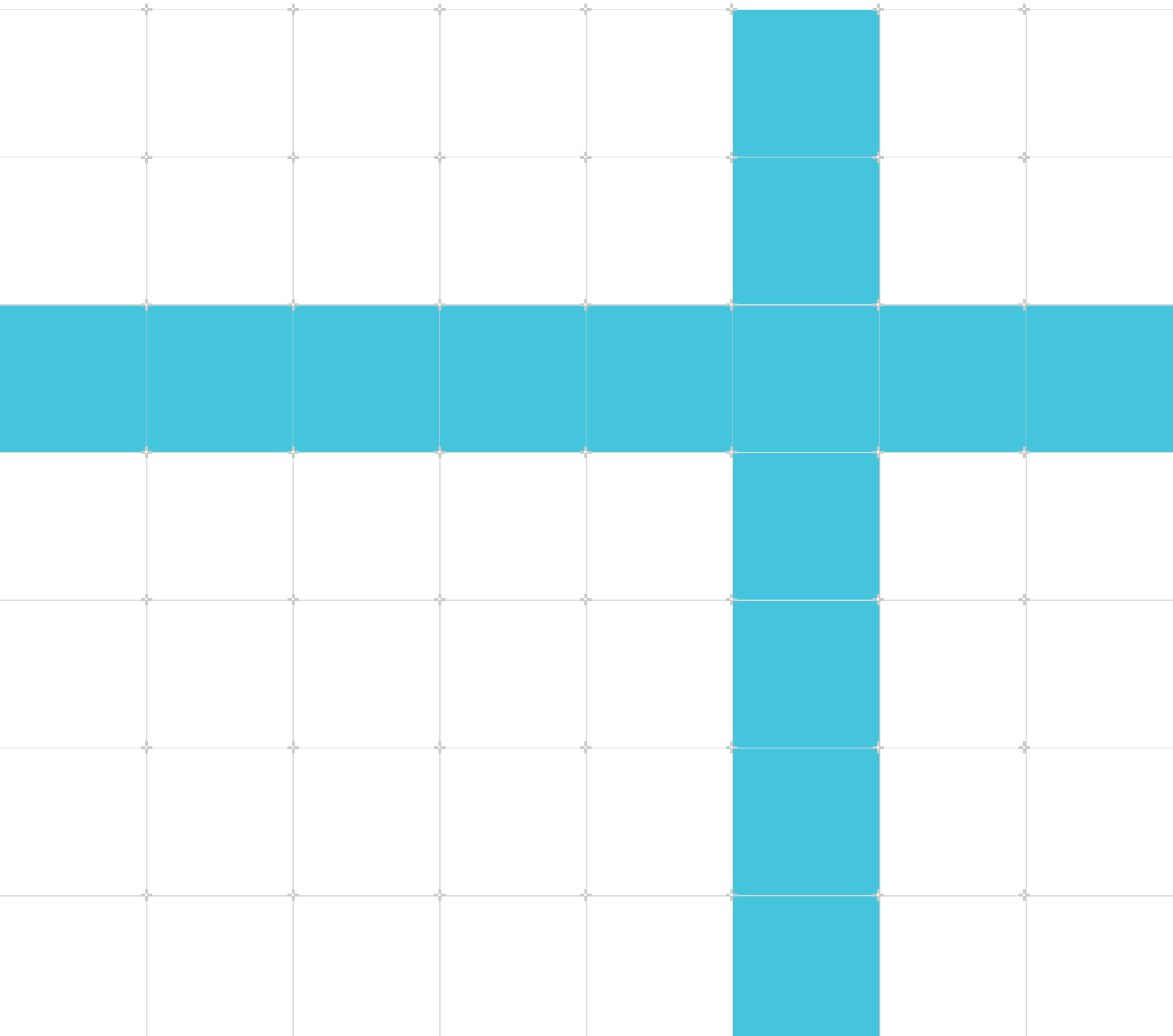
Learn the architecture - Introducing the Arm architecture

Version 2.1

Issue 01
102404_0201_01_en



Learn the architecture - Introducing the Arm architecture

Release information

Document history

Issue	Date	Confidentiality	Change
0100-01	1 April 2019	Non-Confidential	First release
0200-02	30 March 2021	Non-Confidential	Updated for v9
0201-01	23 February 2023	Non-Confidential	Minor modifications

Proprietary Notice

(LES-PRE-20349|version 21.0)

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Feedback

Arm® welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. Overview

The Arm architecture provides the foundations for the design of a processor or core, things we refer to as a Processing Element (PE).

The Arm architecture is used in a range of technologies, integrated into System-on-Chip (SoC) devices such as smartphones, microcomputers, embedded devices, servers and even super computers.

The architecture exposes a common instruction set and workflow for software developers, also referred to as the Programmer's model. This helps to ensure interoperability across different implementations of the architecture, so that software can run on different Arm devices.

This guide introduces the Arm architecture for anyone with an interest in it. No prior knowledge of the Arm architecture is needed, but a general familiarity with processors and programming and their terminologies is assumed.

At the end of this guide you can check your knowledge. You will have learned about the different profiles of the Arm architecture and whether certain features are architecture or micro-architecture specific.

2. About the Arm architecture

The Arm architecture is one of the most popular processor architectures in the world. Billions of Arm-based devices are shipped every year.

The following table describes the three architecture profiles: A, R, and M:

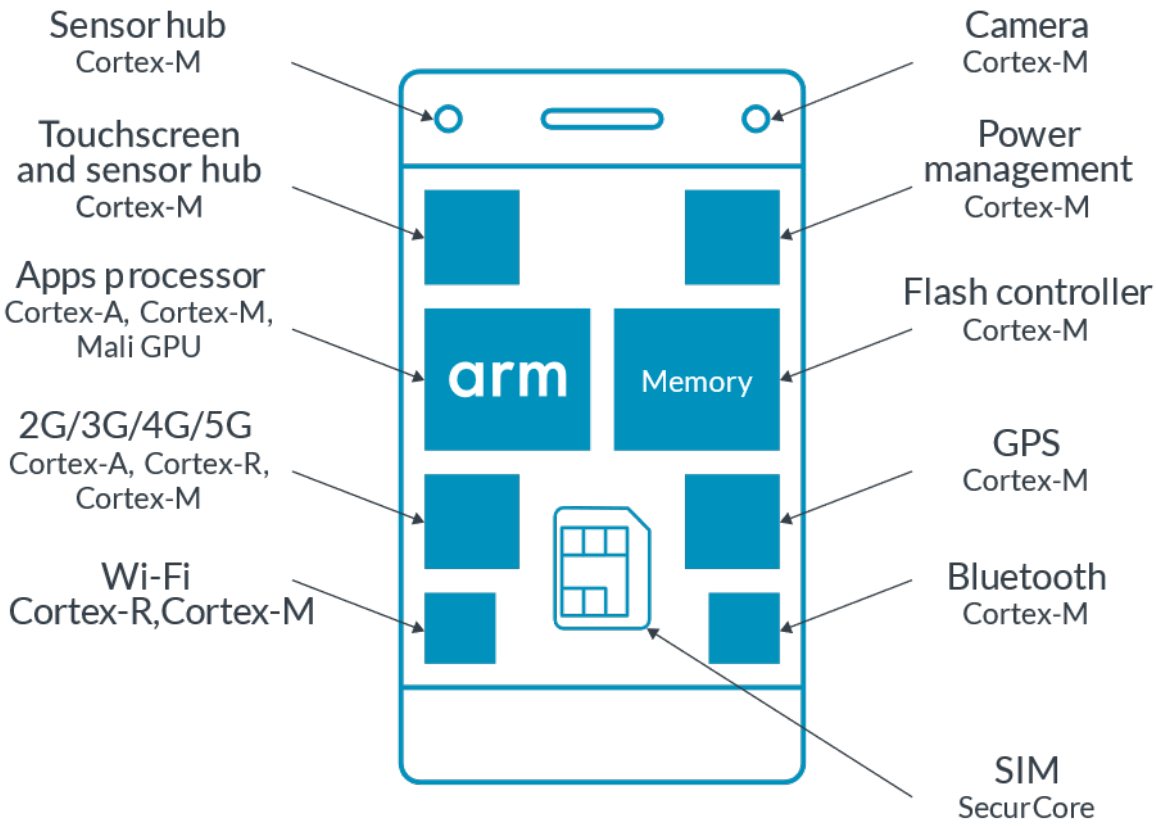
A-Profile (Applications)	R-Profile (Real-Time)	M-Profile (Microcontroller)
High performance	Targeted at systems with real-time requirements	Small, highly power-efficient devices
Designed to run a complex operating system, such as Linux or Windows	Commonly found in networking equipment, and embedded control systems	Found at the heart of many IoT devices

These three profiles allow the Arm architecture to be tailored to the needs of different use cases, while still sharing several base features.

Arm Cortex and Arm Neoverse are the brand names that are used for the Arm processor IP offerings. Our partners offer other processor brands using the Arm architecture.

The following diagram shows an example of an Arm based system:

Figure 2-1: About the Arm Architecture



This example smartphone contains the following processor types:

- An A-profile processor as the main CPU running a rich OS like Android.
- A cellular modem, based on an R-profile processor, provides connectivity.
- Several M-profile processors handle operations like system power management.
- The SIM card uses SecurCore, an M-profile processor with additional security features. SecurCore processors are commonly used in smart cards.

In this guide, we will only look at the A-profile architecture and its two latest versions, Armv8-A and Armv9-A.

3. What do we mean by architecture?

When we use the term architecture, we mean a functional specification. In the case of the Arm architecture, we mean a functional specification for a processor. An architecture specifies how a processor will behave, for example what instructions it has and what the instructions do.

You can think of an architecture as a contract between the hardware and the software. The architecture describes what functionality the software can rely on the hardware to provide. Some features are optional, as we explain in Architecture and micro-architecture.

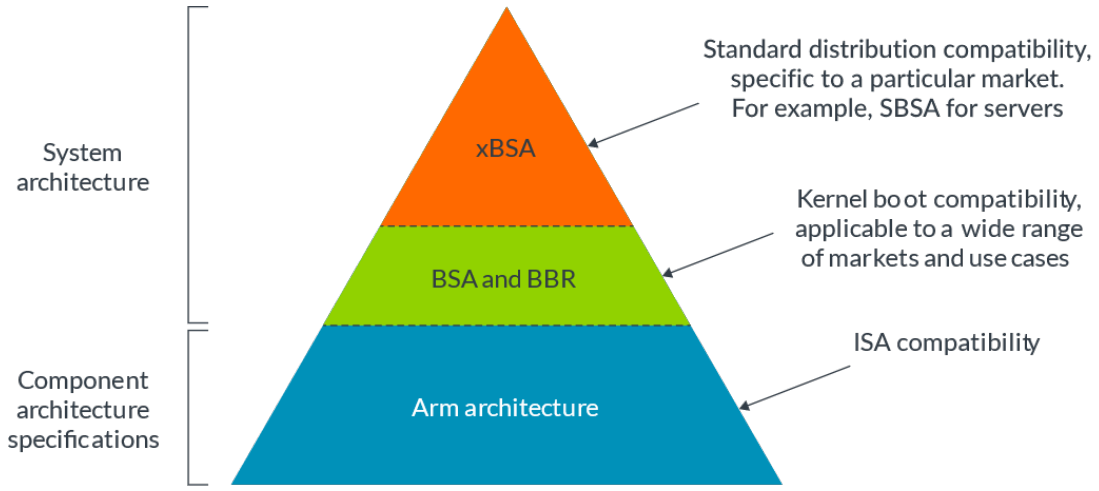
The architecture specifies:

-	Description
Instruction set	The function of each instruction How that instruction is represented in memory (its encoding)
Register set	How many registers there are The size of the registers The function of the registers Their initial state
Exception model	The different levels of privilege The types of exceptions What happens on taking or returning from an exception
Memory model	How memory accesses are ordered How the caches behave, when and how software must perform explicit maintenance
Debug, trace, and profiling	How breakpoints are set and triggered What information can be captured by trace tools and in what format

4. System architecture

Systems include more than just a processor core. Arm also provides specifications to describe the requirements for the system containing the processor as you can see in the following diagram:

Figure 4-1: Development of Arm Architecture



The specifications are the basis of software compatibility. Building hardware according to the specifications means that software can be written to match it. Writing software according to the specifications means that it can run on compliant hardware. The Arm Architecture is the first layer, providing a common programmer's model to software through Instruction Set Architecture (ISA) compatibility.

The Base System Architecture (BSA) specification describes a hardware system architecture that system software can rely on. The BSA covers aspects of the processor and system architecture, for example the interrupt controller, timers, and other common devices that an OS needs. This provides a reliable platform for standard operating systems, hypervisors, and firmware.

The BSA is widely applicable across different markets and use cases. Other standards can build on the BSA to provide market-specific standardization. For example, the Server Base System Architecture (SBSA) is a supplement to the BSA that targets servers. The SBSA describes the hardware and feature requirements for a server OS. The specification contains a set of levels that document hardware features in increasing detail, following the progression of the CPU architecture.

The Base Boot Requirements (BBR) specification covers requirements for systems that are based on Arm architecture and that operating systems and hypervisors can rely on. This specification establishes the firmware interface requirements, like PSCI, SMCCC, UEFI, ACPI, and SMBIOS.

BBR also provides the recipes for targeting specific use cases, for example:

- SBBR: Specifying UEFI, ACPI, and SMBIOS requirements to boot generic, off-the-shelf operating systems and hypervisors like Windows, VMware, RHEL, Oracle Linux, and Amazon Linux. The SBBR also supports other OSes like Debian, Fedora, CentOS, SLES, Ubuntu, openSUSE, FreeBSD, and NetBSD.
- EBBR: Specifying, along with the EBBR specification, UEFI requirements to boot generic, off-the-shelf operating systems, like Debian, Fedora, Ubuntu, openSUSE, and providing benefits for vertically integrated OS platforms.
- LBBR: Specifying potential requirements for the LinuxBoot firmware, to boot the OSes that some hyperscalers use.

5. Architecture and micro-architecture

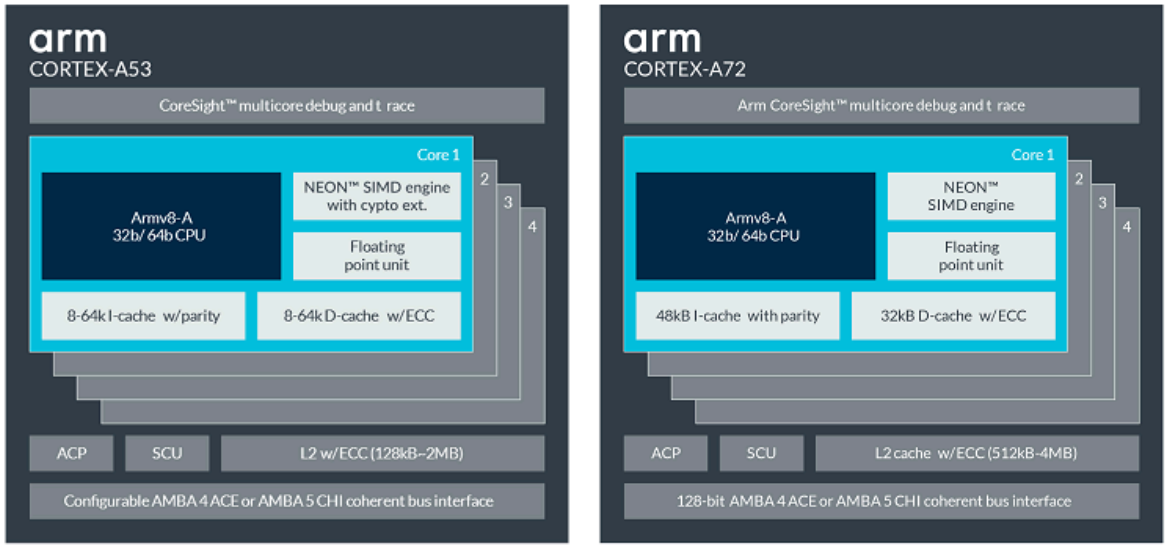
Architecture does not tell you how a processor is built or how it works. The build and design of a processor is referred to as micro-architecture. Micro-architecture tells you how a particular processor works.

Micro-architecture includes things like:

- Pipeline length and layout
- Number and sizes of caches
- Cycle counts for individual instructions
- Which optional features are implemented

For example, Cortex-A53 and Cortex-A72 are both implementations of the Armv8-A architecture. This means that they have the same architecture, but they have very different micro-architectures, as shown in the following illustration and table:

Figure 5-1: Arm Cortex-A53/A72 chip



Architecture	Cortex-A53	Cortex-A72
Target	Optimized for power efficiency	Optimized for performance
Pipeline	8 stages In-order	15+ stages Out-of-order

Architecture	Cortex-A53	Cortex-A72
Caches	L1 I cache: 8KB - 64KB L1 D cache: 8KB - 64KB L2 cache: optional, up to 2MB	L1 I cache: 48KB fixed L1 D cache: 32KB fixed L2 cache: mandatory, up to 2MB

Software that is architecturally compliant can run on either the Cortex-A53 or Cortex-A72 without modification. This is because they both implement the same architecture.

6. Development of the Arm architecture

The Arm architecture has developed over time and each version builds on what came before.

You will commonly see the architecture referred to as something like:

```
Armv8-A
```

This means Version 8 of the architecture, for A-Profile.

Or, in short form:

```
v8-A
```

Armv8-A

Armv8-A was announced in 2011 and was the first 64-bit version of the Arm Architecture. Armv8-A based devices have been deployed in everything from mobile phones to supercomputers.

Armv9-A

Armv9-A is the latest version of the Arm Architecture for A-profile. Armv9-A builds on Armv8-A and adds new features, including:

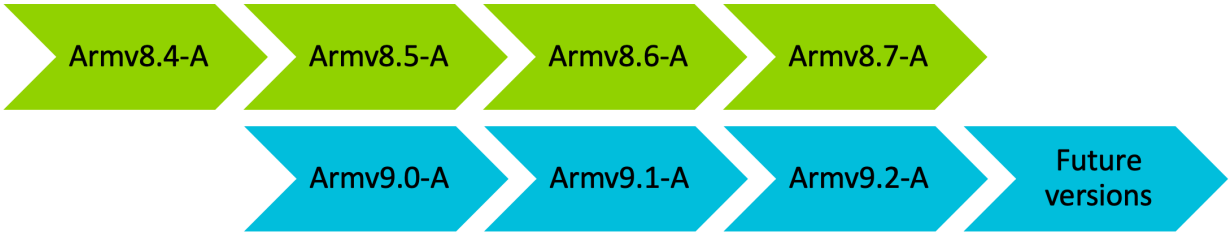
- Scalable Vector Extension, version 2 (SVE2)
- Transactional Memory Extension (TME)
- Branch Record Buffer Extension (BRBE)
- Embedded Trace Extension (ETE)
- Trace Buffer Extension (TRBE)

Also, some of the features that were optional in Armv8-A are mandatory in Armv9-A.

Annual updates

Arm publishes annual updates to the architecture, adding new instructions and features. Armv9.0-A aligns with Armv8.5-A, inheriting all the features from Armv8.5-A and adding new features. After the initial release of Armv9-A, Armv8-A and Armv9-A are updated together. Arm will release new features for Armv9-A, and will continue to update and maintain Armv8-A. The following diagram shows the parallel releases:

Figure 6-1: Releases

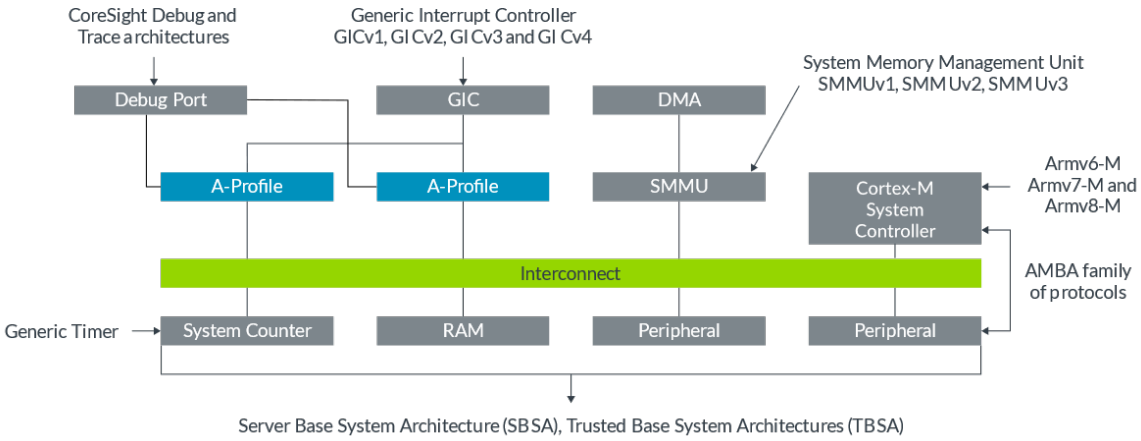


You can find more information about what has been added in each annual update in Understanding the Armv8.x extensions.

7. Other Arm architectures

The Arm architecture is the best-known Arm specification, but it is not the only one. Arm has similar specifications for many of the components that make up a modern System-on-Chip (SoC). This diagram provides some examples:

Figure 7-1: Interconnect diagram



- Generic Interrupt Controller: The Generic Interrupt Controller (GIC) specification is a standardized interrupt controller for use with Armv7-A/R and Armv8-A/R.
- System Memory Management Unit: A System Memory Management Unit (SMMU or sometimes IOMMU) provides translation services to non-processor masters.
- Generic Timer: The Generic Timer provides a common reference system count to all the processors in the system. These timers provide functionality which is used for things like the operating system scheduler tick. The Generic Timer is part of the Arm architecture, but the system counter is a system component.
- Server Base System Architecture and Trusted Base System Architecture: The Server Base System Architecture (SBSA) and Trusted Base System Architecture (TBSA) provide system design guidelines for SoC developers.
- Advanced Microcontroller Bus Architecture: The Advanced Microcontroller Bus Architecture (AMBA) family of bus protocols control how components in an Arm-based system are connected, and the protocols on those connections.

8. Understanding Arm documentation

Arm provides a lot of documentation to developers. In this section of the guide, we explain where to find documentation and other information for developing on Arm.

Where is the documentation?

You can download the Arm architecture and processor manuals from the Arm developer website.

You can ask development questions and find articles and blogs on specific topics from Arm experts in the Arm community.

Which document describes what?

Here is a short description of the different types of documentation:

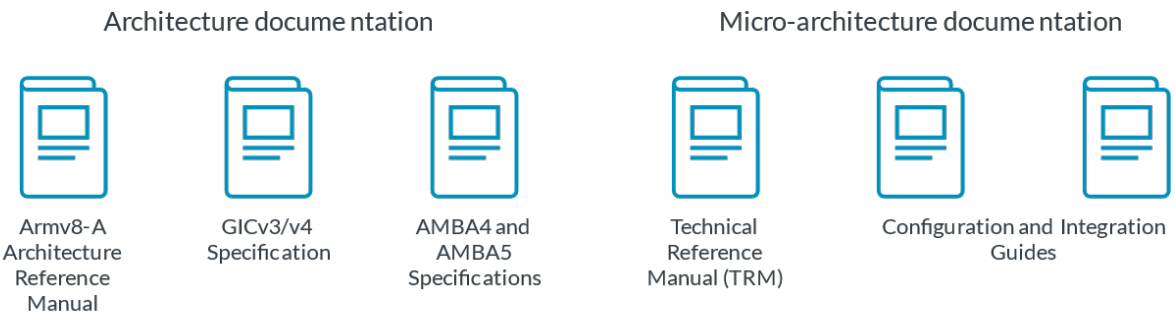
- Each Arm Architecture Reference Manual describes an architecture specification. An Arm Architecture Reference Manual is relevant to any implementation of that architecture.
- Each Arm Cortex processor has a Technical Reference Manual (TRM). The TRM describes the features specific to that processor. In general, the TRMs do not repeat any information given in the Arm Architecture Reference Manuals.
- Each Arm Cortex processor also has a Configuration and Integration Manual (CIM). The CIM describes how to integrate the processor into a system. Generally, this information is only relevant to SoC designers.

The CIMs are only available to IP licensees. The TRMs are available to download from Arm Developer without a license.

So, what does this mean for me?

If you are looking for information on a particular processor, you might need to refer to several different documents. Here we can see the different documents you might need to use with a Cortex-A75 processor:

Figure 8-1: Architecture documentation



Cortex-A75 implements ARMv8.2-A, a GICv4 CPU interface and AMBA bus interfaces, so you must refer to separate documents for each element. You also must refer to the documents that describe the micro-architecture.

If you are working with an existing SoC, you will also use documentation from the SoC manufacturer. This documentation is typically referred to as a datasheet. The datasheet gives information specific to that SoC.

What information will I find in each document?

The following table shows which information is shown in the different types of documents:

-	Architecture	-	-	Micro-architecture	Micro-architecture	-
	Arm Architecture Reference Manual	GIC specifications	AMBA specifications	TRM	CIM	SoC Datasheet
Instruction set	x					
Instruction cycle timings				x		
Architectural registers	x	x				
Processor specific registers				x		
Memory model	x					
Exception model	x					
Support for optional features				x	x (some might be synthesis choice)	
Size of caches/TLBs				x		
Power management				x		
Bus ports				x	x	
All legal bus transactions			x			
Bus transactions generated by processor				x		
Memory map						x
Peripherals						x
Pin-out of SoC						x

Differences between reference manuals and user guides

The documents we have looked at so far, Arm Architecture Reference Manuals, TRMs and CIMs, are reference manuals. This means that they do not provide guidance on how to use the processor. For example, the Arm Architecture Reference Manual does not have a section on how to turn on an MMU.

This structure is deliberate, and is intended to keep a clear divide between the technical detail of what the architecture requires, which is found in reference manuals, and documents that provide more general guidance, for example, this guide. Some general guidance documents will introduce concepts, and others provide instructions for you to follow.

9. Common architecture terms

The architecture uses several terms, usually written in small capital letters in documentation, which have very specific meanings. The Arm Architecture Reference Manuals defines each of these term. In this section, we look at the most common terms and provide addition information about what they mean to programmers.

PE Processing Element

Processing Element (PE) is a generic term for an implementation of the Arm architecture. You can think of a PE as anything that has its own program counter and can execute a program. For example, the Arm Architecture Reference Manual states:

The states that determine how a PE operates, including the current Exception level and security state, and in AArch32 state, the PE mode.

Manuals use the generic term PE because there are many different potential micro-architectures. For example, the following micro-architectures are possible in the Arm Cortex-A processors:

- Cortex-A8 is a single core, single-thread processor. The entire processor is a PE.
- Cortex-A53 is a multi-core processor, each core is a single thread. Each core is a PE.
- Cortex-A65AE is a multi-core processor, each core has two threads. Each thread is a PE.

By using the term PE, the architecture is kept separate from the specific design decisions that are made in different processors.

IMPLEMENTATION DEFINED

A feature which is **IMPLEMENTATION DEFINED** (IMP DEF for short) is defined by the specific micro-architecture. The implementation must present a consistent behavior or value.

For example, the size of the caches is IMP DEF. The architecture provides a defined mechanism for software to query what the cache sizes are, but the size of the cache is up to the processor designer.

Similarly, support for the cryptography instructions is IMP DEF. Again, there are registers to allow software to determine if the instructions are present or not.

In both examples, the choice is static. That is, a given processor either will, or will not, support the features and instructions. The presence of the feature cannot change at runtime.

For Cortex-A processors, some IMP DEF choices will be fixed, and some will be synthesis options. For example, on Cortex-A57 the size of the L1 caches is fixed, and the size of the L2 cache is a synthesis option. However, the decision about the size of the L2 cache is made at design time. It is still static at runtime.

Full details of the IMP DEF options are documented in the Technical Reference Manual (TRM) for the specific processor.

UNPREDICTABLE and CONSTRAINED UNPREDICTABLE

UNPREDICTABLE and **CONSTRAINED UNPREDICTABLE** are used to describe things that software should not do.

When something is UNPREDICTABLE or **CONSTRAINED UNPREDICTABLE**, the software cannot rely on the behavior of the processor. The processor might also exhibit different behaviors if software carried out the bad action multiple times.

For example, providing a misaligned translation table is **CONSTRAINED UNPREDICTABLE**. This represents bad software. Bad software is software that violates the architectural rule that translation tables should adhere to.

Unlike IMP DEF behaviors, the Technical Reference Manual (TRM) does not usually describe all the UNPREDICTABLE behaviors.

Deprecated

Sometimes, we remove a feature from the architecture. There are several reasons that might happen, for example performance or because the feature is no longer commonly used and is unnecessary. However, there may still be some legacy software that relies upon the feature. Therefore, before removing a feature completely, we first mark it as DEPRECATED. For example, the Arm Architecture Reference Manual states:

The uses of the `IT` instruction, and use of the `CP15DMB`, `CP15DSB` and `CP151SB` barrier instructions, are deprecated for performance reasons.

DEPRECATED is a warning to developers that a feature will be removed in the future, and that they should start removing it from their code.

Often, a control will be added to the architecture at the same time, allowing the feature to be disabled. This control allows developers to test for use of the feature in legacy code.

RES0 and RES1

RES0 means Reserved, should be Zero. RES1 means Reserved, should be One.

RES0 and RES1 are used to describe fields that are unused and have no functional effect on the processor.

A reserved field might be used in some future version of the architecture. In this instance, the **RES0**/RES1 field value of 1 will give the new behavior.

A **RES0** field will not always read as 0, and a RES1 field might not always read as 1. **RES0** and RES1 only tell you that the field is unused.

There are times when **RES0** and RES1 fields must be stateful. Stateful means that the fields read back the last written value.

10. Check your knowledge

The following questions will help you test you knowledge:

If you saw Armv7-R referred to in a document, which version and profile of the architecture is being referred to?

Version 7, R-Profile

In which version of the Arm architecture was 64-bit support added to the A-Profile?

Version 8 (Armv8-A)

For each of the following, would you classify them as architecture or micro-architecture: instruction encodings, cache size, and memory ordering?

- Instruction encodings: Architecture
- Cache size: Micro-architecture
- Memory ordering: Architecture

What is a PE?

A PE is a Processing Element: a machine that implements the Arm architecture.

11. Related information

Here are some resources related to material in this guide:

- Arm architecture reference manuals
- Arm Community - Ask development questions, and find articles and blogs on specific topics from Arm experts.

Other Arm architectures

- Generic Interrupt Controller (GIC)
- Server Base System Architecture (SBSA)
- System Memory Management Unit (SMMU or sometimes IOMMU)
- Trusted Base System Architecture (TBSA)

12. Next steps

This guide introduced the fundamental principles of what the Arm architecture is, how it has evolved, and its profiles and their applications. This knowledge helps provide a foundation on which you can build as you learn more about Arm technologies.

We have discussed some of the common terms and concepts that are key to understanding the Arm architecture, and the different profiles of the Arm architecture. We have described features that are specific to architecture, system architecture and micro-architecture, and how Arm architecture terms and concepts appear in Arm Architecture Reference Manuals and other Arm documentation and resources. We have also learned about the different profiles of the Arm architecture and other Arm architectures.

Further guides in this series introduce aspects of the Arm architecture in detail, and provide examples and commentary.

To keep learning about the Arm architecture, see more in our Learn the Architecture series of guides.