior of the program depends on `c`.

2.3.1.60 SDCOMP-59059

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-59059.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch32 state	6.16.1	6.16.2

Description

The compiler can generate incorrect debug information for a global variable that is not `const`.

For example, when compiling with `-frwpi`, the compiler can generate incorrect debug information for `var` in the following:

```
int var = 5;
```

This defect is associated with the issue described in [SDCOMP-60443](#).

Conditions

This defect can occur when all the following are true:

- The program is compiled with `-frwpi`.
- The program is compiled with `-g` or `-gdwarf-version`.
- The program contains a global variable that is not `const`.

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.

2.3.1.61 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	6.16.1, 6.16.2	-

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;
    }
}
```

This defect is associated with the issue described in [SDCOMP-57884](#).

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 .

2.3.1.62 SDCOMP-58773

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-58773.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.16.1	6.16.2

Description

The compiler generates incorrect code for a function that contains an inline assembly statement with a single-copy atomic 64-byte load or store instruction.

For example, the compiler generates code that incorrectly fails to initialize `data` in the following:

```
typedef struct
{
    unsigned long long x[8];
} T;

void func(int *x, void *y)
{
    T data = { x[0], x[1], x[2], x[3], x[4], x[5], x[6], x[7] };
    __asm volatile("st64b %0, [%1]" : : "r" (data), "r" (y) : "memory");
}
```

Conditions

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

- The program is compiled for a target with the Accelerator Support Extension.
- The program is compiled at any optimization level except `-O0`.
- The program contains a function `F`.
- `F` contains an inline assembly statement with one of the following instructions:
 - `LD64B`
 - `ST64B`
 - `ST64BV`
 - `ST64BV0`
- The behavior of the program depends on `F`.

2.3.1.63 SDCOMP-58738

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-58738.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.16.1	6.16.2

Description

The compiler can generate incorrect code for a loop that contains accesses to an array through pointers that overlap each other.

Conditions

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

- The program is compiled for a target that supports SIMD instructions.
- The program is compiled at `-O3`, `-Ofast`, `-Omax`, or with `-fvectorize`.
- The program contains a loop `L`.
- `L` accesses an array through two pointers `A` and `B`.
- An iteration of `L` accesses an address `x` through `A`.
- A different iteration of `L` accesses `x` through `B`.

2.3.1.64 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	6.16.1, 6.16.2	-

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.

2.3.1.65 SDCOMP-57884

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-57884.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.16.1	6.16.2

Description

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

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

```
void no_init_array(int len)
{
    (void)new (std::nothrow) int[len];
}

void init_array(int len)
{
    (void)new (std::nothrow) int[len]{1,2,3};
}

void test(void)
{
    try
    {
        no_init_array(-1); /* Test negative array length */
    }
    catch (const std::bad_array_new_length &)
    {
        std::cout << "Exception caught" << std::endl;
    }
    try
    {
        init_array(1); /* Test too many initializers */
    }
    catch (const std::bad_array_new_length &)
    {
        std::cout << "Exception caught" << std::endl;
    }
}
```

This defect is associated with the issue described in [SDCOMP-58780](#).

Conditions

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

- The program is written in a C++ source language mode.
- The program is compiled with C++ exceptions enabled.
- The program contains a **new** (`std::nothrow`) expression E.

- One of the following is true:
 - `E` specifies a negative array length.
 - `E` specifies more initializers than the array length.
- The behavior of the program depends on a `std::bad_array_new_length` exception not being raised for `E`.

2.3.1.66 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	6.16.1, 6.16.2	-

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 program is compiled with `-gdwarf-2` OR `-gdwarf-3`.
- The program is compiled at any optimization level except `-O0`.
- The program contains a **static** variable of type `T`.
- The size of `T` is larger than 1 byte.

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.

2.3.1.67 SDCOMP-57674

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-57674.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.16.1, 6.16.2	-

Description

The compiler can incorrectly fail to flush a denormal floating-point number to zero.

Conditions

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

- The program is compiled at any optimization level except -o0.
- One of the following is true:
 - The program is not compiled at -ofast or -Omax.
 - The program is compiled with one of the following:
 - -ffast-math
 - -ffp-mode=fast
 - -ffp-mode=std
- The program contains a calculation that involves a denormal floating-point number *x*.
- The behavior of the program depends on *x* being flushed to zero.

2.3.1.68 SDCOMP-57456

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-57456.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
fromelf	AArch64 state	6.16.1, 6.16.2	-

Description

The `fromelf` utility disassembles `movi` and `mvni` instructions incorrectly.

For example, the `fromelf` utility incorrectly disassembles the following valid instructions incorrectly:

```
movi v0.2s, #0x24, msl #16
mvni v0.2s, #0x24, msl #16
```

Conditions

This defect occurs when the program contains a `movi` or `mvni` 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.

2.3.1.69 SDCOMP-57449

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-57449.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
fromelf	AArch64 state	6.16.1, 6.16.2	-

Description

The `fromelf` utility disassembles immediate variants of the SVE `AND`, `BIC`, `EON`, and `EOR` instructions incorrectly.

For example, the `fromelf` utility incorrectly disassembles the following valid `AND` instruction as an invalid `AND` instruction with an immediate operand of `0x10000` instead of `0x1`:

```
and z9.h, z9.h, #0x1
```

Conditions

This defect occurs when the program contains an immediate variant of one of the following SVE instructions:

- `AND`
- `BIC`
- `EON`
- `EOR`

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.

2.3.1.70 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	6.16.1, 6.16.2	-

Description

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

2.3.1.71 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	6.16.1, 6.16.2	-

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.

2.3.1.72 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	6.16.1, 6.16.2	-

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.

2.3.1.73 SDCOMP-57200

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-57200.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch32 state	6.16.1, 6.16.2	-

Description

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

For example, the integrated assembly incorrectly sets the alignment to 1 byte for both `.text.func1` and `.text.func2` in the following:

```
.section .text.func1, "ax"
.arm
nop

.section .text.func2, "ax"
.thumb
nop
```

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

State	Alignment directive
A32	<code>.p2align 2</code>
T32	<code>.p2align 1</code>

This defect is associated with the issue described in [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 explicitly contain one of the following directives:
 - `.align`
 - `.balign`
 - `.p2align`

2.3.1.74 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	6.16.1, 6.16.2	-

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.
- 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.

2.3.1.75 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	6.16.1, 6.16.2	-

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.

2.3.1.76 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	6.16.1, 6.16.2	-

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.

2.3.1.77 SDCOMP-54546

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-54546.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
fromelf	Any	6.16.1, 6.16.2	-

Description

When processing an image which contains debug information 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 debug information.
- `F` contains a `.debug_loc` section `L`.
- `L` contains a base address 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.

2.3.1.78 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	6.16.1, 6.16.2	-

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.

2.3.1.79 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	6.16.1, 6.16.2	-

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.

2.3.1.80 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	6.16.1, 6.16.2	-

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.

- \mathbb{F} contains a global or **static** variable v .
- v contains a bit-field.
- The global and static data addresses in \mathbb{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.

2.3.1.81 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	6.16.1, 6.16.2	-

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 \mathbb{P} .
- \mathbb{P} is within the A64 System Instruction Class encoding space.
- \mathbb{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.

2.3.1.82 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	6.16.1, 6.16.2	-

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.

2.3.1.83 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	6.16.1, 6.16.2	-

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 6.16LTS Qualification Kit Safety Manual*.

2.3.2.1 SDCOMP-65264

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-65264.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.16.1, 6.16.2	-

Description

The compiler and integrated assembler incorrectly fail to report the following error for a 32-bit element `FMMLA` instruction:

- `instruction requires: f32mm`

Subsequently, this can result in unexpected run-time behavior when a 32-bit element `FMMLA` instruction is executed on a target that does not support the Single-precision Matrix Multiplication feature (FEAT_F32MM).

To avoid this issue, specify the `+noF32mm` feature modifier when building for an affected target that does not support FEAT_F32MM.

Conditions

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

- The program is built for an Armv8.6-A or later target with the Scalable Vector Extension (SVE).
- The program is built with an `-march` or `-mcpu` option that does not specify the `+F32mm` feature modifier.
- The program contains a 32-bit element `FMMLA` instruction.
- The program is run on a target that does not support the FEAT_F32MM feature.

2.3.2.2 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	6.16.1, 6.16.2	-

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 `1`.
- `1` has an invalid predication specifier operand.

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

2.3.2.3 SDCOMP-64683

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



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	Armv7-A, Armv7-M, Armv7-R, Armv8-A, Armv8-M with the Main Extension, Armv8-R	6.16.1, 6.16.2	-

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`.

- `ℓ` is one of the following:
 - `LDRD` (literal)
 - `VLDR` (literal)
- `ℓ` 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 `ℓ`.

2.3.2.4 SDCOMP-64255

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



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	6.16.1, 6.16.2	-

Description

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

- `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 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 `ℓ`.
- `ℓ` has an invalid operand.
- The behavior of the system depends on `ℓ` being executed.

2.3.2.5 SDCOMP-63917

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-63917.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.16.1, 6.16.2	-

Description

The inline assembler and integrated assembler incorrectly fail to report an error for a `FMLAL` (by element) or `FMLAL2` (by element) instruction that specifies an invalid second source register of the form `<Vm>.H[<index>]`.

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

```
fmlal    v0.4s, v1.4h, v24.h[0]
fmlal2   v0.4s, v1.4h, v24.h[0]
```

Instead, the inline assembler and integrated assembler generate instructions that use the register `v8` instead of `v24`:

```
fmlal    v0.4s, v1.4h, v8.h[0]
fmlal2   v0.4s, v1.4h, v8.h[0]
```

This can result in unexpected run-time behavior.

This defect is associated with the issue described in [SDCOMP-63752](#).

Conditions

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

- The program contains an instruction `ℓ`.
- `ℓ` is one of the following:
 - `FMLAL`
 - `FMLAL2`
- The second source register operand of `ℓ` is of the form `<Vm>.H[<index>]`.
- `<Vm>` is outside the range `v0-v15`.
- The behavior of the program depends on `ℓ`.

2.3.2.6 SDCOMP-63697

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-63697.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.16.1, 6.16.2	-

Description

The compiler can incorrectly generate an output file and exit with a return code of 0 despite reporting an error. The return code is also referred to as the status code.

For example, when the following file `example.c` is compiled with `--target=arm-arm-none-eabi -mcpu=cortex-m4 -c example.c -o example.o`:

```
int __attribute__((section("var"))) var;
```

the compiler correctly reports the following errors:

```
var changed binding to STB_GLOBAL
invalid symbol redefinition
```

However, it incorrectly generates an output file `example.o` and exits with a return code of 0.

Conditions

The safety-related system is at risk when the build system depends on one of the following:

- The compiler always deleting the output file when an error is reported.
- The compiler always exiting with a non-zero return code when an error is reported.

To avoid this issue, manually inspect the messages reported by the compiler and ensure that the outputs of compiler invocations that report errors are not used.

2.3.2.7 SDCOMP-62234

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-62234.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.16.1, 6.16.2	-

Description

The compiler incorrectly fails to report a warning for an invalid `#undef` preprocessor macro.

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

```
#undef __STDC__
```

Conditions

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

- The program contains an `#undef` preprocessor macro which specifies a name `N` that is lexically identical to a predefined macro name as listed in the *Predefined macro names* section of the relevant C standard specification.
- The behavior of the program depends on `N`.

2.3.2.8 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	AArch64 state	6.16.1, 6.16.2	-

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, 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;
}
```

Conditions

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

- 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.

2.3.2.9 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	6.16.1, 6.16.2	-

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.
- `F` contains an instruction `I`.
- `I` is not compatible with `A`.

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

2.3.2.10 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	6.16.1, 6.16.2	-

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`.
- A and B are incompatible.

2.3.2.11 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	6.16.1, 6.16.2	-

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

2.3.2.12 SDCOMP-61089

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-61089.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch32 state	6.16.1, 6.16.2	-

Description

When compiling with `-mfloat-abi=hard`, the compiler defines the predefined macro `__ARM_PCS_VFP`.

The compiler incorrectly fails to report the following warning when compiling with `-mfloat-abi=hard` for a target that does not support hardware floating-point instructions:

- `'-mfloat-abi=hard': selected processor lacks floating point registers`

Conditions

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

- The program is compiled for a target that does not support hardware floating-point instructions.
- The program is compiled with `-mfloat-abi=hard`.
- The behavior of the program depends on `__ARM_PCS_VFP`.

2.3.2.13 SDCOMP-59605

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-59605.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.16.1	6.16.2

Description

The compiler can incorrectly fail to report a warning for a C++ class or class data member that is annotated with `attribute((packed))` or `#pragma pack`.

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 is compiled with `-mno-unaligned-access`.
- The program contains `x`, where `x` is one of the following:

- A class.
- A data member of a class.
- x is annotated with `attribute((packed))` Or `#pragma pack`.
- The behavior of the program depends on x being accessed using aligned accesses.

2.3.2.14 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	6.16.1, 6.16.2	-

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 τ in a namespace \mathbf{B} .
- \mathbf{A} and \mathbf{B} are not the same.
- The behavior of the program depends on τ being used.

2.3.2.15 SDCOMP-59190

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-59190.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.16.1, 6.16.2	-

Description

The compiler can incorrectly fail to report the following error:

- reference to '<name>' is ambiguous

For example, the compiler incorrectly fails to report an error for the ambiguous reference to $\mathbf{N}()$ from `func()` in the following:

```
namespace A {
    struct N {
        operator int() { return 0; }
    };
}
namespace B {
    int N() { return 1; }
}
namespace C {
    using A::N;
    using B::N;
}
namespace D {
    using A::N;
}
using namespace C;
using namespace D;

bool func(void) {
    return N() == 1;
}
```

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 four namespaces, \mathbf{A} , \mathbf{B} , \mathbf{C} , and \mathbf{D} .
- \mathbf{A} and \mathbf{B} each declare a member with the same name, \mathbf{N} .
- $\mathbf{A}::\mathbf{N}$ and $\mathbf{B}::\mathbf{N}$ declare different entities.

- `c` contains a using-declaration for `A::N`.
- `D` contains a using-declaration for `B::N`.
- The program contains using-directives for `c` and `D`.
- The program contains an unqualified reference to `N`.
- The behavior of the program depends on one of `A::N` or `B::N`.

2.3.2.16 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	6.16.1, 6.16.2	-

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.

2.3.2.17 SDCOMP-57912

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-57912.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.16.1, 6.16.2	-

Description

The compiler can incorrectly fail to report the following error for a `--target=<triple>` option that specifies an unsupported `<triple>`:

- `'--target=<triple>'` is not recognized. Supported values for this option are `'arm-arm-none-eabi'` or `'aarch64-arm-none-eabi'`

Conditions

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

- The program is compiled with `--target=<triple>`.
- `<triple>` is not one of the following:
 - `aarch64-arm-none-eabi`
 - `arm-arm-none-eabi`
- The behavior of the program depends on the program being built for a target supported by Arm Compiler.

2.3.2.18 SDCOMP-57528

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-57528.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.16.1, 6.16.2	-

Description

The inline assembler and integrated assembler can incorrectly fail to report an error for a conditional 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.ne . + 1048576 // An A64 B.cond instruction has the range
                // -1048576 to 1048572
```

Instead, the integrated assembler incorrectly encodes the instruction as:

```
b.ne . - 1048576
```

This defect is associated with the issue described in [SDCOMP-55983](#).

Conditions

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

- The program contains a conditional branch instruction `1` with a destination `D`.

- `D` has one of the following forms:
 - `. + <offset>`
 - `<label> + <offset>`
- `<offset>` is an immediate value provided in the source code.
- `<offset>` is in the range `[-2097152, -1048580]` or `[1048576, 2097148]`.
- The behavior of the program depends on `I` branching to `D`.

2.3.2.19 SDCOMP-57199

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-57199.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armlink	AArch32 state	6.16.1, 6.16.2	-

Description

The linker can incorrectly fail to report a warning for a program that contains a T32 `BL` instruction which branches to an A32 instruction that is not aligned to a 4-byte boundary.

To avoid this issue, manually inspect assembly language source files, and ensure that all A32 code symbols are aligned to a 4-byte boundary.

Conditions

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

- The program contains an A32 instruction `I` at address `A`.
- `A` is not a multiple of 4 bytes.
- The program contains a T32 `BL` instruction that branches to `I`.

You may be able to detect if the safety-related system is at risk using the following methods:

- Link with `--diag_error=L6786W`. The linker reports `L6786W: Mapping symbol #<number> '<symbol>' in <section>(<object>) defined at unaligned offset=<offset> for an affected symbol that is not aligned to a 4-byte boundary.`
- Disassemble the program with `fromelf --text -c`, and manually inspect the output. The output from the `fromelf` utility may contain `<N> bytes skipped because they could not be disassembled for an affected ELF file.`

2.3.2.20 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	6.16.1, 6.16.2	-

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`.

2.3.2.21 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	6.16.1, 6.16.2	-

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 `ι`.
- `ι` specifies one of the following as a general-purpose register operand `ℝ`:
 - `w31`
 - `x31`
- The behavior of the program depends on `ι` accessing `ℝ` as a general-purpose register instead of as the zero register.

2.3.2.22 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	6.16.1, 6.16.2	-

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.

2.3.2.23 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	6.16.1, 6.16.2	-

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`.

2.3.2.24 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	6.16.1, 6.16.2	-

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
```

This defect is associated with the issue described in [SDCOMP-57528](#).

Conditions

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

- The program contains a branch instruction `I` with a destination `D`.
- `D` has one of the following forms:
 - `. + <offset>`
 - `<label> + <offset>`
- `<offset>` is an immediate value provided in the source code.

- The behavior of the program depends on `1` branching to `0`.

2.3.2.25 SDCOMP-55580

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-55580.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	A32 state	6.16.1, 6.16.2	-

Description

The inline assembler and integrated assembler incorrectly fail to report the following error for a `VLDL` instruction that loads the address of a label that is not in the same section as the instruction:

- `unsupported relocation type`

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

```
.arm

.section .data.src, "a", %progbits
.balign 4
.global src
src:
.word 0x11223344

.section .text.func, "ax"
.balign 4
.global func
.type func, %function
func:
    vldr s0, src
    bx lr
```

This defect is associated with the issues described in [SDCOMP-64165](#) and [SDCOMP-63454](#).

Conditions

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

- The program contains a `VLDL` instruction `1`.
- `1` specifies a label `1` as the source operand.
- `1` is in a section `A`.
- `1` is in a section `B`.
- `A` and `B` are not the same.
- The behavior of the program depends on `1` loading the address of `1`.

2.3.2.26 SDCOMP-55267

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-55267.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.16.1, 6.16.2	-

Description

The inline assembler and integrated assembler incorrectly fail to report an error for an MSR instruction that specifies PMMIR_EL1 as the system register to be accessed.

Conditions

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

- The program is assembled for a target that supports the Armv8.4-A additions to the AArch64 Performance Monitors Extension.
- The program contains an MSR instruction that specifies PMMIR_EL1 as the system register to be accessed.

2.3.2.27 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	6.16.1, 6.16.2	-

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 reopened 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.

2.3.2.28 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	6.16.1, 6.16.2	-

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 τ .
- τ contains an anonymous member m of **union** type.
- τ contains a **constexpr** constructor z .
- z does not initialize any member of m .
- The program contains a **constexpr** variable instantiation i of τ .
- The behavior of the program depends on i initializing m .

2.3.2.29 SDCOMP-51180

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-51180.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch64 state	6.16.1, 6.16.2	-

Description

The inline assembler and integrated assembler incorrectly ignore a `.arch` or `.cpu` target selection directive that does not explicitly include or exclude an extension using `+extension` or `+noextension`.

For example, the integrated assembler incorrectly ignores the `.arch armv8-a` target selection directive in the following:

```
.arch armv8-a
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 `-march` or `-mcpu` option A .
 - The program contains a `.arch` or `.cpu` target selection directive A that explicitly includes or excludes an extension using `+extension` or `+noextension`.
- A specifies a target with a set of features x .
- The program contains subsequent a `.arch` or `.cpu` target selection directive B .
- B does not explicitly include or exclude an extension using `+extension` or `+noextension`.

- y specifies a target with a set of features x .
- y is more restrictive than x .

2.3.2.30 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	6.16.1, 6.16.2	-

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.

2.3.2.31 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	6.16.1, 6.16.2	-

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.

2.3.2.32 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	6.16.1, 6.16.2	-

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.

2.3.2.33 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	6.16.1, 6.16.2	-

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()`

2.3.2.34 SDCOMP-46790

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-46790.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch32 state	6.16.1, 6.16.2	-

Description

The compiler incorrectly fails to report an error when compiling with `-mfloat-abi=hard` for a target without a hardware floating-point unit.

For example, the compiler incorrectly fails to report an error when compiling with `-mcpu=cortex-m3 -mfloat-abi=hard`.

Conditions

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

- The program is compiled for a target without a hardware floating-point unit using one of the following options:
 - `-march` with a target architecture that does not select a hardware floating-point unit by default.
 - `-mcpu` with a target CPU that does not select a hardware floating-point unit by default.
 - `-mfpu=none`
- The program is compiled with `-mfloat-abi=hard`.

- The behavior of the program depends on the generated code using hardware floating-point instructions.

2.3.2.35 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	6.16.1, 6.16.2	-

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`.

2.3.2.36 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	6.16.1, 6.16.2	-

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`.

2.3.2.37 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	6.16.1, 6.16.2	-

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 `F`.
- `F` contains an execution region `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 `E` does not end at a 16-byte boundary.
 - The program is built for AArch32 state and `E` does not end at an 8-byte boundary.

2.3.3 Determinism faults

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

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 6.16LTS Qualification Kit Safety Manual*.

2.3.3.1 SDCOMP-57994

This section describes the scope of the determinism fault defect with the unique identifier SDCOMP-57994.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armlink	AArch32 state	6.16.1	6.16.2

Description

The linker can incorrectly select a different implementation of an Arm C library function for two identical linker invocations. Specifically:

- The linker can incorrectly select a different but valid implementation of the Arm C library `strcmp()` function.
- When linking an input object that has been compiled with `-ffp-mode=full`, the linker can incorrectly select a variant of the Arm C library `cbrtf()` function that does not support denormals. 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 to an Arm C library implementation of a function F .
- One of the following is true:
 - All the following are true:
 - F is `cbrtf()`.
 - The program is compiled with `-ffp-mode=full`.
 - All the following are true:
 - F is `strcmp()`.
 - The program is compiled for an Armv8-A or Armv8-R target.
 - The program that is deployed is not identical to the program that was verified and validated during development.
- The behavior of the program depends on F .

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 6.16LTS Qualification Kit Safety Manual*.

2.3.4.1 SDCOMP-65669

This section describes the scope of the documentation synchronization fault defect with the unique identifier SDCOMP-65669.



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	6.16.1, 6.16.2	-

Description

The `-march` and `-mcpu` sections of the *Reference Guide* incorrectly state that a `+<feature>` modifier can only be used with an `-march` or `-mcpu` option if the feature is supported by the target.

Instead, the documentation should state that the compiler and integrated assembler does not prevent the use of `+<feature>` modifiers with an incompatible target.

For example, the integrated assembler does report an error or a warning for the half-precision floating-point `FADD` instruction in the following when assembling with `-march=armv8-a+fp16`:

```
.section .text.func, "ax"
.balign 8
.global func
.type func, %function
func:
    fadd v0.4h, v0.4h, v1.4h
    ret
```

`+fp16` enables the Half-precision floating-point data-processing feature (FEAT_FP16) that is only supported in Armv8.2-A and later.

Subsequently, this can result in unexpected run-time behavior when the `FADD` instruction is executed on a target that does not support FEAT_FP16.

To avoid this issue, you must manually inspect all `-march` and `-mcpu` command-line options and ensure that they do not include features that not supported by the target.

Conditions

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

- The program is built for a target `T` with an `-march` or `-mcpu` option `A`.
- `A` specifies a feature `F`.
- `T` does not support `F`.
- The behavior of the program depends on `F`.

2.3.4.2 SDCOMP-61633

This section describes the scope of the documentation synchronization fault defect with the unique identifier SDCOMP-61633.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	Any	6.16.1, 6.16.2	-

Description

The example in the `-fstack-protector`, `-fstack-protector-all`, `-fstack-protector-strong`, `-fno-stack-protector` section of the *Reference Guide* incorrectly uses an array that is 8 bytes in size to demonstrate stack protection when compiling with `-fstack-protector`.

`-fstack-protector` is only required to enable stack protection for vulnerable functions that contain:

- A character array larger than 8 bytes.
- An 8-bit integer array larger than 8 bytes.

- A call to `alloca()` with either a variable size or a constant size larger than 8 bytes.

To enable stack protection for other functions, consider using one of the following options:

- `-fstack-protector-all`
- `-fstack-protector-strong`

Conditions

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

- The program is compiled with `-fstack-protector`.
- The program contains an array `A`.
- `A` is 8 or less bytes in size.
- The behavior of the program depends on stack protection being applied to a function that uses `A`.

2.3.4.3 SDCOMP-61465

This section describes the scope of the documentation synchronization fault defect with the unique identifier SDCOMP-61465.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang	AArch32 state	6.16.1, 6.16.2	-

Description

The *Predefined macros* section of the *Reference Guide* provides incorrect information about the `SOFTFP` predefined macro.

`SOFTFP` must be defined as follows:

-mfloat-abi=<value>	Targets with hardware floating-point support	Targets without hardware floating-point support
Default	SOFTFP not defined	SOFTFP defined and set to 1
hard	SOFTFP not defined	SOFTFP not defined
soft	SOFTFP defined and set to 1	SOFTFP defined and set to 1
softfp	SOFTFP not defined	SOFTFP defined and set to 1

This defect is associated with the issue described in [SDCOMP-61298](#).

Conditions

The safety-related system is at risk when the behavior of the program depends on `SOFTFP`.

2.3.4.4 SDCOMP-61054

This section describes the scope of the documentation synchronization fault defect with the unique identifier SDCOMP-61054.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armlink	Any	6.16.1, 6.16.2	-

Description

The documentation incorrectly does not state that using both automatic and manual overlays within the same program is not supported.

Conditions

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

- The program is linked with a scatter file `F`.
- `F` specifies an execution region or load region `A`.
- `F` specifies an execution region `B`.
- `A` has the `OVERLAY` attribute.
- `B` has the `AUTO_OVERLAY` attribute.

2.3.4.5 SDCOMP-60826

This section describes the scope of the documentation synchronization fault defect with the unique identifier SDCOMP-60826.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armlink	AArch64 state	6.16.1, 6.16.2	-

Description

The `--cpu=name (armlink)` section of the *Reference Guide* incorrectly does not state that build attribute compatibility checking is supported only for AArch32 state.

The linker cannot report the following error for an input object that is incompatible with the specified AArch64 state `--cpu=name` option:

- `L6366E: <object> attributes are not compatible with the provided attributes`

For example, the linker cannot report an error when the following file is compiled with `--target=aarch64-arm-none-eabi -march=armv8.1-a` and linked with `--cpu=8-A.64`:

```
void func(void)
```

```
{
    return;
}
```

Conditions

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

- The program is linked with `--cpu=name`, where *name* specifies an a target in AArch64 state.
- An object *κ* is specified as an input to the linker.
- *κ* is incompatible with *name*.

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 6.16LTS.

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-65388	Translation fault	Libraries
SDCOMP-64611	Translation fault	Libraries
SDCOMP-64597	Translation fault	Libraries
SDCOMP-64555	Translation fault	Libraries
SDCOMP-64176	Translation fault	Libraries
SDCOMP-64025	Translation fault	Libraries
SDCOMP-62756	Translation fault	Libraries
SDCOMP-60938	Translation fault	Libraries
SDCOMP-60784	Translation fault	Libraries
SDCOMP-60700	Translation fault	Libraries
SDCOMP-60430	Translation fault	Libraries
SDCOMP-60260	Translation fault	Libraries
SDCOMP-60162	Translation fault	Libraries
SDCOMP-60157	Translation fault	Libraries

Identifier	Fault category	Affected components
SDCOMP-59054	Translation fault	Libraries
SDCOMP-58588	Translation fault	Libraries
SDCOMP-58561	Translation fault	Libraries
SDCOMP-58560	Translation fault	Libraries
SDCOMP-58304	Translation fault	Libraries
SDCOMP-58044	Translation fault	Libraries
SDCOMP-57673	Translation fault	Libraries
SDCOMP-53422	Translation fault	Libraries
SDCOMP-53184	Translation fault	Libraries
SDCOMP-52577	Translation fault	Libraries
SDCOMP-50751	Translation fault	Libraries
SDCOMP-50064	Translation fault	Libraries
SDCOMP-49748	Translation fault	Libraries
SDCOMP-47858	Translation fault	armasm
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
SDCOMP-11974	Translation fault	armasm
SDCOMP-61487	Missing diagnostic fault	armasm
SDCOMP-55186	Missing diagnostic fault	armasm
SDCOMP-54893	Missing diagnostic fault	armasm
SDCOMP-30418	Missing diagnostic fault	armasm
SDCOMP-22900	Missing diagnostic fault	armasm
SDCOMP-22142	Missing diagnostic fault	armasm
SDCOMP-17016	Missing diagnostic fault	armasm
SDCOMP-11899	Missing diagnostic fault	armasm

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 6.16LTS 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.



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
Libraries	AArch64 state	6.16.1, 6.16.2	-

Description

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 $(2^{**64})-1$ 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 2^{**64} .

2.4.1.2 SDCOMP-65388

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-65388.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	Any	6.16.1, 6.16.2	-

Description

The Arm C library is not thread-safe for a multithreaded program that contains C++ objects with static storage duration. 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 microlib.
- The program contains two threads `x` and `y`.
- One of the following is true:
 - `x` calls the Arm C library implementation of the `atexit()` function.
 - `x` calls a function which contains a C++ object with static storage duration that must be destroyed upon exit.
- One of the following is true:
 - `y` calls the Arm C library implementation of the `atexit()` function.
 - `y` calls a function which contains a C++ object with static storage duration that must be destroyed upon exit.

2.4.1.3 SDCOMP-64611

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-64611.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	Any	6.16.1, 6.16.2	-

Description

The Arm C and C++ libraries define the constant `math_errhandling` incorrectly. `math_errhandling` incorrectly always includes `MATH_ERREXCEPT`.

`math_errhandling` should be defined as follows:

-ffp-mode=<value>	math_errhandling value
fast	MATH_ERRNO
full	MATH_ERRNO MATH_ERREXCEPT
std (Default)	MATH_ERRNO

`math_errhandling`, `MATH_ERREXCEPT`, and `MATH_ERRNO` are defined in the following system headers:

- `<cmath>` for C++ source language modes.
- `<math.h>` for C source language modes.

Conditions

The safety-related system is at risk if the program uses the value of `math_errhandling` to determine whether floating-point exception signalling is supported.

2.4.1.4 SDCOMP-64597

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-64597.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	AArch64 state	6.16.1, 6.16.2	-

Description

The <arm_neon.h> system header incorrectly defines certain Neon intrinsics with a signed return type instead of an unsigned return type. Subsequently, this can result in unexpected run-time behavior.

For example, the incorrect return type of `vqshrnh_n_s16()` results in `var2` being initialized to `0xffffffff` instead of `0xff` in the following:

```
int16_t var1 = 0x7fff;
uint32_t var2 = vqshrnh_n_s16(var1, 1);
```

To avoid this issue, manually inspect the source code and explicitly cast the return value of each affected intrinsic to the correct unsigned type as specified by the *Arm C Language Extensions* (ACLE). For example:

```
int16_t var1 = 0x7fff;
uint32_t var2 = (uint8_t) vqshrnh_n_s16(var1, 1);
```

Conditions

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

- The program contains a call to a Neon intrinsic `ι` defined the <arm_neon.h> system header.
- `ι` is one of the following:
 - `vqshrund_n_s64()`
 - `vqshrnh_n_s16()`
 - `vqshrns_n_s32()`
 - `vqshrund_n_s64()`
 - `vqshrnh_n_s16()`
 - `vqshrns_n_s32()`
- The behavior of the program depends on the value returned by `ι` being unsigned.

2.4.1.5 SDCOMP-64555

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-64555.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	AArch32 state	6.16.1, 6.16.2	-

Description

The Arm C library implementations of the `__aeabi_ddiv()` function for targets without hardware floating-point support can incorrectly round up a result that is less than halfway between two adjacent representable double-precision numbers.

Conditions

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

- The program is built for a target without hardware floating-point support.
- The program is not linked with microlib.
- The program contains a division operation involving a double-precision value.

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 does not contain a call to `__aeabi_ddiv()`, then the safety-related system is not at risk.

2.4.1.6 SDCOMP-64176

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-64176.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	AArch64 state	6.16.1, 6.16.2	-

Description

The Arm C library implementation of the `nearbyint()` function can return an incorrect result.

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 `nearbyint()` function.
- The program is not compiled with `-fno-builtin`.

- The parameter of `z` has one of the following values:
 - `2251799813685248.5`, that is $2^{51} + 0.5$
 - `-2251799813685248.5`, that is $-(2^{51} + 0.5)$

2.4.1.7 SDCOMP-64025

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-64025.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	Armv8-M with the Main Extension	6.16.1, 6.16.2	-

Description

The Arm C library variant for a target that supports integer MVE only has the following incorrect behavior:

- `longjmp(<env>)` fails to restore callee-saved vector registers from `<env>`.
- `setjmp(<env>)` fails to save callee-saved vector registers to `<env>`.

Conditions

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

- The program is built for a target with the M-profile Vector Extension (MVE).
- The program is built for a target that supports integer MVE.
- The program is built for a target that does not support floating-point MVE.
- The program is built for a target without hardware floating-point support.
- The program contains a call `z` to one of the following functions:
 - `longjmp(<env>)`
 - `setjmp(<env>)`
- The behavior of the program depends on `z`.

2.4.1.8 SDCOMP-62756

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-62756.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	AArch64 state	6.16.1, 6.16.2	-

Description

The Arm C library implementation of the `memmove()` function can incorrectly corrupt memory when copying at least 4GB.

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 `memmove()` function.
- `z` specifies the following:
 - A destination address, `d`.
 - A source address, `s`.
 - The number of bytes to copy, `n`.
- `n` is at least 4GB.
- `d` is lower than `s`.
- `d+n` is higher than `s`.
- The behavior of the program depends on `z`.

2.4.1.9 SDCOMP-60938

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-60938.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	Any	6.16.1, 6.16.2	-

Description

The Arm C library implementations of the `lround()`, `lroundf()`, `llround()`, and `llroundf()` functions can incorrectly fail to set `errno` to `EDOM`.

For example, the `llround()` function incorrectly fails to set `errno` to `EDOM` in the following:

```
double var = <value>;
llround(var);
```

where `<value>` is equivalent to `0x1.p+63`.

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, where the parameter value is equivalent to the specified parameter value. The value does not need to be a constant literal value specified at compile-time:

State	Function	Parameter Type	Parameter Value
AArch64	<code>lround()</code> or <code>llround()</code>	double	<code>0x1.p+63</code>
AArch64	<code>lroundf()</code> or <code>llroundf()</code>	float	<code>0x1.p+63F</code>
AArch32	<code>lroundf()</code>	float	<code>0x1.p+31F</code>
AArch32	<code>llround()</code>	double	<code>0x1.p+63</code>
AArch32	<code>llroundf()</code>	float	<code>0x1.p+63F</code>

- The behavior of the program depends on `z` setting `errno` to `EDOM`.

2.4.1.10 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	AArch32 state	6.16.1, 6.16.2	-

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 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.

2.4.1.11 SDCOMP-60700

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-60700.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	Any	6.16.1, 6.16.2	-

Description

The Arm C library implementation of the `fwrite()` function incorrectly always returns zero when an error occurs instead of returning the number of objects successfully written.

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 `fwrite()` function.
- The behavior of the program depends on `z` always returning the number of objects successfully written.

2.4.1.12 SDCOMP-60430

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-60430.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	AArch64 state	6.16.1, 6.16.2	-

Description

The Arm C++ library can allocate memory at a fallback address that is incorrectly not 8-byte aligned. 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 compiled in a C++ source language mode.
- The program is linked with the C++ libraries supplied with Arm Compiler.
- The program throws a C++ exception `E`.
- The amount of free memory in the heap is not enough to allocate a C++ exception object when `E` is thrown.

2.4.1.13 SDCOMP-60260

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-60260.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	AArch64 state	6.16.1, 6.16.2	-

Description

The Arm C library implementations of the `fma()` and `fmaf()` functions can incorrectly set `errno` to `ERANGE` when the result is exactly equal to zero.

For example, the `fma()` function incorrectly sets `errno` to `ERANGE` in the following function:

```
double func(double x)
{
    return fma(x, -1.0, x);
}
```

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:
 - `fma()`
 - `fmaf()`
- The product of the first two parameters of `z` is exactly equal to the negation of the third parameter of `z`.
- The behavior of the program depends on `z` setting `errno` to `ERANGE`.

2.4.1.14 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	6.16.1, 6.16.2	-

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_TYPE_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.

2.4.1.15 SDCOMP-60157

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-60157.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	Any	6.16.1	6.16.2

Description

The Arm C library implementation of the POSIX `mbstowcs()` function can incorrectly update the source pointer `s` when the destination pointer `d` is a null pointer. `s` should not be updated when `d` is a null pointer.

This defect is associated with the issue described in [SDCOMP-58588](#).

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 POSIX `mbstowcs()` function.
- The first (destination pointer) argument of `z` is a null pointer.
- The third (input buffer size) argument of `z` is less than the length of the source string pointed to by the second argument.

2.4.1.16 SDCOMP-59054

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-59054.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	Any	6.16.1	6.16.2

Description

The Arm C++ library implementation of the `std::allocator<T>::allocate()` function incorrectly raises a `std::length_error` exception instead of `std::bad_alloc` OR `std::bad_array_new_length`.

For example, the Arm C++ library implementation of the `std::allocator<T>::allocate()` function incorrectly raises a `std::length_error` exception for the following:

```
std::allocator<int> f;  
std::allocator<int>::size_type size = f.max_size();  
int *p = f.allocate(size + 1, 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 is compiled with C++ exceptions enabled.
- The program contains a call `z` to the `std::allocator<T>::allocate()` function `F`.
- `F` is defined in the `<allocator.h>` Arm C++ library header file.

- The size parameter of `z` exceeds the maximum supported size, `numeric_limits<size_t>::max()/sizeof(T)`.
- The behavior of the program depends on a `std::bad_alloc` or `std::bad_array_new_length` exception being raised for `z`.

2.4.1.17 SDCOMP-58588

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-58588.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	Any	6.16.1	6.16.2

Description

The Arm C library implementations of the `mbstowcs()` and `wcsrtombs()` functions incorrectly update the pointer object `p` pointed to by the source pointer when the destination pointer `d` is a null pointer. `p` should not be updated when `d` is a null pointer.

This defect is associated with the issue described in [SDCOMP-60157](#).

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:
 - `mbstowcs()`
 - `wcsrtombs()`
- The destination pointer argument of `z` is a null pointer.
- The behavior of the program depends on the pointer object pointed to by the source pointer remaining unchanged.

2.4.1.18 SDCOMP-58561

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-58561.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	Any	6.16.1	6.16.2

Description

The Arm C library implementations of the `snprintf()`, `swprintf()`, `vsnprintf()`, and `vswprintf()` functions can incorrectly write an empty string to the buffer but still return a non-negative result.

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:
 - `snprintf()`
 - `swprintf()`
 - `vsnprintf()`
 - `vswprintf()`
- The first argument to `z` specifies a pointer `p` to a buffer `B`.
- The second argument to `z` specifies a buffer size `s`.
- Incrementing `p` by `s-1` causes an integer overflow.
- The behavior of the program depends on `z` writing the intended output string to `B`.

2.4.1.19 SDCOMP-58560

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-58560.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	AArch64 state	6.16.1	6.16.2

Description

The Arm C library implementation of the `scalblnf()` function can return an incorrect result.

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 `scalblnf()` function.
- The value of the `exp` argument of `z` is outside the range of a 32-bit signed integer.

2.4.1.20 SDCOMP-58304

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-58304.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	Any	6.16.1	6.16.2

Description

The Arm C library implementations of the `fgets()` and `fgetws()` functions incorrectly return a null pointer and incorrectly fail to set the read buffer to the terminating null character, `'\0'`.

For example, the `fgets()` function incorrectly sets `p` to null and incorrectly fails to set `buffer` to `'\0'` in the following:

```
char *p = fgets(buffer, 1, file_to_read);
```

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:
 - `fgets()`
 - `fgetws()`
- The read buffer size parameter of `z` is 1.
- The behavior of the program depends on one of the following:
 - `z` returning a non-null pointer.
 - `z` setting the read buffer to the terminating null character, `'\0'`.

2.4.1.21 SDCOMP-58044

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-58044.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	Any	6.16.1	6.16.2

Description

The Arm C library implementations of functions declared in the `<stdio.h>` system header can incorrectly result in a deadlock between threads in a multithreaded environment.

To avoid this issue, do not use line-buffered or unbuffered input streams in a multithreaded environment.

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 consists of multiple threads.
- More than one thread contains a call to a function declared in the `<stdio.h>` system header.
- The program contains a read from an input stream `s`.
- `s` is either line-buffered or unbuffered.

2.4.1.22 SDCOMP-57673

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-57673.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	AArch64 state	6.16.1, 6.16.2	-

Description

The Arm C library incorrectly fails to configure the floating-point unit as follows:

- Enable Default NaN propagation.
- Enable flushing denormalized numbers to zero.

Subsequently, a floating-point operation can generate an unexpected result at run-time.

To avoid this issue, ensure that your reset handler sets the `DN` and `FZ` bits of the `FPCR` register to 1. For example:

```
mrs x0, FPCR
orr x0, #0x03000000
msr FPCR, x0
isb
```

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 floating-point operation `F`.
- The behavior of the program depends on one of the following for `F`:

- Default NaN propagation.
- Flushing denormalized numbers to zero.

2.4.1.23 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	6.16.1, 6.16.2	-

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`.

2.4.1.24 SDCOMP-53184

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-53184.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	Any	6.16.1, 6.16.2	-

Description

The Arm C library implementation of the `fclose()` function can incorrectly fail to disassociate the stream it closes when the `_sys_close()` function returns a non-zero value.

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 an implementation of the `_sys_close()` function that can return a non-zero value.
- The program contains a call `z` to the `fclose()` function.
- `z` closes a stream `s` that is used to access a file `F`.
- The behavior of the program depends on `s` being disassociated from `F`.

2.4.1.25 SDCOMP-52577

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-52577.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	Any	6.16.1, 6.16.2	-

Description

The default constructor of an instantiation of the Arm C++ library template `std::forward_list<T, x>` incorrectly fails to call the default constructor of `x`, where `x` is an allocator for type `T`.

For example, the default constructor of `y` incorrectly fails to call the default constructor of `x` in the following:

```
struct X : public std::allocator<char> {
    X() { /* default constructor code */ };
};

std::forward_list<char, X> Y = {};
```

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 is linked with the C++ libraries supplied with Arm Compiler.
- The program contains an instantiation `y` of the class template `std::forward_list<T, x>`, where `x` is an allocator for type `T`.
- The program contains a call `z` to the default constructor of `y`.
- The behavior of the program depends on `z` automatically calling the default constructor of `x`.

2.4.1.26 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	6.16.1, 6.16.2	-

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.

2.4.1.27 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	6.16.1, 6.16.2	-

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

2.4.1.28 SDCOMP-49748

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-49748.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
Libraries	Any	6.16.1, 6.16.2	-

Description

The Arm C++ library implementation of the `const` version of the `std::bitset<N>` class template `operator[]` incorrectly returns a value of `std::bitset<N>::const_reference` type instead of `bool` type.

For example, `var` is incorrectly initialized as a variable of `std::bitset<N>::const_reference` type instead of `bool` type in the following:

```
const std::bitset<8> obj;  
auto var = obj[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 contains a `const` object `c` of type `std::bitset<N>`.
- The program uses `operator[]` to access a bit at a specific position in `c`.
- The behavior of the program depends on `operator[]` returning a value of `bool` type.

2.4.1.29 SDCOMP-47858

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-47858.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armasm	Any	6.16.1, 6.16.2	-

Description

The legacy assembler incorrectly generates a `code` type symbol instead of a `data` type symbol for an address associated with the contents of a file included using an `INCBIN` directive. The `INCBIN` directive correctly always includes file contents as data.

For example, the legacy assembler incorrectly generates a `code` type symbol for `sym` in the following:

```

AREA |.text|, CODE
EXPORT sym
ADDS r0, r0, #1
sym
  INCBIN src.bin
END

```

Conditions

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

- An assembly language source file contains an exported symbol `s`.
- `s` is defined in a `CODE` section.
- `s` is immediately followed by an `INCBIN` directive.
- `s` is not immediately preceded by a data definition directive.
- The behavior of the program depends on `s` being used as a symbol of `data` type instead of `code` type.

2.4.1.30 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	6.16.1, 6.16.2	-

Description

The Arm C library implementation of the `qsort()` function can corrupt the stack when sorting 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 the `qsort()` function.
- `z` specifies an array containing `m` members of size `n` each.
- `m * n` is larger than 4GB.

2.4.1.31 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	6.16.1, 6.16.2	-

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`.

2.4.1.32 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	6.16.1, 6.16.2	-

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`.

2.4.1.33 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	6.16.1, 6.16.2	-

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.

2.4.1.34 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	6.16.1, 6.16.2	-

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.

2.4.1.35 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	6.16.1, 6.16.2	-

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`.

2.4.1.36 SDCOMP-11974

This section describes the scope of the translation fault defect with the unique identifier SDCOMP-11974.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armasm	Any	6.16.1, 6.16.2	-

Description

The legacy assembler generates a dependency file that incorrectly does not contain files included using the C preprocessor `#include` directive.

For example, the legacy assembler generates a dependency file that incorrectly does not contain `include.h` for the following:

```
AREA |.text|, CODE
#include "include.h"
BX      lr

END
```

Conditions

This defect occurs when all the following are true:

- The program is assembled with all the following options:
 - `--cpreproc`
 - `--cpreproc_opts=<option>[,<option>, ...]`
 - `--depend=D`
- The program contains a C preprocessor `#include` directive.

The safety-related system is only at risk when `D` is used in the build process.

2.4.2 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 6.16LTS Qualification Kit Safety Manual*.

2.4.2.1 SDCOMP-61487

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-61487.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armasm	Any	6.16.1, 6.16.2	-

Description

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

For example, the legacy assembler incorrectly fails to report an error when assembling with `--cpu=Cortex-A9` and `--fpu=FPv5`.

Conditions

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

- The program contains an assembly language source file `F`.
- `F` contains an instruction `I`.
- `F` is built with `--cpu=A`.
- `F` is built with `--fpu=B`.
- `I` is not compatible with `A`.
- `A` is not compatible with `B`.
- The behavior of the program depends on `I` being successfully executed.

2.4.2.2 SDCOMP-55186

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-55186.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armasm	AArch64 state	6.16.1, 6.16.2	-

Description

The legacy assembler incorrectly fails to report an error for an UNPREDICTABLE LDRAA or LDRAB instruction.

For example, the legacy assembler incorrectly fails to report an error for each of the following:

```
LDRAA X0, [X0, #8]!  
LDRAB X0, [X0, #8]!
```

Conditions

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

- The program is assembled for an Armv8.3-A target.
- The program contains an instruction with one of the following forms:
 - LDRAA <Xt>, [<Xn>{, #<simm>}]!
 - LDRAB <Xt>, [<Xn>{, #<simm>}]!
- <xt> and <xn> are the same.

2.4.2.3 SDCOMP-54893

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-54893.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armasm	Armv8-A	6.16.1, 6.16.2	-

Description

The legacy assembler incorrectly fails to report an error for a half-precision floating-point scalar instruction or a half-precision floating-point SIMD instruction that is in an `IT` block and uses a `D` register.

For example, the legacy assembler incorrectly fails to report an error for the `VADDNE.F16` instruction in the following:

```
AREA |.text.func|, CODE  
  
THUMB  
  
IT NE  
VADDNE.F16 d0, d1, d2  
  
END
```

Conditions

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

- The program is assembled for an Armv8.2-A or later target and T32 state.
- The program contains an instruction `IT` in an `IT` block.
- `IT` is one of the following:

- A half-precision floating-point scalar instruction.
- A half-precision floating-point SIMD instruction.
- `I` uses a `D` register.
- `I` is `UNPREDICTABLE` in an `IT` block.

2.4.2.4 SDCOMP-30418

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-30418.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armasm	Armv8-A	6.16.1, 6.16.2	-

Description

The legacy assembler incorrectly fails to report an error instead of a warning for a ThumbEE instruction.

To avoid this issue, assemble with `--diag_error=1929`.

Conditions

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

- The program is assembled for AArch32 state.
- The program contains a ThumbEE instruction.

2.4.2.5 SDCOMP-22900

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-22900.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armasm	AArch64 state	6.16.1, 6.16.2	-

Description

The legacy assembler incorrectly fails to report an error for an invalid `MRS` or `MSR` instruction.

For example, the legacy assembler incorrectly fails to report an error for the following invalid `MRS` instruction:

```
mrs x4, S3__C15_C2_
```

Conditions

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

- The program contains an `MRS` or `MSR` instruction `I`.
- `I` accesses an implementation-defined register `R`.
- `R` must be specified using an operand `K` of the form `S<op0><op1><Cn><Cm><op2>`.
- Any of the following are missing from `K`:
 - `<op0>`
 - `<op1>`
 - `<Cn>`
 - `<Cm>`
 - `<op2>`
- The behavior of the program depends on `I` accessing `R`.

2.4.2.6 SDCOMP-22142

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-22142.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armasm	Any	6.16.1, 6.16.2	-

Description

The legacy assembler incorrectly fails to report an error for an Advanced SIMD or an AArch64 floating-point instruction.

For example, when assembling with `--cpu=8-A.64 --fpu=none`, the legacy assembler incorrectly fails to report an error for the following AArch64 floating-point instruction:

```
FADD s0, s1, s2
```

Conditions

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

- The program is assembled with `--fpu=none`.
- One of the following is true:

- The program contains an Advanced SIMD instruction, and the target does not support Advanced SIMD instructions.
- The program contains an AArch64 floating-point instruction, and the target does not support AArch64 floating-point instructions.

2.4.2.7 SDCOMP-17016

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-17016.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armasm	Armv7-A, Armv7-R	6.16.1, 6.16.2	-

Description

The legacy assembler can incorrectly fail to report an error for an `MRS` or `MSR` instruction that specifies a memory-mapped debug register that is not visible in the CP14 interface as the special register to be accessed.

For example, the legacy assembler incorrectly fails to report errors for the following instructions when assembling for a Cortex-A15 target:

```
MRS    r0, DBGPID1
MSR    DBGVIDSR, r0
```

Conditions

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

- The program contains an `MRS` or `MSR` instruction `1`.
- `1` specifies a memory-mapped debug register `D` as the special register to be accessed.
- The *Arm Architecture Reference Manual Armv7-A and Armv7-R edition* describes `D` as not visible in the CP14 interface.

2.4.2.8 SDCOMP-11899

This section describes the scope of the missing diagnostic fault defect with the unique identifier SDCOMP-11899.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armasm	AArch32 state	6.16.1, 6.16.2	-

Description

The legacy assembler incorrectly fails to report an error for a `RELOC` directive that has a second operand containing an invalid expression of the form `symbol+offset`. Instead, the legacy assembler incorrectly generates a relocation that ignores the specified `offset`.

Conditions

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

- The program contains a `RELOC` directive `R`.
- The second operand of `R` is an expression of the form `symbol+offset`.
- The behavior of the program depends on `R`.

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 6.16LTS.

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.



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
SDCOMP-62359	Documentation synchronization 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 6.16LTS 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	6.16.1, 6.16.2	-

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 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 6.16LTS Qualification Kit Safety Manual*.

2.5.2.1 SDCOMP-62359

This section describes the scope of the documentation synchronization fault defect with the unique identifier SDCOMP-62359.

The following table describes the scope of this defect:

Affected components	Target environment	Affected releases	Unaffected releases
armclang, Libraries	Any	6.16.1, 6.16.2	-

Description

The *Integer division-by-zero errors in C code* section of the *User Guide* incorrectly states that you can trap and identify integer division-by-zero errors using the Arm C library helper function `__aeabi_idiv0()`.

Integer division-by-zero in C code is undefined behavior, and the compiler does not guarantee a specific behavior for such code.

To trap and identify integer division-by-zero errors, you must manually test the denominator before the division operation takes place. For example:

```
#include <signal.h>

int divide(const int numerator, const int denominator)
{
    if (denominator == 0)
    {
        return raise(SIGFPE);
    }
    else
    {
        return numerator / denominator;
    }
}
```

Conditions

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

- The program contains an integer division operation with a denominator `d`.
- The behavior of the program depends on one of the following when `d` is zero:
 - The `SIGFPE` signal being raised.

- The `__aeabi_idiv0()` function being called.

Appendix A Changes since the Arm Compiler for Embedded FuSa 6.16.2 Qualification Kit Defect Report

This appendix provides information about changes made to the defect lists compared to the Qualification Kit *Defect Report* included in the release of Arm Compiler for Embedded FuSa 6.16.2 in May 2022.

A.1 Defects added

This section contains a list of defects that have been added to this document compared to the Qualification Kit *Defect Report* included in the release of Arm Compiler for Embedded FuSa 6.16.2 in May 2022.

Identifier	Fault category	Affected components	Target environment
SDCOMP-66632	Translation fault	armclang	AArch64 state
SDCOMP-66256	Translation fault	armclang	AArch64 state
SDCOMP-66090	Translation fault	Libraries	AArch64 state
SDCOMP-65669	Documentation synchronization fault	armclang	Any
SDCOMP-65517	Translation fault	armlink	Any
SDCOMP-65388	Translation fault	Libraries	Any
SDCOMP-65264	Missing diagnostic fault	armclang	AArch64 state
SDCOMP-65243	Missing diagnostic fault	armclang	AArch64 state
SDCOMP-65172	Translation fault	armclang	Any
SDCOMP-64999	Translation fault	armlink	AArch32 state
SDCOMP-64683	Missing diagnostic fault	armclang	Armv7-A, Armv7-M, Armv7-R, Armv8-A, Armv8-M with the Main Extension, Armv8-R
SDCOMP-64611	Translation fault	Libraries	Any
SDCOMP-64597	Translation fault	Libraries	AArch64 state
SDCOMP-64595	Translation fault	armlink	AArch64 state
SDCOMP-64591	Translation fault	armclang	AArch32 state
SDCOMP-64590	Translation fault	armlink	Any
SDCOMP-64555	Translation fault	Libraries	AArch32 state
SDCOMP-64255	Missing diagnostic fault	armclang	AArch32 state
SDCOMP-64176	Translation fault	Libraries	AArch64 state
SDCOMP-64165	Translation fault	armclang	A32 state

Identifier	Fault category	Affected components	Target environment
SDCOMP-64066	Translation fault	armclang	Armv8-M with the Main Extension
SDCOMP-64059	Translation fault	armclang	Armv8-M with the Main Extension
SDCOMP-64025	Translation fault	Libraries	Armv8-M with the Main Extension
SDCOMP-63984	Translation fault	armclang	T32 state
SDCOMP-63952	Translation fault	armclang	AArch32 state
SDCOMP-63948	Translation fault	armclang, Libraries	AArch64 state
SDCOMP-63946	Translation fault	armclang	AArch64 state
SDCOMP-63917	Missing diagnostic fault	armclang	AArch64 state
SDCOMP-63913	Translation fault	armclang	AArch64 state
SDCOMP-63912	Translation fault	armclang	AArch32 state
SDCOMP-63911	Translation fault	armclang	AArch32 state
SDCOMP-63894	Translation fault	armclang	AArch64 state
SDCOMP-63761	Translation fault	armclang	AArch64 state
SDCOMP-63752	Translation fault	armclang	AArch64 state
SDCOMP-63738	Translation fault	fromelf	Armv8-M with the Main Extension
SDCOMP-63697	Missing diagnostic fault	armclang	Any
SDCOMP-63688	Translation fault	armclang	Armv6-M, Armv8-M without the Main Extension
SDCOMP-63454	Translation fault	armclang	T32 state
SDCOMP-62791	Translation fault	armclang	AArch32 state
SDCOMP-62769	Translation fault	armclang	AArch32 state
SDCOMP-62756	Translation fault	Libraries	AArch64 state
SDCOMP-62725	Translation fault	armclang	Armv8-M with the Main Extension
SDCOMP-62692	Translation fault	armclang	Armv7-A, Armv7-M, Armv7-R, Armv8-A, Armv8-M with the Main Extension, Armv8-R
SDCOMP-62661	Translation fault	armclang	AArch64 state
SDCOMP-62378	Translation fault	armclang	AArch32 state
SDCOMP-62359	Documentation synchronization fault	armclang, Libraries	Any
SDCOMP-62352	Translation fault	armclang	Armv8-M with the Main Extension
SDCOMP-62330	Translation fault	armclang	Any
SDCOMP-62251	Translation fault	armlink	AArch64 state
SDCOMP-62234	Missing diagnostic fault	armclang	Any
SDCOMP-62221	Translation fault	armclang	Armv6-M
SDCOMP-62217	Translation fault	fromelf	Armv8-M with the Main Extension
SDCOMP-62201	Missing diagnostic fault	armclang	AArch64 state
SDCOMP-62176	Translation fault	armclang	AArch32 state
SDCOMP-62133	Translation fault	armclang	AArch64 state
SDCOMP-62123	Translation fault	armclang	AArch32 state
SDCOMP-62028	Translation fault	armclang	Armv8-M without the Main Extension

Identifier	Fault category	Affected components	Target environment
SDCOMP-61633	Documentation synchronization fault	armclang	Any
SDCOMP-61514	Translation fault	armclang	AArch32 state
SDCOMP-61489	Missing diagnostic fault	fromelf	Any
SDCOMP-61488	Missing diagnostic fault	armlink	Any
SDCOMP-61487	Missing diagnostic fault	armasm	Any
SDCOMP-61486	Translation fault	armclang	Any
SDCOMP-61465	Documentation synchronization fault	armclang	AArch32 state
SDCOMP-61461	Missing diagnostic fault	armclang	A32 state
SDCOMP-61299	Translation fault	armclang	AArch32 state
SDCOMP-61298	Translation fault	armclang	AArch32 state
SDCOMP-61150	Translation fault	armlink	AArch32 state
SDCOMP-61089	Missing diagnostic fault	armclang	AArch32 state
SDCOMP-61080	Translation fault	armclang	Any
SDCOMP-61054	Documentation synchronization fault	armlink	Any
SDCOMP-60938	Translation fault	Libraries	Any
SDCOMP-60897	Translation fault	armclang	Armv8-M
SDCOMP-60826	Documentation synchronization fault	armlink	AArch64 state
SDCOMP-60784	Translation fault	Libraries	AArch32 state
SDCOMP-60725	Translation fault	fromelf	Armv8-M with the Main Extension
SDCOMP-60700	Translation fault	Libraries	Any
SDCOMP-60659	Translation fault	armlink	Any
SDCOMP-60632	Translation fault	armclang	AArch64 state
SDCOMP-60589	Translation fault	armclang	Any
SDCOMP-60443	Translation fault	armclang	AArch32 state
SDCOMP-59938	Translation fault	armlink	AArch64 state

A.2 Defects removed

This section contains a list of defects that have been removed from this document compared to the Qualification Kit *Defect Report* included in the release of Arm Compiler for Embedded FuSa 6.16.2 in May 2022.

Any defect may be removed from future versions of this document without prior notice. For the details about a specific removed defect, see the Qualification Kit *Defect Report* included in the release of Arm Compiler for Embedded FuSa 6.16.2 or [contact Arm support](#).

Identifier	Fault category	Affected components	Target environment
SDCOMP-56990	Translation fault	armclang	AArch64 state
SDCOMP-51624	Translation fault	armclang	AArch64 state
SDCOMP-30395	Missing diagnostic fault	armlink	AArch64 state
SDCOMP-25815	Translation fault	fromelf	Any
SDCOMP-25192	Missing diagnostic fault	armasm, armlink	Any
SDCOMP-17692	Translation fault	Libraries	Any
SDCOMP-15257	Translation fault	Libraries	Any

A.3 Defects updated

This section contains a list of defects that have been updated in this document compared to the Qualification Kit *Defect Report* included in the release of Arm Compiler for Embedded FuSa 6.16.2 in May 2022.

Identifier	Fault category	Affected components	Target environment
SDCOMP-60430	Translation fault	Libraries	AArch64 state
SDCOMP-60342	Translation fault	armclang	Any
SDCOMP-60326	Translation fault	fromelf	Any
SDCOMP-60260	Translation fault	Libraries	AArch64 state
SDCOMP-60162	Translation fault	Libraries	AArch64 state
SDCOMP-60157	Translation fault	Libraries	Any
SDCOMP-60117	Translation fault	armlink	Any
SDCOMP-59974	Translation fault	armclang	AArch64 state
SDCOMP-59788	Translation fault	armclang	Armv7-M, Armv8-M
SDCOMP-59656	Translation fault	armclang	AArch64 state
SDCOMP-59605	Missing diagnostic fault	armclang	Any
SDCOMP-59521	Translation fault	armclang	Any
SDCOMP-59512	Missing diagnostic fault	armclang	Any
SDCOMP-59190	Missing diagnostic fault	armclang	Any
SDCOMP-59074	Translation fault	armclang	AArch64 state
SDCOMP-59059	Translation fault	armclang	AArch32 state
SDCOMP-59054	Translation fault	Libraries	Any
SDCOMP-58780	Translation fault	armclang	Any
SDCOMP-58773	Translation fault	armclang	AArch64 state
SDCOMP-58738	Translation fault	armclang	Any
SDCOMP-58588	Translation fault	Libraries	Any
SDCOMP-58561	Translation fault	Libraries	Any
SDCOMP-58560	Translation fault	Libraries	AArch64 state
SDCOMP-58367	Missing diagnostic fault	armclang	T32 state
SDCOMP-58354	Translation fault	armlink	Any

Identifier	Fault category	Affected components	Target environment
SDCOMP-58304	Translation fault	Libraries	Any
SDCOMP-58044	Translation fault	Libraries	Any
SDCOMP-57994	Determinism fault	armlink	AArch32 state
SDCOMP-57912	Missing diagnostic fault	armclang	Any
SDCOMP-57884	Translation fault	armclang	Any
SDCOMP-57725	Translation fault	armclang	Any
SDCOMP-57674	Translation fault	armclang	Any
SDCOMP-57673	Translation fault	Libraries	AArch64 state
SDCOMP-57528	Missing diagnostic fault	armclang	AArch64 state
SDCOMP-57456	Translation fault	fromelf	AArch64 state
SDCOMP-57449	Translation fault	fromelf	AArch64 state
SDCOMP-57255	Translation fault	armclang	Any
SDCOMP-57229	Translation fault	armclang	Any
SDCOMP-57213	Translation fault	armlink	Any
SDCOMP-57200	Translation fault	armclang	AArch32 state
SDCOMP-57199	Missing diagnostic fault	armlink	AArch32 state
SDCOMP-56812	Missing diagnostic fault	armclang	Any
SDCOMP-56435	Translation fault	armlink	Any
SDCOMP-56331	Missing diagnostic fault	armclang	AArch64 state
SDCOMP-56220	Missing diagnostic fault	armclang	Any
SDCOMP-56212	Missing diagnostic fault	armclang	Any
SDCOMP-55983	Missing diagnostic fault	armclang	A32 state
SDCOMP-55580	Missing diagnostic fault	armclang	A32 state
SDCOMP-55460	Translation fault	armclang	Any
SDCOMP-55267	Missing diagnostic fault	armclang	AArch64 state
SDCOMP-55186	Missing diagnostic fault	armasm	AArch64 state
SDCOMP-55184	Translation fault	fromelf	Any
SDCOMP-54893	Missing diagnostic fault	armasm	Armv8-A
SDCOMP-54546	Translation fault	fromelf	Any
SDCOMP-53903	Missing diagnostic fault	armclang	Any
SDCOMP-53422	Translation fault	Libraries	Any
SDCOMP-53184	Translation fault	Libraries	Any
SDCOMP-52627	Missing diagnostic fault	armclang	Any
SDCOMP-52577	Translation fault	Libraries	Any
SDCOMP-51180	Missing diagnostic fault	armclang	AArch64 state
SDCOMP-50968	Translation fault	fromelf	Any
SDCOMP-50751	Translation fault	Libraries	Any
SDCOMP-50408	Translation fault	armclang	AArch32 state
SDCOMP-50064	Translation fault	Libraries	Any
SDCOMP-50017	Missing diagnostic fault	armclang	Any

Identifier	Fault category	Affected components	Target environment
SDCOMP-49961	Missing diagnostic fault	armclang	Any
SDCOMP-49919	Missing diagnostic fault	armclang	Any
SDCOMP-49763	Missing diagnostic fault	armclang	Any
SDCOMP-49748	Translation fault	Libraries	Any
SDCOMP-47858	Translation fault	armasm	Any
SDCOMP-46790	Missing diagnostic fault	armclang	AArch32 state
SDCOMP-45879	Translation fault	Libraries	AArch64 state
SDCOMP-44980	Translation fault	fromelf	Any
SDCOMP-30903	Translation fault	Libraries	Any
SDCOMP-30418	Missing diagnostic fault	armasm	Armv8-A
SDCOMP-30359	Translation fault	Libraries	Any
SDCOMP-29077	Translation fault	Libraries	Any
SDCOMP-28728	Translation fault	fromelf	AArch64 state
SDCOMP-25238	Missing diagnostic fault	armclang	Any
SDCOMP-24899	Translation fault	fromelf	T32 state
SDCOMP-22900	Missing diagnostic fault	armasm	AArch64 state
SDCOMP-22142	Missing diagnostic fault	armasm	Any
SDCOMP-18689	Missing diagnostic fault	armlink	Any
SDCOMP-18016	Translation fault	Libraries	Any
SDCOMP-17355	Missing diagnostic fault	armlink	Any
SDCOMP-17016	Missing diagnostic fault	armasm	Armv7-A, Armv7-R
SDCOMP-13831	Translation fault	Libraries	Armv7-M
SDCOMP-11974	Translation fault	armasm	Any
SDCOMP-11947	Translation fault	fromelf	Any
SDCOMP-11899	Missing diagnostic fault	armasm	AArch32 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
0	26 July 2024	Non-Confidential	Initial release

Change history

To be completed later. For a list of technical changes to this document, refer to the Arm Developer website.

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.



Tip

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



Remember

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

Useful resources

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

Access to Arm documents depends on their confidentiality:

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

Arm product resources	Document ID	Confidentiality
Arm Compiler for Embedded FuSa 6.16LTS documentation index	KA005062	Non-Confidential
Arm Compiler for Embedded FuSa 6.16LTS Qualification Kit Safety Manual	102288	Confidential
Does Arm document all known issues that affect each Arm Compiler release?	KA005052	Non-Confidential