Real-time Operating System Design Considerations

Version 1.0

Confidential
Real-time Operating System Design Considerations

Confidential Proprietary Notice

This document is CONFIDENTIAL and any use by you is subject to the terms of the agreement between you and ARM or the terms of the agreement between you and the party authorised by ARM to disclose this document to you.

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: (i) for the purposes of determining whether implementations infringe any third party patents; (ii) for developing technology or products which avoid any of ARM’s intellectual property; or (iii) as a reference for modifying existing patents or patent applications or creating any continuation, continuation in part, or extension of existing patents or patent applications; or (iv) for generating data for publication or disclosure to third parties, which compares the performance or functionality of the ARM technology described in this document with any other products created by you or a third party, without obtaining ARM’s prior written consent.

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/trademarks/guidelines/index.php

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


110 Fulbourn Road, Cambridge, England CB1 9NJ.
Confidentiality Status
This document is Confidential. This document may only be used and distributed in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

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

Web Address
http://www.arm.com
# Real-time Operating System Design Considerations

## Preface
- About this book .................................................................................................................. 8
- Feedback ........................................................................................................................... 10

## Chapter 1 Introduction
1.1 ARM®v8-M architecture changes ............................................................................. 1-12

## Chapter 2 ARM®v8-M Architectural features
2.1 Thread and Handler Stack Pointers ........................................................................... 2-14
2.2 Stack and Stack limit ................................................................................................. 2-15
2.3 SVCall and PendSV exceptions ............................................................................... 2-16
2.4 SysTick timer .......................................................................................................... 2-17
2.5 EXC_RETURN ......................................................................................................... 2-18
2.6 OS configurations ..................................................................................................... 2-20
2.7 Extension of CMSIS-RTOS for Non-secure RTOS ................................................ 2-22

## Chapter 3 Context switching operations
3.1 RTOS design requirements ....................................................................................... 3-24
3.2 RTOS running in the Secure world ......................................................................... 3-25
3.3 RTOS running in the Non-secure world .................................................................... 3-26
3.4 Impact of the AIRCR.PRIS bit ................................................................................ 3-27
3.5 Supporting multiple Secure software libraries ....................................................... 3-28
3.6 Summary .................................................................................................................. 3-29
List of Figures
Real-time Operating System Design Considerations

Figure 1  Key to timing diagram conventions ................................................................. 9
Figure 2-1 Exception return with stack frame ............................................................. 2-18
List of Tables

Real-time Operating System Design Considerations

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 2-1</td>
<td>Stack pointers and stack limit registers</td>
<td>2-15</td>
</tr>
<tr>
<td>Table 2-2</td>
<td>EXC_RETURN updates for the ARMv8-M architecture</td>
<td>2-18</td>
</tr>
<tr>
<td>Table 2-3</td>
<td>Possible OS configurations</td>
<td>2-20</td>
</tr>
</tbody>
</table>
This preface introduces the Real-time Operating System Design Considerations.

It contains the following:

About this book

ARM®v8-M architecture is for the next generation ARMv8-M processor family of real time deterministic embedded processors. It sets the foundation for security and productivity for embedded solutions.

Product revision status

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

- \textit{rm} Identifies the major revision of the product, for example, r1.
- \textit{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:

\textbf{Chapter 1 Introduction}

Describes the changes in ARM®v8-M architecture compared to the previous ARMv6-M and ARMv7-M architectures.

\textbf{Chapter 2 ARM®v8-M Architectural features}

Describes the Stack Pointers, SVC call and PendSV exceptions, SysTick timer, and other features of the ARMv8-M architecture.

\textbf{Chapter 3 Context switching operations}

Describes the basic idea of context switching operations. Actual implementations can be different based on the RTOS design requirements.

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 \textit{ARM Glossary} for more information.

Typographic conventions

\textit{italic}

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

\textbf{bold}

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

\texttt{monospace}

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

\texttt{monospace}

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

\texttt{monospace italic}

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

\texttt{monospace bold}

Denotes language keywords when used outside example code.
Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example:

\[
\text{MRC p15, 0, } <\text{Rd}>, <\text{CRn}>, <\text{CRm}>, <\text{Opcode}_2>
\]

**Smaller 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 conventions](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 Real-time Operating System Design Considerations.
• The number ARM 100689_0100_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.
Chapter 1
Introduction

Describes the changes in ARM®v8-M architecture compared to the previous ARMv6-M and ARMv7-M architectures.

It contains the following sections:

• 1.1 ARM®v8-M architecture changes on page 1-12.
1.1 ARM®v8-M architecture changes

The ARMv8-M architecture has brought a range of changes compared to the previous ARMv6-M and ARMv7-M architectures. As a result, existing RTOS code for ARMv6-M and ARMv7-M architecture must be updated to run on the ARMv8-M architecture.

Some of the changes that are required are generic to RTOS designs:

- The change of Memory Protection Unit (MPU) programmers’ model to Protected Memory System Architecture (PMSA) v8 means that an RTOS with MPU support must update MPU support code.
- The extension of the EXC_RETURN (Exception Return) value definition means that if the OS creates a thread by exception return with an artificially created stack frame, the value of EXC_RETURN used is likely to change.

Additional changes are required to take advantage of the new capabilities in the ARMv8-M architecture. For example:

- Stack limit features are available for both Secure and Non-secure states in ARMv8-M architecture with Main Extension, and are available in Secure state for ARMv8-M architecture. The stack limit feature enables stack overflow scenarios to be detected. Stack overflows are one of the most common sources of software errors in embedded systems.
- Depending on the configuration of the system, an RTOS can support both Secure and Non-secure threads. For such cases, you must update the RTOS to support extra stacks.
- For RTOS previous written for ARMv6-M architecture, moving to ARMv8-M architecture enables the OS to use exclusive access instructions for semaphore variable updates. This architecture enhancement also enables ARMv8-M architecture with Main Extension and ARMv8-M architecture versions of the OS to share semaphore code.
- If the RTOS is delivered in compiled library form, recompilation of the RTOS code enables the software to be optimized for ARMv8-M processors.

Depending on the system-level design around the ARMv8-M processor, the Secure software and associated resources might be locked down. This means that software developers can only update the Non-secure program address space and access to Non-secure hardware resources. Secure resources can only be accessed using APIs in the Secure firmware. Such restrictions affect RTOS designs and the Secure firmware. Multiple design configurations are possible.
Chapter 2
ARM®v8-M Architectural features

Describes the Stack Pointers, SVCall and PendSV exceptions, SysTick timer, and other features of the ARMv8-M architecture.

It contains the following sections:
• 2.1 Thread and Handler Stack Pointers on page 2-14.
• 2.2 Stack and Stack limit on page 2-15.
• 2.3 SVCall and PendSV exceptions on page 2-16.
• 2.4 SysTick timer on page 2-17.
• 2.5 EXC_RETURN on page 2-18.
• 2.6 OS configurations on page 2-20.
• 2.7 Extension of CMSIS-RTOS for Non-secure RTOS on page 2-22.
2.1 Thread and Handler Stack Pointers

An RTOS can use the two Stack Pointers that the ARMv8-M architecture provides within each security state. The *Main Stack Pointer* (MSP) can be used by software running in Thread mode and is always used by Handler mode. A second register, the *Process Stack Pointer* (PSP) can be used by Thread mode software. The SPSel bit in the CONTROL register configures the Stack Pointer register that is used by Thread mode.

Using both registers allows an RTOS to separate the stack that is used by a user application from the OS stack, and protects the OS stack from errors in a user thread that might cause stack corruption and affect the stack pointed to by PSP.

--- Note ---

Simple applications without an OS do not have to use this configuration and instead can configure Thread mode to use the MSP. This use of MSP is the default stack configuration out of reset.

---

If the Security Extension is implemented, MSP and PSP are banked between security states. The notations MSP_NS, PSP_NS, MSP_S and PSP_S are used to indicate which banked version is being referred to, either *Secure* (S) or *Non-secure* (NS). In addition, the CONTROL.SPSel bit is also banked, allowing different stack configurations in Secure and Non-secure Thread modes.
2.2 Stack and Stack limit

If the ARMv8-M architecture with Security Extension is implemented, a stack limit feature is provided using stack limit registers accessible using MSR and MRS instructions.

\[
\text{MRS } RN, <x> \; \text{; copy the value of } <x> \text{ into } RN \\
\text{MSR } <x>, RN \; \text{; write the value in } RN \text{ into } <x>.
\]

Where \(<x>\) is one of: MSP, PSP, MSPLIM, or PSPLIM.

For example:

\[
\text{MRS } R0, \text{ MSPLIM} \; \text{; copy MSPLIM value into } R0
\]

Secure software can also use MSP_NS, PSP_NS, MSPLIM_NS and PSPLIM_NS to access the Non-secure version of those registers.

\[
\text{; Execute in Secure state} \\
\text{MSR MSP, r1 \; set Secure MSP} \\
\text{MSR MSP_NS, R2 \; set Non-secure MSP}
\]

The following table lists the available Stack Pointers and Stack limit registers.

<table>
<thead>
<tr>
<th>Stack</th>
<th>Stack Pointers</th>
<th>Corresponding stack limit register</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure Main Stack</td>
<td>MSP_S</td>
<td>MSPLIM_S</td>
</tr>
<tr>
<td>Used by Secure handlers, and Secure thread when bit[1] of the Secure CONTROL register is 0.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Secure Process Stack</td>
<td>PSP_S</td>
<td>PSPLIM_S</td>
</tr>
<tr>
<td>Non-secure Main Stack</td>
<td>MSP_NS</td>
<td>MSPLIM_NS</td>
</tr>
<tr>
<td>Used by Non-secure handlers, and Non-secure thread when bit[1] of the Non-secure CONTROL register is 0. (available on ARMv8-M architecture only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-secure Process Stack</td>
<td>PSP_NS</td>
<td>PSPLIM_NS</td>
</tr>
<tr>
<td>Used by Non-secure threads when bit[1] of the Non-secure CONTROL register is 1. (available on ARMv8-M architecture only)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.3 SVCall and PendSV exceptions

The SVCall and PendSV exceptions are banked between Secure and Non-secure states.

When the processor is in Secure state, the SVC exception handling sequence fetches the exception vector from the Secure vector table and executes the SVCall handler in Secure state. When the processor is in Non-secure state, the SVC exception handling sequence fetches the exception vector from the Secure vector table and executes the SVCall handler in Non-secure state.

Similarly, the PendSVSet and PendSVClr bit in the Interrupt Control and State Register (ICSR) is banked. Secure software can also trigger Non-secure PendSV using the Non-secure alias of the ICSR (0xE002ED04).

The priority level registers for SVC and PendSV are also banked between Secure and Non-secure states.
2.4 SysTick timer

A timer is a requirement of most operating systems. ARMv8-M processors provide a basic countdown timer called SysTick that can be used to generate a periodic interrupt. SysTick reduces the need for software porting, because the SysTick programmers’ model is the same on all Cortex®-M devices. In ARMv8-M processors, the SysTick timer is an optional feature and designers can decide if the chip must include this timer or not.

In the ARMv8-M architecture, the programmers’ model of the SysTick timer remains unchanged. However, the SysTick timer is banked, so there can be two physical SysTick timers in the design:

- When the processor is in Secure state, the Secure SysTick timer is accessed.
- When the processor is in Non-secure state, the Non-secure SysTick timer is accessed.

Each of these timers can trigger SysTick exceptions in their corresponding security states. Secure software can also access the Non-Security SysTick registers using an alias address.

For the ARMv8-M architecture with Main Extension there is the option of implementing just one SysTick. In this case, the SysTick timer that is implemented can be configured to be Secure or Non-secure using bit[24] of the Interrupt Control and State Register (ICSR).

For bit[24] of this register, SysTick Targets Non-secure State (STTNS):

- If this bit is 0 SysTick targets the Secure state (default).
- If this bit is 1 SysTick targets the Non-secure state.

The STTNS bit is accessible only when the processor is in Secure state, and only if the design configuration of the processor selects one SysTick implementation.

An RTOS can also use device-specific timer peripherals for time keeping and task scheduling. In such case, the security domain of the timer must match the RTOS configuration. For example, if the RTOS is running in a Secure domain, the timer that is used by the RTOS must be configured to be Secure accesses only, and the corresponding interrupt must also be configured as Secure.
2.5 EXC_RETURN

EXC_RETURN is an exception return mechanism.

Many real-time operating systems create threads using an exception return with an artificially created stack frame, as shown in the following figure.

![Figure 2-1 Exception return with stack frame](image)

In the ARMv8-M architecture, EXC_RETURN has been updated as follows:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field</th>
<th>Changes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:24</td>
<td>PREFIX</td>
<td>-</td>
<td>Must be 0xFF to be a valid EXC_RETURN code.</td>
</tr>
<tr>
<td>23:7</td>
<td>Reserved</td>
<td>fix to 1</td>
<td>Implementation defined</td>
</tr>
<tr>
<td>6</td>
<td>S</td>
<td>New</td>
<td>Secure or Non-secure stack. This bit indicates whether a Secure or Non-secure stack is used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 Non-secure stack used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 Secure stack used.</td>
</tr>
<tr>
<td>5</td>
<td>DCRA</td>
<td>New</td>
<td>Default callee register stacking. This bit indicates whether the default stacking rules apply, or whether the callee registers, for example, R4-R11, are already on the stack.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 Stacking of the callee saved registers skipped.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 Default rules for stacking the callee registers are followed.</td>
</tr>
<tr>
<td>Bits</td>
<td>Field</td>
<td>Changes</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>4</td>
<td>FType</td>
<td>-</td>
<td>Stack Frame Type. This bit indicates whether the stack frame is a standard integer only stack frame, or an extended stack frame with floating-point register contents.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 Extended stack frame.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 Standard stack frame.</td>
</tr>
<tr>
<td>3</td>
<td>Mode</td>
<td>-</td>
<td>Mode. This bit indicates the Mode that was stacked from.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 Handler mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 Thread mode.</td>
</tr>
<tr>
<td>2</td>
<td>SPSEL</td>
<td>-</td>
<td>Stack Pointer selection. This bit indicates which stack point the exception frame resides on.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 Main Stack Pointer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 Process Stack Pointer.</td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>0</td>
<td>ES</td>
<td>New</td>
<td>Exception Secure. The security domain the exception was taken to.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 Non-secure.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 Secure.</td>
</tr>
</tbody>
</table>
2.6 OS configurations

Various operating system configurations are possible with the ARMv8-M architecture. Since the ARMv8-M architecture with Security Extension is optional, a chip design with ARMv8-M architecture might have no TrustZone® technology support.

Without TrustZone technology support, the processor:

- Is always in Non-secure state.
- Has no Security Attribution Unit (SAU).
- Has no Secure MPU, Secure SysTick, or SCB registers.

Designs which include the Security Extension allow multiple configuration arrangements, several of which are described in the following table.

### Table 2-3 Possible OS configurations

<table>
<thead>
<tr>
<th>Configuration</th>
<th>RTOS Design requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTOS running in Secure state. Some threads are Secure, and some threads are Non-secure. Cross domain APIs calls are also possible between domains</td>
<td>Each thread might have Secure and Non-secure stack space allocation.</td>
</tr>
<tr>
<td>RTOS running in Non-secure state. Threads are Non-secure, but potentially some of the threads can call Secure API.</td>
<td>All threads have a Non-secure stack, and some threads that call Secure API also must have Secure stack allocation. Secure firmware must include CMSIS-RTOS API (version 2) to support handling of stack switching on the Secure side.</td>
</tr>
<tr>
<td>RTOS running in Non-secure state. Threads are Non-secure and there is no calling to Secure APIs. (Secure state is not used by the application).</td>
<td>All threads have Non-secure stack only. The design is identical to RTOS for chip designs without TrustZone technology support. The RTOS code still needs to be updated due to changes in EXC_RETURN and MPU programmers’ model.</td>
</tr>
<tr>
<td>RTOS and all applications running in Secure state. Non-secure state is not used.</td>
<td>All threads have Secure stack only. The RTOS code still needs to be updated due to changes in EXC_RETURN and the MPU programmers' model.</td>
</tr>
<tr>
<td>RTOS on Secure domain and use the idle thread of that OS to run a second OS (not real time) in the Non-secure domain</td>
<td>Secure RTOS Each thread might have Secure and Non-secure stack space allocation. Non-secure OS All threads have a Non-secure stack, and some threads that call Secure API also must have Secure stack allocation. Secure firmware must include CMSIS-RTOS API (v2) to support handling of stack switching on the Secure side.</td>
</tr>
</tbody>
</table>

Additional configurations are possible, for example, by having a bare-metal or interrupt driven application in one security domain and running an RTOS in the other domain.

This section contains the following subsections:

- 2.6.1 Case 1 – Secure state on page 2-20.
- 2.6.2 Case 2 – Non-secure state on page 2-21.
- 2.6.3 Case 3 – RTOS and application in Non-secure state only on page 2-21.
- 2.6.4 Case 4 – RTOS and application in Secure state only on page 2-21.

2.6.1 Case 1 – Secure state

When a real-time operating system is running in Secure state, threads can be Secure or Non-secure. It is also possible to have cross-domain API calls between Secure and Non-secure software, which means each thread can have both Secure and Non-secure stack allocation.
When creating a thread from Secure handler mode, the EXC_RETURN code can be:
• 0xFFFFFFFD if the new thread is Secure. The stack frame is PSP_S.
• 0xFFFFFFBD if the new thread is Non-secure. The stack frame is PSP_NS.

2.6.2 Case 2 – Non-secure state
In Non-secure state threads are Non-secure by default. However, threads can call APIs in Secure firmware memory, so threads can have both Secure and Non-secure stack allocation.

Because the RTOS is on a Non-secure domain, it cannot directly access the Secure Stack Pointers and is therefore unable to handle context switching of Secure stacks. To solve this problem, a new version of CMSIS-RTOS APIs is being developed which supports Secure stack management.

When creating a thread from Non-secure handler mode, the EXC_RETURN code must be 0xFFFFFFBC (thread is Non-secure). The stack frame must be pointed to by PSP_NS.

2.6.3 Case 3 – RTOS and application in Non-secure state only
Threads are always Non-secure when the RTOS and application are in Non-secure state only.

In Non-secure state only, the RTOS can be based on existing code that was written for ARMv8-M processors. However, the following modifications are still required:
• EXC_RETURN for creating a thread must be 0xFFFFFFBC (thread is Non-secure). The stack frame must be pointed to by PSP_NS.
• MPU Support code must be updated to support the new programmers’ model.

2.6.4 Case 4 – RTOS and application in Secure state only
It is possible for a microcontroller vendor to ship a blank device without locking down the Secure memory. In this case, software developers can create an application with an RTOS which runs entirely in Secure domain.

In Secure state only:
• EXC_RETURN for creating a thread must be 0xFFFFFFFD (thread is Secure).
• The stack frame must be pointed to by PSP_S.
2.7 Extension of CMSIS-RTOS for Non-secure RTOS

In some microcontroller devices the Secure memory space has been locked down and cannot be modified. As a consequence, many embedded applications that are based on ARMv8-M architecture might have an RTOS running from the Non-secure side.

In this configuration, Non-secure threads can still call Secure APIs in the Secure firmware. This means that these threads need Secure stack space allocation and the context switching of the RTOS must also switch the Process Stack Pointer on the Secure side (PSP_S). To meet such requirements, the CMSIS-RTOS API is extended to support context switching of Secure stack for Non-secure RTOS. The APIs are used by the Non-secure RTOS in initialization, thread creation, and context switching.

The API is standardized so that:

- The operations are identical across different chip products (enabling RTOS products to work on a range of ARMv8-M processor chips from different microcontroller vendors).
- The API is open, so all RTOS designers can create RTOS running in the Non-secure domains.

The CMSIS-RTOS API supplies function prototypes to perform the following functions:

- Initialize Secure Process Stack management.
- Allocate Secure stack space for a thread. Since Non-secure software developers have no visibility of the Secure software details, this function does not have stack size requirement information. Typically, this API must allocate the maximum stack size required by API calls.
- Free Secure stack space for a thread. Frees the allocated Secure stack space when a thread is removed or disabled.
- Store Secure content. If a context switch occurs when the current thread is in Secure state, the Non-secure RTOS calls this function to save the context of the thread before it is swapped out. Technically the registers are in the Secure stack already, but the PSP_S value must be saved to the Trace Control Block (TCB) in the Secure world.
- Load Secure content. If the OS has to switch to a context that has previously been saved, use this function to restore the context by setting PSP_S.
Chapter 3
Context switching operations

Describes the basic idea of context switching operations. Actual implementations can be different based on the RTOS design requirements.

It contains the following sections:
• 3.1 RTOS design requirements on page 3-24.
• 3.2 RTOS running in the Secure world on page 3-25.
• 3.3 RTOS running in the Non-secure world on page 3-26.
• 3.4 Impact of the AICR. PRIS bit on page 3-27.
• 3.5 Supporting multiple Secure software libraries on page 3-28.
• 3.6 Summary on page 3-29.
3.1 RTOS design requirements

Several areas of RTOS design requirements require particular attention. For example, to prevent Secure stack overflow, stacks for Secure software must be protected with stack limit registers.

To reduce stack size requirements, Process Stack Pointers (PSP) must be used for threads. The thread only needs to allocate stack for the thread and first level of exception stack frame, which avoids the need to reserve stack for exceptions and interrupt handlers.

Since Secure software can force Secure exceptions to have higher priority than Non-secure exceptions using AIRCR.PRIS, the Secure PendSV exception handler might be able to preempt a Non-secure interrupt. Therefore, the Secure RTOS scheduler might have to check EXC_RETURN bit[3], or the ICSR.RETTOBASE (Return to base) status bit in handling context switching.
3.2 RTOS running in the Secure world

The RTOS kernel can access both Secure and Non-secure stack and Stack Pointers when the RTOS is running in the Secure world. The Task Control Blocks (TCBs) might contain Secure information, including register contents in Secure state and the PSP_S values.

It is easier to place all TCBs in Secure memory as it can hold the register content of a thread, even if the thread is in a Non-secure state.

Assuming that the RTOS design uses PSP for thread stack management, the list of registers that must be saved in the TCB include:

- Callee saved registers (R4 to R11) in integer register banks.

  **Note**
  
  Caller saved registers R0-R3, r12 are on the stack frame already.

- If CONTROL_FPCA is 1 and EXC_RETURN bit [4] is 0, then floating-point registers S16 to S31. (S0 to S15, and the Floating-point Status and Control register (FPSCR) are saved in the stack frame with lazy stacking.
- PSP_S and PSP_NS, PSPLIM_S and PSPLIM_NS (if using ARMv8-M architecture with Main Extension).
- CONTROL_S, CONTROL_NS and EXC_RETURN values. These registers hold the state information, which can be privileged or unprivileged, and Secure or Non-secure.
- MPU region configurations (optional).

If bit 3 of EXC_RETURN is 0b0, an exception handler other than PendSV is active. Here context switching cannot be performed and the RTOS must schedule context switch operations in the next tick. If bit 4 (FType) of EXC_RETURN is 0b0, floating-point context is active and context switching can be implemented in Secure PendSV at lowest Secure priority level, as follows:

1. Store callee saved registers (R4 to R11) in TCB.
2. Save FPU S16-S31 to the TCB. S0-S15 and FPSCR would be saved automatically with lazy stacking.
3. Save PSP_S, PSP_NS, PSPLIM_S, PSPLIM_NS, EXC_RETURN, CONTROL_S, and CONTROL_NS in the TCB.
4. Set PSPLIM_S and PSPLIM_NS to 0 to disable the limit check.
5. Load PSP_S, PSP_NS, then PSPLIM_S and PSPLIM_NS for new thread.
6. Load callee save registers and FPU registers from the new TCB.
7. Load EXC_RETURN and CONTROL from the new TCB.

At the end of this sequence, the exception returns to the new thread.
3.3 RTOS running in the Non-secure world

When the RTOS runs in the Non-secure world, the RTOS kernel can only access the Non-secure stacks and Stack Pointers, and must use the CMSIS-RTOS v2 Security Extension API to handle context switching. In this arrangement, there is no other active exception running when the Non-secure PendSV handler executes, and this avoids the need for checking for running exception handlers before starting context switch.

Note

The detail is based on the CMSIS-RTOS API v2 and the APIs have not been finalized.

When the RTOS design uses PSP for thread stack management the list of registers to be saved in the TCB include:

- Call saved registers (R4 to R11) in integer register banks.
  
  Note
  
  Caller saved registers R0-R3, and R12 are on the stack frame already.

- If FPCA is 1 and EXC_RETURN bit [4] is 0, then the floating-point registers S16 to S31, S0 to S15, and FPSCR are saved in the stack frame with lazy stacking.
- PSP_NS and PSPLIM_NS (if using ARMv8-M architecture with Main Extension).
- CONTROL_NS and EXC_RETURN values. These hold the state information, privileged or unprivileged.
- MPU region configurations (optional).

The RTOS can also add additional information in the TCB, if necessary.

Depending on the value of EXC_RETURN bit 6 (S bit), the thread is running a Secure API (0b1), or a Non-secure API (0b0).

If running a Secure API, call TZ_Store_Context_S() to save the Secure contexts. Otherwise, it is common practice to implement context switching in Non-secure PendSV at lowest Secure priority level, as follows:

1. Store call saved registers (R4 to R11) in the TCB.

   Note

   Caller saved registers R0-R3, and R12 are on the stack frame already.

2. If EXC_RETURN bit 4 (FType) is 0, the floating-point context is active. Save FPU S16-S31 to the TCB. S0-S15 and FPSCR are saved automatically with lazy stacking.
3. Save PSP_NS, PSPLIM_NS, EXC_RETURN and CONTROL_NS in the TCB.
4. Set PSPLIM_NS to 0 to disable the limit check.
5. Load PSP_NS, then PSPLIM_NS for new thread.
6. If the EXC_RETURN bit 6 (S bit) of the new thread is 1, call TZ_Load_Context_S() to update PSP_S and PSPLIM_S. Otherwise, load the following register from TCB:

   - Load callee save registers.
   - FPU registers S16-S31 if the EXC_RETURN bit 4 of the new thread is 0.
7. Load EXC_RETURN and CONTROL_NS from new TCB.

At the end of this sequence, the exception returns to the new thread.
3.4 Impact of the AIRCR.PRIS bit

In the ARMv8-M architecture Secure software can use the programmable Prioritize Secure Exceptions (PRIS) bit in the Application Interrupt and Reset Control Register (AIRCR) to shift the Non-secure exception priority by 1 bit so that the Non-secure exceptions are mapped to lower half of priority levels.

As a result of the PRIS bit, the Non-secure exceptions can have lower priority than the lowest priority Secure exceptions. For example, if Secure PendSV is set to lowest priority, the priority level is 0xC0. For Non-secure exceptions, the lowest priority level is 0xE0. As a result, if Secure PendSV is used for context switching, for example triggered by Secure SysTick exception, the following sequence can occur:

- Non-secure IRQ is running at lowest priority level.
- Secure SysTick is triggered and executed.
- Secure SysTick handler set Secure PendSV pending status.
- Secure SysTick handler exit, tail chained into Secure PendSV.

In this situation, the Secure PendSV must not execute context switching because a Non-secure interrupt handler is still running.

To solve this issue, there are several possible solutions:

- Option 1: Make sure that AIRCR.PRIS is not used.
- Option 2: Check EXC_RETURN and ICSR.RETTOBASE before context switches.
- Option 3: Use Non-secure PendSV at lowest priority level to call a Secure API to handle context switching.

RTOS running in Non-secure state do not have the same issue.
3.5 Supporting multiple Secure software libraries

Secure RTOS designers must also consider cases where the Secure MPU is used for supporting multiple Secure software libraries. Here the Secure MemManage fault is used to control context switching between different libraries. A Secure RTOS must be aware of such MPU context changes.
3.6 Summary

With the ARMv8-M architecture, several processor features are extended, which can affect RTOS designs in several ways. Usually, the RTOS code must be updated to run on ARMv8-M architecture.

There are several different possible systems and RTOS configurations. RTOS vendors might need to implement different configurations of their RTOS to address different product requirements.

To help RTOS operations in devices with locked down Secure memories, CMSIS-RTOS is extended with a new set of APIs. These APIs enable RTOS running in Non-secure domain to handle context switching.