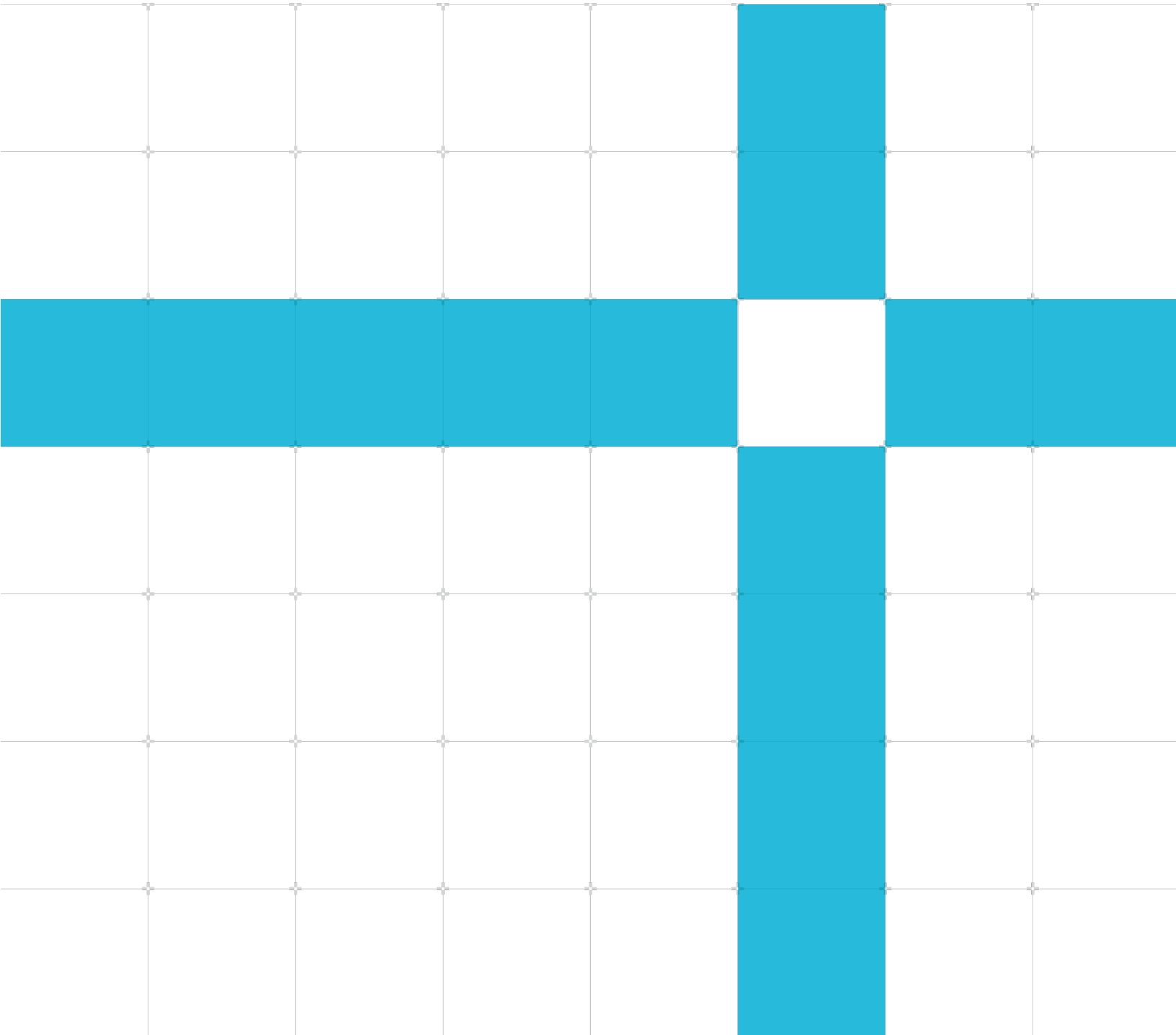
arm

ISAs, commonality and differentiation

Non-Confidential

Issue 1.0
Document ID: 102252



ISAs, commonality and differentiation

Release information

Document history

Issue	Date	Confidentiality	Change
01	9th September 2020	Non-Confidential	First release

Non-Confidential Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349)

Confidentiality Status

Contents

1 Overview

This guide describes some of the features that are specific to each Arm Instruction Set Architecture (ISA), and considers which applications make best use of those features. The focus of the guide is Cortex-R. However, we also consider Cortex-A and Cortex-M, considered in the context of helping a developer make the right choice for their project.

Cortex-R processors are niche, but have some unique features that make them well-suited to their target markets. Even within this niche, Arm has continued to innovate, with features like support for real-time hypervisors on Cortex-R52. Now, Cortex-R82 takes these features even further.

This guide also considers the context in which Cortex-R82 was developed, by reviewing some of the other innovations that Arm has introduced during the last twenty years. Rather than considering the processor in isolation, we describe how Arm has evolved instruction sets, and carefully chosen how to add new functionality. We hope that this information helps you to understand why Cortex-R82 includes the features that it does.

This wider context may also be useful in choosing a device for your own application, especially if you are considering how the features of our processors may evolve in the future. The section in this document on How to choose a processor outlines some of the factors to consider when you choose an Arm processor for your projects.

1.1 Before you begin

If you are not familiar with the basics of the Arm architecture, we recommend that you read **Introducing the Arm architecture** before you read this guide. Introducing the Arm architecture provides a high-level overview of some of the features that we describe in this guide.

2 The Cortex families

In 2004, Arm introduced the Cortex-M family with Cortex-M3. Cortex-M3 gave developers an energy-efficient MCU with a simpler programming model that does not require assembly coding.

Cortex-M3 features include automatic register stacking on interrupt entry, and a simple but low-latency vectored interrupt controller. Over time, the Cortex-M family expanded to add processors with additional features including Floating Point Unit (FPU), Single Instruction Multiple Data (SIMD), and most recently, TrustZone followed by vector processing.

When Cortex-M launched, there was already demand for a higher level of performance. Developers wanted a programmer's model that was closer to Cortex-A, but suitable for markets with hard real-time requirements. Instead of a Memory Management Unit (MMU), Cortex-R processors were introduced with memory protection and tightly coupled memories. These features made it easier for Cortex-R processors to guarantee worst-case execution time and meet the needs of markets like storage, automotive, and modems. Other architectural optimizations include the ability to interrupt load/store multiple instructions. Interrupt controllers are often optimized for very low latency.

2.1 Adding 64-bit

One of the biggest changes in recent years was the introduction of 64-bit processing to the Cortex-A family. This change was first incorporated in Cortex-A53 and Cortex-A57, in 2012. This development was in response to partner needs for the mobile phone market. Since then, the ecosystem has matured, and the market has grown to include HPC and hyperscale.

2.2 Cortex-R82 and the Armv8-R architecture

Cortex-R52, the first Armv8-R architecture processor, was announced in 2016. Cortex-R52 added an extra Exception level, EL2, to allow software to run a hard real-time hypervisor. Although virtualization is nothing new, the ability to execute virtualization without compromising real-time responsiveness is unique. This feature particularly benefits the automotive industry. Cortex-R82 continues with support for a Memory Protection Unit (MPU) at EL2, and MMU or MPU at EL1.

Cortex-R82 includes performance enhancements, and two major new features:

- 64-bit instruction set: For storage, 64-bit support provides the ability to address more than 4GB of memory. Modern SSDs need around 1GB of cache memory for every TB of disk capacity. This means that, as storage capacities continue to grow, the ability to support large cache memories is increasingly important. There is a second use case, too. Cortex-R82 can share the same address map and memory system as a 64-bit Cortex-A applications processor.
- MMU: The MMU gives Cortex-R82 the ability to run Linux. The processor still retains the classic Cortex-R features, including an MPU and Tightly Coupled Memories (TCMs). These features allow a cluster of cores to run a mix of hard real-time and rich OSs. This addresses new opportunities like computational storage.

The new features in Cortex-R82 provide access to the rich ecosystem of Linux and applications that are already ported to the Armv8-A architecture, but still provides the classic Cortex-R features of

support for hard real time. Having an MMU might seem contradictory to this, given the risk of translation table walks at critical times. Cortex-R82 supports both MPU and MMU. Therefore, we might expect that hard real-time event handling is done with software using the MPU, which also gives access to TCMs, and that less critical code runs using the MMU. Translation table walks can be interrupted to switch to critical routines. Using TCMs for critical code and data guarantees that no walks are required by the handler routines themselves.

In many systems, we expect that, rather than operating multiple OSs on a single core, we will instead see multiple cores in a cluster with partitioning between dedicated real-time and Linux cores. This flexibility means that a single SoC can address multiple application use cases. This becomes more valuable as process geometries shrink and the cost of new tapeouts continues to rise.

3 Right-sized innovation

The Arm architecture families have evolved over the years to add new features or reach higher performance points. Arm always considers solutions holistically, so that a feature does not compromise another aspect of the family. For example, MVE on Cortex-M55 was carefully architected to consider both raw compute performance and available bus bandwidth and energy envelope. Trying to fit The Armv8-A Scalable Vector Extension (SVE) into a Cortex-M core is not feasible.

Similarly, **Trustzone for Cortex-M** was designed to retain the key aspects of microcontrollers: power efficiency, simple software stack, and fast context switch. For Cortex-R82, an important goal was to preserve the ability to handle hard real-time tasks while still enabling Linux support.

4 Software ecosystems

A software ecosystem and developer community has developed around each Cortex family. These ecosystems contain many of the software building blocks that you need, from component libraries to tools like compilers. The **Arm Developer Community** includes support forums where you can ask for help with your own projects.

One of the benefits of the variety of Arm-based chips is the wide range of commercial and open source software that supports them. If you need to find additional engineering expertise, many engineers have experience working with Arm-based devices.

For Cortex-M devices, **CMSIS libraries** include **machine learning** and **DSP** functionality, many real-time operating systems, and middleware libraries. For Cortex-A, there is extensive support for Linux and commercial operating systems. Arm and partners actively contribute to Linux development through the **Linaro** organization.

To allow software ecosystems to grow successfully, the instruction set needs to be well-defined and governed. This enables software to be readily portable between different implementations of the same architecture. For example, mobile phone app stores include apps that are developed once, but are deployed on many devices.

5 How to choose a processor

As a developer choosing an Arm-based processor, how do you decide which one you need? If you have been using Arm for a while, you probably already have your favorite family. If you are new to Arm, the choice might seem overwhelming. Let's consider some of the factors that might affect your choice.

Your starting point might depend on whether you are a software engineer or a hardware engineer. Arm processors cover a vast performance spectrum, but there is overlap in the middle ground of high-end Cortex-M, low-end Cortex-A, and Cortex-R. However, the software ecosystems for each family are quite different, and this factor might affect your choice. The overlap between families is partly because partners want to retain their ecosystem investment, but address new applications that require higher performance or push rich IoT devices into smaller energy footprints.

If your primary focus is low-power, energy-efficient applications, or perhaps a battery-powered IoT device, then Cortex-M is probably the right choice. Cortex-M includes a rich ecosystem of real-time operating systems and CMSIS libraries. **Trusted Firmware-M** provides a secure foundation if you are developing a device with connectivity. It is easy to scale up to more performant devices like Cortex-M7 or Cortex-M55. Cortex-M55 also introduces extensions that can help with machine learning and DSP tasks.

If you need Linux or another rich operating system that requires an MMU, then Cortex-A is the best option. Until the launch of Cortex-R82, Cortex-A was the only option. Cortex-A still scales down to energy-efficient CPUs like Cortex-A5 and Cortex-A32. The Armv8-A architecture is used by partners like Fujitsu in its A64FX supercomputer processor, and Amazon in its Graviton hyperscale platform. You can see that there are plenty of options for high-end devices. The Linux ecosystem has also made Cortex-A processors popular in single board computers, including the Raspberry Pi. Cortex-A processors have the richest instruction sets of the Arm Cortex families. You can learn more in **Introducing the Arm architecture**.

With the introduction of Cortex-R82, there is now an increased overlap between features in the Cortex-R and Cortex-A families. Indeed, the raw instruction sets for both are similar. Cortex-R82 runs user-mode Arm v8.4 binary files unmodified, if required optional features, like Neon, are present. The main differences between Cortex-A processors and Cortex-R82 are in the memory model. The memory model is where the focus for Cortex-R82 is designed for the needs of developers with hard real-time requirements.

6 Models for software development

Models allow you to run code without needing hardware. Some models are just the architecture, for example the **Arm Ecosystem Models**. Other models include device-specific features and are configurable to match the hardware. **Fast Models** are ideal for programmers, because they execute quickly, but they do not provide performance data. **Cycle Models** execute more slowly. However, because Cycle Models are derived from the design data (RTL), they can be used for performance analysis and extend to include features like performance counters.

Cortex-R processors are mostly used in deeply embedded products. This means that it is harder to find development silicon if you want something to build or test software on. For Cortex-R52, you can use the **MPS3 development board**. Another option, which is available for other Arm cores too, is to use a model for software development work.

7 Arm platform solutions

Arm does not provide cores in isolation. For the Cortex-M family, our **Corstone** products provide a subsystem around the CPU. At the high end, Corstone-700 includes both Cortex-A and Cortex-M processors. Developers can use Corstone-700 to run something in low power mode, but wake up an applications processor when needed.

For rich IoT endpoints using Cortex-A processors, we work with partners on Project Cassini, which will provide a cloud-native experience to edge computing.

Perhaps your project needs an app ecosystem, or the ability for the user to run new applications dynamically. An RTOS-based system is typically composed and compiled together. This type of system allows very high-density code, and makes good use of compiler optimizations. This type of system is well suited to Cortex-M-based or Cortex-R-based systems. Firmware can be updated, and Arm Platform Security Architecture (PSA) provides a framework for doing this securely. But if you want user applications, Linux, Android, or another rich operating system is a better choice.

If you are a hardware engineer trying to decide which Cortex family to use, talk to your software engineering colleagues. This is because ecosystems do matter. From a hardware perspective, considerations might include:

- Requirements for interrupt latency
- Whether you need tightly coupled memory
- Whether you need symmetric multiprocessing (SMP) support. All our current Cortex-A cores support SMP. Within the Cortex-R family, only Cortex-R8 and Cortex-R82 support this feature.
- What sort of performance level you need

Benchmarks and their relevance are also a consideration, but are beyond the scope of this guide.

8 Related information

Here are some resources related to material in this guide:

- **Arm architecture and reference manuals -** Find technical manuals and documentation relating to this guide and other similar topics.
- **Arm Community** - Ask development questions, and find articles and blogs on specific topics from Arm experts.
- **Arm Ecosystem Models**
- **CMSIS on GitHub**
- **Corstone**
- **Cortex-M55**
- **Introducing the Arm architecture**
- **Making Helium: Why not just add Neon?** - Learn more about MVE in this blog.
- **MPS3 development board**
- **Taming of the Edge** - Learn more about Project Cassini in this blog.
- **Trusted Firmware-M**
- **Trustzone for Cortex-M**, including comparisons with Cortex-A

9 Next steps

This guide has briefly explained some of the differences between the different Arm processor families and their features. We have described some of the reasons that you might choose one processor rather than another for your project. You can learn more about our processors and their features on the **processors area** of Arm Developer.

Arm continues to evolve its architectures, adding features in response to customer demands and preventing fragmentation within each segment to ensure that software ecosystems can grow, too. Armv8-R AArch64 and Cortex-R82 continues this trend. What uses can you find for Cortex-R82?