Introduction to the ARMv8-M architecture

Version 2.0

Revision Information

The following revisions have been made to this User Guide.

Date	Issue	Confidentiality	Change
08 July 2016	0100-00	Non-Confidential	First release
01 September 2016	0101-00	Non-Confidential	Second release
20 February 2017	0200-00	Non-Confidential	Third release

Proprietary Notice

Confidentiality Status

This document is 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

1 Introduction to the ARMv8-M architecture

The ARM®v8-M architecture is used for the next-generation ARMv8-M processor family of real-time deterministic embedded processors. It is aimed at low cost deeply embedded systems, where low-latency interrupt processing is vital. The ARMv8-M architecture reduces the complexity of developing secure embedded solutions that scale all the way from the smallest IoT device to complex SoCs.

ARM uses the term architecture for the definitions of such things as the instruction set, programmers' model and memory model, but not implementation details such as pipeline stages. Some of the features of the architecture, including parts of the instruction set, are optional.

ARMv8-M architectural enhancements enable better software design in several ways. For example, the new designs have a simplified programmers' model for the MPU, so setting up a memory region is as simple as setting up a start and end address.

The ARMv8-M architecture provides enhancements to the debug components inside the cores. For example, the programmers' model for breakpoint and watchpoint units permits higher flexibility for breakpoint configuration and improved trace support.

The most significant enhancement in the ARMv8-M architecture is the inclusion of the optional Security Extension. The ARMv8-M architecture Security Extension can also be referred to as ARM TrustZone technology for ARMv8-M. TrustZone technology enables multiple security domains within a single processor system.

With existing ARMv6-M and ARMv7-M processor-based products, an application can execute various software components, such as communication stacks in an unprivileged state, and use the MPU feature to protect the system from memory corruption. In this way, even if the software stack suffers an attack and fails the rest of the system can still be functional because of the separation of stacks between privileged and unprivileged states.

While it is possible to create Secure embedded system designs with processors based on the ARMv6-M and ARMv7-M architectures, systems that require multiple applications or multiple security domains on a single Cortex-M series processor using these architectures can be challenging to design.

Some designs, such as complex SoCs, use multiple Cortex-M-series processors for system management and offloading I/O tasks. Of these, several processors can be in a permanent Secure domain, for example system management, and others in a permanent Non-secure domain, such as offloading of peripheral tasks. It is impractical to run the application entirely in unprivileged state as there are many restrictions on programs executing in this state.

TrustZone technology for the ARMv8-M architecture is designed to simplify such systems without the need for multiple processors, and can enable these systems to be built at lower cost.

The ARMv8-M architecture is a 32-bit architecture. The registers in the register bank, most data operations, and addresses are 32-bit. The 32-bit linear address provides 4GB of address space, which is architecturally pre-defined into several regions with different memory attributes.

Some portions of the memory space are used by the internal components of the processor core, such as programmable registers for:

- Nested Vectored Interrupt Controller (NVIC).

- SysTick timer.
- System Control Block (SCB).
- Memory Protection Unit (MPU).
- Security Attribution Unit (SAU). Present only if the ARMv8-M Security Extension is implemented.
- Debug components.

The rest of the memory space is utilized by chip designers in various ways. Architecturally there is no restriction on what type of memories or peripherals can be connected to the systems, so products from different chip vendors can all have different types of memories and peripherals.

Although the ARMv8-M architecture is 32-bit, it also supports data types of various sizes such as 8-bit and 16-bit. ARMv8-M also supports a limited set of 64-bit operations.

The architecture supports memory access instructions for various sizes, and there are instructions for converting between different data types.

Note

The ARMv8-M architecture does not support the A32 instruction set.

The instruction set supported by M-profile processors is the T32 instruction set, previously called the Thumb instruction set. It contains a range of 16-bit and 32-bit instructions. The ARMv8-M architecture only supports a subset of the T32 instruction set, with mostly 16-bit instructions. The ARMv8-M architecture with Main Extension supports a wider range of instructions, including an optional:

DSP Extension	A range of instructions including SIMD operations targeting DSP applications.
Floating-point extension	Which can be further divided into single-precision and double-precision, or single-precision only options.

Exception and interrupt handling are also defined by the architecture. In ARMv8-M processors, peripheral interrupts are a subset of exceptions. There are also extra system exceptions for:

- Non-Maskable Interrupt (NMI).
- Fault handling exceptions.
- Software exceptions, for example, a Supervisor Call to the OS.
- System Tick timer (SysTick exception).

  The SysTick timer is a 24-bit built-in timer that can be used by the OS to generate periodic interrupts for task scheduling, or by application code for timing control.

2 ARMv8-M subprofiles

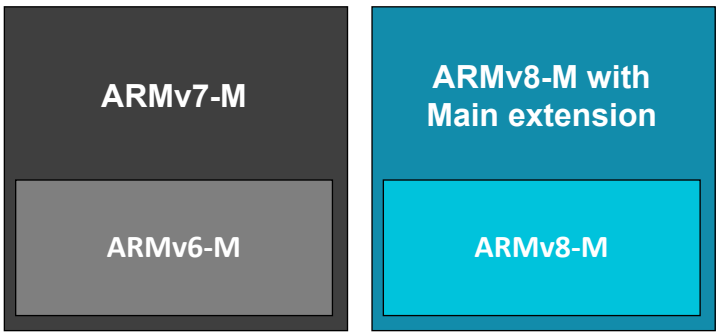
The ARMv8-M architecture is divided into two subprofiles. A processor can implement the ARMv8-M architecture, or it can implement the ARMv8-M architecture with Main Extension.

A processor with the Main Extension is often referred to as a *Mainline* implementation.

A processor without the Main Extension is often referred to as a *Baseline* implementation.

The ARMv8-M architecture is a subset of the ARMv8-M architecture with Main Extension, and has a subset of the instructions, registers, and features of an ARMv8-M implementation with Main Extension.

This similar to the way that the ARMv6-M architecture is a subset of the ARMv7 architecture. This is shown in the following figure:



Both subprofiles support an optional Security Extension that is based on ARM TrustZone technology. This feature enables software components to be partitioned into separated security domains for better security capability.

ARMv8-M processors can use either subprofile, and a number of further optional extensions, to address a wide range of requirements, exceeding the range addresses by ARMv6-M and ARMv7-M processors.

The ARMv8-M architecture

The ARMv8-M architecture extends the ARMv6-M architecture with instruction set and system level feature enhancements.

Instruction set enhancements include:

- Signed and unsigned integer divide.
- Wide immediate moves.
- Compare and branch, and long branch instructions.
- Exclusive memory access instructions to enhance semaphore support for multiprocessor systems.

- Instructions to support TrustZone technology for ARMv8-M.
- Memory access instructions for C++11 atomic data types (load-acquire and store-release instructions).

System level feature enhancements include:

- Support for up to 496 interrupts. For power efficiency reasons, so many interrupts might never be fully implemented. By comparison, the ARMv6-M architecture only supports up to 32 interrupts.
- Interrupt active bits for dynamic reprioritization of interrupts.
- Optional TrustZone technology for the ARMv8-M architecture.
- Improved flexibility in MPU region definition using the Protected Memory System Architecture (PMSA) v8.
- Better debug capability, including enhancements in breakpoint and watchpoint units.

ARMv8-M architecture with Main extension.

The ARMv8-M architecture with Main Extension is the full feature subprofile of the ARMv8-M architecture for mainstream microcontroller products and high performance embedded systems. Additional instructions address the demands of complex data processing.

The ARMv8-M architecture with Main Extension is similar to the ARMv7-M architecture, and in addition to the ARMv8-M architecture fundamentals includes:

- An expanded 32-bit instruction set that can give performance improvements when compared to the ARMv8-M architecture.
- An optional integer DSP for efficient signal processing.
- An optional floating-point Extension architecture with support for single- and optional support for double-precision floating-point operations.
- Optional coprocessor support for hardware acceleration.

The ARMv8-M architecture with Main Extension has the following enhancements over the ARMv7-M architecture, for example:

- Test Target Memory (TT) instruction.
- Memory access instructions for C++11 atomic data types (load-acquire and store-release instructions).

System level feature enhancements include:

- Optional TrustZone technology for the ARMv8-M architecture.
- The optional MPU uses the Protected Memory System Architecture (PMSA) v8 enabling improved flexibility in MPU region definition.
- Better debug capability. Enhancements in breakpoint and watchpoint units.

- Improvements to the *Instrumentation Trace Macrocell* (ITM).
- Comprehensive trace and self-hosted debug extensions to make embedded software easier to debug and trace, dramatically reducing the design effort and release cycle.

3 Operational modes and states

ARMv8-M processors run in different operation modes and states when executing application software, handling exceptions, or when accessing Secure memory.

The following modes and states are available to the processor:

Thread and Handler modes

The processor runs in Thread mode when executing application software, and runs in Handler mode when handling exceptions. The processor enters Thread mode when it comes out of reset, and returns to Thread mode when it has finished processing exceptions.

When in Thread mode, execution can be privileged or unprivileged. When in Handler mode, execution is Privileged.

Secure and Non-secure states

When the core is in Secure state, it can access both Secure and Non-secure memories. If the ARMv8-M architecture with Security Extension is implemented, the processor starts up in Secure state.

When the core is in Non-secure state, it can access Non-secure memories only.

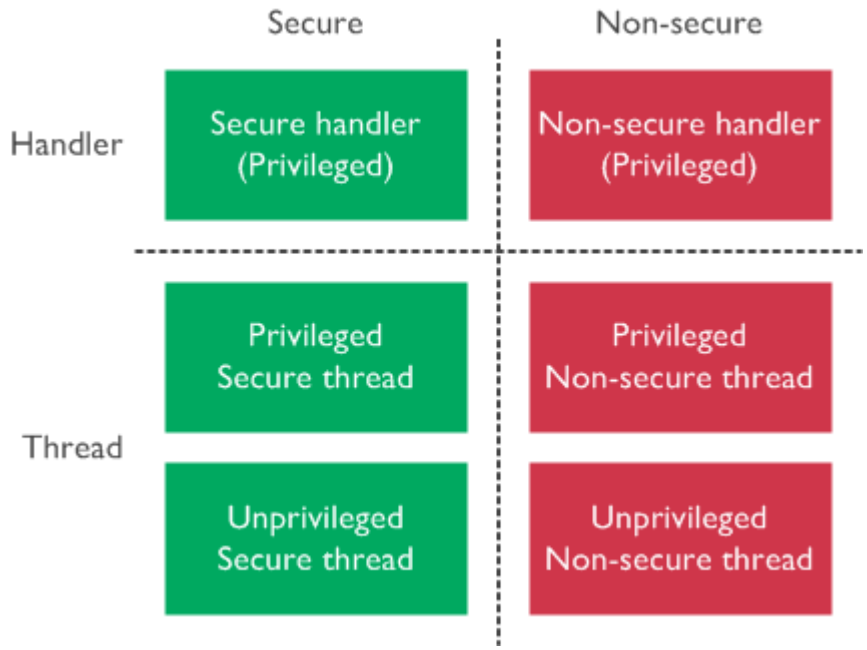
If the ARMv8-M architecture with Security Extension is not implemented, there is no Secure state and the processor starts up in Non-secure state.

Privileged and Unprivileged levels

In Privileged level, the software can use all the instructions and has access to all resources. Privileged software executes at the privileged level. Only privileged software can write to the CONTROL register to change the privilege level for software execution in Thread mode.

In Unprivileged level, the software can use the SVC instruction to make a Supervisor Call to transfer control to privileged software. In unprivileged mode, the software has limited access to instructions that change processor state. It cannot access the system timer, NVIC, or System Control Block, and might have restricted access to memory or peripherals. Unprivileged software executes at the unprivileged level.

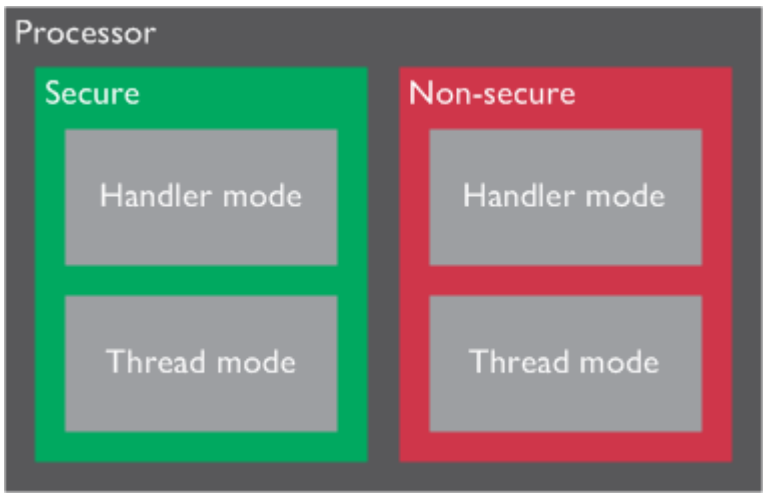
The following figure shows the possible combinations of the processor modes, states, and security levels:



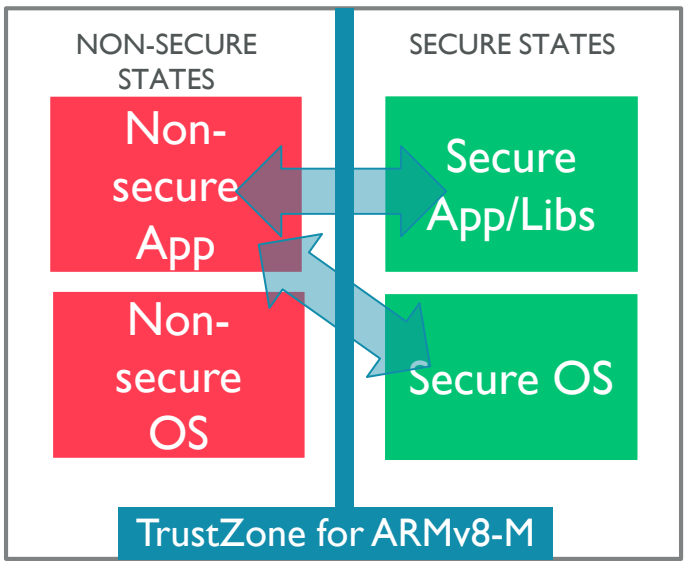
4 Secure and Normal worlds

The Security Extensions divides the system and the software into Secure and Normal worlds.

The following figure shows how TrustZone technology for ARMv8-M adds Secure and Non-secure states to processor operation:



Secure software can access both Secure and Non-secure memories and resources, while Normal software can only access Non-secure memories and resources. These security states are separate from the existing Thread and Handler modes, enabling both a Thread and Handler mode in Secure and Non-secure states.



Note

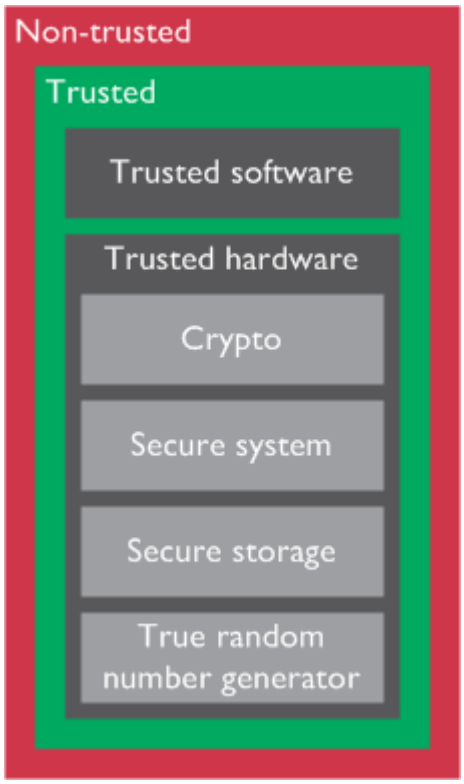
Thread mode can also be either Privileged or Unprivileged.

If the Security Extension is implemented, the system starts up in Secure state by default. If the Security Extension is not implemented, the system is always in Non-secure state.

In designs with the ARMv8-M architecture Security Extension, components that are critical to the security of the system such can be placed in the Secure world. These critical components include:

- A Secure boot loader.
- Secret keys.
- Flash programming support.
- High value assets.

The remaining applications are placed in the Normal world.

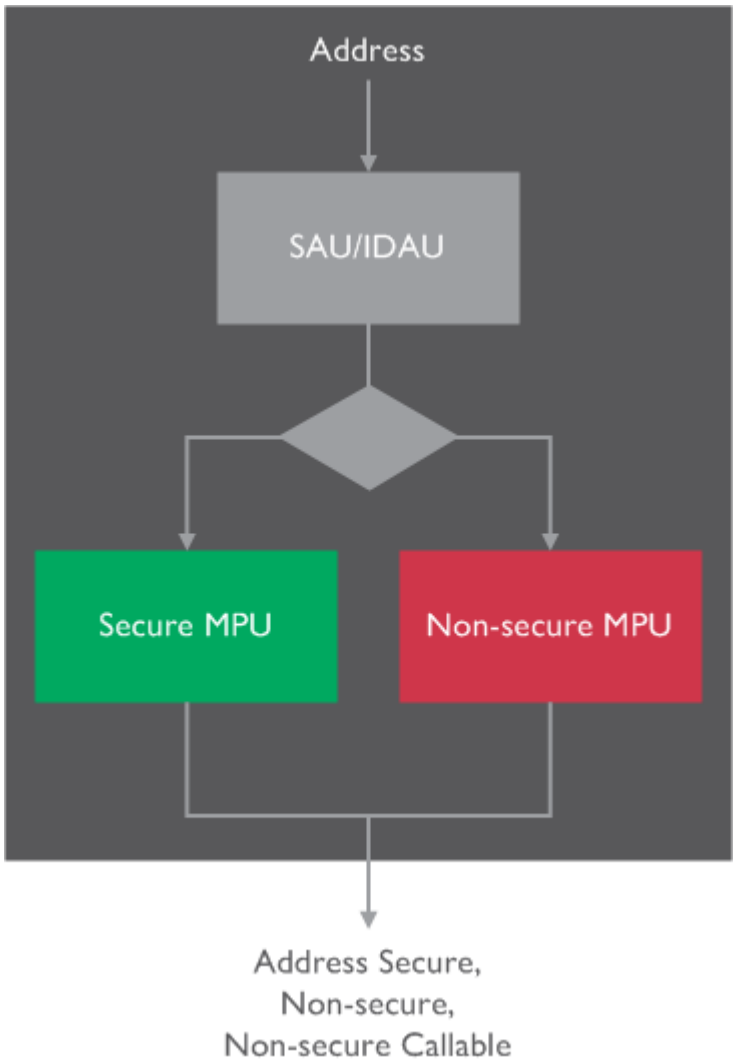


Secure (Trusted) and Non-secure (Non-trusted) software can work together, but Non-secure applications cannot access Secure resources directly. Instead, any access to Secure resources can go through APIs provided by Secure software, and these APIs can implement authentications to decide if the access to the Secure service is permitted. By having this arrangement, even if there are vulnerabilities in the Non-secure applications, hackers cannot compromise the whole chip.

5 Secure and Non-secure MPUs

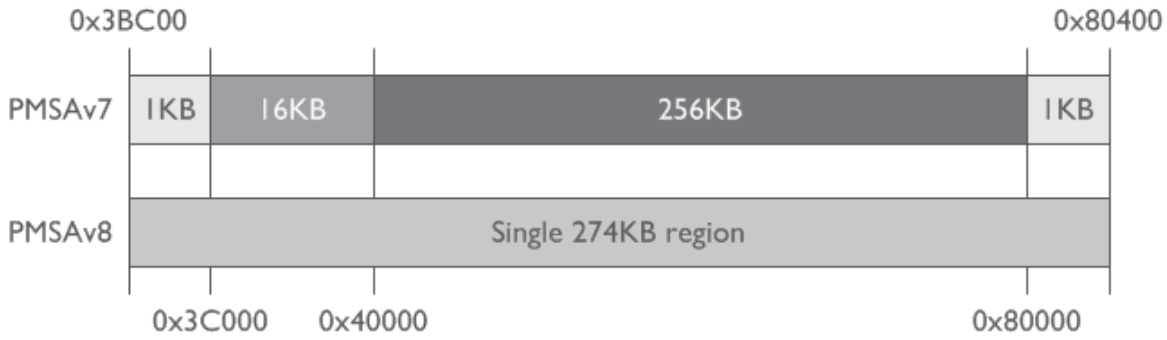
The Memory Protection Unit (MPU) is a programmable unit in the processor that allows privileged software, typically an OS kernel, to define memory access permissions and memory attributes. It monitors memory access, including instruction fetches and data accesses from the processor, and can trigger a fault exception when an access violation is detected

As in earlier M-series processors, the Memory Protection Unit (MPU) is optional. Based on application requirements, designers can exclude the MPU to reduce area and power, or include either Secure or Non-secure MPU, or both if necessary.



Although the concepts for the MPU operations are similar, the MPU in the ARMv8-M architecture has a different programmers' model compared to the MPU in previous architectures.

- The size of an MPU region can be any size with a granularity of 32 bytes.
- The MPU in ARMv6-M and ARMv7-M requires that an MPU memory region must be aligned to an address which is a multiple of the region size, and that the region size must be a power of two, which can be tuned by excluding sub-regions of $^1/_8$ of the region. For example, when creating a memory region from an address 0x3BC00-0x80400, multiple MPU region registers are required, as in the following figure.



- PMSEv8 does not include subregions as the region size is now more flexible.
- Regions are now not allowed to overlap. As the MPU region definition is much more flexible, overlapping MPU regions are not necessary.
- Memory type definitions are different.
- Memory regions define memory attributes using an index value which is then looked up in a set of memory attribute registers.

As the Security Extension was not previously available, legacy configuration code must also be updated to reflect the new features.

6 The ARMv8-M Programmers' model

The ARM architecture provides sixteen 32-bit registers (R0-R15) for software use. Thirteen of these registers (R0-R12) can be used for general-purpose storage, while R13 to R15 have special uses. Additionally, the Combined Program Status Register, the Exception mask registers, and the CONTROL register are designated as special registers in the ARMv8-M architecture.

R13 - Stack pointer

The processor uses full descending stacks, which means that register R13, the stack pointer, holds the address of the last stacked item in memory. When the processor pushes a new item onto the stack, it decrements the stack pointer and then writes the item to the new memory location.

Physically there are four stack pointers when the ARMv8-M architecture with Security Extension is implemented, or two stack pointers when the Security Extension is not implemented. In both the Secure and Non-secure states, the processor implements the main stack and the process stack, with a pointer for each held in independent registers. The stack pointers are banked.

	Secure state	**Non-secure stack**
Main stack pointer	MSP_S	MSP_NS
Process stack pointer	PSP_S	PSP_NS

A programmer does not have to use both main and process stacks for all operations, a single stack can be sufficient. The _S and _NS suffixes identify whether the resource is for the Secure state or Non-secure state.

If the ARMv8-M Security Extension is not implemented there is no Secure state and the stack pointers are MSP and PSP.

R14 - Link Register

The Link Register (LR) is register R14. It stores the return information for subroutines, function calls, and exceptions.

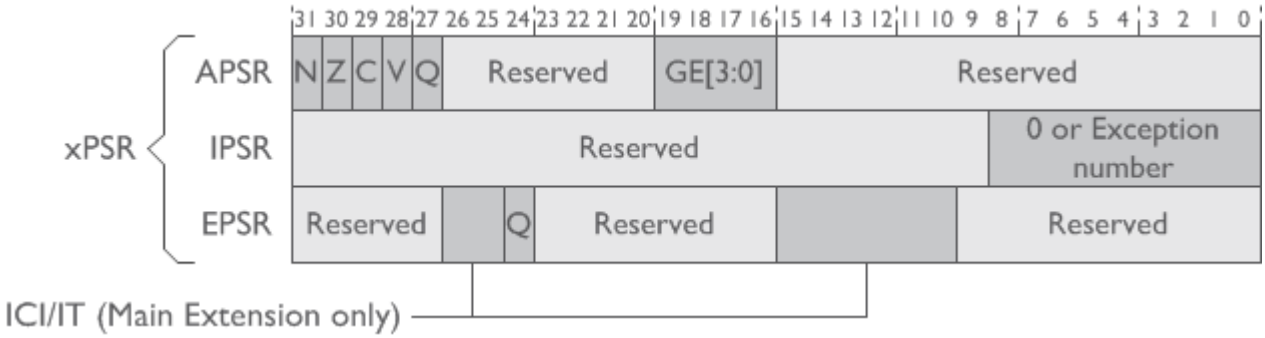
R15 - Program Counter

The Program Counter (PC) is register R15. It contains the current program address. On reset, the processor loads the PC with the value of the reset vector.

Combined Program Status Register

The Combined Program Status Register (XPSR) consists of the Application Program Status Register (APSR), Interrupt Program Status Register (IPSR), and Execution Program Status Register (EPSR) concatenated into a single register to give efficient context saving on and exception entry and exit.

Accesses to the XPSR are really to the APSR, IPSR, and EPSR. In addition, there is a concatenation of these three registers in the xPSR or Combined PSR. The APSR and IPSR are accessed by software.



Application Program Status Register

The APSR contains the current state of the condition flags from previous instruction executions.

Interrupt Program Status Register

The IPSR contains the exception type number of the current Interrupt Service Routine (ISR).

Execution Program Status Register

The EPSR contains the state bit (T) and the execution state bits for either the If-Then (IT) instruction, or the Interruptible-Continuable Instruction (ICI) field for an interrupted load multiple or store multiple instruction.

Exception mask registers

The exception mask registers disable the handling of exceptions by the processor. An example of the use of these registers would be to disable exceptions when they might affect timing critical tasks.

All exception mask registers can be modified using MRS and MSR instructions. The PRIMASK and FAULTMASK registers can be modified with the CPS instruction. The behavior differs depending on whether ARMv8-M architecture with Security Extension is implemented or not. All exceptions have an associated priority, with a low value indicating a high priority. Low priority exceptions take precedence.

Priority Mask Register

The PRIMASK register prevents servicing of all exceptions with configurable priority.

Fault Mask Register (only with ARMv8-M architecture with Main Extension)

The FAULTMASK register prevents servicing of all exceptions except Non-maskable Interrupts, HardFaults, or Resets.

Base Priority Mask Register (only with ARMv8-M architecture with Main Extension)

The BASEPRI register defines the minimum group priority for exception processing. When BASEPRI is set to a nonzero value, it prevents the servicing of all exceptions with the same or lower group priority level as the BASEPRI value.

CONTROL register

The CONTROL register controls the stack that is used and the privilege level for software execution in Thread Mode when the processor is accessed through privileged access only and indicates whether the Floating-point Extension is active.

6.1 Banked internal resources

Several internal resources are banked between Secure and Non-secure states.

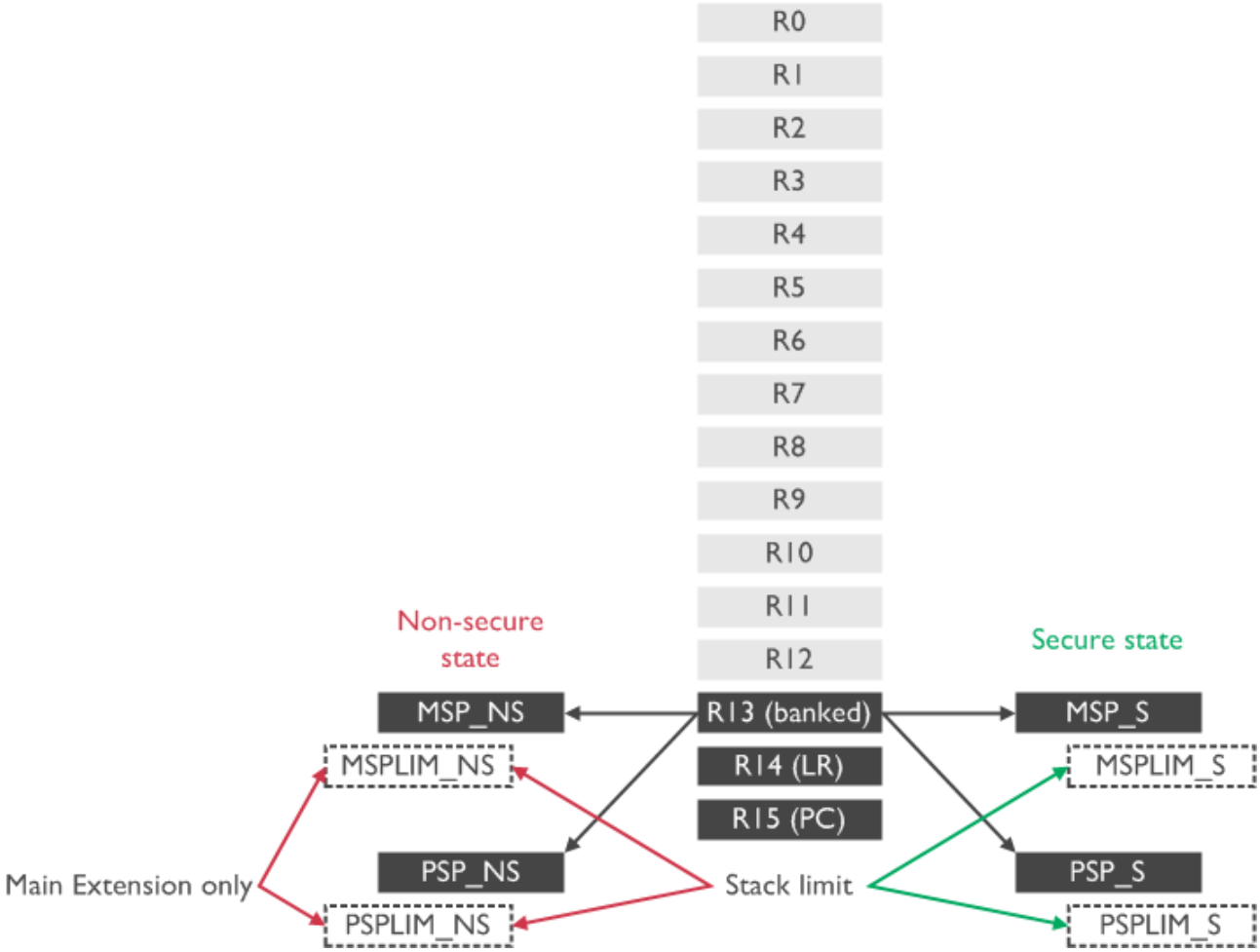
For registers inside the processor core:

- The Stack Pointers are banked between Secure and Non-secure states. This enables separations of Secure and Non-secure stacks.
- Interrupt masking registers like PRIMASK, FAULTMASK, and BASEPRI are banked.
- FAULTMASK and BASEPRI are available on the ARMv8-M architecture with Main Extension only. This allows existing software to be reused, but Non-secure software cannot influence the operation of Secure software.
- Bit 0 and 1 of the special CONTROL register are banked. Secure and Non-secure software can have different Stack Pointer control and privileged settings.

In addition, the MPU, SysTick timer, and some of the registers in the System Control Block (SCB) are also banked. For example, the Vector Table Offset Register (VTOR) is banked to allow the vector tables for Secure software and Non-secure software to be separated. Software that accesses these registers using an existing address accesses the corresponding view of the peripheral based on the current processor state. Secure software can also access Non-secure versions of these components using alias addresses.

6.2 Stack and stack limit registers

To separate Secure and Non-secure stacks, processors that are based on the ARMv8-M architecture support four stack pointers if the ARMv8-M architecture with Security Extension is implemented. In addition, a stack limit feature is provided using stack limit registers (accessible using `MSR` and `MRS` instructions) in Privileged level.



The following table summarizes the ARMv8-M architecture support for stack pointers.

Stack	**Stack pointers**	**Corresponding stack limit register**
Secure Main Stack – used by Secure handlers, and Secure thread if CONTROL_S[1] is 0.	MSP_S	MSPLIM_S
Secure Process Stack – used by Secure threads if CONTROL_S[1] is 1.	PSP_S	PSPLIM_S
Non-secure Main Stack – used by Non-secure handlers, and Non-secure thread if CONTROL_NS[1] is 0.	MSP_NS	MSPLIM_NS (available on ARMv8-M Mainline only)
Non-secure Process Stack – used by Non-secure threads if CONTROL_NS[1] is 1.	PSP_NS	PSPLIM_NS (available on ARMv8-M Mainline only)

Stack Limit registers

If a stack overflow occurs, the ARMv8-M architecture stack limit feature triggers an exception so the error can be handled by software.

The _S and _NS suffixes are used in the ARMv8-M architecture to identify whether the resource is for the Secure state or Non-secure state. Stack limit registers are available for all stack pointers in

the ARMv8-M architecture with Main Extension. In the ARMv8-M architecture the stack limit registers are available for Secure stack pointers only.

The following table summarizes the stack limit registers for the ARMv8-M architecture.

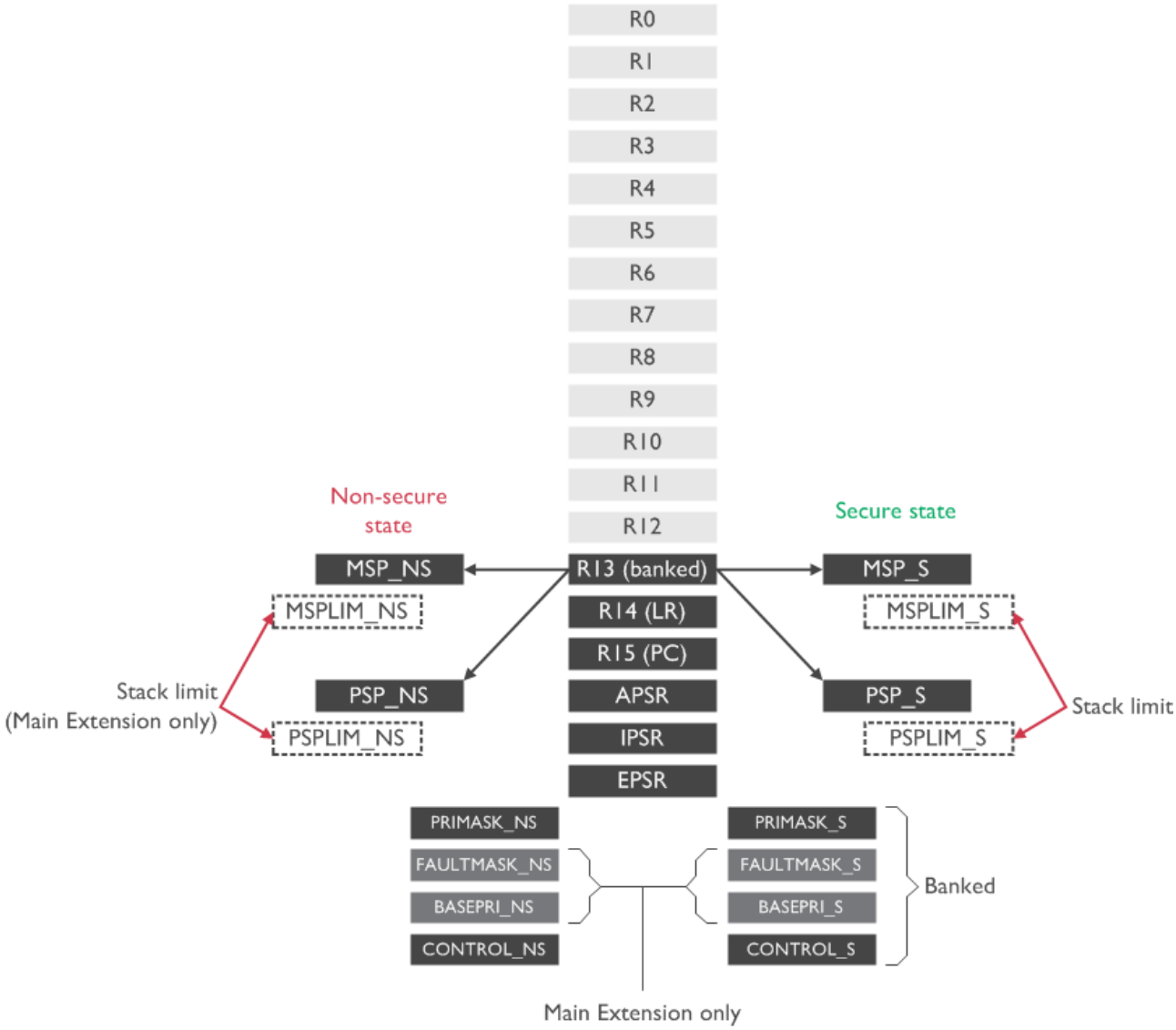
State	Secure	Non-secure
Stack limit configuration for MSP	MSPLIM_S	MSPLIM_NS (ARMv8-M Mainline only)
Stack limit configuration for PSP	PSPLIM_S	PSPLIM_NS (ARMv8-M Mainline only)

6.3 Effect of TrustZone on the Programmer's model

The processor can operate in Secure state or Non-secure state when the TrustZone architecture security extension is enabled. When running a program in Secure memory, the processor is in Secure state. When running a program in Non-secure memory, the processor is in Non-secure state.

With TrustZone technology for the ARMv8-M architecture most of the register banks are shared between Secure and Non-secure states except for stack pointers. Some of the special registers are also banked.

For example, some bits in the CONTROL register which defines the stack pointer selection (MSP/PSP) and privileged state selection are banked.



In the ARMv8-M architecture, the memory space is also partitioned into Secure and Non-secure sections. Only Secure software can have access to the Secure memory and peripherals. An extra level of memory protection can also be implemented by having optional Secure MPU and Non-secure MPU support, which define the memory access permission of privileged and unprivileged software.

The following built-in components inside the processor are banked:

- MPU.
- SysTick timer.
- Some of the registers in the System Control Block (SCB).

Secure software can also access the Non-secure MPU, SysTick, or SCB using alias addresses.