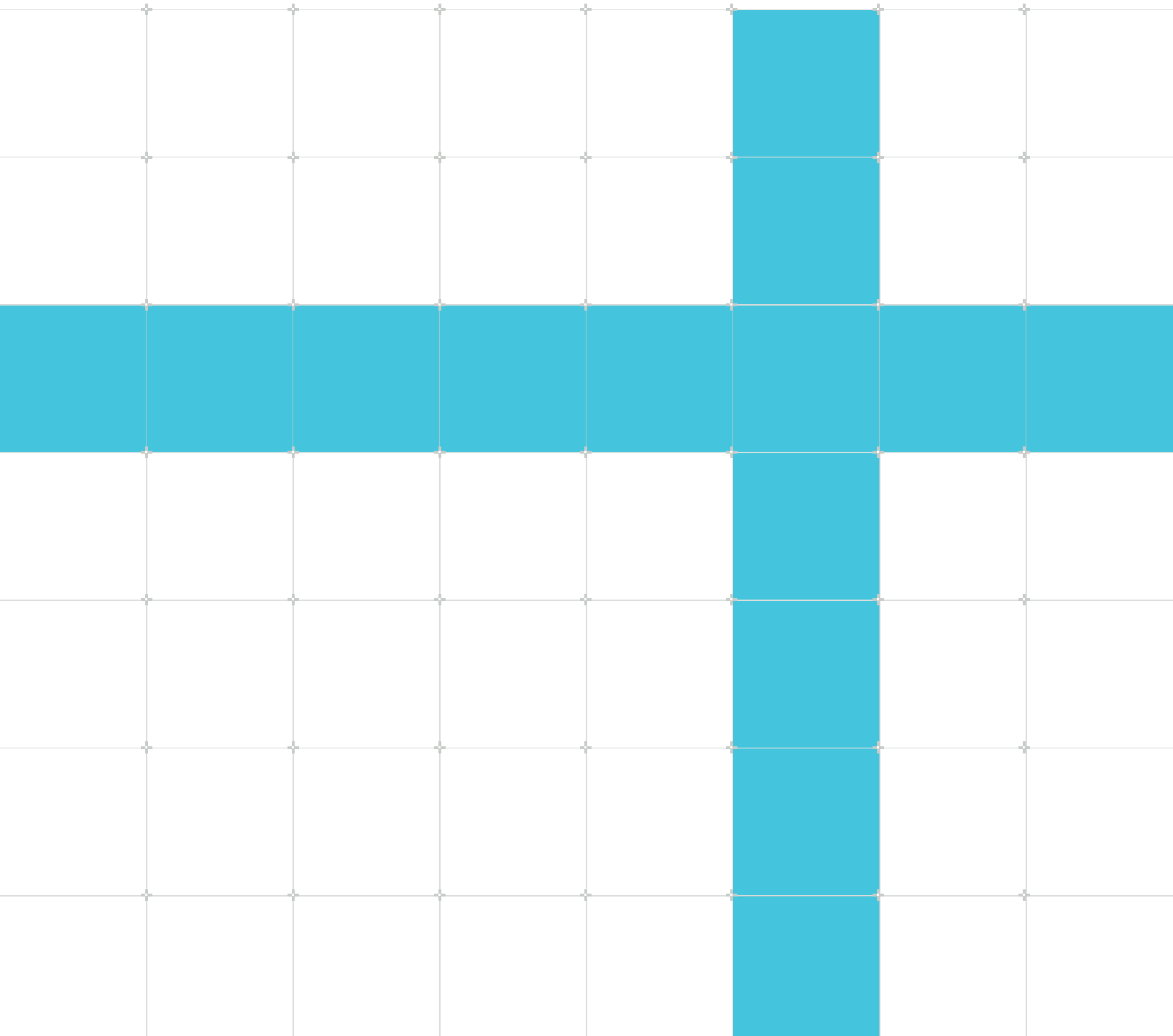
Learn the architecture - Introducing the Arm architecture

Version 2.3

Issue 01
102404_0203_01_en



Learn the architecture - Introducing the Arm architecture

Release information

Document history

Issue	Date	Confidentiality	Change
0203-01	25 July 2025	Non-Confidential	Expand description of system architecture and general editorial changes
0202-01	9 January 2025	Non-Confidential	Clarify meaning of microarchitecture in section “Architecture and microarchitecture”
0201-02	23 September 2024	Non-Confidential	Image updates
0201-01	23 February 2023	Non-Confidential	Minor modifications
0200-02	30 March 2021	Non-Confidential	Updated for v9
0100-01	1 April 2019	Non-Confidential	First release

Proprietary Notice

represent Arm's view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

This document may include technical inaccuracies or typographical errors. THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, any patents, copyrights, trade secrets, trademarks, or other rights.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Reference by Arm to any third party's products or services within this document is not an express or implied approval or endorsement of the use thereof.

This document consists solely of commercial items. You shall be responsible for ensuring that any permitted use, duplication, or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of this document shall prevail.

The validity, construction and performance of this notice shall be governed by English Law.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-1121-V1.0

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. Introducing the Arm architecture guide overview

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

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

The architecture exposes a common instruction set and features 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. No prior knowledge of the Arm architecture is needed. However, we assume that you are familiar with processors, programming and their terminologies.

At the end of this guide you can check your knowledge. This guide describes different profiles of the Arm architecture and whether certain features are architecture or microarchitecture 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.

To address the needs of different markets, the architecture has three different profiles; A, R, and M. Table 2-1: Arm architecture profiles on page 7 describes these profiles.

Table 2-1: Arm architecture profiles

A-Profile (Applications)	R-Profile (Real-Time)	M-Profile (Microcontroller)
High performance and feature-rich	Targeted at systems with real-time or deterministic 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

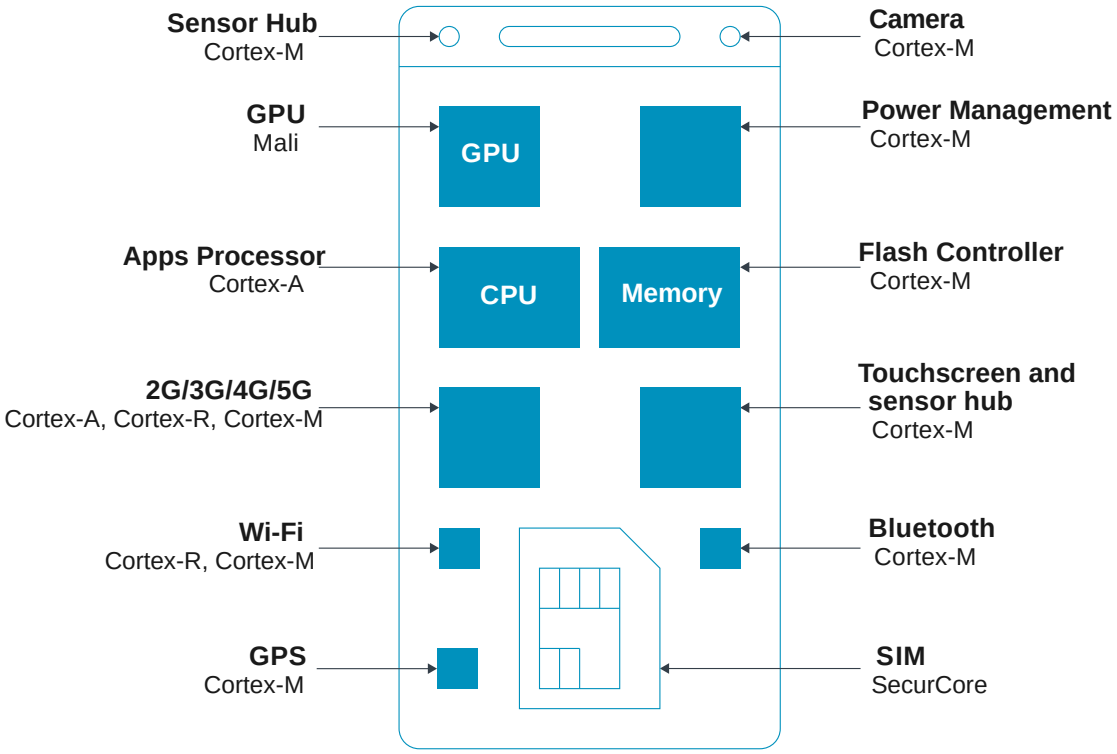
These three profiles allow the Arm architecture to be tailored to the needs of different use cases, while still sharing several base features.

Note

Arm® Cortex®, Arm® Neoverse™ and Arm® SecurCore® are examples of brand names that are used for IP offerings. Our partners offer other processor brands using the Arm architecture.

Figure 2-1: About the Arm Architecture on page 8 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.

This guide describes only 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 behaves, 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 describe in Architecture and microarchitecture.

Table 3-1: Key elements defined by the Arm architecture on page 9 describes what the architecture specifies.

Table 3-1: Key elements defined by the Arm architecture

Feature	Description
Instruction set	The list of available instructions The function of each instruction, including any exceptions it can trigger How an instruction is represented in memory, its encoding
Register set	Set of available registers The size of the registers The function of the registers The reset behaviour of those registers
Exception model	The different levels of privilege and functions available at each level of privilege The types of exceptions and what triggers them 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. Architecture and microarchitecture

Architecture tells you what the operation of a processor is. Microarchitecture tells you how a processor performs the operations required by the architecture.

Compliance to the Arm architecture is essential to ensure operability with the broad ecosystem of Arm software and tools and interoperability of all Arm-compliant processors.

This means that each Arm-compliant processor built and designed in accordance with a microarchitecture must execute each and every instruction of the Arm architecture, and no additional instructions.

Microarchitecture includes:

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

For example, Arm® Cortex®-A53 processor and Arm® Cortex®-A72 processor are both implementations of the Armv8-A architecture. This means that they have the same architecture, but they have very different microarchitectures, as shown in Figure 4-1: Arm Cortex-A53 and Arm Cortex-A72 cores on page 10 and Table 4-1: Example microarchitectural differences between Arm Cortex-A53 and Arm Cortex-A72 on page 11.

Figure 4-1: Arm Cortex-A53 and Arm Cortex-A72 cores

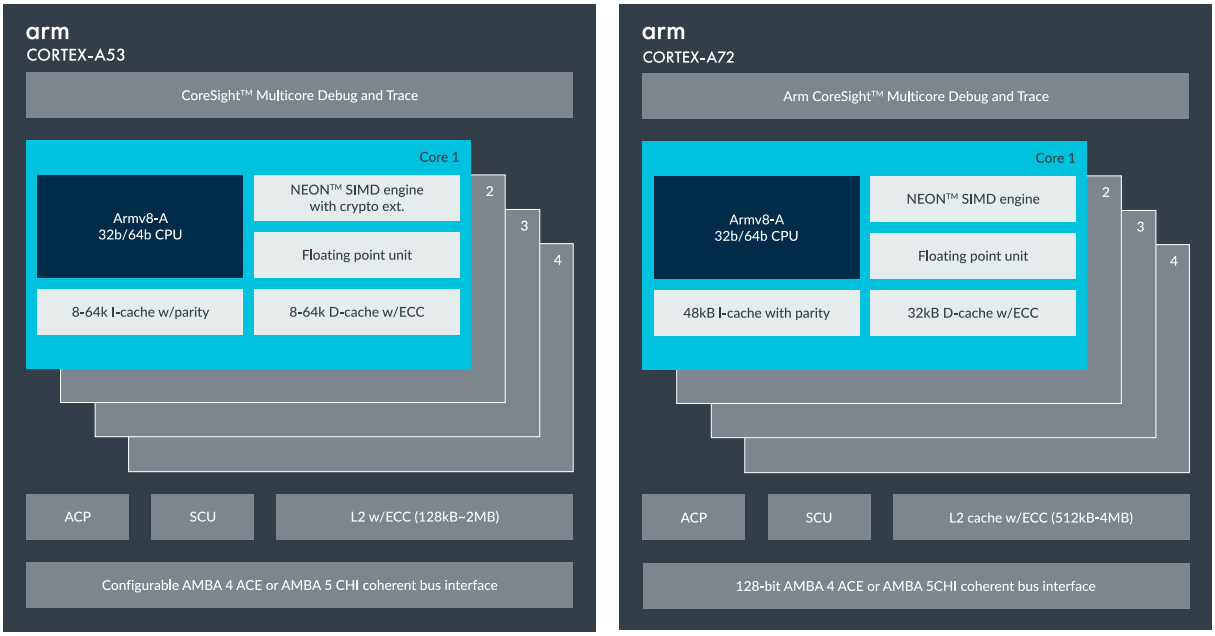


Table 4-1: Example microarchitectural differences between Arm Cortex-A53 and Arm Cortex-A72

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

5. Development of the Arm architecture

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

We refer to the architecture in the following ways:

```
Armv9-A
v9-A
```

This means Version 9 of the architecture, for A-profile.

The two main versions of the Arm architecture for A-profile are Armv8 and Armv9.

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 also introduced changes to existing features, for example restricting support to AArch32 to EL0.

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)
- Scalable Matrix Extensions (SME, SME2)
- Realm Management Extension (RME)
- Guarded Control Stack (GCS)
- Branch Record Buffer Extension (BRBE)
- Embedded Trace Extension (ETE)
- Trace Buffer Extension (TRBE)

Armv9-A also introduced changes to existing features in Armv8-A, for example restricting support to AArch32 to EL0.

Annual updates

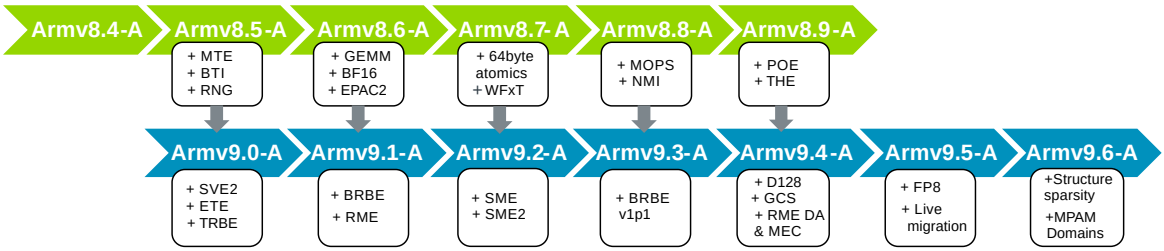
Arm releases an annual update to the architecture adding new features and extending existing ones. These annual updates are identified using a minor version number, for example:

```
Armv9.3-A, which was announced in 2022.
```

When Armv9-A was introduced, Armv9.0-A aligned to Armv8.5-A. This means that Armv9.0-A includes all the features in Armv8.5-A, plus Armv9-only features such as SVE2.

Armv8-A and Armv9-A were updated in parallel. Any feature added in Armv8 was inherited by the corresponding release of Armv9, but some features were only added in Armv9-A. Figure 5-1: Releases on page 13 shows the alignment between Armv8 and Armv9, and gives examples of the features added in different releases.

Figure 5-1: Releases



For convenience, we sometimes refer to updates to Armv8 and Armv9 collectively by the year they were announced. For example, Arm8.9-A and Armv9.4-A are collectively known as the 2022 extensions.

You can find more information on the features added in each annual update in the *Arm Architecture Reference Manual*, and via the blogs posted on the Arm website:

- Arm A-Profile Architecture Developments 2024
- Arm A-Profile Architecture Developments 2023
- Arm A-Profile Architecture Developments 2022
- Arm A-Profile Architecture Developments 2021
- Arm A-Profile Architecture Developments 2020
- Developments in the Arm A-Profile Architecture: Armv8.6-A

Identifying features

In Arm documentation, features are referred to as FEATs using the particular notation:

```
FEAT_<name>
```

Often, one high-level feature is represented by a collection of FEATs. For example, the Realm Management Extension (RME) consists of:

- FEAT_RME
- FEAT_RME_GDI
- FEAT_RME_GPC2
- FEAT_RME_GPC3

For each FEAT, the architecture defines:

- Which version of the architecture introduced the feature
- From which version of the architecture is it permitted to be built
- In some cases, from which version of the architecture becomes mandatory
- Any constraints on the feature, for example FEAT_RME_GPC2 requires FEAT_RME

Typically, a feature can be built from the version before it was introduced. For example, the FEAT_MOPS:

- Introduced in the Armv8.8-A as part of the 2022 extensions
- It is permitted to be built from Armv8.7-A or Armv9.2-A
- It is mandatory from Armv8.8-A and Armv9.3-A

You can explore the full feature list on Arm's website:

- HTML
- JSON

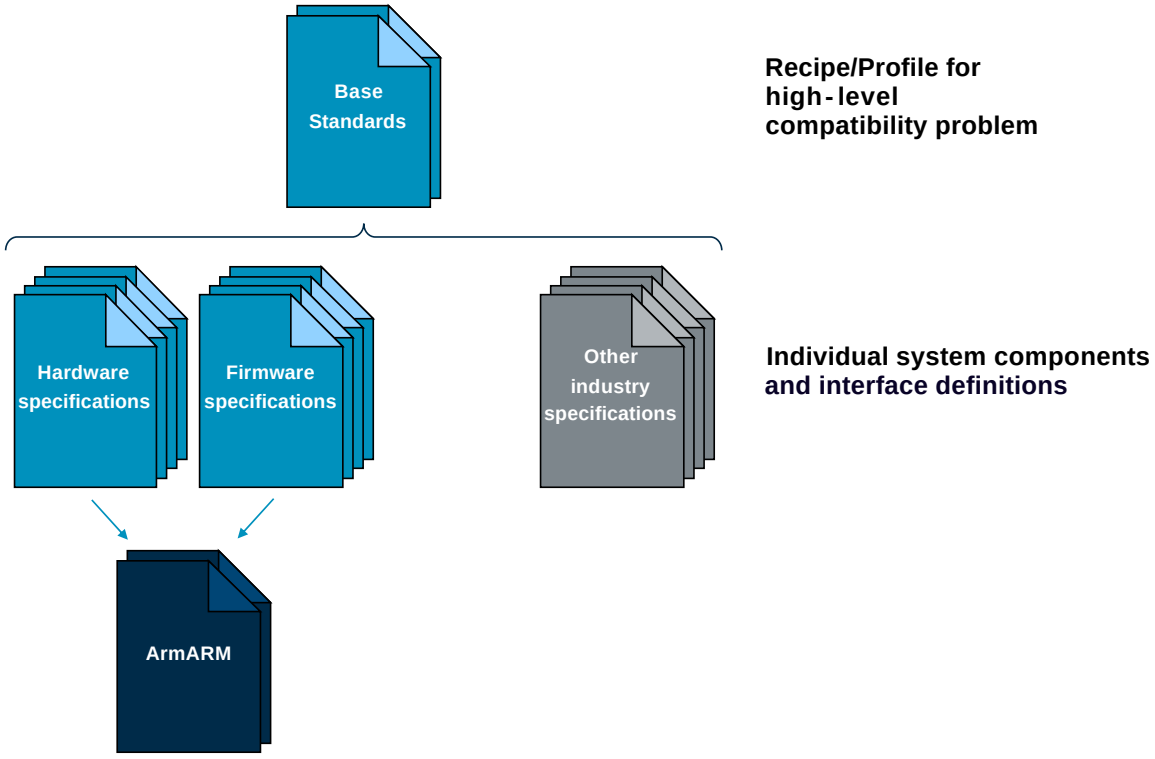
6. System architectures

An *Arm Architecture Reference Manual*, sometimes referred to as the Arm ARM, is the best-known Arm specification. It offers designers a significant level of flexibility to develop processors specifically optimized for a wide range of applications, ranging from tiny embedded systems to supercomputers. These systems require more than just a processor core. For example, they also need memory, input/output devices, power control, monitoring tools, debugging features, and the hardware to connect everything. In addition, system software like an operating system is needed to manage all these components.

Arm, and its partners, have created open and royalty-free system architectures to encourage hardware and software reuse. These architectures define software and hardware interfaces for various components, addressing common issues and fostering innovation. They complement other industry standards by accommodating Arm's unique characteristics,

Figure 6-1: System architecture groups on page 15 shows how we can group the system architectures.

Figure 6-1: System architecture groups



Arm ARM is a set of specifications defining standardized system components and their interfaces. Hardware specifications include additional elements of the System-on-chip (SoC), describing

interfaces to adjacent hardware components and their management by software, for example using registers. They comprise:

- System components supporting the processor core, in particular the Generic Interrupt Controller (GIC), System Memory Management Unit (SMMU), and CoreSight debug and trace.
- Power management: a specialized subsystem featuring an embedded System Control Processor (SCP) for controlling and monitoring power across the SoC. This interfaces to the main system software using the System Control and Management Interface (SCMI).
- The AMBA family of interfaces, including efficient access to memory and memory-mapped devices such as peripherals. Widely adopted, feature rich and architecture neutral, AMBA has a comprehensive and thriving ecosystem of 3rd party partners offering compatible products and solutions.

Firmware is low-level software, typically provided with the SoC or board, that supports higher-level system software like an Operating System or Hypervisor. Often, the firmware needs to align with the hardware capabilities being implemented. Firmware interface specifications targeting A-profile systems include:

- Enabling processor core transitions through privileged exception levels and security states. Relevant specifications are the SMC Calling Convention (SMCCC), Firmware Framework for A-profile (FF-A) and the Arm Confidential Computing Architecture (CCA).
- A set of standard secure and monitor interfaces, such as the Power State Coordination Interface (PSCI) for managing system and CPU power transitions
- Arm-specific supplements for the industry-defined Advanced Configuration and Power Interface (ACPI)

The hardware and firmware specifications described above offer system designers a toolkit of individual building blocks to use as needed for their use case. Some common high-level compatibility problems require a set of capabilities to be standardized, so some of the system architecture specifications define a base set of hardware and firmware functionality. These base standards define profiles of Arm-specific and industry specifications, allowing system designers to add their own value on top. For example, the Base Boot Requirements (BBR), Base System Architecture (BSA), and their market-specific supplements enable support for standard off-the-shelf operating systems.

For a more comprehensive overview of the various system architectures, see Learn the architecture - Arm System Architectures.

7. Understanding Arm documentation

Arm provides a lot of documentation to developers. This section describes where to find documentation and other information for developing on Arm.

7.1 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.

7.2 Which document describes what?

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

- Arm Architecture Reference Manuals describes the PE architecture. An *Arm Architecture Reference Manual* is relevant to any implementation of that architecture.
    - Some components, such as the Generic Interrupt Controller (GIC) and System MMU (SMMU) have their own architecture specifications.
- Each Arm Cortex or Arm Neoverse processor has a Technical Reference Manual (TRM). The TRM describes the features specific to that processor. In general, the TRMs do not repeat information given in the Arm Architecture Reference Manuals.
    - Some processors also provide a Software Optimization Guide with information on instruction timing and scheduling.
- Each Arm Cortex or Arm Neoverse processor also typically 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.
- If applicable, the System Architecture specifications give system or software rules and definitions.

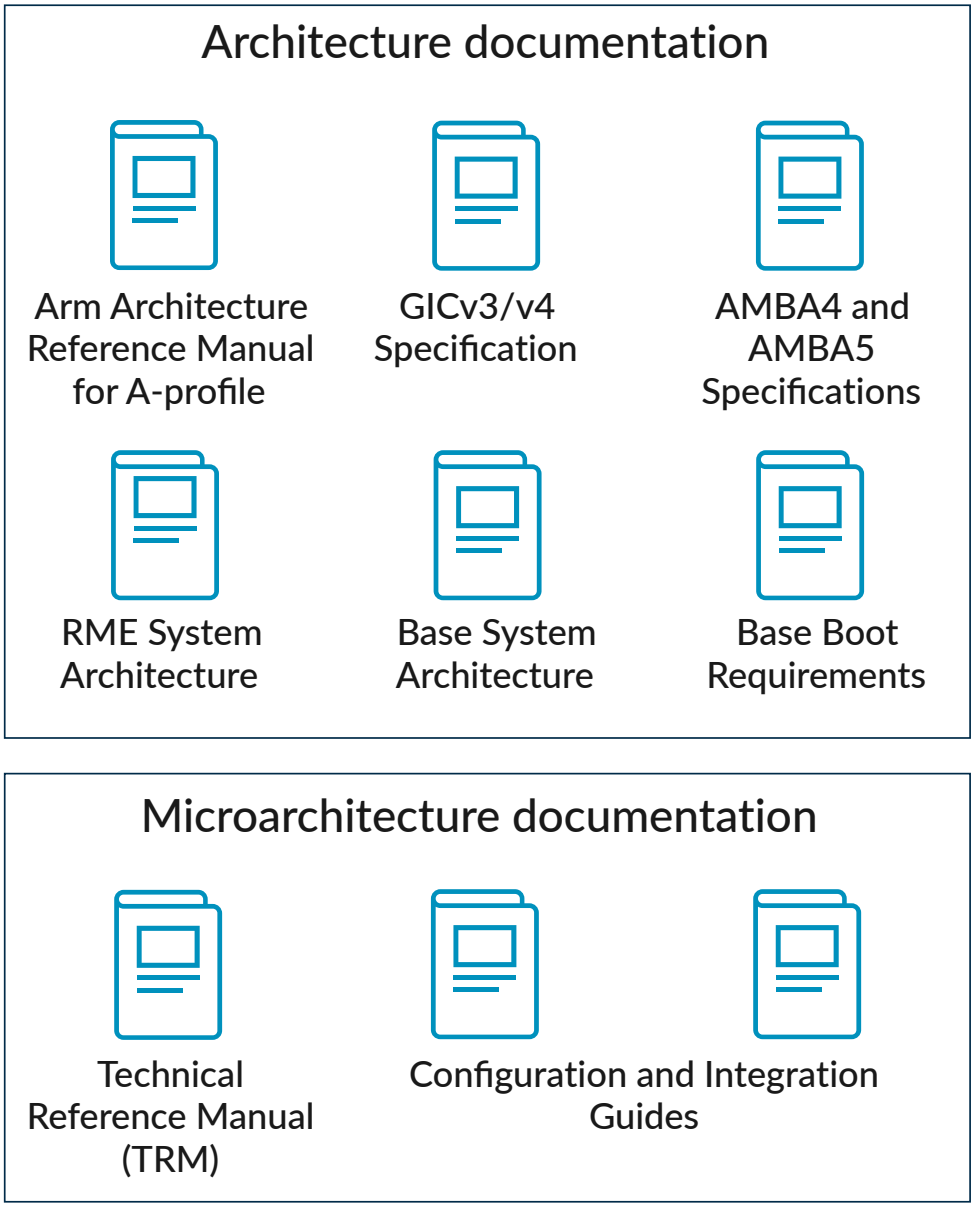
Note

The CIMs are only available to IP licensees. The TRMs are available to download from Arm Developer.

7.3 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. Figure 7-1: Architecture documentation on page 19 shows the documents you might need to use with a Neoverse V3 processor:

Figure 7-1: Architecture documentation



The *Arm® Neoverse™ V3 Core Technical Reference Manual* states that the processor implements Armv9.2-A along with some optional features such as Realm Management Extension. You will need to refer to the *Arm Architecture Reference Manual for A-profile architecture* for details, including how

software can detect if features are enabled. The TRM also provides other microarchitecture details such as the number and size of caches.

If you are designing a SoC using Neoverse V3 processor and associated Arm IP, you will need documents such as:

- The TRMs describe the available hardware interfaces, such as GICv4.1 CPU Interface and AMBA CHI Issue E. You will need to connect IP blocks which comply with the corresponding architecture specifications.
- Some optional processor features, such as the Realm Management Extension (RME), depend on capabilities in components outside the core. If you want to use RME, then you must ensure that your system meets the requirements in RME System Architecture specification. Otherwise you will need to see the *Arm® Neoverse™ V3 Core Configuration and Integration Manual* to ensure that RME is disabled.
- If you want to support standard off-the-shelf operating system images then, depending upon your requirements, you might need to ensure that your overall hardware design complies with Base System Architecture (BSA) and possibly one of its market-specific supplements. You will also need to ensure that the firmware accompanying the SoC complies with the Base Boot Requirements (BBR).

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

7.4 What information will I find in each document?

Table 7-1: Example contents of different Arm document types on page 20 shows the kind of information found in the different types of documents:

Table 7-1: Example contents of different Arm document types

Feature	Arm Architecture Reference Manual	GIC specifications	AMBA specifications	TRM	CIM	SoC datasheet
Instruction set	x	-	-	-	-	-
Instruction cycle timings	-	-	-	x or separate document	-	-
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 and TLBs	-	-	-	x	x some might be synthesis choice	-
Power management	-	-	-	x	-	-
Bus ports	-	-	-	x	x	-

Feature	Arm Architecture Reference Manual	GIC specifications	AMBA specifications	TRM	CIM	SoC datasheet
Bus transactions description	-	-	x	-	-	-
Bus transactions generated by processor	-	-	-	x	-	-
Memory map	-	-	-	-	-	x
Peripherals	-	-	-	-	-	x
Pin-out of SoC	-	-	-	-	-	x

7.5 Differences between reference manuals and user guides

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, Arm Architecture Reference Manuals 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 different types of information. The technical detail of what the architecture requires is found in reference manuals. More general guidance is provided in other documents, for example, this guide. Some general guidance documents introduce concepts, and others provide instructions for you to follow.

8. Common architecture terms

The architecture uses several terms, sometimes written in small or capital letters, which have very specific meanings. The Arm Architecture Reference Manuals define each of these terms. This section describes the most common terms and what they mean to programmers.

8.1 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, *Arm Architecture Reference Manual for A-profile architecture* 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 microarchitectures. For example, the following microarchitectures are possible in the Arm Cortex-A processors:

- Arm Cortex-A8 is a single core, single-thread processor. The entire processor is a PE.
- Arm Cortex-A53 is a multi-core processor, each core is a single thread. Each core is a PE.
- Arm 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.

8.2 IMPLEMENTATION DEFINED

A feature which is **IMPLEMENTATION DEFINED** (IMP DEF) is defined by the specific microarchitecture. 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. There are registers to enable software to determine if the instructions are present or not.

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

For Cortex and Neoverse processors, some IMP DEF choices are fixed, and some are synthesis options. For example, on the Arm Cortex-A57 processor 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.

The TRM for the specific processor provides full details of the IMP DEF options.

8.3 UNPREDICTABLE and CONSTRAINED UNPREDICTABLE

UNPREDICTABLE and **CONSTRAINED UNPREDICTABLE** describe what software should not do.

When something is **UNPREDICTABLE** or **CONSTRAINED UNPREDICTABLE**, 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 must adhere to.

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

8.4 Deprecated

Sometimes, Arm removes a feature from the architecture. This might happen because the feature is no longer commonly used and is unnecessary. However, there might still be some legacy software that relies upon the feature. Therefore, before removing a feature completely, we first mark it as DEPRECATED.

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

Often, a control is 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.

8.5 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 field might be conditionally reserved, for example it is **RES0** only when a feature is disabled by software or not implemented. Similarly, a field that is currently RESx might have a defined use in a future version of the architecture.

A **RES0** field might 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.

8.6 Future Architecture Technologies

Arm sometimes releases advanced information on new features that will be included in unannounced future releases of the architecture. These releases are intended to enable early software and tools development. Such releases are labeled as Future Architecture Technologies (FAT).

9. Check your knowledge

The following questions help you test your knowledge:

If you see 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 microarchitecture: instruction encodings, cache size, and memory ordering?

- Instruction encodings: Architecture
- Cache size: Microarchitecture
- Memory ordering: Architecture

What is a PE?

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

10. Related information

Here are some resources related to material in this guide:

- Learn the Architecture Guides
- Arm Architecture Reference Manual for A-profile architecture
- All Arm Architecture Reference Manuals
- Learn the architecture - Arm System Architectures
- Arm Community: Ask development questions, and find articles and blogs on specific topics from Arm experts.

11. Next steps

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

This guide describes some of the common terms and concepts that are key to understanding the Arm architecture, and the different profiles of the Arm architecture. We describe:

- Features that are specific to architecture, system architecture and microarchitecture
- How Arm architecture terms and concepts appear in Arm Architecture Reference Manuals and other Arm documentation and resources

You have also learned about the different profiles of the Arm architecture and other Arm architectures.

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

To continue learning about the Arm architecture, see our Learn the Architecture series of guides.