WHITE PAPER

# ARMv8-M Architecture Technical Overview

**10-Nov-2015**

**Joseph Yiu**

Senior Embedded Technology Manager, CPU Product Group, ARM

## Introduction

ARM Cortex®-M Processors are the most popular processor series in the electronics industry. With over 300 licenses, the Cortex-M processors are available in over 3500 microcontroller parts from most of the microcontroller vendors, and are also used in wide range of embedded applications including sensors, wireless communication ASICs, power management ICs and as companion processors within complex SoCs.

The existing Cortex-M processors are based on two architecture versions:

- Cortex-M3, Cortex-M4 and Cortex-M7 are based on ARMv7-M architecture
- Cortex-M0, Cortex-M0+ and Cortex-M1 are based on ARMv6-M architecture

The architecture specifications define the behavior of the processors from both software and debug points of view. For example, the instruction set, programmers' model, exception model, and debug registers, which are visible to debug tools, are all defined by the architecture specifications. Each architecture can result in multiple processor implementations, which in turn can be used in multiple SoC products.

Building on the success of the existing ARMv6-M and ARMv7-M architectures, ARM has created the ARMv8-M architecture. The ARMv8-M architecture remains a 32-bit architecture, and is highly compatible with existing ARMv6-M and ARMv7-M architectures to enable easy migration of software within the Cortex-M processor family.

The partitioning between ARMv6-M and ARMv7-M is essential due to the diverse requirements of embedded processors across different applications. To allow for the continuation of diversity, the new ARMv8-M architecture is divided into two profiles:

- **ARMv8-M Baseline** – A sub-profile of the ARMv8-M architecture for processor designs with low gate count and a simpler instruction set. It is similar to the ARMv6-M but with some significant enhancements. This is ideal for a wide range of ultra low power designs.
- **ARMv8-M Mainline** – This is the full feature sub-profile of the ARMv8-M architecture for mainstream microcontroller products and high performance embedded systems. It has a richer instruction set to address the demands in complex data processing. It is similar to the ARMv7-M but with additional enhancements.

By having two different profiles, different type of processors can be designed to address different requirements, as addressed by the range of Cortex-M processors available today. The Baseline sub-profile is a subset of the Mainline sub-profile and by including even more in this sub-profile (compared to ARMv6-M), it is even easier to migrate application code between the sub-profiles.

As in previous versions of the architecture, the specification does not restrict the implementation details. Each of the future generation Cortex-M processors can implement additional implementation specific features. This document outlines the enhancements in ARM's M-profile architecture and does not cover processor specific implementation details.

## Architectural Enhancements Overview

A range of enhancements have been incorporated in the ARMv8-M architecture. One of the key enhancements available in both ARMv8-M Baseline and ARMv8-M Mainline is a security extension called TrustZone® technology.

The addition of connectivity in to embedded systems is enabling a whole new range of connected intelligent devices. One of the key challenges of this demand for connectivity is the security of embedded devices. To help address the security requirements of embedded devices, ARM developed TrustZone for ARMv8-M, and is included as part of the ARMv8-M architecture.

Conceptually TrustZone for ARMv8-M is similar to the TrustZone technology found in ARM Cortex-A Processors. The underlying operations of TrustZone for ARMv8-M are however very different as they are optimized for embedded systems that requires real-time responsiveness, whilst at the same time allowing for high energy efficiency and low silicon area overhead.

In addition to TrustZone technology, there are a range of additional architecture enhancements.

ARMv8-M Baseline enhancements include:

- Hardware divide instructions
- Compare and branch and 32-bit branch instructions

- Exclusive access instructions
- 16-bit immediate data handling instructions (MOVW, MOVT)
- Load acquire, store release instructions (C11 atomic variable handling)
- New instructions for TrustZone technology support
- Support for more interrupts
- New style MPU (Memory Protection Unit) programmer's model. The 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.

Many of the instructions added to the ARMv8-M Baseline (relative to ARMv6-M) are available in the ARMv7-M architecture.

ARMv8-M Mainline enhancements (relative to ARMv7-M) include:

- Load acquire, store release instructions (C11 atomic variable handling)
- Floating point extension architecture v5 (Cortex-M4 processor is based on FPv4)
- New instructions for TrustZone technology support
- New style MPU programmer's model - using the Protected Memory System Architecture (PMSA) v8 – enabling improved flexibility in MPU region definition
- Better debug capability – enhancements in breakpoint and watchpoint units.

These architectural enhancements enable better software design in a number of ways. For example, under PMSAv8 the new MPU programmer's model removed some of the previous restrictions in the definition of MPU memory regions.

For example, the MPU in ARMv6-M / ARMv7-M requires that an MPU memory region starts from an address which is a multiple of the region size, and the region size must be a power of two. Thus, when creating a memory region from an address 0x3BC00 to 0x80400, multiple MPU region registers are required, as shown in Figure 1.

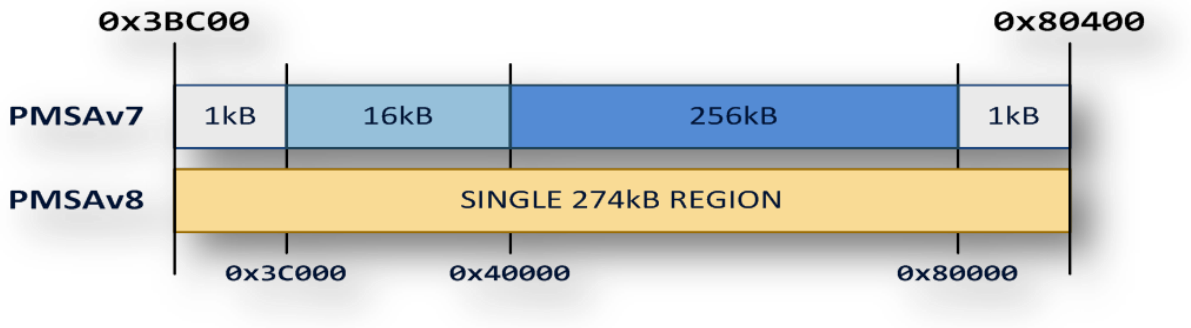


Figure 1:PMSAv8 enable flexible memory region definitions

With PMSA v8, memory regions are defined using the starting address and ending address (with 32 byte region granularity). Thus the same memory region can be defined by using only one set of MPU region registers.

Comparing ARMv8-M Baseline to ARMv6-M architecture, there are a number of new instructions taken from ARMv7-M:

| Instructions | Advantages |
|---|---|
| **Hardware divide** | Faster integer-divide operations and removes the needs for run time library functions for integer divide handling |
| **Compare and branch** | More efficient conditional branches in some cases |
| **32-bit branch instruction** | Longer branch ranges |
| **Exclusive accesses** | Better semaphore support for multi-processor systems |

The inclusion of MOVW and MOVT instructions allows larger immediate values to be generated with a fewer numbers of instructions. This also enables a firmware protection technique called eXecute-Only-Memory (XOM, Figure 2) to be used efficiently. Some embedded systems protect on-chip firmware with XOM which only permits instruction fetches and hence the data cannot be read using data/debug accesses. In this way, application code can utilize the on-chip firmware by calling an API, but the on-chip firmware cannot be reverse engineered.

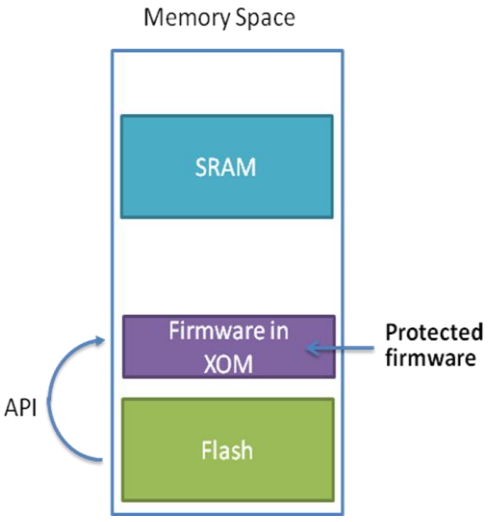


Figure 2:XOM technique for firmware protection

Since data read is not possible for such protected firmware, these program images cannot contain

literal data and hence constant data is generated with instructions with immediate data. The inclusion of MOVW and MOVT enables large immediate values to be generated efficiently.

The load acquire and store release instructions are for handling atomic data types as defined in the C11 standard. These memory access instructions implement the release-consistency (RCsc) memory model, reducing the need for explicit memory barrier instructions. The enhancements added include load and store instructions for different data sizes, as well as exclusive variants. Support for load-acquire and store-release was first available in the ARMv8-A architecture, and is added to ARMv8-M for consistency across architectures. While most microcontroller application code today might not use C11 atomic types, they are increasingly becoming common in high-end software development and inclusion of these instructions help make next generation Cortex-M products even more future proof.

## Overview of security technologies in embedded processors

The most significant enhancement in the ARMv8-M architecture is the inclusion of the TrustZone security extension, a technology that adds a new dimension of security control allowing multiple security domains within a single processor system, and which cannot be implemented with legacy processor solutions. It is important to understand that it is possible to create secure embedded system designs with processors based on the ARMv6-M and ARMv7-M architectures – TrustZone technology enhances the security and simplifies development. It is also important to understand that TrustZone technology and the anti-tampering features in ARM SecurCore processor products are independent technologies.

The following table explains various forms of security in embedded systems.

| Security feature level | Solutions |
|---|---|
| **Application level security**<br><br>The processor based system is created with the application software executing in a single security domain. Interfaces to the system are designed with security in mind, and communication with other systems (e.g. cloud server) is protected with sufficient security mechanisms (authentication, encryption, etc). | All existing processors can achieve this providing that the software does not have vulnerabilities and the communications and interfaces use appropriate protection methods. |
| **Privilege level security**<br><br>OS kernel and application code are partitioned into privileged and unprivileged states, with various types of access restrictions imposed on unprivileged application tasks (e.g. memory protection using MPU). This provides a safety net that prevents a single point of failure in an unprivileged task from leading to a complete | Processors with privileged and unprivileged execution states, with MPU or MMU for memory protection. |

| | |
|---|---|
| compromise of the system. | |
| **TrustZone security**<br>An additional security state to allow full isolation of two security levels. The processor architecture is designed with additional security state(s). This, combined with hardware level protection measures, makes it impossible for normal software (in Non-Secure state) to access or modify information (both instruction and data) in Secure state(s). This enables a root of trust in an embedded system, and addresses a wider range of security requirements in next generation embedded products. | ARM Cortex-A Processors and next generation Cortex-M Processors. |
| **Anti-tampering security**<br>Product system level and chip level design techniques to prevent lab environment hacking attempts. Processors are designed with features to prevent physical form of information leaks such as power and timing signatures. | ARM SecurCore processors and specialized chip design techniques such as memory contents encryption / scrambling. |

With the existing Cortex-M0+, Cortex-M3, Cortex-M4 and Cortex-M7 based products, an application can execute various software components such as communication stacks in unprivileged states and use the MPU feature to protect system from memory corruptions. In this way, even if the software stack suffers from an attack and fails, the rest of the system can still be functional because of the separation of privileged and unprivileged states.

Systems that require multiple applications or multiple security domains on a single Cortex-M processor can be challenging to design. This is because it is often impractical to run the application entirely in unprivileged state as there are many restrictions on programs executing in this state. Some designs, for example a complex SoC, use multiple Cortex-M processors for system management and offloading I/O tasks. Thus, some of these Cortex-M processors can be in a permanent Secure domain (e.g. system management) and others in a permanent Non-Secure domain (e.g. offloading of peripheral tasks). TrustZone for ARMv8-M is designed to simplify such systems without the need for multiple processors, and potentially allow these systems to be built at lower cost.

## Introducing TrustZone® for ARMv8-M

TrustZone for ARMv8-M adds an extra state to the operation of the next generation Cortex-M processors so that there is both a Secure and Non-Secure state. These security states are orthogonal to the existing Thread and Handler modes, thereby having both a Thread and Handler mode in both Secure and Non-Secure states. Note that the Thread mode can also be either Privileged or Unprivileged.

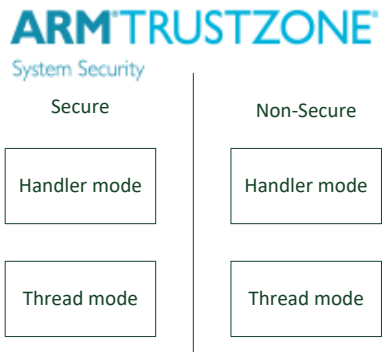


**Figure 3: TrustZone for ARMv8-M adds Secure and Non-Secure states to a processor's operation**

Similar to TrustZone in Cortex-A processors, code running in Secure state can access both Secure and Non-Secure information, whereas Non-Secure programs can only access Non-Secure information.

TrustZone for ARMv8-M is an optional architecture extension. By default the system starts up in Secure state if TrustZone security extension is implemented. If TrustZone security extension is not implemented, the system is always in Non-Secure state.

TrustZone for ARMv8-M is designed with small energy efficient systems in mind. Unlike TrustZone in Cortex-A processors, the division of Secure and Non-Secure worlds is memory map based and the transitions takes place automatically without the need of a Secure Monitor exception handler, thus eliminate switching overhead.

A designer of a microcontroller or SoC device defines the memory spaces into Secure and Non-Secure areas. Some of the regions can be defined by software using a new unit inside the processor called the Security Attribution Unit (SAU), or by device specific controller logic connected to a special Implementation Defined Attribution Unit (IDAU) interface on the processor.

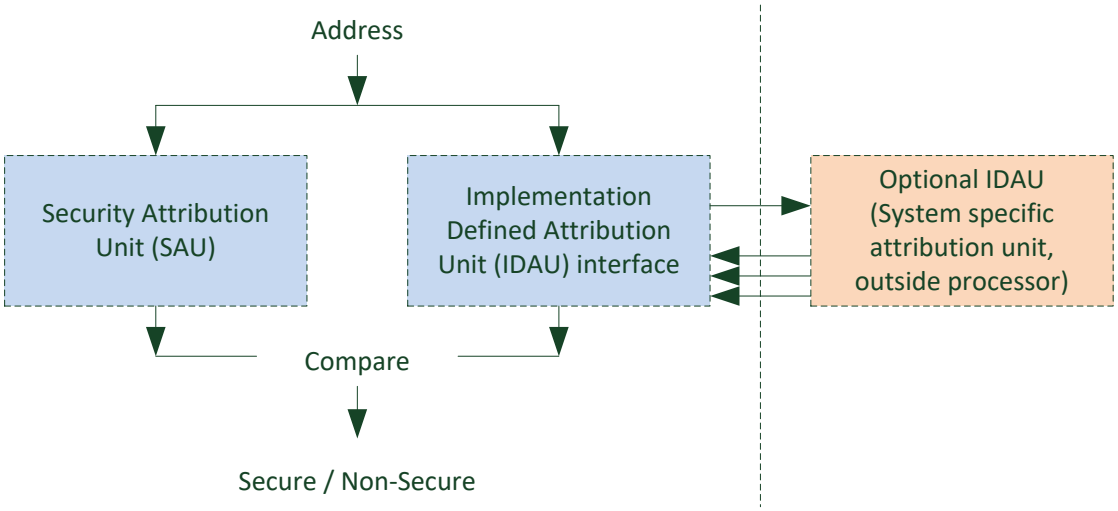


**Figure 4: Security attribute defined by optional SAU and IDAU**

The SAU is programmable in Secure state and has a programmer's model similar to the Memory Protection Unit (MPU). The SAU implementation is configurable by chip designers: it is always presented but the number of regions is defined by the chip designer. Alternatively, chip designers can use an IDAU to define a fixed memory map, and use a SAU optionally to override the security attributes for some parts of the memory.

The processor state is dependent on the memory space definition: when the processor is running code in a Secure region it is in Secure state, otherwise it is in Non-Secure state. Application code can branch to/call code in the other domain, and the processor detects the security domain switches automatically. Since an application can access functions in the other domain directly, TrustZone for ARMv8-M is easy to use and is very flexible.

The Secure memory space is further divided into two types:

- Secure – contains Secure program code or data (including Secure stack, heap and any other Secure data)
- Non-Secure Callable (NSC) – contains entry functions (e.g. entry point for APIs) for Non-Secure programs to access Secure functions.

Typically NSC memory regions contain tables of small branch veneers (entry points). In order to prevent Non-Secure applications from branching into invalid entry points, a new instruction called SG (Secure Gateway) is introduced. When a Non-Secure program calls a function in the Secure side:

- The first instruction in the API must be an SG instruction.
- The SG instruction must be in a NSC region (defined by the SAU or IDAU).

The reason for introducing NSC memory is to prevent other binary data (e.g. a look up table) which has a value the same as the opcode as the SG instruction being used as an entry function in to the Secure state. By separating NSC and Secure memory types, Secure program code containing binary data can be securely placed in a Secure region without direct exposure to the Non-Secure world, and can only be accessed via valid entry points in NSC.

If a Non-Secure program attempts to branch / call into a Secure program address without using a valid entry point, a fault event is asserted. In ARMv8-M Mainline, a new fault exception type SecureFault is introduced (exception number 7). For ARMv8-M Baseline, the fault event is handled by HardFault in Secure state.

Two other new instructions are introduced for Secure state programs to switch to Non-Secure state: BXNS and BLXNS. When a Non-Secure program calls a Secure API, the API completes by returning to Non-Secure state using a BXNS instruction.

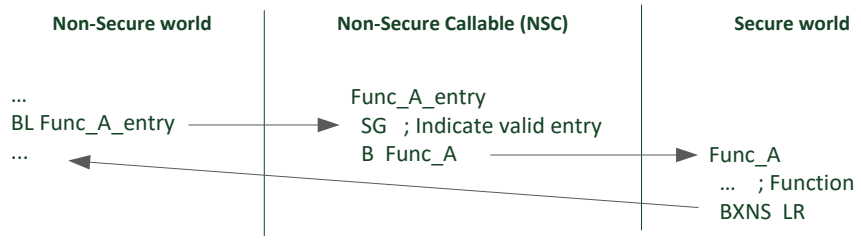


Figure 5: Software flow when Non-Secure program call a function (e.g. API) in Secure side

TrustZone for ARMv8-M also allows a Secure program to call Non-Secure software. In such case, the Secure program use a BLXNS instruction to call a Non-Secure program. During the state transition, the return address and some processor state information are pushed onto the Secure stack and the return address on the Link Register (LR) is set to a special value called FNC_RETURN:

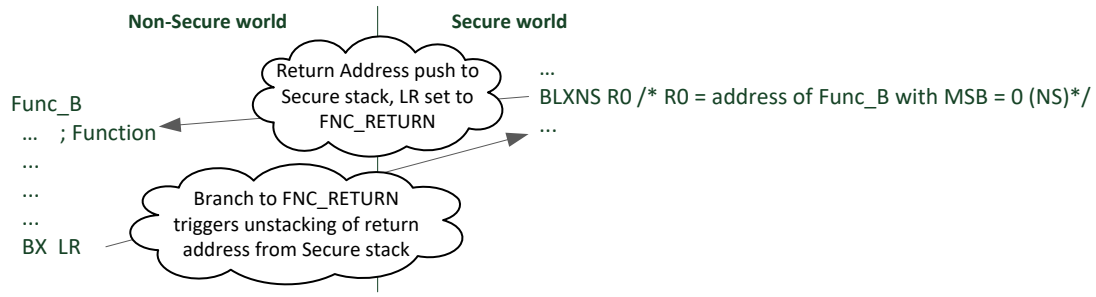


Figure 6: Software flow when a Secure program (e.g. middleware) calls a Non-Secure function (e.g. hardware driver)

The Non-Secure function completes by performing a branch to the FNC_RETURN address; this automatically triggers the unstacking of the true return address from the Secure stack and returns to the calling function. The state transition mechanism automatically hides the return address of the Secure software. Secure software can choose to transfer some of the register values to the Non-Secure side as parameters, and clears other secure data from the register banks before the function call.

State transitions can also happen due to exceptions and interrupts. Each interrupt can be configured as Secure or Non-Secure, as determined by a register which is programmable from the Secure side only, called the Interrupt Target Non-Secure (NVIC_ITNS) register. There are no restrictions regarding whether a Non-Secure/Secure interrupt can take place when the processing is running Non-Secure/Secure code.

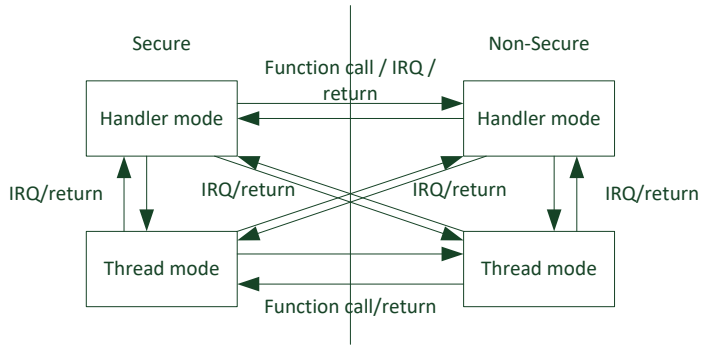


**Figure 7: Various forms of transition between Secure and Non-Secure worlds in TrustZone for ARMv8-M**

If the arriving exception/interrupt has the same state as the current state, then the exception sequence is almost identical to the current Cortex-M processors, allowing very low interrupt latency. The main difference occurs when a Non-Secure interrupt takes place, and is accepted by the processor during execution of Secure code. In this case the processor automatically pushes all Secure information onto the Secure stack and erases the contents from the register banks, thus avoiding an information leak.

All existing interrupt handling features such as nesting of interrupts, vectored interrupt handling, and vector table relocation are supported. TrustZone for ARMv8-M maintains the low interrupt latency characteristics of the existing Cortex-M processor family, with only Secure to Non-Secure interrupts incurring a slightly longer interrupt latency due to the need to push all Secure contents to the Secure stack.

The enhancement of the exception model also works with the lazy stacking of registers in the floating point unit (FPU). In the Cortex-M4 and Cortex-M7, lazy stacking is used to reduce the interrupt latency in exception sequences so that stacking of floating point registers is avoided unless the interrupt handler also uses the FPU. In the ARMv8-M architecture, the same concept is applied to avoid the stacking of the Secure floating point context. In the case that Secure software does use the FPU and Non-Secure interrupt handler does not use FPU, the stacking and unstacking of FPU registers are skipped to provide a faster interrupt handling sequence.

TrustZone for ARMv8-M brings a number of additional registers and components to the processors, examples include:

- A TrustZone capable ARMv8-M processor has four stack pointers: MSP_S (Secure Main Stack Pointer) and PSP_S (Secure Process Stack Pointer) for Secure state, and MSP_NS and PSP_NS for Non-Secure state.
- The processor can optionally implement two SysTick timers (banked), one for each state.
- There can be two separate sets of MPU configuration registers (the number of regions implement can be different between the two states).
- A number of processor internal registers (e.g. a number of registers in the System Control Space (SCS) are banked). For example, the Vector Table Offset Register (VTOR) is banked

to allow two separated vector tables (one in Secure memory, the other one in Non-Secure memory) for handling of Secure and Non-Secure exceptions and Interrupts.

As part of TrustZone for ARMv8-M, a stack limit checking feature is also added. This detects the erroneous case where an application uses more stack than expected, which can potentially cause a security lapse as well as the possibility of a system failure. For ARMv8-M Mainline, all stack pointers have corresponding stack limit registers. For ARMv8-M Baseline, Secure stack pointers have corresponding stack limit registers; Non-Secure programs can use the MPU for stack overflow prevention.

In order to allow software to determine the security attribute of a memory location, a new instruction called TT (Test Target) is introduced. For each memory region defined by the SAU and IDAU, there is an associated region number generated by the SAU or by the IDAU. This region number can be utilized by software to determine if a contiguous range of memory shares common security attributes.

The TT instruction returns the security attributes and region number (as well as MPU region number) from an address value. By using a TT instruction on the start and end addresses of the memory range, and identifying that both reside in the same region number, software can quickly determine that the memory range (e.g. data array or data structure) is located entirely in Non-Secure space (Figure 8). Note: Unlike MPU regions in ARMv6-M and ARMv7-M, SAU/IDAU in ARMv8-M does not allow overlapping of regions.

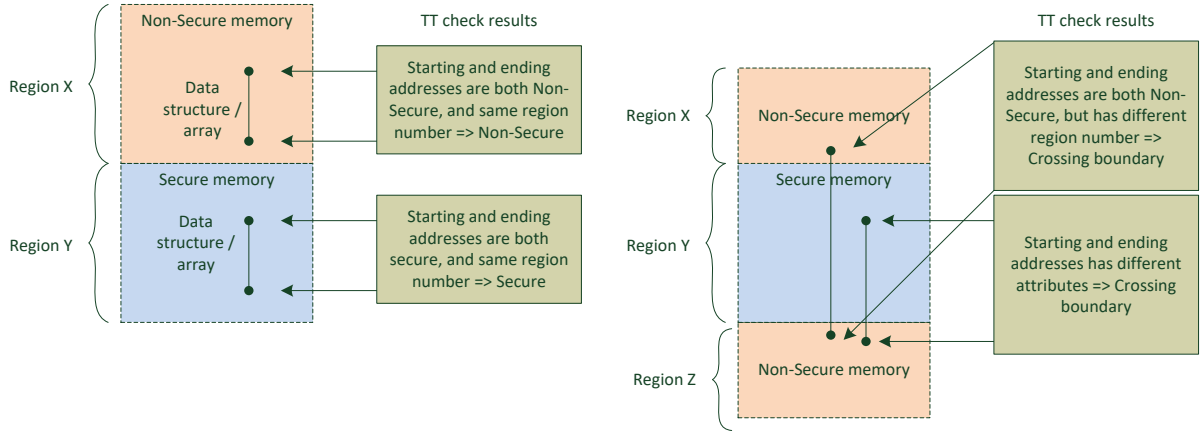


**Figure 8: TT instruction allow software to determine if a data object is placed entirely in a Non-Secure region**

Using this mechanism, secure code servicing APIs in to the Secure side can determine if the memory referenced by a pointer from Non-Secure software has the appropriate security attribute for the API. This prevents Non-Secure software from using APIs in Secure software to read out or corrupt Secure information.

Many additional security checking mechanisms are added as part of the ARMv8-M architecture. It is impossible to cover all details here, additional documents including the ARMv8-M architecture specification will be released in the near future.

## System Design with TrustZone for ARMv8-M

In a TrustZone capable ARMv8-M microcontroller or SoC, the processor is only part of the story. Additional hardware enhancements are needed to ensure Security integrity at a system level. In order to allow security awareness at the system level, ARM has released the AMBA® 5 AHB5 specification, which is an enhancement of the AHB Lite specification from AMBA 3.0. The AHB5 specification adds the HNONSEC signal to indicate if a bus transaction is Secure (HNONSEC=0) or Non-Secure (HNONSEC=1). In addition, AHB5 contains the following enhancements:

- Adding exclusive access sideband signals (HEXCL and HEXOKAY) and HMASTER.
- Expanding the HPROT signal to 7 bits to include additional memory attributes
- Adding User signals for system specific sideband signal requirements (e.g. for error correction code)
- A bus slave can have multiple select bits (HSEL) to allow multiple views of a bus slave when accessed from different address ranges
- Adding a number of properties to define bus components capabilities to allow ESL (Electronic System Level) design tools to handle different components correctly
- A range of clarifications.

The inclusion of HNONSEC allows a TrustZone capable ARMv8-M system to work together with TrustZone based systems containing Cortex-A processors using an AXI interconnect (which uses A*x*PROT[1] as security state indication).

Within a system level design, additional components are needed, for example, to allow memory blocks to be partitioned into Secure memory regions and Non-Secure memory regions. Similarly, access permission control logic is needed to manage access permission of peripherals. Legacy peripherals and legacy bus masters can be reused with appropriate bus wrapper logic.

## Usage of TrustZone technology in ARMv8-M processors

TrustZone for ARMv8-M can be utilized in a number of ways:

**Firmware protection** – microcontroller vendors are increasingly adding firmware to their products to enhance their overall value and to make it easier for their customers to create applications. This firmware can include valuable IP which needs to be protected from reverse engineering or IP theft. TrustZone technology enables such protection by allowing chip vendors to put their firmware in protected, Secure memory space, while still allowing users to use the firmware via APIs.

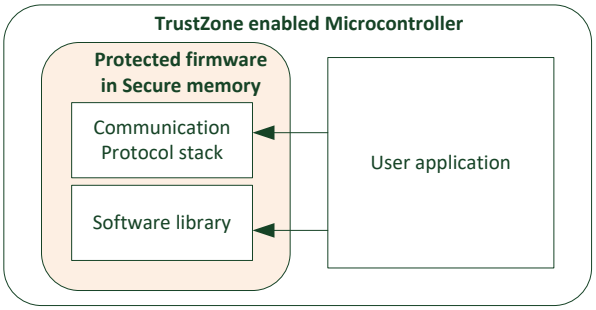


Figure 9: TrustZone technology for firmware IP protection

Microcontroller vendors can also create microcontrollers with blank flash in protected memory space, and provide a means to allow third parties to integrate firmware in protected Secure memory space. The device can then be sold to system developers for end user application development with the firmware protected.

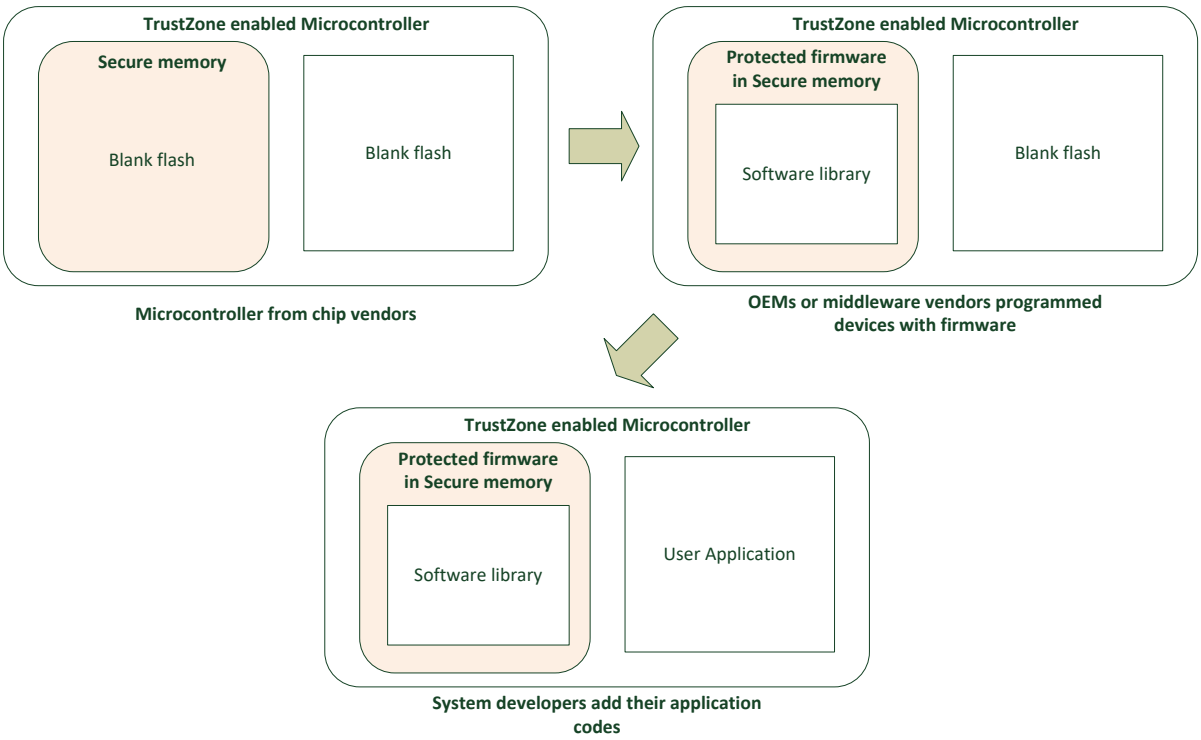


Figure 10: TrustZone technology enables a secure way to distribute firmware/middleware

**Security management in IoT devices** – Many IoT devices need to handle security sensitive information such as user information and security keys. TrustZone technology allows this information and associated firmware (that can have direct access to this data) to be stored in protected Secure memory space. TrustZone technology enables the application code running in Non-Secure mode access to the Secure information via predefined APIs only and (if provided in the Secure software) via an authentication process.

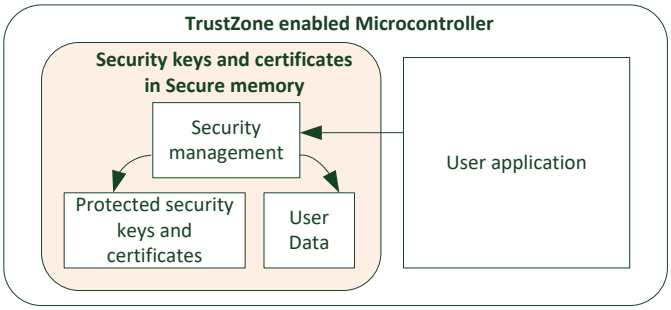


Figure 11: TrustZone technology for protection of sensitive data

**Root of Trust implementation** – Connected devices with authentication requirements require a root of trust in the system architecture. This is particularly important for devices that can be updated over the air. In a system with TrustZone technology, code providing firmware-update support and associated authentication can be placed in secure space and hence protected. Even if a device is compromised at the application level it cannot be wiped out and replaced with spurious firmware.

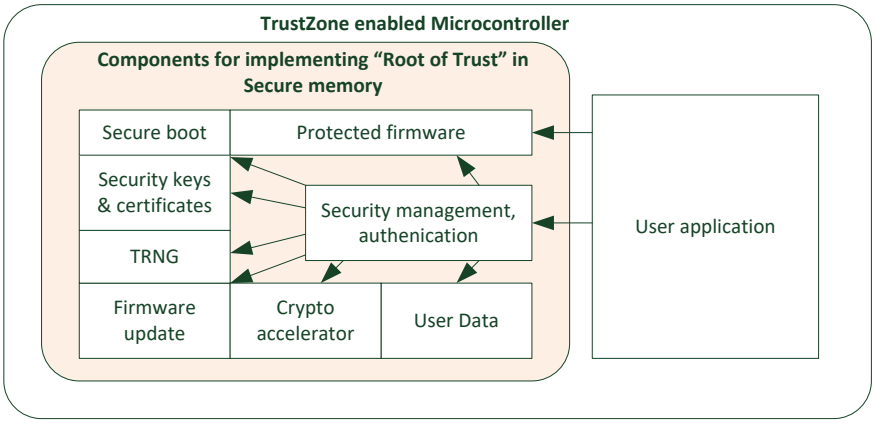


Figure 12: Use of TrustZone technology to implement "Root of Trust"

**Sandboxing for devices with certified software** – Many ASSPs, such as a Bluetooth® chipset, contain preloaded software while also allowing developers to add their own software components. Using TrustZone technology, the preloaded firmware can be placed in the Secure side and its behavior prevented from being altered by applications running on the Non-Secure side. This helps in ensuring that certified firmware remains in its certified state. In addition placing the firmware in the Secure side of ARMv8-M architecture based processor helps protect it from being reverse engineered.

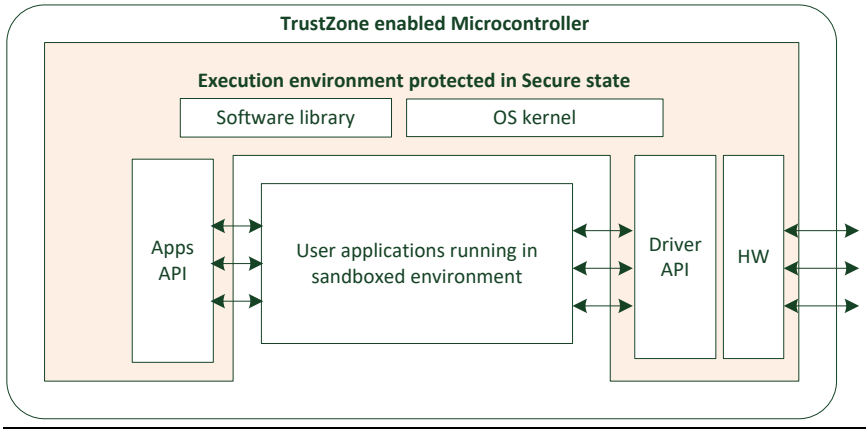


Figure 13: TrustZone technology for creating sand-boxed execution environments

**Consolidation of multiple helper processors in complex SoCs** – Complex application processor SoCs often contain Cortex-M processors for off-load or for managing system functions. Sometimes multiple Cortex-M processors are used to isolate secure software from non-secure software. TrustZone technology makes it possible to merge secure and non-secure processors to reduce system cost and software complexity.

## Software migration and software enablement

ARMv8-M and TrustZone technology affects Cortex-M development tools and middleware in a number of ways. Changes include enhancements in instruction set support and in the programmer's model of debug support components. In addition, an RTOS will require changes to take advantage of the TrustZone security extension. A number of activities to support the Cortex-M ecosystem have already started in ARM. These include:

- Update of ARM compiler and gcc to support the instruction set enhancements
- Update of ARM C Language Extension (ACLE) to support software development in TrustZone technology
- Update of CMSIS-CORE to support future generation processors
- Update of CMSIS-RTOS API specification to support RTOS operations in microcontrollers with TrustZone for ARMv8-M
- ARM mbed™ OS enhancement

ARM is in the process of informing third party ecosystem partners about ARMv8-M and TrustZone technology, and some partners have already started developing their next generation products to support ARMv8-M. If you are developing software development tools or middleware for ARM Cortex-M microcontrollers and would like to start working on enhancing your products to support ARMv8-M, please contact the ARM ecosystem partner management team: [www.arm.com/contact-us/](www.arm.com/contact-us/)).

## Summary

The next generation of ARM Cortex-M processors will be based on the ARMv8-M architecture. The new architecture is upward compatible with existing ARMv6-M and ARMv7-M architectures. The new architecture contains a range of enhancements including:

- Additions to the instruction set
- Protected Memory System Architecture (PMSAv8)
- TrustZone security extensions
- Debug support enhancements

The TrustZone security extension adds Secure and Non-Secure states to the processor's operation. It allows multiple security domains to be established within a single processor system and can be utilized in many different ways.

Updates to tools, middleware and RTOS are already underway and ARM welcomes ecosystem partners to start working on ARMv8-M. Please contact the ARM ecosystem partner management team for more information.