embeddedworld2016
Exhibition&Conference
...it's a smarter world

The Next Steps in the Evolution of Embedded Processors for the Smart Connected Era

Joseph Yiu
Senior Embedded Technology Manager,
CPU Group, ARM
Cambridge, United Kingdom
Version 1.1. April-2016

***Abstract*— The next era of embedded products is demanding more from embedded processors. The breadth of chip solutions is growing as system designers demand higher integration and right-fit solutions. Security in the connected era has become a key requirement, and the demand for secure software and crypto functions has never been greater.**

This technical presentation looks at the capabilities of the next generation of microcontroller products, looking at how they address the needs of system designers and software developers so that they can create the energy efficient, scalable and secure systems that their customers demand.

Keywords—ARMv8-M Architecture; embedded processors; IoT; security; TrustZone

1 BACKGROUND

ARM® Cortex®-M processors are used in around 11 billion devices (as of the end of 2014) and their uses span everything from generic microcontrollers to numerous custom System-on-Chip (SoC) designs. The current generations of Cortex-M processors are based on the ARMv6-M and the ARMv7-M architectures. While ARMv6-M and the ARMv7-M architecture based SoCs will continue to power many embedded products, ARM has developed a new version of the architecture to prepare for future embedded application requirements. In November 2015, ARM announced the ARMv8-M architecture, an architecture version which will be used in the next generation of the ARM Cortex-M processors. This paper focuses on the security enhancements of this new architecture.

2 INTRODUCTION TO THE ARMV8-M ARCHITECTURE

ARM uses the term 'architecture' for the definitions of the instruction set, programmer's model, memory model, etc. The architecture specification does not, however, include exact implementation details (micro-architecture), such as pipeline stages. Typically some of the features of the architecture, including part of the instruction set, are optional. For example, both Cortex-M3 and Cortex-M7 processors are based on the ARMv7-M architecture, but have different features, pipeline designs and performance.

The ARMv8-M architecture is divided into two sub-profiles (Figure 1), similar to the partitioning of ARMv6-M and ARMv7-M today.

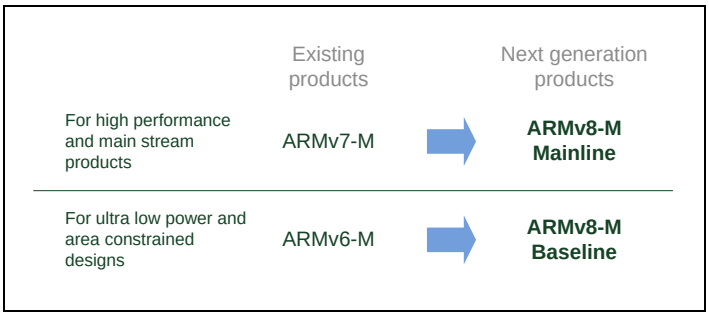

Figure 1: Separation of sub-profiles in ARMv8-M architecture

Existing Cortex-M processors cover a wide spectrum of applications. The separation of sub-profiles in ARMv8-M enables the next generation Cortex-M processors to continue to address broad and diverse use cases, while at the same time enabling easier migration of designs across the sub-profiles.

Both sub-profiles retain almost all of the characteristics of Cortex-M processors today, for example:

- A 32-bit architecture.
- An unchanged architecturally predefined memory space.
- A Nested Vectored Interrupt Controller (NVIC) for interrupt and exception handling, with capability for handling interrupts with low latency.
- Architectural sleep modes support.
- Optional floating point support in ARMv8-M Mainline.
- Optional DSP instruction support in ARMv8-M Mainline.

Both sub-profiles will have the following enhancements:

- Support for an optional security technology called ARM TrustZone® for ARMv8-M.
- An enhanced Memory Protection Unit (MPU) design.
- Various enhancements in debug components to improve debug capability.
- Support for load acquire and store release instructions. These instructions are memory access instructions with memory ordering restrictions, with variants for exclusive access support. This support is targeted at the handling of atomic variables (a feature of C11).

In addition, ARMv8-M Baseline includes a range of instructions which were previously available in ARMv7-M but not in ARMv6-M, and enable a richer instruction set functionality within the constrained profile. These include:

- Hardware integer divide instructions.
- Compare and branch instructions.
- 32-bit branch instruction (allows a wider branch address offset).
- Exclusive access instructions (for semaphore operations).
- More choice of instructions for immediate data generation (MOVW and MOVT).

The instruction set and architectural enhancements in ARMv8-M Baseline ensure consistency between the system level features in the ARMv8-M Mainline and Baseline sub-profiles. This consistency of features, such as exclusive accesses between multiple processors, further enhances the scalability of Cortex-M system designs.

Most of the details of these enhancements are covered in the whitepaper "ARMv8-M Architecture Technical Overview" (reference 1). In the rest of this paper, we will focus on the key security enhancement: TrustZone for ARMv8-M.

3 SECURITY REQUIREMENTS IN EMBEDDED SYSTEMS

The word "security" can mean many different things in embedded system designs. In most embedded systems, security can include, but is not limited to:

- Communication protection – prevents data transfers from being visible to, or intercepted by unauthorized parties and uses techniques such as cryptography.
- Data protection – prevents secret data stored inside devices from being accessed by unauthorized parties.
- Firmware protection – prevents on-chip firmware from being reverse engineered.
- Operation protection – prevents critical operations from malicious intentional failure.
- Tamper protection – In many security sensitive products, anti-tampering features are needed to prevent the device's operation or protection mechanisms from being overridden.

Traditionally, security measures for embedded systems are focused on the first three items. In many cases, an embedded device is typically considered as a single security domain, in which the software running inside is delivered by a single party (with the exceptions of systems running Linux or another rich OS).

Running all software components in a device as a single security domain is simple to do, but has some disadvantages. For example, if there is vulnerability in one of the software components, like a communication protocol stack, it is possible for a hacker to take advantage of such vulnerability to compromise the entire system. Also, many modern microcontrollers or SoC devices have pre-loaded on chip firmware that needs protection, while at the same time needing to allow third party software developers to add their own application software. In such cases, chip designs with a single security domain can be inadequate.

In order to improve the security handling capability, most modern processors support privileged and unprivileged execution states. By partitioning software into privileged (e.g. critical system control operations, security operations) and unprivileged (e.g. general operations, user interface,

communication protocol stacks) domains, and by utilizing a MPU to manage memory access permissions, critical operations and data can be protected from unprivileged software. In this way, even if a software component running in an unprivileged state is compromised, the rogue application can still be terminated by the OS and a restart triggered, and hence the system can recover.

With the arrival of the IoT era, the requirements for security in embedded systems have dramatically changed. The combination of various protection requirements needs more than the privileged and unprivileged arrangement can offer. For example, APIs (software functions) in pre-loaded on-chip firmware need to be accessible by third parties (microcontroller software developers), but a protection mechanism is needed to prevent the reverse engineering of this pre-loaded firmware. In addition, some of the internal functions of this on-chip firmware should not be directly accessible. In some instances OS services (via a Supervisor Call exception) can be used, but this brings additional overhead and has dependency on the OS.

At the same time, in many instances interrupt handlers created by third parties must also be serviced with low interrupt latency, and secret data in preloaded firmware must remain hidden from the third party's software. Third party software must still be able to access certain privileged operations, for example, power management and interrupt control. As a result, an extra level of security level separation is highly desirable.

4 TrustZone for ARMv8-M concepts

4.1 Processor states

As a result of these requirements, a more robust security boundary is needed in the processor architecture. TrustZone for ARMv8-M introduces an additional partitioning of Secure and Non-Secure states, which are orthogonal to the privileged and unprivileged states. The Secure execution state is designed to be used for the execution of protected firmware, including the range of security critical functions provided by microcontroller vendors (Figure 2).

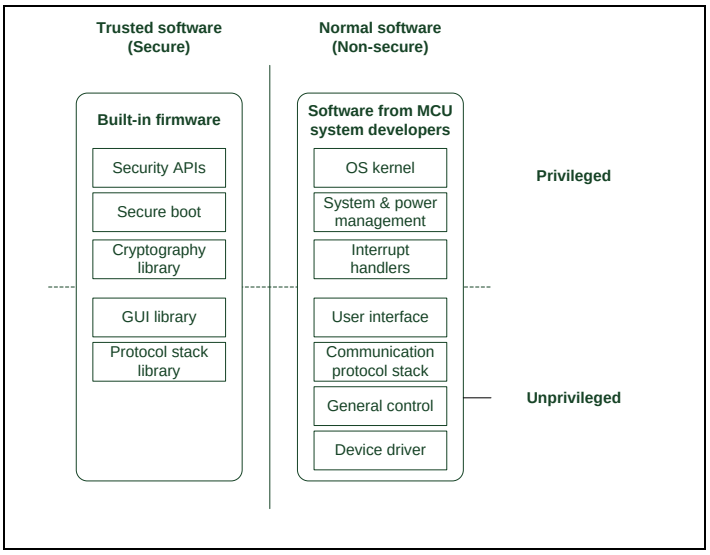

Figure 2: A possible use case of TrustZone for ARMv8-M in microcontroller designs.

4.2 Memory partitioning

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

When the processor executes software code from the Secure memory, it is then in the Secure state (with exceptions at the boundaries of certain state transition sequences). Otherwise the processor is in a Non-secure state. TrustZone for ARMv8-M also introduces a new memory attribute called Non-Secure Callable (NSC). Some portions of executable secure memory can be marked as NSC, which indicates that this area contains valid entry points for Secure APIs.

The partitioning of memory space is defined by using two mechanisms (both of them are optional):

- A hardware unit called the Implementation Defined Attribution Unit (IDAU) which is closely coupled to the processor. The IDAU is designed by chip designers developing the chip.
- A programmable unit called the Security Attribution Unit (SAU) inside the processor. This is accessible only in Secure state.

For each memory access, the look up results from IDAU and SAU are combined and the higher security attribute is selected. In typical systems, the primary memory partitioning is handled by the IDAU, and then SAU is optionally used to override the security settings of certain memory areas (e.g. to

change Secure NSC regions to Secure to restrict memory space allocated for entry points).

4.3 Processor's built-in hardware

The Cortex-M processors contain a range of internally programmable components. Under the ARMv8-M architecture, some of the hardware resources are banked (two separate units for Secure and Non-secure states). For example:

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

When the processor is in the Secure state, it can access the Secure SysTick, Secure MPU settings and Secure SCB registers. The same code running in Non-secure can only access the Non-secure version of these units. Secure state software can also access the Non-Secure SysTick, MPU and SCB using alias addresses, and has access to the SAU registers.

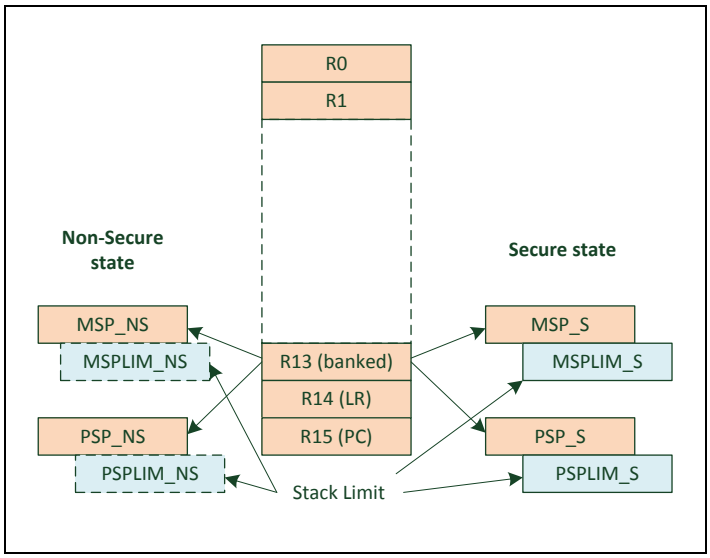

Figure 3: Shared register bank in ARMv8-M

The register bank is shared with the exception of stack pointers (Figure 3). Instead of having two stack pointer registers (MSP and PSP) in ARMv6-M and ARMv7-M, there are four stack pointers in ARMv8-M (Table 1):

	Secure	Non-secure
Main Stack Pointer (MSP)	MSP_S	MSP_NS
Process Stack Pointer (PSP)	PSP_S	PSP_NS

Table 1: Banked stack pointers in ARMv8-M architecture

The "_S" and "_NS" suffixes are used in ARMv8-M to identify whether the resource is for the Secure state or Non-Secure state. 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 also banked.

A stack limit detection feature is also added to ARMv8-M. Stack limit registers (Table 2) are available for all stack pointers in ARMv8-M Mainline; in ARMv8-M Baseline the stack limit registers are available for Secure stack pointers.

	Secure	Non-Secure
Stack limit configuration for MSP	MSPLIM_S	MSPLIM_NS
Stack limit configuration for PSP	PSPLIM_S	PSPLIM_NS

Table 2: Stack limit registers

4.4 Interrupt and exception management

Each of the interrupts, including the Non-Maskable Interrupt (NMI), can be programmed by secure software to target the Secure or Non-Secure states. Secure and Non-Secure interrupts can share the same priority level range or, optionally, the Non-Secure interrupts can be forced to use the lower half of the exception priority range under software control. Some system exceptions are banked; for example, the SysTick exception is banked because the SysTick hardware is banked.

The architecture supports separated exception vector tables for the Secure and Non-Secure exceptions. The starting address of the vector tables are defined using Vector Table Offset Registers (VTOR_S and VTOR_NS), and the initial values for the VTOR registers can be defined by chip designers (this is different from Cortex-M0/M0+/M3/M4, which are fixed to address 0x00000000).

A new system exception called SecureFault is introduced in the ARMv8-M Mainline sub-profile for handling of security violations. For ARMv8-M Baseline, such violations are handled by the existing HardFault exception targeting the Secure state.

4.5 State transitions

Secure and Non-Secure software can directly interact with each other by means of function calls, as outlined below.

4.5.1 Non-Secure code to Secure functions

When the processor is in the Non-Secure state, software can call a function in Secure memory directly provided that:

- The entry point is in a Secure memory region marked with the NSC attribute.
- The first instruction is Secure Gateway (SG)

The SG instruction is introduced in the ARMv8-M architecture to indicate valid entry points into the Secure region. Secure software can have multiple entry points (with multiple SG instructions) and NSC regions. When such transitions take place, the LSB of the Link Register (LR, R14)

is used to indicate the return state (0 indicates Non-secure). The return state information is then used in the function return with the BXNS instruction to ensure that the function returns to the correct security domain. BXNS (Branch and Exchange to Non-secure) and BLXNS (Branch with Link and Exchange to Non-secure) are new instructions introduced to the ARMv8-M architecture to support TrustZone operations.

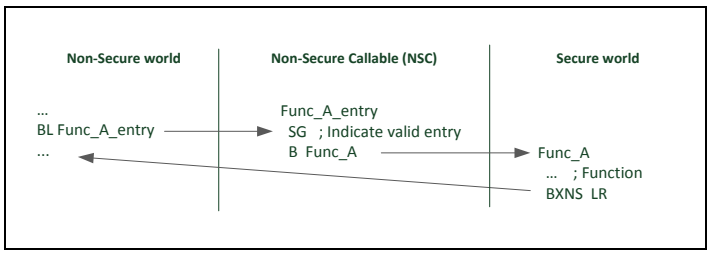

Figure 4: Software flow when Non-secure software calls a function (e.g. API) in secure side

4.5.2 Secure code to Non-Secure functions

When the processor is in the Secure state, it can call a function in Non-Secure memory directly using the BLXNS instruction. In such cases, the return address and part of the PSR (Program Status Register) is pushed to the current Secure stack, and at the same time the Link register (LR, R14) is switched to a special value called FNC_RETURN.

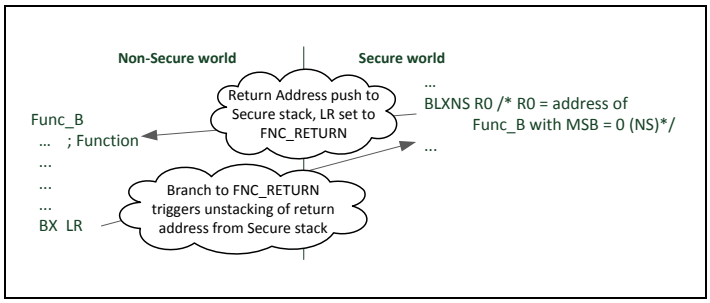

Figure 5: Software flow when Secure program call a function (e.g. API) in Non-secure side

When the Non-Secure function is completed, the value FNC_RETURN is loaded into the PC and this triggers the processor to automatically restore the return address and partial PSR from the Secure stack. This mechanism protects the Secure program addresses from being visible to Non-Secure software.

4.5.3 Exception sequences

The Security state of the processor can also switch at exception entry, exit and tail-chaining scenarios. Since the register bank is shared, if a Non-Secure interrupt occurs during the execution of a Secure program, the processor hardware automatically pushes all registers to the Secure stack and clears them to ensure that no Secure information is leaked to the Non-Secure side. At the end of the Non-Secure exception/interrupt handler, the processor then restores all these registers from the Secure stack and resumes the interrupted task.

If the current security state of the processor and the target state of the exception/interrupt are the same, or if the exception/interrupt is Secure, then only the caller save registers are pushed to the current stack, as in ARMv6-M and ARMv7-M designs. This enables interrupt handlers to be programmed as normal C functions.

5 KEY CHARACTERISTICS OF TRUSTZONE FOR ARMV8-M

The design of TrustZone for ARMv8-M is optimized for microcontrollers and low power SoC applications. In these applications the requirements of low power, low memory footprint and low latency are important factors. TrustZone for ARMv8-M technology is optimized to meet these requirements and has been designed from the ground up instead of reusing the existing TrustZone technology for ARM Cortex-A processors.

The design of TrustZone for ARMv8-M has the following key characteristics:

- Non-Secure code can call Secure functions via valid entry points only, with the protection of the function return domain. There is no limitation on the number of entry points.
- Very low switching overhead in cross security domain calls. There is only one extra instruction (SG) when calling from the Non-Secure to the Secure domain, and only a few extra clock cycles when calling from the Secure state to Non-Secure functions.
- Non-Secure interrupt requests can still be served during the execution of Secure code, with minimal impact on the interrupt latency (stacking of the full register banks is required instead of just the caller saving registers).
- The processor starts up in Secure state by default. This enables “root-of-trust” implementations such as secure boot.
- Low power - There is no need for separate register banks for Secure and Non-Secure states, whilst Non-Secure interrupt handlers are still prevented from peeking into data used by secure operations.
- Ease of use – Interrupt handlers remain programmable in C, and Non-Secure software can access Secure APIs with standard C/C++ function calls.

- High flexibility - The design allows Secure software to call Non-Secure functions (this is often needed when protected middleware on the Secure side needs to access device driver code in the Non-Secure side). In addition, the Secure state can also have privileged and unprivileged execution states, making it possible to support multiple Secure software components with a protection mechanism between them (see section 7, Support of multiple secure software components).

At a high level, the concepts of TrustZone for ARMv8-M are similar to the TrustZone technology in ARM Cortex-A processors. In both designs, the processor has Secure and Non-Secure states, with Non-Secure software able to access to Non-Secure memories only. However, there are a number of differences in the implementation:

- TrustZone for ARMv8-M allows direct function calls between security domains. In Cortex-A processors, TrustZone requires a Secure Monitor exception to switch from the Non-Secure to the Secure state (with higher execution cycle overhead), and Secure software is not able to directly call Non-Secure functions.
- TrustZone for ARMv8-M supports multiple Secure function entry points, whereas in TrustZone for Cortex-A processors, the Secure Monitor handler is the sole entry point.
- In ARMv8-M architecture, Non-Secure interrupts can still be serviced when executing a Secure function.

As such TrustZone for ARMv8-M is optimized for low power microcontroller type applications:

- In many microcontroller applications with real-time processing, deterministic behaviour and low interrupt latency are important requirements. Therefore the ability to service interrupt requests while running Secure code is critical.
- By allowing the register banks to be shared between Secure and Non-Secure states, the power consumption of ARMv8-M implementations can be similar to ARMv6-M or ARMv7-M implementations.
- The low overhead of state switching allows Secure and Non-Secure software to interact frequently, which is expected to be common place when Secure firmware contains software libraries such as GUI firmware or communication protocol stacks.

TrustZone for ARMv8-M is also different from the virtualization approach as supported in the ARMv8-R architecture. In systems with virtualization, each of the virtualized software environments (Virtual Machines, or VMs) are isolated from each other, and there is no direct interaction between VMs other than going through the hypervisor, interrupts or shared memories. As a result, interaction between software in different VMs requires additional execution cycles and software overhead, and while this is perfectly acceptable in the high-performance Cortex-R processors, it is not ideal for resource constrained Cortex-M applications. In addition, the memory footprints for virtualized systems are often significantly larger - as they require hypervisor software and often contain multiple operating systems.

6 How TrustZone for ARMv8-M addresses security requirements

The security requirements of embedded systems were outlined earlier in this document. This section looks at how TrustZone for ARMv8-M helps address these Security requirements:

- Data protection – Sensitive data can be stored in secure memory spaces and can only be accessed by Secure software. Non-Secure software can only gain access to secure APIs providing services to the Non-Secure domain, and only after security checks or authentication.
- Firmware protection – firmware that is built-in (preloaded) on chips can be stored in secure memories to prevent it from being reverse engineered and compromised by malicious software developers using the chips. TrustZone for ARMv8-M can also work with additional protection techniques. For example, device level read-out protection, a technique commonly used in the industry today, can be used in conjunction with TrustZone for ARMv8-

M to protect the completed firmware of the final product.

- Operation protection – software for critical operations can be preloaded to chips as secure firmware and the appropriate peripherals can be configured to permit access from the Secure state only. In this way, the operations can be protected from intrusion from the Non-Secure side.

Since TrustZone is only a barrier between security domains, some security requirements cannot be addressed by TrustZone technology alone. For example:

- Communication protection still requires cryptography techniques which might be handled by software or assisted by hardware crypto accelerators (e.g. ARM TrustZone Cryptocell® products). TrustZone can help support such techniques, as certain crypto software and hardware can be configured to only be accessible within the Secure state.
- Anti-tampering, if required in a product, still requires specialized chip design techniques as well as product level design arrangements (e.g. circuit boards and product enclosures). Whether anti-tampering is applied is dependent on system requirements and the value of the assests being protected.

Nonetheless, TrustZone technology enables a better foundation for system level security. In the simplest example, TrustZone for ARMv8-M can be used to protect firmware from being reverse engineered (Figure 6). Today, many microcontrollers already have built-in firmware such as USB or Bluetooth stacks, and TrustZone makes the firmware protection implementation easier and more secure (e.g. by ensuring that untrusted software cannot branch to the middle of secure APIs to bypass any initial checking).

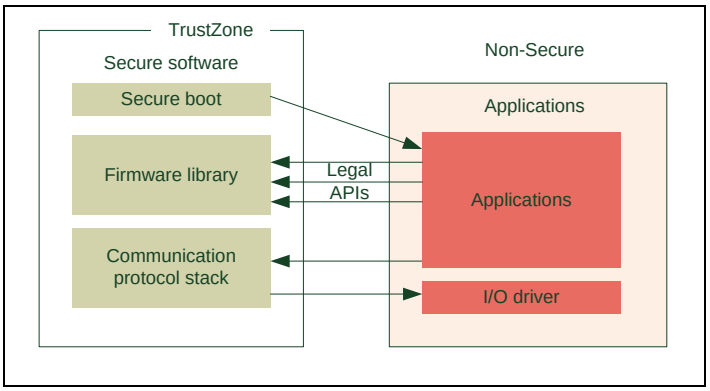

Figure 6: Firmware protection – a simple use case of TrustZone for ARMv8-M

TrustZone can also be used with the additional protection features used in advanced microcontrollers targeting the next generation IoT products. For example, a microcontroller developed for IoT applications can include a range of security features, and the use of TrustZone technology can help ensure that all those features can only be accessed via APIs with valid entry points (Figure 7).

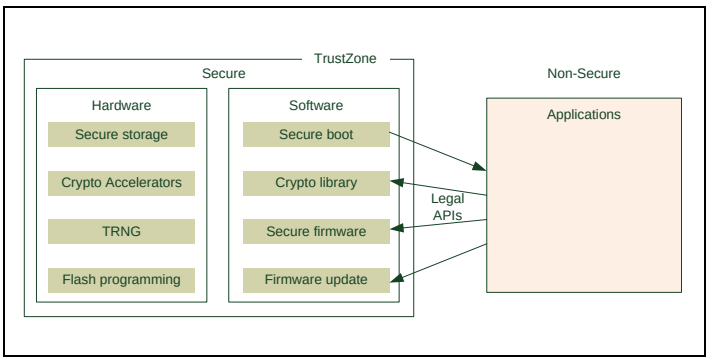

Figure 7: Security features in a microcontroller for IoT applications can utilize TrustZone technology in various ways

By using TrustZone to safe guard these security features, chip designers can:

- Prevent untrusted applications from directly accessing security critical resources.
- Ensure that a Flash image is reprogramed only after authentication / checking.
- Prevent firmware from being reverse engineered.
- Store secret information with protection at the software level.

In some other application scenarios, such as a wireless SoC with a certified built-in radio stack, TrustZone technology can protect the standardised operations (e.g. wireless communication behaviour) and therefore help ensure that customer defined applications cannot void the certification (Figure 8).

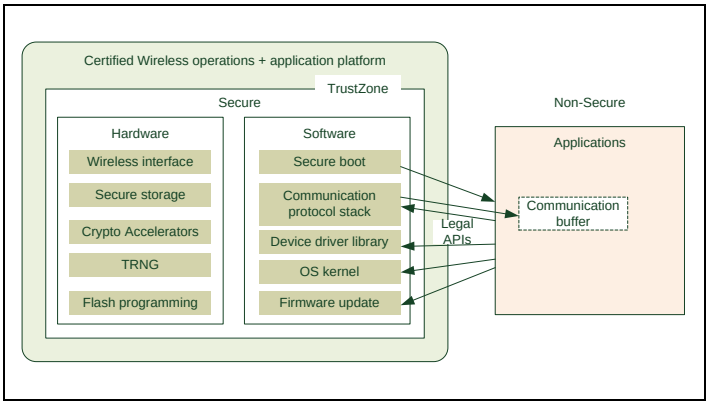

Figure 8: TrustZone for ARMv8-M can be utilized to protect the operations of certified wireless communication interface

7 SUPPORT OF MULTIPLE SECURE SOFTWARE COMPONENTS

One of the key differences between TrustZone in Cortex-A processors and TrustZone for ARMv8-M is that the Secure state in ARMv8-M can be privileged or unprivileged, depending upon the setting of a programmable bit in a special register called CONTROL. (Note: The CONTROL register is banked between the Secure and Non-Secure states). When combined with the optional MPU support, it is then possible to execute a number of firmware libraries in the Secure unprivileged state, and use the MPU to protect them from accessing other other's memory.

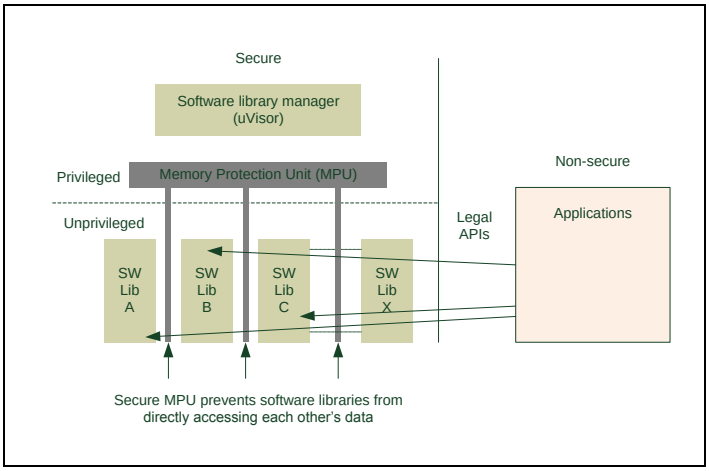

Figure 9: TrustZone for ARMv8-M can support multiple Secure software libraries with memory protection between them

With this arrangement Non-Secure software can, as usual, gain access to APIs provided by each library. On the Secure side, a software library manager is used to context switch the Secure MPU configuration when a Secure Memory Management fault is detected.

For example, when the secure MPU is currently set up for software library A, the Non-Secure applications can access APIs in software library A straight away. However, if a Non-Secure application accesses a API function in software library B, the API execution triggers a Memory Management fault and the fault handler in the software library manager context switches the secure MPU configuration to work with software library B, and the API function in library B can then resume. Since the software library can tell which secure library is currently selected, it can block the library from accessing other libraries' data using the MPU.

The context switches incur a small delay for the API calls. However, there is no context switch if an API from the currently selected library is called.

Since this operation requires a number of fault status registers and a fault address register, this technique is supported by ARMv8-M Mainline but not ARMv8-M Baseline.

8 HOW SECURE IS TRUSTZONE FOR ARMV8-M?

One of the common questions when talking about secure system designs is: "How secure is it?" Many aspects of attack scenarios have been considered in the development of TrustZone for ARMv8-M. For example:

- Software accesses – with additional system level components, memories can be partitioned between Secure and Non-Secure spaces, and can disable Non-Secure software from accessing secure memories and resources.
- Branch to arbitrary secure address locations – the SG instruction and NSC memory attribute ensures that a Non-Secure to Secure branch can happen only at valid entry points.
- Inadvertent SG instruction in binary data – NSC memory attribute ensures that only secure address spaces that are intended to be used as entry points can be used to switch the processor into the Secure state. Branching to an inadvertent SG instruction in an address location not marked as NSC will result in a fault exception.
- Faking of a return address when calling a Secure API – when the SG instruction is executed, the return state of the function is stored in the LSB of the return address in the Link Register (LR). At the return of the function, this bit is checked against the return state to prevent the Secure API function (which was called from Non-Secure side) from returning to a fake return address pointing to a Secure address.
- Attempting to switch to the Secure side using FNC_RETURN (function return code) – When

switching from non-returnable Secure code (for example a Secure boot loader) to Non-Secure the BLXNS instruction should be used to ensure that there is a valid return stack. The return stack can then be used to enter an error handler. This prevents Non-Secure malicious code from trying to switch the processor to Secure code using the FNC_RETURN mechanism and crashing the Secure software if there is no valid return address in the Secure stack. This recommendation does not apply when returning from a Secure API to Non-Secure software, as this can use the BXNS instruction.

- Faking of EXC_RETURN (exception return code) to return to Secure state illegally – if a Non-Secure interrupt takes place during Secure code execution, the processor automatically adds a signature value to the Secure stack frame. If the Non-Secure software attempts to use the interrupt return to switch to the Secure side illegally, the signature check at the exception return will fail and hence the error is detected.
- Attempt to create stack overflow in Secure software – A stack limit feature is implemented for Secure stack pointers in both ARMv8-M Mainline and Baseline sub-profiles. Therefore such stack overflows can be detected and handled by the fault exception handler.

On the debug side, the architecture also handles the security requirements from the ground up:

- Debug access management – Debug authentication signals are implemented on the processors so that chip designers can control if debug and trace operations are allowed for Secure and Non-Secure states respectively. AMBA bus interface protocols also support sideband signals in bus transactions, so the system can filter transfers to prevent debuggers from directly accessing Secure memories.
- Debug and trace management – the debug authentication signals can be setup to disable debugging and tracing operations when the processor is running in the Secure state.

Although the architecture is designed to handle many types of attack scenarios, Secure software always needs to be written with care and needs to utilize security features (e.g. stack limit checks) to prevent vulnerabilities. The ARM C Language Extensions (ACLE, reference 2) have been extended to include additional features to support the ARMv8-M architecture, and software developers writing secure software should utilize these features to enable their development tools to generate code images for ARMv8-M devices. Some examples of these extensions can be found in the ARM Compiler document (reference 3).

9 THE NEXT STEPS IN IOT MICROCONTROLLERS

TrustZone for ARMv8-M enables silicon providers to include hardware and software in their chips in a secure manner so that their IP can be protected. This will lead to new developments in the next generation of microcontrollers, and accelerate some of the trends in current microcontroller markets. Such trends include:

- Inclusion of middleware in microcontroller devices to provide value added differentiation features.
- IoT microcontroller products redefined as IoT platforms – microcontroller devices with built-in internet connectivity solutions, OS, and additional application specific middleware (e.g. audio).

With these changes, IoT devices will be able to have much shorter development cycles. For example, microcontroller software developers need spend less time in integrating middleware from multiple vendors themselves, and software development environments like mbed™OS will enable software developers to utilize the connectivity features and create secure IoT devices with a shorter development project cycle.

The role of TrustZone technology is more than just IP protection. Since secure operations can also be protected, the same technology can be used to safe guard critical system operations. As a result, it is possible to deploy ARMv8-M based systems in a wide range of applications, such as industrial and automotive, with TrustZone being used as a system reliability enhancement feature. This could lead to new ranges of industrial and automotive microcontroller devices.

In conclusion, the next generation of the ARM Cortex-M processors will be powered by a new architecture version called the ARMv8-M Architecture. The TrustZone technology in this architecture update addresses some of the most

important challenges in IoT and embedded systems. This paper has shown how TrustZone can be used to offer significant enhancements in embedded system designs, with examples provided across a wide range of scenarios.

REFERENCES

The following documents are referenced in this whitepaper, and can provide useful information and further details the on ARMv8-M architecture.

	Documents
1	ARMv8-M Architecture Technical Overview https://community.arm.com/docs/DOC-10896
2	ARMv8-M Security Extensions: Requirements on Development Tools http://infocenter.arm.com/help/topic/com.arm.doc.ecm0359818/ECM0359818_armv8m_security_extensions_reqs_on_dev_tools_1_0.pdf
3	ARM Compiler Software Development Guide 6.4 Chapter 8 Building Secure and Non-secure Images Using ARMv8-M Security Extensions http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0773e/page1446115999905.html