Fault Handling and Detection

Version 1.1
Fault Handling and Detection

Copyright © 2016 ARM Limited or its affiliates. All rights reserved.

Release Information

<table>
<thead>
<tr>
<th>Issue</th>
<th>Date</th>
<th>Confidentiality</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100-00</td>
<td>07 July 2016</td>
<td>Non-Confidential</td>
<td>First release.</td>
</tr>
<tr>
<td>0101-00</td>
<td>01 September 2016</td>
<td>Non-Confidential</td>
<td>Second release.</td>
</tr>
</tbody>
</table>

Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice 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. 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 implementations infringe any third party patents.

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, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

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.

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

If any of the provisions contained in these terms conflict with any of the provisions of any signed written agreement covering this document with ARM, then the signed written agreement prevails over and supersedes the conflicting provisions of these terms. 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 the Agreement shall prevail.

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM Limited or its affiliates in the EU and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow ARM’s trademark usage guidelines at http://www.arm.com/about/trademark-use-guidelines.php

Copyright © 2016, ARM Limited or its affiliates. All rights reserved.


110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Unrestricted Access is an ARM internal classification.
**Product Status**

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

**Web Address**

http://www.arm.com
Contents

Fault Handling and Detection

Preface

About this book ................................................................. .................................................. 6
Feedback .................................................................................................................. 8

Chapter 1 
Introduction

1.1 The ARM®v8-M architecture ........................................................... 1-10

Chapter 2 
Fault exceptions and registers

2.1 Fault exceptions .................................................................................. 2-12
2.2 Fault Status Registers and Fault Address Registers ..................... 2-15
2.3 Synchronous and asynchronous bus faults ................................... 2-17
2.4 Lockup state ......................................................................................... 2-18

Chapter 3 
Secure and Non-secure system considerations

3.1 Secure fault handler considerations ................................................ 3-20
3.2 Using AIRCR.BFHFNMINS ................................................................. 3-21
3.3 Fault handler compatibility in ARM®v8-M ....................................... 3-22
3.4 Non-secure fault handler considerations ....................................... 3-23
Preface

This preface introduces the Fault Handling and Detection.

It contains the following:

• About this book on page 6.
• Feedback on page 8.
About this book

Write a short description in the book map to render in the "About this book" section of the preface.

Product revision status

The rm:pn identifier indicates the revision status of the product described in this book, for example, r1p2, where:

rm  Identifies the major revision of the product, for example, r1.
pn  Identifies the minor revision or modification status of the product, for example, p2.

Intended audience

Using this book

This book is organized into the following chapters:

Chapter 1 Introduction
Error detection and correction techniques can be used to help mitigate the effect of errors in silicon devices. ARMv8-M processors include features that provide a means of detecting some of these errors.

Chapter 2 Fault exceptions and registers
This describes fault exceptions and fault types. It also provides the register description of relevant registers.

Chapter 3 Secure and Non-secure system considerations
This describes when to use AIRCR.BFHFNMINS and the changes required to ARMv6-M and ARMv7-M Fault handlers for use in ARMv8-M processors.

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.

Typographic conventions

italic  Introduces special terminology, denotes cross-references, and citations.

bold  Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

monospace  Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.

monospace  Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

monospace italic  Denotes arguments to monospace text where the argument is to be replaced by a specific value.

monospace bold  Denotes language keywords when used outside example code.
<and>

Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example:

\[
\text{MRC p15, 0, \text{<Rd>}, \text{<CRn>}, \text{<CRm>}, \text{<Opcode_2>}}
\]

**SMALL CAPITALS**

Used in body text for a few terms that have specific technical meanings, that are defined in the *ARM glossary*. For example, *IMPLEMENTATION DEFINED*, *IMPLEMENTATION SPECIFIC*, *UNKNOWN*, and *UNPREDICTABLE*.

**Timing diagrams**

The following figure explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.

![Timing Diagram Figure](image)

**Figure 1  Key to timing diagram conventions**

**Signals**

The signal conventions are:

**Signal level**

The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means:

- **HIGH** for active-HIGH signals.
- **LOW** for active-LOW signals.

**Lowercase n**

At the start or end of a signal name denotes an active-LOW signal.
Feedback

Feedback on this product
If you have any comments or suggestions about this product, contact your supplier and give:

• The product name.
• The product revision or version.
• An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content
If you have comments on content then send an e-mail to errata@arm.com. Give:

• The title Fault Handling and Detection.
• The number ARM 100691_0101_00_en.
• If applicable, the page number(s) to which your comments refer.
• A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Note
ARM tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.
Error detection and correction techniques can be used to help mitigate the effect of errors in silicon devices. ARMv8-M processors include features that provide a means of detecting some of these errors.

In silicon devices, errors can occur because of:
- Software bugs.
- Usage errors, where the conditions are outside normal operational conditions. For example, temperature or supply voltage, or unexpected operations, such as invalid input data or operator errors.
- Memory corruptions, where stray radiation and other effects can cause the data that is stored in RAM to be corrupted.

Features of ARM®v8-M processors can enable software to manage or even correct some of the error conditions, and alert the users of the device to the event so that corrective or protective actions can be taken. Some of the ARMv8-M devices are designed to detect more types of error conditions, and can handle the detected errors in a predictable manner, making them suitable for use in safety-related systems.

The features to detect and handle errors are divided into architectural features and implementation-specific features. The architecture for ARMv8-M processors incorporates fault handling features by exceptions, and a Non-Maskable Interrupt (NMI) for handling system level errors, for example, brown out detection. Implementation-specific features such as Error Correcting Code (ECC) for memories are not covered here.

It contains the following sections:
- 1.1 The ARM®v8-M architecture on page 1-10.
1.1 The ARM®v8-M architecture

The ARMv8-M architecture is designed for devices with a small silicon footprint.

In a similar way to ARMv6-M, all fault events are considered as unrecoverable. There are no fault status registers in the ARMv8-M architecture, as there are in the ARMv8-M architecture with Main Extension. However, software developers can still analyze errors during software development using debug features like the Micro Trace Buffer (MTB) or Embedded Trace Macrocell (ETM). These features provide recent execution history, and therefore enable issues to be identified easily.

It is also possible for silicon chip designers create their own fault status registers and fault address registers to capture information about bus errors.
Chapter 2
Fault exceptions and registers

This describes fault exceptions and fault types. It also provides the register description of relevant registers.

It contains the following sections:
- 2.1 Fault exceptions on page 2-12.
- 2.2 Fault Status Registers and Fault Address Registers on page 2-15.
- 2.3 Synchronous and asynchronous bus faults on page 2-17.
- 2.4 Lockup state on page 2-18.
2.1 Fault exceptions

In ARMv8-M processors, error conditions are handled by fault exceptions. Fault exceptions are similar to interrupts, where piece of software that is called a handler is executed when fault event taken place.

Events can generate faults, for example:
- A bus error on:
  - An instruction fetch or vector table load.
  - A data access.
- An internally detected error like an UNDEFINED instruction.
- Attempting to execute an instruction from a memory region marked as Execute Never (XN).
- A privilege violation or an attempt to access an unmanaged region causing an MPU fault.

2.1.1 Description of fault types

Based on the processor implementation and the optional features included, there can be different type of fault exceptions available.

<table>
<thead>
<tr>
<th>Fault type</th>
<th>Short name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard Fault</td>
<td>HardFault</td>
<td>Default Fault exception. Occurs because of an error during exception processing, or because an exception cannot be managed by any other exception mechanism.</td>
</tr>
<tr>
<td>Bus Fault</td>
<td>BusFault</td>
<td>Occurs because of a memory-related fault for an instruction or data memory transaction. This fault might be from an error that is detected on a bus in the memory system. Not available in ARMv8-M or ARMv6-M processors.</td>
</tr>
<tr>
<td>Usage Fault</td>
<td>UsageFault</td>
<td>Occurs because of a memory protection-related fault. The MPU or the fixed memory protection constraints determine this fault, for both instruction and data memory transactions. This fault is always used to abort instruction accesses to Execute Never (XN) memory regions. Not available in ARMv8-M or ARMv6-M processors.</td>
</tr>
<tr>
<td>Memory Management Fault</td>
<td>MemManage</td>
<td>Caused by violation of access permission set by the Memory Protection Unit (MPU) or attempting to execute code from XN address regions. Not available in ARMv8-M or ARMv6-M processors.</td>
</tr>
<tr>
<td>Secure Fault</td>
<td>SecureFault</td>
<td>Caused by Security violations. Available in ARMv8-M architecture with Main Extension only, and present only if the ARMv8-M architecture with Security Extension is implemented. In the ARMv8-M architecture, security violations are handled by HardFault.</td>
</tr>
</tbody>
</table>

Note

- The Bus Fault, Usage Fault, Memory Management Fault, and Secure Fault exceptions are configurable fault exceptions. They are disabled by default and can be enabled by software.
- Configurable Fault exceptions have programmable exception priority levels similar to interrupts and other system exceptions.
• HardFault exception has a priority level of 1 (higher than all other exceptions apart from the NMI), or can be priority level 3 (higher than NMI) if it is triggered by a Security violation in Non-secure NMI in the ARMv8-M architecture.
• If a fault event is triggered and the corresponding Configurable Fault exception is disabled, or if the priority level of the configurable fault exception is not high enough to trigger a preemption, the HardFault exception would be triggered instead. This is called escalation.

2.1.2 Security state of fault exceptions

If the Security Extension is implemented in the processor, fault exceptions can target either the Secure state or Non-secure state. This behavior is a change to the behavior in the ARMv6-M and ARMv7-M architectures.

Table 2-2  Fault exceptions and target states

<table>
<thead>
<tr>
<th>Fault exception</th>
<th>Target state</th>
</tr>
</thead>
<tbody>
<tr>
<td>HardFault</td>
<td>Default to Secure state. This can be configured to allow Non-secure HardFault by Secure software. However, security violations are still handled by the HardFault exception in Secure state.</td>
</tr>
<tr>
<td>BusFault</td>
<td>Default to Secure state. This can be configured to allow Non-secure BusFault by Secure software.</td>
</tr>
<tr>
<td>UsageFault</td>
<td>Can either be Secure or Non-secure state. Depends on the current state when the error event occurs.</td>
</tr>
<tr>
<td>MemManage</td>
<td>Can either be Secure or Non-secure state. Depends on the current state when the error event occurs.</td>
</tr>
<tr>
<td>SecureFault</td>
<td>Secure state only.</td>
</tr>
</tbody>
</table>

A programmable bit in the Application Interrupt and Reset Control Register (AIRCR) called BFHFNMINS (BusFault, HardFault, and NMI Non-secure enable), bit [13] is used in the ARMv8-M architecture to permit Secure software to define whether fault exceptions and NMI are handled by Non-secure software.

The target state of the fault exception determines which vector table is used for fetching the exception vector and the execution state of the ISR. Because the vector tables for Secure and Non-secure software are separated, different fault handlers can be used when the fault events are targeted at different states.

2.1.3 Fault escalation and HardFault

All fault exceptions except for HardFault have a configurable exception priority. Software can disable execution of the configurable fault handlers, but not the HardFault handler.

Usually, the exception priority, and the values of the exception mask registers, determine whether the processor enters the fault handler, and whether one fault handler can preempt another fault handler.

In some situations, a fault with configurable priority is treated as a HardFault. This situation is called priority escalation, and the fault is described as being escalated to HardFault. Escalation to HardFault occurs when:

• A fault handler causes the same kind of fault as the one it is servicing. The escalation to HardFault occurs because a fault handler cannot preempt itself because it must have the same priority as the current priority level.
• A fault handler causes a fault with the same or lower priority as the fault it is servicing. This is because the handler for the new fault cannot preempt the currently executing fault handler.
• An exception handler causes a fault for which the priority is the same as or lower than the currently executing exception.
• A fault occurs and the handler for that fault is not enabled.

If a Bus Fault occurs during a stack push when entering a Bus Fault handler, the Bus Fault does not escalate to a HardFault. This means that if a corrupted stack causes a fault, the fault handler executes even though the stack push for the handler failed. The fault handler operates but the stack contents are corrupted.

Note

Only Reset and NMI can preempt the fixed priority HardFault. A HardFault can preempt any exception other than Reset, NMI, or another HardFault. In the ARMv8-M architecture a Security violation in a Non-secure NMI handler (if BFHFNMIINS is set to 1) would trigger and preempted by Secure HardFault exception.

In the case where a bus error occurred during an exception vector fetch, this error is always handled by HardFault exception and not BusFault.
2.2 Fault Status Registers and Fault Address Registers

In ARMv7-M and ARMv8-M architecture with Main Extension, several Fault Status Registers (xFSR) are available to allow fault handlers to identify the cause of the fault exceptions. Fault Address Registers (xFAR) are available to indicate the address of the access that triggers the fault.

The fault status registers indicate the cause of a fault. For synchronous BusFaults and MemManage faults, the fault address register indicates the address that is accessed by the operation that caused the fault.

2.2.1 Register descriptions

Description of fault status registers and fault address registers.

Table 2-3 Register descriptions

<table>
<thead>
<tr>
<th>Handler</th>
<th>Status register name</th>
<th>Address register name</th>
<th>Register description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HardFault</td>
<td>HFSR</td>
<td></td>
<td>HardFault Status Register</td>
</tr>
<tr>
<td>MemManage</td>
<td>MMFSR</td>
<td>MMFAR</td>
<td>MemManage Fault Status Register</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMFAR</td>
<td>MemManage Fault Address Register</td>
</tr>
<tr>
<td>BusFault</td>
<td>BFSR</td>
<td>BFAR</td>
<td>BusFault Status Register</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BFAR</td>
<td>BusFault Address Register</td>
</tr>
<tr>
<td>UsageFault</td>
<td>UFSR</td>
<td></td>
<td>UsageFault Status Register</td>
</tr>
</tbody>
</table>

2.2.2 Fault types

The following table shows:
- The type of fault.
- The handler that is used for the fault.
- The corresponding fault status register.
- The register bit that indicates that the fault has occurred.

Table 2-4 Faults and the corresponding fault status registers

<table>
<thead>
<tr>
<th>Fault</th>
<th>Handler</th>
<th>Bit name</th>
<th>Fault status register</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus error on a vector read</td>
<td>HardFault</td>
<td>VECTTBL</td>
<td>HardFault Status Register</td>
</tr>
<tr>
<td>Fault that is escalated to a hard fault</td>
<td>HardFault</td>
<td>FORCED</td>
<td></td>
</tr>
<tr>
<td>MPU or default memory map mismatch:</td>
<td>MemManage</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>On instruction access</td>
<td>MemManage</td>
<td>IACCVIOL</td>
<td>MemManage Fault Status Register</td>
</tr>
<tr>
<td>On data access</td>
<td>MemManage</td>
<td>DACCVIOL</td>
<td></td>
</tr>
<tr>
<td>During exception stacking</td>
<td>MemManage</td>
<td>MSTKERR</td>
<td></td>
</tr>
<tr>
<td>During exception unstacking</td>
<td>MemManage</td>
<td>MUNSKERR</td>
<td></td>
</tr>
<tr>
<td>During lazy floating-point state preservation</td>
<td>MemManage</td>
<td>MLSPERR</td>
<td></td>
</tr>
<tr>
<td>Fault</td>
<td>Handler</td>
<td>Bit name</td>
<td>Fault status register</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>----------</td>
<td>----------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>Bus error:</td>
<td>BusFault</td>
<td>STKERR</td>
<td>BusFault Status Register</td>
</tr>
<tr>
<td>During exception stacking</td>
<td></td>
<td>UNSTKERR</td>
<td></td>
</tr>
<tr>
<td>During exception unstacking</td>
<td></td>
<td>BUSERR</td>
<td></td>
</tr>
<tr>
<td>During instruction prefetch</td>
<td></td>
<td>LSPERR</td>
<td></td>
</tr>
<tr>
<td>During lazy floating-point state preservation</td>
<td></td>
<td>PRECISERR</td>
<td></td>
</tr>
<tr>
<td>Precise data bus error</td>
<td></td>
<td>IMPRECISERR</td>
<td></td>
</tr>
<tr>
<td>Imprecise data bus error</td>
<td></td>
<td>NOCP</td>
<td>UsageFault Status Register</td>
</tr>
<tr>
<td>Attempt to access a coprocessor</td>
<td>UsageFault</td>
<td>UNDEFINSTR</td>
<td></td>
</tr>
<tr>
<td>UNDEFINED instruction</td>
<td></td>
<td>INVSTATE</td>
<td></td>
</tr>
<tr>
<td>Attempt to enter an invalid instruction set state</td>
<td></td>
<td>INVPIC</td>
<td></td>
</tr>
<tr>
<td>Invalid EXC_RETURN value</td>
<td></td>
<td>UNALIGNED</td>
<td></td>
</tr>
<tr>
<td>Illegal unaligned load or store</td>
<td></td>
<td>DIVBYZERO</td>
<td></td>
</tr>
<tr>
<td>Divide By 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.3 Synchronous and asynchronous bus faults

Bus Faults are subdivided into two classes: synchronous and asynchronous bus faults.

**Synchronous bus fault**
This bus fault is also described as a precise bus fault in older versions of ARM documents. This bus fault refers to the fault exception that takes place immediately after the bus transfer is carried out.

**Asynchronous bus fault**
This bus fault is also described as an imprecise bus fault in older versions of ARM documents. This bus fault refers to the fault exception that can take place a certain time after the bus transfer is carried out, where the processor could have executed further instructions before the exception sequence started.

2.3.1 Synchronous bus fault

A synchronous BusFault can escalate into lockup if it occurs inside an NMI or HardFault handler. Cache maintenance operations can also trigger a bus fault.

The fault handler can use BFSR to determine whether faults are asynchronous or synchronous. Asynchronous faults are often unrecoverable, as you do not know which code caused the error.

2.3.2 Asynchronous bus fault

An asynchronous bus fault can happen when there is write buffering in the processor design. As a result, the processor pipeline proceeds to subsequent instruction execution before the bus error response is observed.

Debug accesses can also trigger Bus Faults. Debugger load or store accesses are synchronous, and are visible to the debugger interface only.

When an asynchronous bus fault is triggered, the BusFault exception is pended. If another higher priority interrupt event arrived at the same time, the higher priority interrupt handler is executed first, and then the Bus Fault exception takes place. If the BusFault handler is not enabled, the HardFault exception is pended instead. The HardFault caused by the asynchronous BusFault never escalates into lockup.
2.4 Lockup state

The processor enters into lockup state if a fault occurs when executing the NMI or HardFault handlers. When NMI is Non-secure and a Security violation is detected, it triggers a Secure HardFault at priority level 3 instead. When the processor is in lockup state, it does not execute any instructions.

The processor remains in lockup state until either:
• It is reset.
• An NMI occurs.
• A debugger halts it.

——— Note ————
If lockup state occurs from the NMI handler, a subsequent NMI does not cause the processor to leave lockup state.
Chapter 3
Secure and Non-secure system considerations

This describes when to use AIRCR.BFHFNMINS and the changes required to ARMv6-M and ARMv7-M Fault handlers for use in ARMv8-M processors.

It contains the following sections:
• 3.1 Secure fault handler considerations on page 3-20.
• 3.2 Using AIRCR.BFHFNMINS on page 3-21.
• 3.3 Fault handler compatibility in ARM®v8-M on page 3-22.
• 3.4 Non-secure fault handler considerations on page 3-23.
3.1 Secure fault handler considerations

In general, software developers for Secure software:

- Must always implement fault handlers to ensure that software developers creating Non-secure software can be aware of problems during software development.
- Secure software must utilize stack limit registers to ensure that stack overflow in Secure software can be detected and handled by a Secure HardFault or Usage Fault handler.
- Secure fault handlers must not call Non-secure functions to prevent Non-secure code gaining access to the Secure state which can be used for DoS attack.
3.2 Using AIRCR.BFHFNMINS

In ARMv8-M processors with ARM TrustZone® technology for ARMv8-M implemented, the BFHFNMINS bit in the Application Interrupt and Reset Control Register (AIRCR) is available and is programmable by Secure privileged code only.

The possible values of the BFHFNMINS bit are:

0  BusFault, HardFault, and NMI are Secure.

1  BusFault and NMI are Non-secure and exceptions can target Non-secure HardFault.

It can be used in ARMv8-M devices where TrustZone technology is used for firmware protection. This is common where software assets are preloaded and can be utilized by third-party software developers, but at the same time it must not be possible to reverse engineer the preloaded software.

In such applications, the protected firmware and RAM memory that is used by protected firmware is placed in Secure memory. Fault events can be handled by Non-secure software created by third-party software developers. When the core leaves the reset state, Secure boot code can set the AIRCR.BFHFNMINS bit before starting third-party applications.

If the application contains a Secure RTOS, then the fault handling code interacts with the Secure RTOS kernel and therefore the fault exception must be executed in Secure state. In such case, AIRCR.BFHFNMINS must not be set to 1.

It is likely that the fault handlers are handled in Secure state, that is, AIRCR.BFHFNM is set to 0. The Secure software developer can allocate a Non-secure interrupt as a second-level Non-secure fault handler.

Note

Software must clear the Bus Fault Address Register (BFAR) when changing the value of AIRCR.BFHFNMINS so as not to expose the last accessed address to the other security state.

When a fault event is triggered, the Secure fault handler first analyzes the type of fault that has occurred. If it finds no security thread, the handler can then set the pending status of the Non-secure second-level fault handler so that the fault event can be handled by Non-secure software. It is not recommended to call the Non-secure fault handler from Secure fault handler directly, as this could allow Non-secure software to gain a higher than expected priority level.
3.3 Fault handler compatibility in ARM®v8-M

Some of the fault handlers that are written for the ARMv6-M and ARMv7-M architecture contain code that extracts stacked program counters from the stack frame. Some of these handlers might need to be changed because the fault handler might be in the Secure world, while the stack frame is in the Non-secure stack, so the code to locate the stack frame must be changed.
3.4 Non-secure fault handler considerations

When migrating software from ARMv6-M or ARMv7-M to ARMv8-M architecture, software developers that create Non-secure software must be aware of some possible changes.

These changes are the following:

- The HardFault and BusFault handler might no longer be used because the Secure software on chip has its own HardFault and BusFault handlers (AIRCR.BFHFNMINS might be set to 0).
- Self-reset might not be accessible from Non-secure software. When TrustZone technology for ARMv8-M is implemented, Secure software can configure whether Non-secure software can access the SYSRESETREQ bit in AICR. Also, the AIRCR.VECTRESET bit in ARMv7-M is not available in ARMv8-M architecture.