embeddedworld2017
Exhibition&Conference
... it's a smarter world

Optimizing ARM Cortex-A and Cortex-M Based Heterogeneous Multiprocessor Systems for Rich Embedded Applications

Kinjal Dave
Senior Product Manager
CPU Group, ARM

Abstract— Heterogeneous multiprocessor (HMP) systems using application processors and microcontrollers in the same SoCs are used extensively across a wide range of embedded markets. An increasing number of embedded applications now benefit from the combination of ultra-low-power ARM® Cortex®-M processors alongside higher performance Cortex-A processors. Next-generation embedded applications require improved performance and security features without sacrificing the overall efficiency in the system. This paper highlights motivations, benefits and system design choices available to SoC architects. The software development considerations and solutions for such HMP systems are also discussed.

Keywords— Heterogeneous processing, Cortex-A, Cortex-M, HMP, low-power, multi-core, ARM

I. INTRODUCTION

We encounter more and more compute systems every day-starting with the smartphone and perhaps a smart-watch tethered to the phone. In our homes, we interact with smart and connected televisions, refrigerators, washing machines and thermostats. In the gym, smart and connected equipment is becoming the norm. All these are transforming the way in which we live for better.

A common requirement for all these devices is the variety of tasks expected from these devices in an energy-efficient manner. This requirement translates into architecting systems such that they can handle these diverse compute requirements. This key requirement to designing compute systems capable of handling diversity of workloads spans across several markets. Some common examples include embedded applications, in-vehicle-infotainment systems, healthcare and industrial applications.

This means that modern compute systems must be designed to meet conflicting requirements. These must be designed to provide high performance for running several demanding applications while being able to respond quickly to a real-time event. These must be designed to handle general data processing as well as specialized multimedia processing tasks efficiently. These must be designed to support different software environments – for example running Linux on one compute element and running a RTOS on another.

To meet these conflicting requirements, modern system designers rely heavily on building heterogeneous compute systems. Heterogeneous compute is fundamentally about using the right processor optimized for a set of tasks. Some of the benefits of such HMP systems include an increased overall system performance, increased system efficiency and reduced system cost due to integration of different compute elements.

II. TERMINOLOGY FOR HMP SYSTEMS

A. ARM processors-based HMP systems

There are several types of HMP systems. In a generic sense, HMP system refers to a complex system that combines several different compute elements like a general-purpose processor, a graphics processor, an image processor, a video processor, a display processor and possibly several accelerators. Fig. 1 shows a typical HMP compute system that includes several compute elements. "Heterogeneous multi-processor system" is also used to denote compute systems that use various combinations of ARM's Cortex processors (e.g. Cortex-A, Cortex-R and Cortex-M).

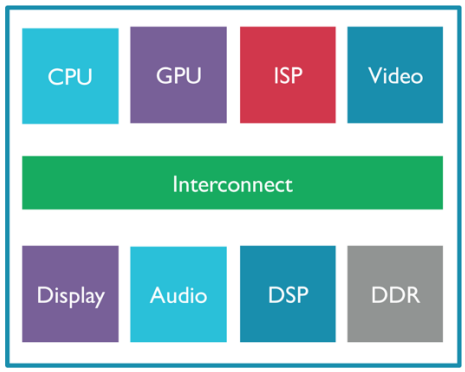
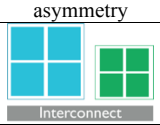


Fig. 1. A generic heterogeneous multiprocessor (HMP) compute system

The scope of this paper is to discuss the heterogeneous compute systems using ARM Cortex processors. It is commonplace for several compute systems to use different combinations of Cortex-A, Cortex-R and Cortex-M processors to provide the right functionality for a given application. Table 1 below depicts the different possible combinations of HMP systems using Cortex processors and lists the key differences between these systems.

This paper discusses the HMP systems exhibiting functional symmetry. More specifically, this paper discusses the details of Cortex-A and Cortex-M processors-based HMP systems.

TABLE I. MULTI-CORE PROCESSORS TERMINOLOGY

Multi-core		
Homogeneous	***Heterogeneous***	
	Performance asymmetry	Functional asymmetry
		
Same ISA	Same ISA	Different ISA
Same microarchitecture	Different microarchitecture	Different microarchitecture
Same view of memory	Same view of memory	Different view of memory
Same software	Software symmetry	Different software environments – software asymmetry

B. *ARM architecture for diverse computing needs*

The three different Cortex processor families from ARM are optimized for different compute requirements as shown in table 2.

- The Cortex-A processors are optimized for running rich operating systems like Linux and Android and can provide high performance for demanding applications across a wide range of applications.
- The Cortex-R processors are optimized for hard real-time applications with high performance requirements.
- The Cortex-M processors are optimized for ultra-low power, low-cost compute required for a wide variety of embedded applications with real time capabilities.

Heterogeneous systems using all three Cortex processors are common today and used across many applications. Some common examples include smartphones, wearable devices with a rich GUI and Advanced Driver Assist Systems (ADAS). Furthermore, embedded systems that were traditionally based on simple MCUs are now increasingly required to support rich graphical user interfaces.

TABLE II. ARCHITECTURAL DIFFERENCES AMONGST ARM CORTEX PROCESSOR FAMILIES

	Cortex-A	Cortex-R	Cortex-M
Architecture profile	ARMv7-A ARMv8-A	ARMv7-R ARMv8-R	ARMv7-M ARMv8-M
Instruction set	32-bit/64-bit	32-bit	32-bit
Interrupts	Software managed	Deterministic software managed	Hardware managed
Bus interface	AMBA® AXI	AMBA AXI	AMBA AHB/AXI
Operating system support	Rich OS/RTOS	RTOS	RTOS
Examples	Cortex-A7 Cortex-A35	Cortex-R8 Cortex-R52	Cortex-M7 Cortex-M33

This requires the use of Cortex-A processors in addition to Cortex-M processors. Therefore, an increasing number of embedded systems use Cortex-A processors extensively to address the high performance, rich user interface and running a rich operating system (like Linux) requirements across general purpose embedded, industrial, consumer and medical applications. These applications also use Cortex-M processors to address deterministic, real-time control requirements in industrial, medical and consumer applications like intelligent thermostats.

III. SYSTEM DESIGN CONSIDERATIONS

This section discusses the pros and cons of various system design choices available for architecting Cortex-A and Cortex-M processors-based HMP systems. First, the key architectural differences between the Cortex processor families are highlighted. Thereafter, the system design choices available to architect these systems are discussed.

A. *Key system design considerations*

To highlight the design considerations for such HMP systems, consider an example generic HMP compute subsystem using Cortex-A and Cortex-M processors as shown in figure 2. The Cortex-M subsystem uses a local memory. This allows the Cortex-M processor to run in the background without going through the main interconnect. This results in reduced bus transfers crossing the clock domains and thereby reduces system power consumption. The fundamental architectural differences between Cortex-A and Cortex-M processors are highlighted in Table II. The system designer needs to consider the following for architecting heterogeneous systems that combine Cortex-A and Cortex-M processors:

- How to address the impact of memory map differences between Cortex-A and Cortex-M processors at the system level?
- How to manage and distribute interrupts across Cortex-A and Cortex-M processors?
- How to handle inter-processor communication between Cortex-A and Cortex-M processor subsystems?
- How to handle security and secure/non-secure state communications between Cortex-A and Cortex-M processor subsystems?

This paper discusses these design considerations in further detail.

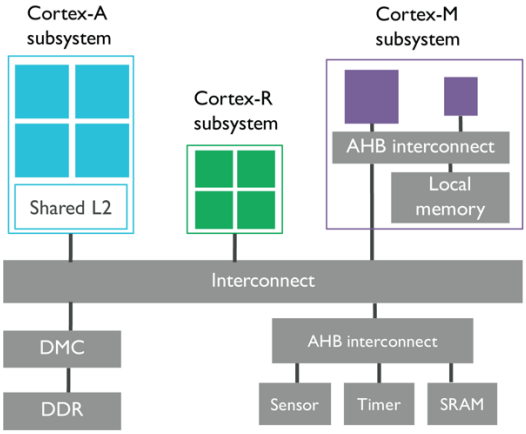


Fig. 2. A generic HMP system using Cortex-A, Cortex-R and Cortex-M processors

B. Addressing differences in memory space addresses

Table III enumerates the differences in address spaces between the Cortex-A and Cortex-M processors. The Cortex-A processors support a significantly larger physical memory address space as compared to Cortex-M processors. This section discusses two approaches to address these differences at the system level.

TABLE III. ADDRESS SPACE DIFFERENCES BETWEEN CORTEX-A AND CORTEX-M PROCESSORS

	Cortex-A	Cortex-M
Physical addressing	ARMv7-A: Upto 40-bits ARMv8-A: Upto 48-bits	ARMv7-M, ARMv8-M: 32-bits
Addressing spaces	Secure and Non-secure	ARMv7-M: single ARMv8-M: Secure and Non-secure

The first approach is to specify a shared memory address space for both the processors as shown in the Fig. 3. The Cortex-M subsystem has its own local memory to enable deterministic capabilities, and allows the main memory system to reduce power when the Cortex-A processor system is idle. Essentially, this approach bridges the Cortex-M subsystem to a point in the system hierarchy where it has access to a limited portion of the address space. The system can then be architected to connect peripherals in this shared address space thus enabling both the Cortex-A and Cortex-M systems to access the common peripherals when needed. However, this approach requires a design decision on what main system resources the Cortex-M subsystem should be able to access.

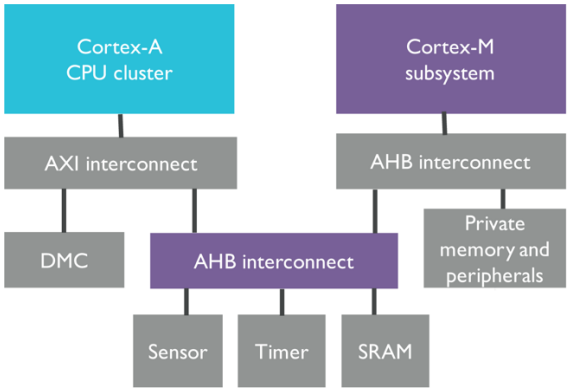


Fig. 3. Sharing a region of memory space between Cortex-A and Cortex-M processors in an HMP system

Another approach is to use a System Memory Management Unit (SMMU) to enable the Cortex-M subsystem to access the entire address space supported by the Cortex-A processor. The Cortex-M subsystem can be bridged to the top level of the main memory system, providing full access to system resources.

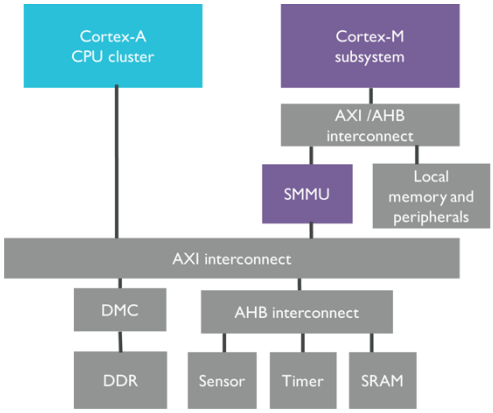


Fig. 4. Using an SMMU allows more flexibility for a Cortex-M processor to access a wider memory addressing space

The SMMU is used to overcome the address size and security state issues. The SMMU can be configured to provide one or more windows on to the main memory system. These windows can be re-configured as different areas of the memory system are needed by the Cortex-M subsystem. Using the SMMU to perform stage 1 address translation also allows the security attribute for ARM TrustZone® security to be set. Using an SMMU might seem to be a costly addition to a Cortex-M subsystem, however, it is possible to share the SMMU with other masters that need an SMMU. For example, ARM's CoreLink™ MMU-500 provides a modular design, allowing it to be shared by several masters or sub-systems. A single shared Translation Control Unit (TCU) is responsible for performing translation table walks. The TCU can either

have a dedicated connection to the interconnect for these walks, or share the connection of one of the masters.
The Translation Buffer Units (TBUs) provide caches of translations recently used by that master or subsystem. The TBUs can be individually sized, based on the requirements of the master or subsystem connected to them. For the Cortex-M subsystem being considered here, access to the main memory system is likely to be infrequent and to limited address ranges. This implies that a relatively small TBU could be provided to keep area cost to a minimum.

C. Managing interrupts in HMP systems

The Cortex-A and Cortex-M processors manage interrupts in different ways. The Cortex-A processors typically support up to four cores in a processor cluster, sharing a common GIC (Generic Interrupt Controller). As shown in Fig. 4, a GIC allows software to control the prioritization and distribution of interrupts. The Cortex-M processors include a Nested Vectored Interrupt Controller (NVIC), as shown in Fig. 5. The NVIC provides the same basic controls as a GIC, but is tailored to match the ARMv7-M and ARMv8-M exception models and without the ability to support multiple cores. When an interrupt is only of interest to software running on the Cortex-A processor cluster or a Cortex-M processor, the interrupt source can be directly connected to the GIC or the NVIC.

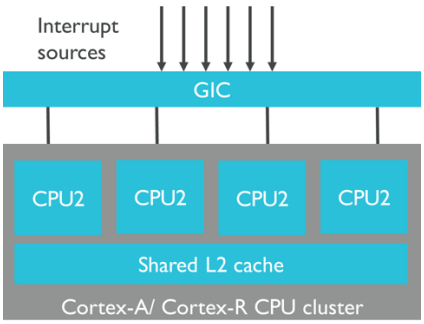


Fig. 5. Interrupt handling in Cortex-A processors

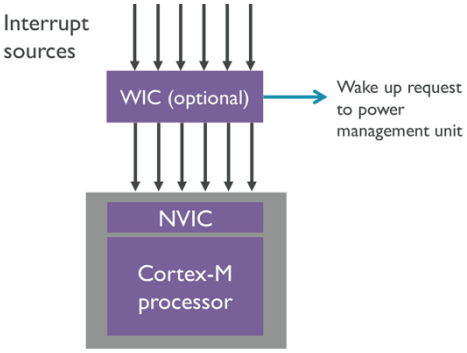


Fig. 6. Interrupt handling in Cortex-M processors

However, for some use cases it is required to run the interrupt handlers on different processors at different times. For example, the handling of a sensor interrupt might be handed over to a Cortex-M processor when the Cortex-A processor cluster is in retention or powered down. Two approaches can be considered to connect shared interrupts between the Cortex-A and Cortex-M systems
One simple approach (Fig. 7) is to wire the interrupt sources to both processor subsystems i.e. connecting the interrupt sources to both the GIC and the WIC in the Cortex-A and Cortex-M processors respectively. This approach requires the software to ensure that the interrupt is serviced by only one of the processor subsystems.

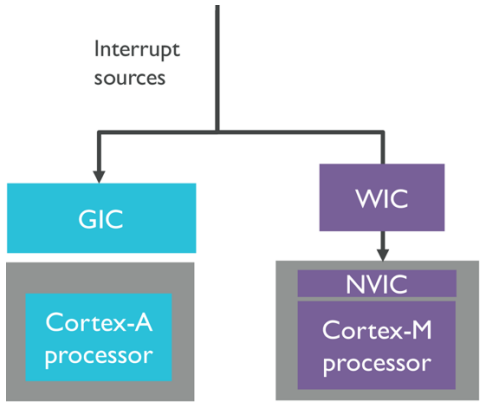


Fig. 7. Wired interrupt scheme for Cortex-A and Cortex-M processors

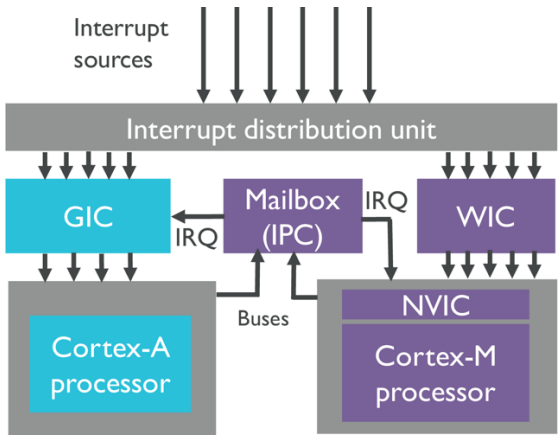


Fig. 8. Managing interrupts across Cortex-A and Cortex-M processors using a custom-build interrupt distribution unit

The second approach is to design a custom interrupt distribution unit in hardware to route the interrupts to the appropriate processor subsystem. (Fig. 8) Such a scheme can optionally benefit by using software level interrupt passing using a mail box scheme as shown in the figure. The first approach is simpler from a system design perspective. However, this also results in higher software overhead when switching the interrupt allocation from one processor subsystem to another. The second approach does incur a small hardware cost of the interrupt design unit, but results in significant reduction of software overhead for interrupt

allocation between the two processor subsystems. It also provides more flexibility in allocating interrupts to and from Cortex-A to Cortex-M subsystems

D. Handling inter-processor communication in HMP systems

The software running on the Cortex-A processors and Cortex-M sub-system needs to be able to communicate with each other. For example:

- To initiate hand over of a shared peripheral from one system to the other.
- Cortex-A processor requesting system control activities from a Cortex-M system controller.
- Cortex-M sensor hub reporting sensor information to the Cortex-A processors.

Such communication would typically be via mail boxes in shared memory. This would need to be memory that is part of the main system's address space, so that both the Cortex-A processors and the Cortex-M subsystem have visibility.
Such mail boxes might be complimented by door-bell interrupts, to signal the presence of new messages or the completion of previous commands. This requires a mechanism for each processor to generate interrupts in the other's interrupt controller.

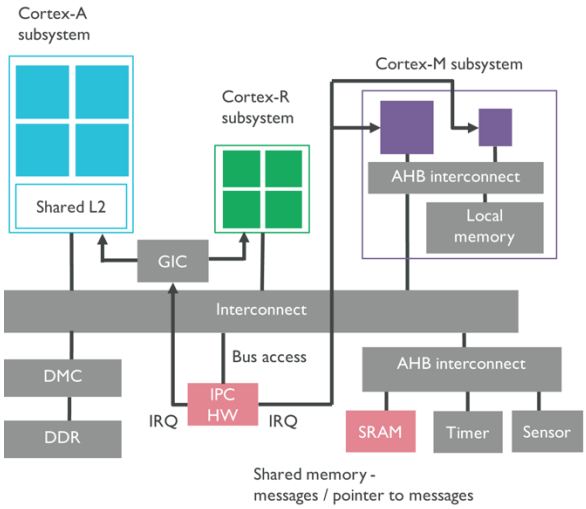


Fig. 9. Handling inter-processor commuication between Cortex-A, Cortex-R and Cortex-M processor subsystems

E. Security considerations for HMP systems

Architecting security in modern compute systems is a necessary requirement to enable devices to counter specific threats that it might experience. ARM offers TrustZone technology as a foundation to architect system-wide hardware isolation for trusted software and critical resources. TrustZone technology has been supported by all the ARMv7-A and ARMv8-A processors and is commonly used to run trusted boot and a trusted OS to create a Trusted Execution Environment (TEE). Typical use cases include the protection of authentication mechanisms, cryptography, key material and digital rights management (DRM). Applications that run in the secure world are called Trusted Apps.

At the heart of the TrustZone approach is the concept of secure and non-secure worlds that are hardware separated, with non-secure software blocked from accessing secure resources directly. Within the processor, software either resides in the secure world or the non-secure world; a switch between these two worlds is accomplished via software referred to as the secure monitor (Cortex-A) or by the core logic (Cortex-M). This concept of secure (trusted) and non-secure (non-trusted) worlds extends beyond the processor to encompass memory, software, bus transactions, interrupts and peripherals within an SoC.

The ARMv8-M architecture extends TrustZone technology to Cortex-M class systems enabling robust levels of protection at all cost points. TrustZone for ARMv8-M has the same high-level features as TrustZone on applications processors with the key benefit that context switching between secure and non-secure worlds is done in hardware for faster transitions and greater power efficiency. There is no need for any secure monitor software.

Designers planning to integrate Cortex-A and Cortex-M processors to build HMP systems need to consider the architectural differences between TrustZone technology supported by these processors. Table IV summarizes these differences.

TABLE IV. TRUSTZONE FOR ARMV7-A/ARMV8-A AND ARMV8-M

	TrustZone for ARMv7-A and ARMv8-A	TrustZone for ARMv8-M
Security states	SEL0 – Trusted Apps SEL1 – Trusted OS EL3 – Trusted Boot and Firmware (ARMv8-A)	Secure Thread – Trusted code/data Secure Handler – Trusted device drivers, RTOS,etc
Secure interrupts	Yes	Yes (deterministic)
State transition	Software transition	Hardware transition (fast)
Memory management	Virtual memory MMU with secure attributes	Secure Attribution Unit (SAU) & MPU memory partitions
System interconnect security	Yes	Yes
Trusted Boot	Yes	Yes
Secure code, data and memory	Yes	Yes
Software	ARM Trusted Firmware (and 3rd party TEEs)	ARM Keil CMSIS, ARM mbed OS, mbed uVisor, 3rd party software

When integrating ARMv6-M or ARMv7-M processors (e.g Cortex-M0, Cortex-M7) within the HMP system, the designer needs to be mindful of a few things. Since ARMv6-M and ARMv7-M processors do not support TrustZone, the compute

subsystem using Cortex-M processors must be defined as always secure (e.g. system control processor subsystem) or always non-secure (e.g. audio subsystem). The designer also needs to ensure that the debug system matches the security domains for each processor.

Using ARMv8-M processors in HMP systems provides more flexibility and configurability for security features in the system. ARMv8-M processors like Cortex-M33 or Cortex-M23, support TrustZone for ARMv8-M. This removes the restriction of defining the Cortex-M subsystem as always secure/non-secure. Furthermore, the designer also has the flexibility to either share the secure worlds or keep them separate between the Cortex-A and Cortex-M processor subsystems. It is also important to consider the system memory partitioning and interrupt distribution in the Secure/Non-secure worlds across the two processor subsystems.

The designer also needs to ensure that Secure and non-secure memory partitioning must match between the Cortex-A and Cortex-M processor subsystems. In ARMv8-M processors, a certain address can only be Secure or Non-Secure (unlike Cortex-A processors). Therefore:

- The Cortex-A processor's MMU page setup needs to match the memory map observed by the Cortex-M processor (ARMv8-M)
- When creating the memory map for Cortex-M, the designer needs to be aware of the MMU page size on Cortex-A processor (e.g. 4KB page size granularity)

The interrupts also need to be managed appropriately between the Secure/Non-secure states across the Cortex-A and Cortex-M processor subsystems. The interrupt distribution unit described in Fig. 8 needs to ensure the following are true:

- Must route interrupt to the correct security domains (and be consistent between cores)
- Interrupt's security domain and peripheral's security domain must match
- Non-secure software must not be able to change Secure interrupt routing

IV. Software Considerations for HMP Systems

A. *Overview of software development requirements for HMP systems*

The presence of two functionally asymmetric compute subsystems in HMP systems poses some challenges for software development. The key challenge that needs to be addressed is developer productivity. Software developers would want their software to be portable across different HMP SoC platforms. It is common to see HMP systems running different software environments across the Cortex-A (running Linux, for example) and Cortex-M processors (RTOS). Therefore, making it easier to debug issues on HMP systems is important for developer productivity.

B. *Addressing software challenges for HMP systems*

The three key challenges faced by software developers for HMP systems are:

- How to communicate between two different software environments running on the Cortex-A and Cortex-M processor subsystems?
- How to debug issues across two different software environments?
- How to identify performance hot-spots and improve performance for these HMP systems?

To ensure that developers can efficiently develop programs for heterogeneous ARM SoCs, it is important to standardize some of the frameworks for communication between the different software environments running on Cortex-A and Cortex-M processor subsystems. The MCA working group is focused on standardizing the APIs, providing detailed documentation for the specification, and expanding the functionality on OpenAMP. More details can be found at [2] and [3].

With the Linux operating system, when the user wants to start, stop, or execute another task, the *remoteproc* command is used. When one application needs to communicate with another application, the *rpmsg* APIs are used, which are now a part of mainstream Linux and found in all current kernel distributions. This hides the complexities of managing heterogeneous hardware and software environments and provides a simplified application level interface to the user. The Cortex Microcontroller Software Interface Standard (CMSIS) is adopting the OpenAMP (Open Asymmetric Multi Processing) framework.

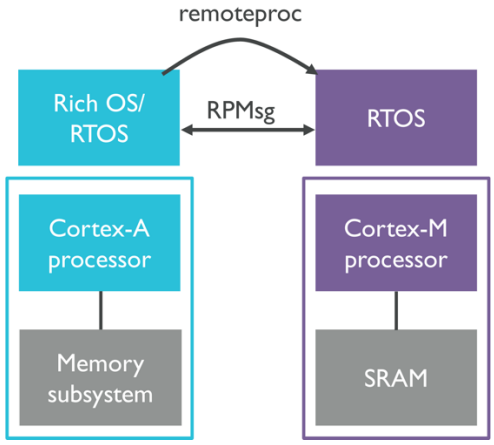


Fig. 10. Standardizing communication between different software environments running on Cortex-A and Cortex-M processor subsystems.

The other challenge faced by developers working on HMP systems is debugging issues across different software environments. To enable the ease of debugging issues across different software environments, the debugger should support

capabilities to provide a detailed visibility into the HMP system. For example, ARM's DS-MDK debugger offers complete visibility of all software applications running on Cortex-A and Cortex-M processors, thus making it easier to debug issues on such systems.

The third challenge for developers is tuning for performance by identifying performance hotspots at the processor and the system level for these HMP systems. Having access to detailed performance metrics from the applications running on the HMP system is critical in spotting the performance bottlenecks for the processor subsystem. In addition, several other CPU performance counters such as cache hits/misses, branch prediction and memory usage are also collected to provide insights into performance issues at system level. The Streamline analyser within DS-MDK helps with performance tuning and hot-spot identification for ARM processors based HMP system.

V. SUMMARY

A clear majority of embedded applications are transitioning to heterogeneous multiprocessor systems (HMP) SoCs using Cortex-A and Cortex-M processors. This expands the reach of traditional embedded systems by providing rich functionality and higher performance in addition to meeting the deterministic, real-time control requirements for such systems. These systems also benefit from the two largest and growing embedded software ecosystems for Cortex-A and Cortex-M processors. This paper highlights the design considerations and different approaches to design such HMP systems. In addition, the introduction of ARMv8-M architecture expands the capabilities of such HMP systems making them more robust and flexible for covering a wide variety of current and future use cases.

VI. REFERENCES

[1] ARM TrustZone, https://developer.arm.com/technologies/trustzone

[2] Open Asymmetric Multiprocessing (OpenAMP) overview http://www.multicore-association.org/workgroup/oamp.php

[3] OpenAMP overview on github https://github.com/OpenAMP/open-amp/wiki

[4] Programmer's guide for ARMv8-A http://infocenter.arm.com/help/topic/com.arm.doc.den0024a/DEN0024A_v8_architecture_PG.pdf

[5] ARMv8-M architecture reference manual