WHITE PAPER

# ARM Cortex-R Architecture

*For Integrated Control and Safety Applications*

Simon Craske, Senior Principal Engineer October, 2013

## Foreword

The ARM® architecture continuously evolves to support deployment of energy-efficient computation devices in a growing spectrum of applications that can take advantage of progress in semiconductor technology.

Recent advances included the Large Physical Address Extensions (LPAE) for the ARMv7-A applications architecture and the new 32-/64-bit ARMv8-A applications architecture.

This whitepaper discloses, for the first time, preliminary details of ARM's next step in architecture development for the ARM Cortex®-R real-time processor series.

These developments will lead to a new generation of Cortex-R processors that will meet the needs of integrated control and safety systems in applications such as automotive Advanced Driver Assistance Systems (ADAS), Hybrid Electric Vehicle (HEV) power train control and factory automation.

This paper provides an opportunity for SoC and MCU designers to factor these architectural developments into their product planning, and for OEMs and ecosystem partners to be informed about what is driving the Cortex-R processor roadmap forward.

## Introduction

ARM is known primarily for its range of Cortex-A processors used in significant numbers of consumer devices such as smartphones and tablets. Each of these processors is based on evolutions of ARM's application profile (A profile) architecture, with each generation adding new features for increased performance and capabilities, while ensuring compatibility with a broad software ecosystem. Some features -- for example, the addition of 64-bit processing in ARMv8-A -- benefit significantly from being presented to the ARM ecosystem ahead of products containing this feature becoming available; this provides an opportunity for discussion and development of associated software, including operating systems, and system IP collateral, as well as providing guidance on ARM's intended direction.

In addition to the A profile, the ARM architecture also contains profiles targeting the specific needs of embedded, real-time processors and microcontrollers, respectively called the Real-time (R profile) and Microcontroller profiles (M profile). Found in systems ranging from anti-lock-braking to white goods to cell phone radios, the Cortex-R and Cortex-M series processors, based on these profiles, form the relatively unknown masses of CPUs that make the everyday world work in a safe and reliable way.

While not announcing any new specific processors, this white paper aims to provide an introduction to the next evolution in the ARM R architecture profile used by ARM's Cortex-R series processors, and in doing so,  kick-start changes in the ecosystem to accommodate the benefits this new architecture brings to integrated control and safety systems.

## Today's Cortex-R Architecture

ARM's current lineup of embedded, real-time processors is based on the ARMv7-R architecture, and is formed of three complementary processors: the Cortex-R4, Cortex-R5 and Cortex-R7 processors. In common with processors based on the ARMv7-A architecture, these processors execute both the ARM (A32) and Thumb® (T32) instruction sets, but differ by offering a range of features for safety and real-time applications.

From an architectural point of view, the key difference between the A and R profiles are the memory system capabilities. As shown in Table 1, the ARMv8-A profile provides full virtual memory support via the Virtual Memory System Architecture (VMSA). VMSA uses translation tables located in memory and cached in a Translation Lookaside Buffer (TLB), and is capable of running rich operating systems like those commonly found on desktop and mobile-phone platforms. The ARMv8-R profile implements memory protection without translation via the Protected Memory System Architecture (PMSA). The PMSA scheme uses registers tightly coupled to the processor in a Memory Protection Unit (MPU) to provide protection of memory without the non-deterministic behavior introduced by potential TLB misses. PMSA provides support for running Real-Time Operating Systems (RTOS) with the ability to prevent erroneous execution of tasks corrupting either other tasks or the kernel.

| ***A profile*** | ***R profile*** | ***M profile*** |
|---|---|---|
| Application profile:<br>- 32-bit and 64-bit registers<br>- A32 (ARM), T32 (Thumb) and A64 instruction sets<br>- Virtual memory support<br>- Runs rich operating systems<br>- Virtualization extension | Real-time profile:<br>- 32-bit register width<br>- ARM and Thumb instructions<br>- Protected memory support<br>- Runs real-time OS | Microcontroller profile:<br>- 32-bit register width<br>- Thumb instruction set only<br>- Protected memory support<br>- Microcontroller applications |

**Table 1 - Comparison of profile features**

| *Cortex-R4* | *Cortex-R5* | *Cortex-R7* |
|---|---|---|
| Introduced 2005:<br>- ARMv7-R architecture<br>- High-performance, real-time, deeply embedded processor<br>- Deterministic event response<br>- Soft and hard error handling<br>- Configurable feature set | Introduced 2010:<br>- ARMv7-R architecture<br>- Low-latency peripheral port<br>- Accelerator coherency port<br>- Dual core in split or lock step<br>- Bus error management<br>- Smaller floating-point unit<br>- Enhanced memory protection | Introduced 2012:<br>- ARMv7-R architecture<br>- Large performance increase<br>- Advanced micro-architecture<br>- Higher clock frequency<br>- Symmetric multiprocessing<br>- Accelerator coherency port<br>- Quality of service features<br>- Enhanced error management<br>- Integrated interrupt controller |

**Table 2 – ARM Cortex-R portfolio evolution**

As shown in Table 2, since the introduction of the Cortex-R4 processor in 2005, the features and capabilities of ARM's embedded processors have been driven to provide a portfolio capable of supporting high-performance computing solutions for embedded systems, where high availability, fault tolerance, maintainability and real-time responses are required, such as:

| | |
|---|---|
| Automotive: | Airbag, braking, stability, dashboard, engine management |
| Storage: | Hard disk drive controllers, Solid state drive controllers |
| Mobile handsets: | 3G, 4G, LTE, WiMax smartphones and baseband modems |
| Embedded: | Medical, industrial, high-end microcontroller units (MCU) |
| Enterprise: | Networking and printers; Inkjet and multi-function printer |
| Home: | Digital TV, BluRay players and portable media players |
| Cameras: | Digital still camera (DSC) and digital video camera (DVC) |

In addition to the requirements of today's systems, ARM sees four new challenges for real-time application developers. These can be summarized as:

**Desire for consolidation**: The aim of providing more capable systems results in a desire to merge previously multiple discrete systems onto a single processor, bringing new challenges regarding isolation of these previously independent systems.

**Increased safety and integrity**: The evolution of safety standards and a requirement for more robust systems are resulting in demands for improved isolation between tasks along with guaranteed quality of service for interrupts.

**Demand for feature-rich software**: The ability to reuse communication stacks, such as Ethernet or WIFI, along with other libraries and applications from the application world, in a real-time system compatible way.

**New higher-performance applications**: New application markets, such as radar processing in automotive and improved graphics in human machine interfaces (HMI), result in new computational requirements.

## ARMv8-R Architecture

The ARMv8-R architecture represents the latest evolution of the ARM real-time architecture profile. While adopting some features from the ARMv8-A architecture announced in 2011, ARMv8-R remains a 32-bit architecture using the AArch32 exception model (compatible with that used in ARMv7-R) and executing the A32 (ARM) and T32 (Thumb) instruction sets. In addition to the real-time features already present in ARMv7-R, the ARMv8-R architecture adds a number of key architectural capabilities aimed at addressing the requirements of future integrated control and safety applications.

## Consolidation via Virtualization

To address the requirement to be able to consolidate multiple systems onto a single processor, ARMv8-R architecture brings support for hardware virtualization. In common with ARMv8-A, this results in the addition of a new exception level of higher priority than any that already exists on current Cortex-R processors. Figure 1 illustrates the new exception level's register file entries.

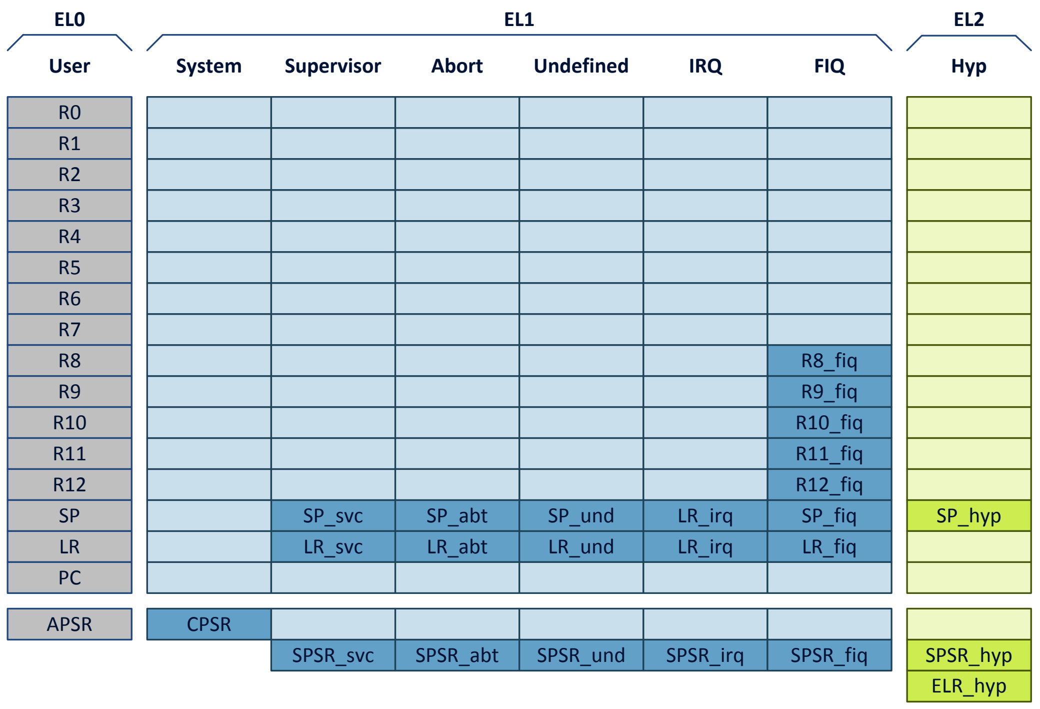

**Figure 1 - AArch32 register file**

The addition of virtualization, when combined with appropriate hypervisor software (also known as a virtual machine monitor), provides the ability to isolate memory and processing time between numerous different guest operating systems and their tasks/applications.

As show in Figure 2, a single hypervisor at exception level 2 (EL2) is capable of hosting multiple guest operating systems, with each guest operating system (OS) still retaining two distinct exception levels, typically one for the kernel (EL1) and one for its applications/tasks (EL0). Flexibility in the architecture also permits time critical tasks, such as interrupts and device drivers, to be run directly by the Hypervisor as its own tasks at EL0.

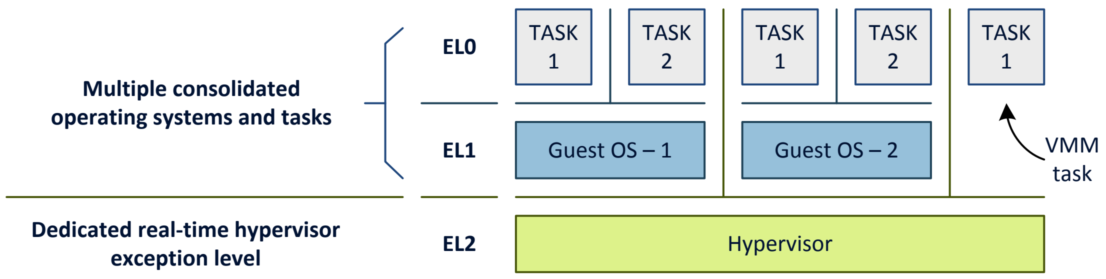
Multiple consolidated operating systems and tasks
Dedicated real-time hypervisor exception level
EL0
EL1
EL2
TASK 1
TASK 2
TASK 1
TASK 2
TASK 1
Guest OS – 1
Guest OS – 2
VMM task
Hypervisor

**Figure 2 - Three exception level model**

In order to maintain determinism, the stage-2 memory translation scheme of the ARMv8-R architecture differs from that of a conventional system, such as ARMv8-A. In particular, a conventional scheme would support translation at stage-2, i.e. every memory address produced by EL1 would traditionally be translated to a new address by a second stage of translation; whereas the stage-2 scheme in the ARMv8-R architecture only supports protection. As a result, the intermediate physical address (IPA) generated by a given guest OS is always the same as the final Physical Address (PA). Like the PMSA Memory Protection Unit (MPU) of ARMv7-R architecture, the MPU is register-based and tightly coupled to the processor, providing fast, deterministic responses. This avoids cases, for example, causing failures in hard real-time systems when an interrupt is not serviced in the required time due to the overhead of page-table walks required to refill the TLB on a miss.

The stage-2 MPU ensures the integrity of the hypervisor software and isolates each of the guest operating systems to its own physical address space, controlling whether or not a particular OS can assess any given peripheral or memory location.

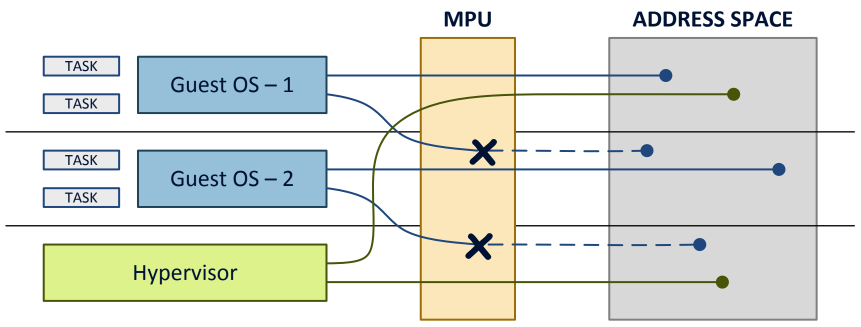

**Figure 3 - Stage-2 MPU capability**

## Safety and Integrity via PMSAv8

The ARMv8-R architecture replaces the PMSA of ARMv7-R with a new scheme intended to increase flexibility and ease-of-use, as well as reduce reprogramming time, and by association, context switch times.

The PMSA is fundamentally based on the use of an MPU. In the ARMv7-R architecture, the MPU is specified such that it supports memory regions defined as a power of two in size, where a given region has to be aligned to a base address equal to an integer multiple of its size. While this scheme yields a very low logic gate count implementation, it requires effort on the part of software to conform to these strict usage rules. The new scheme provided by PMSAv8 offers a significant improvement in flexibility for software: the MPU permits a region to start and end at any address equal to an integer multiple of 64 bytes. As Figure 4 illustrates, this permits a single region to describe a given memory region that requires the concatenation of numerous regions in PMSAv7.

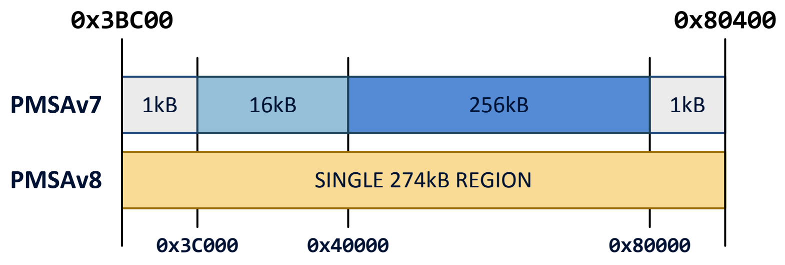

**Figure 4 - PMSAv7 vs PMSAv8 MPU example**

The potential to describe the required memory map with fewer regions allows for faster programming of the MPU, and thus context switching. This is further enhanced by PMSAv8 providing direct access to each of the MPU region registers which, when compared with the indirect mechanism used by PMSAv7, reduces the number of system register writes required to program the same number of regions.

The ARMv8-R architecture implements separate PMSAv8 compatible EL1 controlled stage-1 and EL2 controlled stage-2 MPUs. This allows EL1 operating systems to reconfigure their memory protection without intervention by the hypervisor software. However, the stage-2 MPU always makes the final decision about whether or not a memory transaction is permitted, ensuring a robust system.

## Feature-Rich Software via VMSA

In order to enable use of the broader range of software assets available in the application processor and rich operating system world, the ARMv8-R architecture includes support for the full Virtual Memory System Architecture (VMSA) at EL1 and EL0. This permits one of the guest operating systems to be a rich OS, while retaining real-time responsiveness for the hypervisor and other RTOS guests.

The VMSA guest is compatible with AArch32 of the ARMv8-A architecture, permitting use of operating systems compatible with ARMv7-A architecture. The VMSA provides full support for both short and long descriptor types (as introduced with the large physical address space extensions), though the IPA is always policed by the stage-2 MPU, and is thus limited to 32-bits.

## Computation via Advanced SIMD

Although permitted by the ARMv7-R architecture, no Cortex-R processor implementations currently include support for ARM’s advanced SIMD extensions (also known as NEON™ technology). The option for inclusion of VMSA support in the ARMv8-R architecture implementations, and the ability to easily reuse rich OS applications and libraries, provide an opportunity for increased computation capability.

## Interrupts and System Errors

Additional new features in the ARMv8-R architecture include system register mapping of the interrupt control registers and the addition of a System Error Interrupt (SEI). The mapping of interrupt control to system registers improves interrupt response time by removing the need for memory transactions to be performed in order to determine which interrupt pin caused the interrupt exception to be taken, while the addition of a dedicated SEI provides a means of handling critical system errors.

## Conclusion

The ARMv8-R architecture brings new technology to the world of integrated control and safety applications. Providing virtualization and new memory protection features, theARMv8-R architecture offers the ability to consolidate and safely isolate multiple systems. At the same time, the ARMv8-R architecture offers the addition of VMSA capabilities to real-time systems, allowing the combination of communication and other software stacks with embedded software for fully featured systems.

This white paper provides an early preview of the architecture and its capabilities ahead of any product announcement with the intention of facilitating discussion and development of the associated eco-system collaterals.

## References

For further details on ARM's architecture profiles and Cortex-R processor portfolio, please visit the ARM website at www.arm.com or the new online ARM Connected Community.