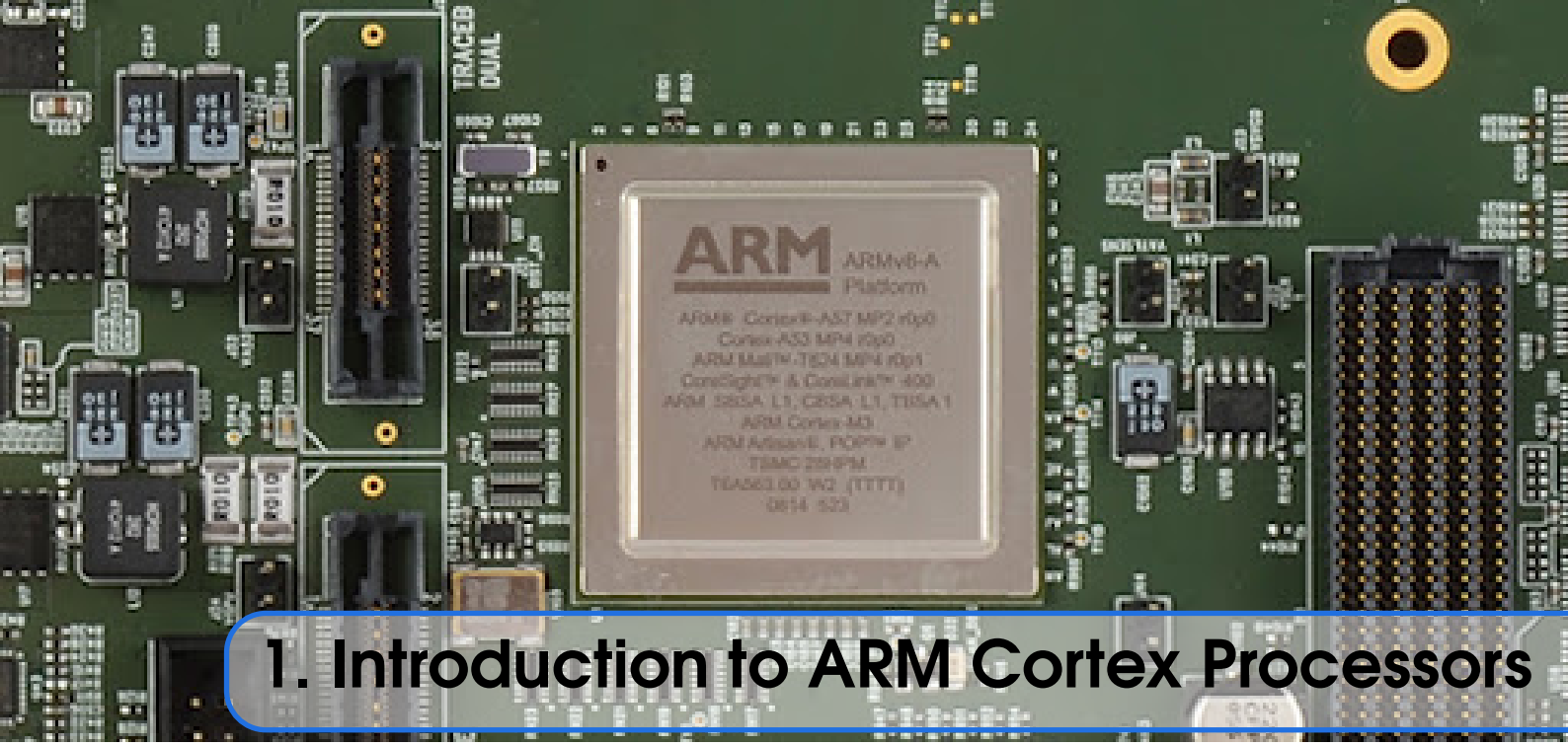


1. Introduction to ARM Cortex Processors

1.1 Introduction to Embedded Systems

In our daily lives, we encounter types of computers that are familiar to us, such as desktops, laptops, and mobile phones. Simultaneously, there are other types of computing systems hidden within other mechanical or electrical system, such as washing machines, refrigerators, cameras, vehicles/airplanes, and printers. These are known as embedded systems.

So, an embedded system is a specialized computer system (other than desktop and laptop) that is designed to perform dedicated functions or tasks within a larger mechanical or electrical system.

These specialized computers (embedded systems) are designed for special-purpose or single-function tasks, executing a single program, possibly with inputs from the environment. They are characterized by being low-cost, low-power, small in size, limited memory and relatively fast to reacting to events in real-time.

They are commonly found in a wide range of applications, including consumer electronics, automotive systems, industrial machines, medical devices, and more.

More precisely, an embedded system can be defined as a microcontroller-based system designed to control a specific function or range of functions. It is not intended to be programmed or modified by the end user.

Figure 1.1 illustrates how the embedded system interacts with its environment through different analog/digital inputs/outputs and how it works through the processing of the given input by specified software.

Figure 1.2 illustrates the different subsystems inside the embedded system processor as boxes, where we can see:

- Analog to digital (ADC) interfaces
- Digital to analog (DAC) interfaces
- Pulse-width-modulation (PWM) interfaces

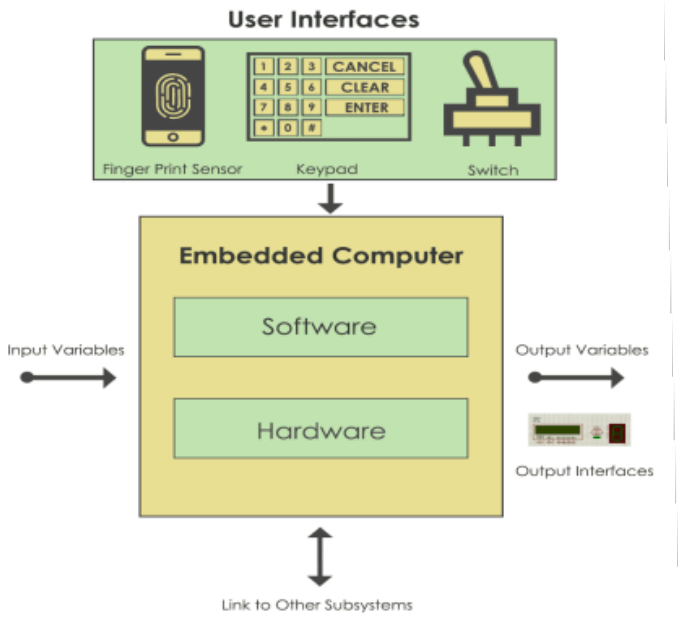


Figure 1.1: Block diagram of an embedded system.

- Timers and counters
- In addition to the processor, memory, digital I/O, etc.

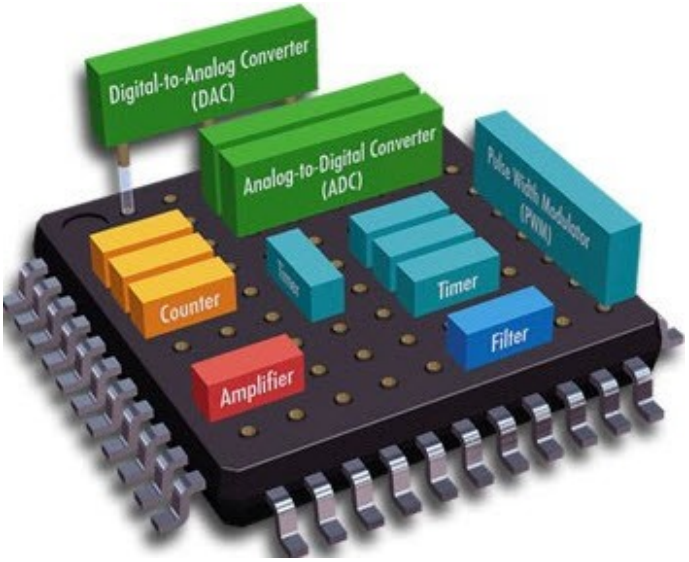


Figure 1.2: Block diagram of an embedded system.

1.1.1 Example embedded system: 1. bike computer

A bike computer is an example of a small embedded system (see **??**). The bike computer senses wheel rotation, mostly magnetically, and outputs calculated values such as speed, distance, and time to the user. The specifications of this system are as follows:

- Functions

  - Speed and distance measurement
- Constraints
  - Size
  - Cost
  - Power and energy
  - Weight
- Inputs
  - Wheel rotation indicator
  - Mode key
- Output
  - Liquid Crystal Display
- Use Low Performance Microcontroller
  - 8-bit, 10 MIPS

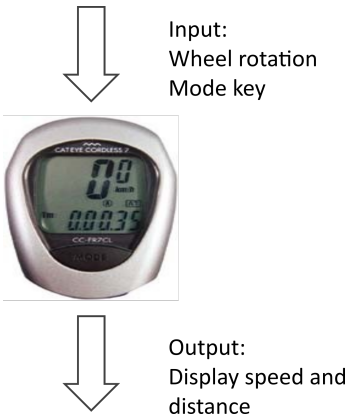


Figure 1.3: Example of an Embedded System.

1.1.2 Example embedded system: 2. Gasoline automobile engine control unit

In this example, the functional requirements and the constraints are much greater than for the bike computer. Requirements such as spark timing and fuel injection need real time calculations. This means that every calculation must be finished at a defined point in time. Additional constraints are a high reliability in a harsh environment, for relatively low cost. Furthermore, limited installation space requires a small device. Many sensors and actuators have to be operated and networked to the rest of the car by the microcontroller. Consequently, a powerful, reliable and high-performance microcontroller is needed.

1.2 Basic Operation of a Computing System

The principal unit in each computing system is the central processing unit (CPU) that carries out all computations. It fetches instructions from the program memory and executes them. During execution, it may require transferring data to/from data memory. In addition to that, the computing system needs input/output blocks to provide an interface with the outside world, allowing users to interact with it and observe the output results (Figure 1.4).

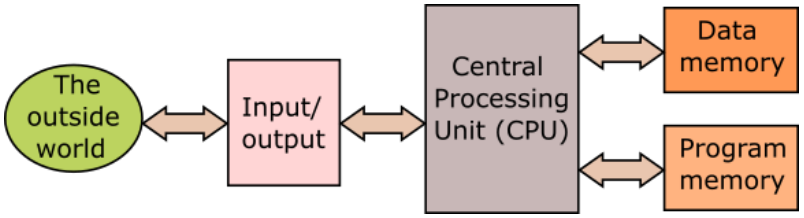


Figure 1.4: Principal blocks in computing system.

1.2.1 Instruction Set Architecture

The instruction set executed by the CPU can have two main types of architectures:

Complex Instruction Set Computing (CISC):

In this architecture, each instruction can take multiple clock cycles to execute and perform a variety of low-level operations. Typically used in desktops and laptops (x86 architecture).

Reduced Instruction Set Computing (RISC):

This architecture focuses on a smaller set of simple and highly optimized instructions. Each instruction typically performs only one low-level operation, and they are designed to be executed in a single clock cycle. Typically used in microcontrollers, which are used to build embedded systems (ARM and MIPS).

1.2.2 Memory types

The central memory consists of two types:

Random Access Memory (RAM):

This type is volatile memory and is used for data storage in microcontrollers.

Read-Only Memory (ROM):

This type is non-volatile memory used for program storage in microcontrollers. It has been replaced by flash memory in newer versions of microcontrollers.

1.2.3 CPU Classification Architecture

Broadly, there are two types of CPU architecture that are fundamental in computing system design (see Figure 1.5):

Von Neumann Architecture:

In this architecture, both program instructions and data are stored in the same memory. This model is followed in conventional computing systems.

Harvard Architecture:

In this architecture, data and instructions are stored in separate memories. Typically used for building embedded systems, followed in microcontrollers, and digital signal processors (DSPs).

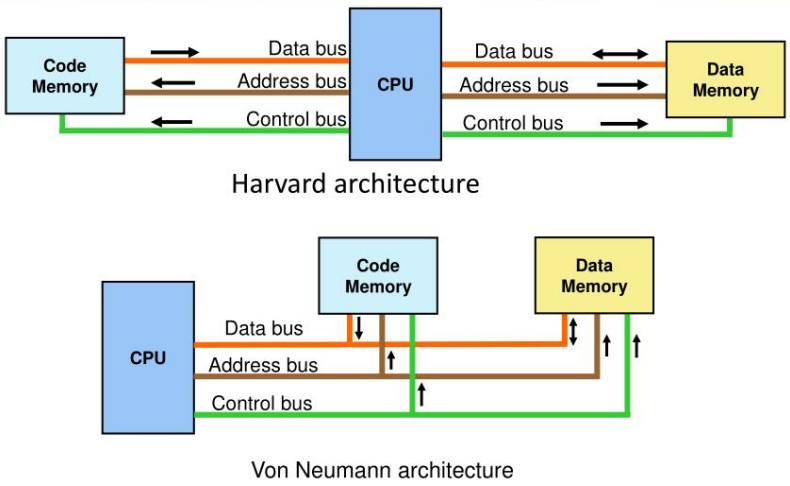


Figure 1.5: Von Neumann and Harvard Architectures.

1.2.4 What is a microprocessor ?

The microprocessor is essentially the entire CPU fabricated on a single chip. It consists of (see Figure 1.6) :

- Set of registers to store temporary data.
- An arithmetic logic unit (ALU), where all arithmetic and logical computations are carried out.
- Mechanisms to interface with external devices (memory and I/O) through buses (address, data, and control).
- A control unit that synchronizes the operation.

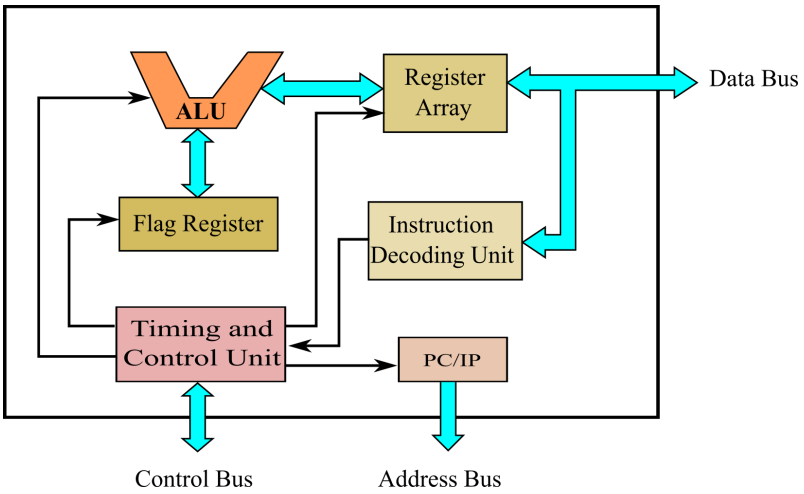


Figure 1.6: Block Diagram of Basic Functional Blocks of a Microprocessor.

1.2.5 What is a microcomputer ?

A microcomputer is a computing system built using a microprocessor. However, since a microprocessor does not contain memory and I/O, we have to interface these components to build a microcomputer. For this reason, it becomes complex and expensive for very

small and low-cost embedded systems (see Figure 1.7).

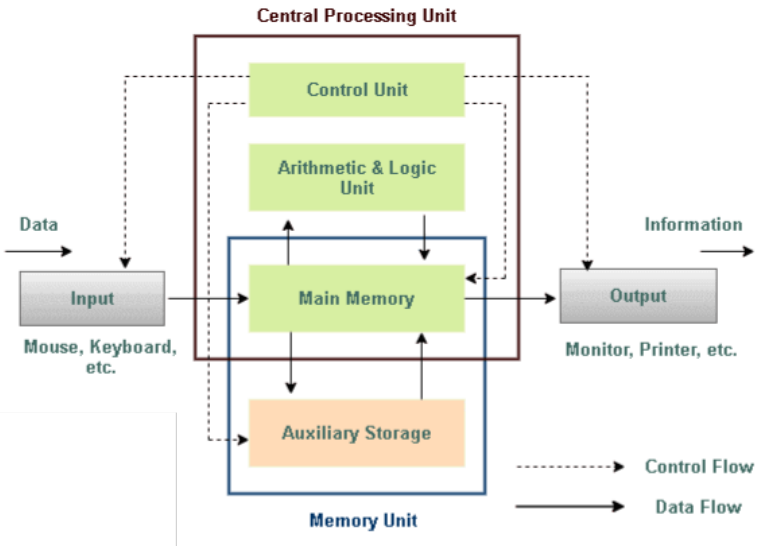


Figure 1.7: Schematic diagram of microcomputer.

1.2.6 What is a microcontroller ?

A microcontroller is essentially a computer on a single chip, it has extra hardware to allow it to interface with the outside world more easily. It is very inexpensive, small, low power, and convenient for use in embedded system design. It operates on data fed through its serial or parallel input ports, controlled by the software stored in on-chip memory. It often has analog input pins, timers, and other utility circuitry built-in (see Figure 1.8).

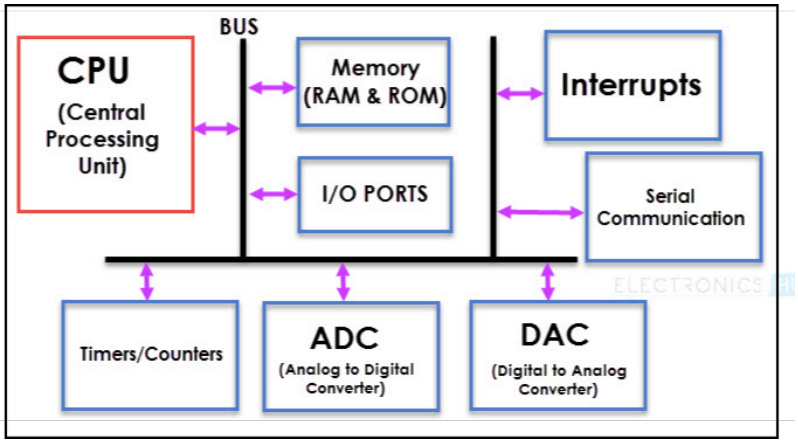


Figure 1.8: Schematic diagram of microcontroller.

A more detailed schematic diagram of a microcontroller is represented in Figure 1.9.

1.2.7 What differentiates microcontrollers from PCs?

For the PC, the executed program is first loaded from disk/SSD into an allocated section of memory. It is usually loaded part by part to conserve memory space. The PC

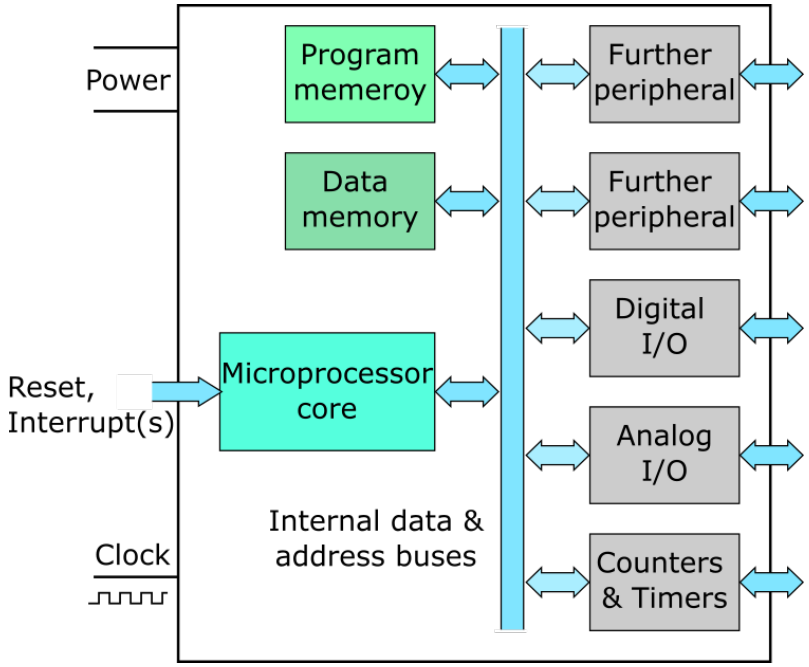


Figure 1.9: Detailed schematic diagram of microcomputer.

also has a complicated operating system that handles all low-level operations (including low-level driver codes for interfacing with various devices).

For the microcontroller, there is no disk to read from. The executed program is stored in on-chip ROM, and the ROM size limits the maximum size of the application. The microcontroller has no operating system, and the program in ROM is the only program that is running (must include low-level routines).

1.2.8 Where are microcontrollers used for?

Microcontrollers are typically used in applications where processing power is not critical. In the modern household, they can be found in 10 to 50 such devices embedded in various devices and equipment. These applications can be divided as:

- One-third are in the office automation segment.
- One-third are in consumer electronics goods.
- One-third are used in automotive and communication applications.

1.2.9 Evolution of microcontrollers

Microcontrollers evolved from a microprocessor-based board-level design to a single chip in the mid-1970s. As the process of miniaturization continued, all the components needed for a controller were integrated into a single chip.

In the mid-1980s, microcontrollers became embedded into larger ASICs (Application-Specific Integrated Circuits). They are fabricated as a module inside a larger chip.

1.2.10 Advantages of using microcontrollers

The main advantages of microcontrollers can be summarized as follows:

- **Fast and effective:** Because the architecture correlates closely with the problem being solved (control systems).
- **Low cost / Low power:** Due to a high level of system integration within one component, and only a handful of components needed to create a working system.
- **Compatibility:** Their opcodes and binaries are the same for all 80x51 / ARM / PIC variants.

1.3 Architecture of ARM Microcontrollers

1.3.1 History of ARM Series of Microcontrollers

The architectural ideas of ARM Microcontrollers were developed in 1983 by Acorn Computers to replace the 8-bit 6502 microprocessor in BBC computers, where the first commercial RISC implementation took place. In 1990, the Advanced RISC Machine (ARM) company was founded. Initially owned by Acorn, Apple, and VLSI.

1.3.2 Importance of ARM Processors

The ARM Microcontrollers are one of the most widely used processor cores. They are mainly used in battery-operated devices due to their low power consumption and reasonably good performance. Until 2010, 90% of 32-bit embedded RISC processors were ARM processor. Here are some examples of their applications:

- ✓ **ARM7:** Used in iPod.
- ✓ **ARM9:** Used by BenQ and Sony Ericsson to fabricate TV sets and other equipment.
- ✓ **ARM11:** Used in Apple iPhone, Nokia N93, N100.

1.3.3 Specificities of ARM Processors

The main specificities of ARM Processors can be summarized in the following points:

- ▶ They have a simple RISC-based architecture with a powerful design.
- ▶ The whole family of ARM processors shares similar design principles and a common instruction set.
- ▶ They are small processors with lower power consumption and reasonable performance (MIPS / watt), making them very suitable for embedded system and portable devices applications.
- ▶ They have high code density which fits within limited memory and physical size restrictions.
- ▶ They can interface with slow and low-cost memory systems.
- ▶ They have a reduced die size for the processor to accommodate more peripherals.

1.3.4 ARM Architectures and Families

ARM processors can be classified based on architecture, application domain, performance characteristics, and intended use cases. Figure 1.10 illustrates the different architecture versions of ARM processors. Under each version, different processor families can be found, and each family has many processor cores.

The first architecture family was introduced in 1980 under ARMv1 (Version 1), continuing through the latest architecture announced in 2021 under ARMv9. These different families can be divided into the classic ARM processors (ARM7, ARM9, ARM10 and

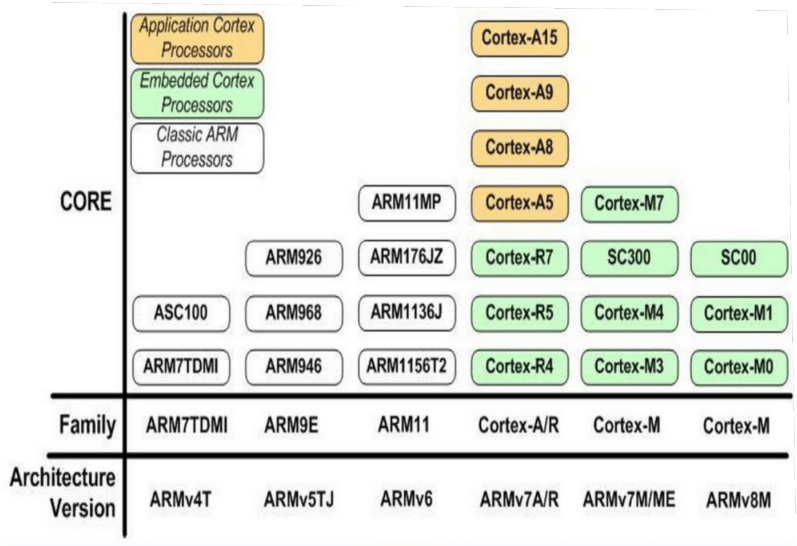


Figure 1.10: ARM Processors Architectures and Families.

ARM11) and the Cortex processors, which first developed on ARMv7. There are 3 sub-families within the ARM cortex family (profiles), each designed for specific applications, as detailed in Table 1.1.

Table 1.1: ARM Cortex Processor Calcification.

A-profile (Applications)	R-profile (Real-time)	M-profile (Microcontroller)
- High performance	- Targeted at systems with real-time requirements.	- Smallest/lowest power. Small, highly powerefficient 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 and SoC applications with independent power supply

This course focus on the **Cortex-M series (specifically M4 processor subfamily)**, as they target smaller-scale applications such as microcontrollers and mixed-signal design. Criteria such as low cost, low power, energy efficiency, and low interrupt latency are important in these applications. Simultaneously, the processor design must be user-friendly and able to provide deterministic behavior, as required in many real-time control systems. Figure 1.11 shows how this processor family might be used in a smartphone.

1.3.5 Arm-based system-on-a-chip (SoC)

An Arm-based SoC is designed by selecting elements from Arm Intellectual P roperty (IP) libraries and/or other third party IP vendors. Different components such as core, RAM, ROM, peripherals, etc. are selected from licensable IP. These IP components are combined with company-owned or additional third-party IP to create a SoC as shown in Figure 1.12 .

1.3.6 Cortex-M4 processor features

ARM Cortex-M4 Processors are characterized by:

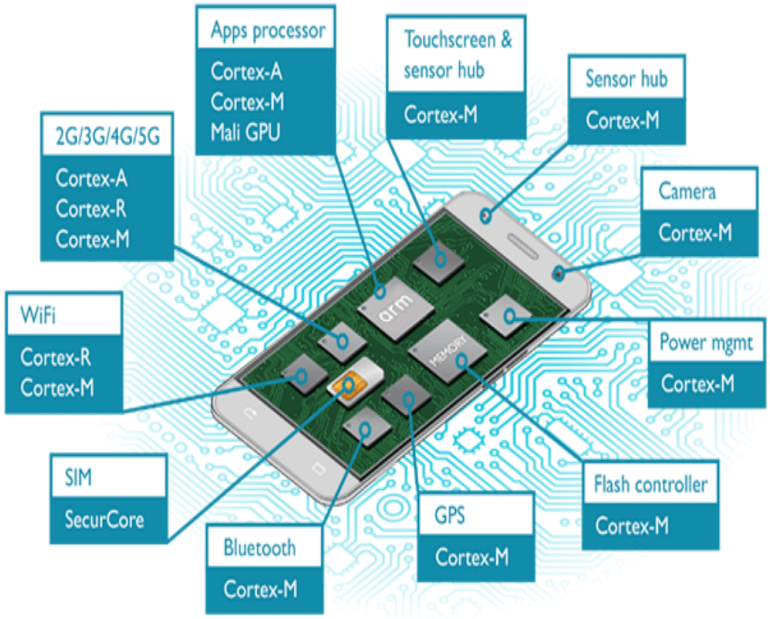


Figure 1.11: Example of ARM Cortex family applications in handheld devices.

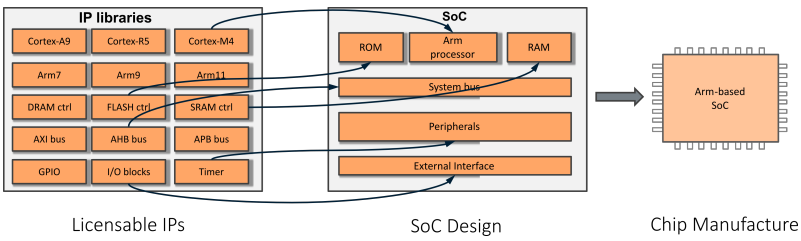


Figure 1.12: ARM-based SoC designing.

- It is 32-bit RISC (Reduced Instruction Set Computing) processors with :
  - 32-bit registers
  - 32-bit internal data path
  - 32-bit bus interface
- It is Thumb-2 technology allows 16-bit instructions and 32-bit instructions to work together without any state switching overhead.
- It has low power consumption, leading to longer battery life – especially critical in mobile products.
- It can handle 8-bit and 16-bit data efficiently and supports hardware divide instructions, as well as Multiply-and-Accumulate (MAC).
- It is three-stage pipeline design (instruction fetch, decode, and execution).
- It is Harvard bus architecture, which allows simultaneous instruction fetches and data accesses.
- It uses 32-bit addressing with a 4GB address space, utilized by the program code, data, and peripherals.
- It follows a load-store architecture, where data is loaded from memory, processed, and then written back to memory.
- It supports from 8 to 256 interrupt priority levels and includes non-Maskable Interrupt (NMI) in addition to 1 to 240 physical interrupts.
- It supports sleep modes with up to 240 wake-up interrupts.
- It can implement a complete hardware debug solution with up to 8 breakpoints and 4 watchpoints.

1.3.7 Properties of the RISC ARM Architecture

The key design features of the RISC architecture used by ARM processors can be summarized as follows:

- **Instructions:** Reduced set / Single cycle / Fixed length.
- **Pipeline:** Decode in one stage / No need for microcode.
- **Registers:** Large number of general-purpose registers (GPRs).
- **Load/Store Architecture:** Data processing instructions work on registers only; load/store instructions to transfer data from/to memory.

1.3.8 Distinctive ARM Features

Although ARM processors are based on RISC architecture, advanced and useful features that differ from pure RISC have been introduced in ARM architecture, including:

- ✓ Variable cycle execution for certain instructions (multiple-register load/store for higher code density).
- ✓ In-line barrel shifter results in more complex instructions (improves performance and code density).
- ✓ Thumb 16-bit instruction set (results in an improvement in code density by about 30
- ✓ Conditional execution (reduces branching and improves performance).
- ✓ Enhanced instructions (some DSP instructions are present).

1.3.9 Cortex-M4 block diagram

Figure 1.13 shows the block diagram of the Cortex-M4 processor. It follows a Harvard architecture as it incorporates separate data and instruction buses. The Cortex-M4 pro-

cessor provide generic bus interfaces based on AMBA (Advanced Microcontroller Bus Architecture). The AMBA specification supports several bus protocols, the AHB(AMBA High-performance Bus) Lite protocol is used for the main bus interfaces, and the APB (Advanced Peripheral Bus) protocol is used for the Private Peripheral Bus (PPB). Data accesses and instruction fetches are carried out in parallel two bus interfaces (D-CODE and I-CODE) based on the AHB Lite bus protocol. The SRAM and peripherals are connected to the system bus. Additionally, there are many sophisticated debugging features that utilize the PPB and Debug Access Port (DAP) interface. Common functional blocks found in the Cortex-M4 architecture include:

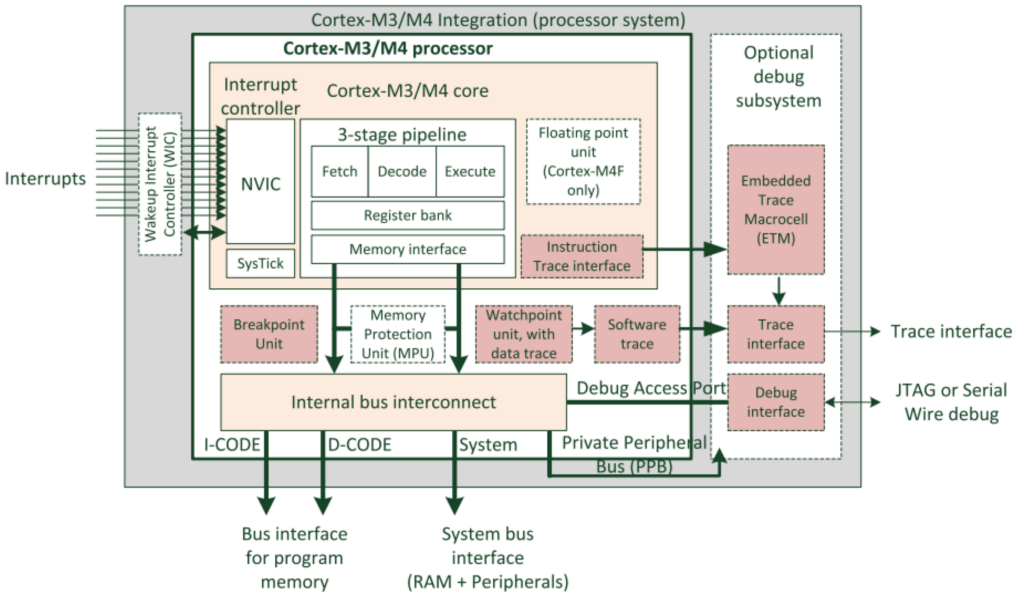
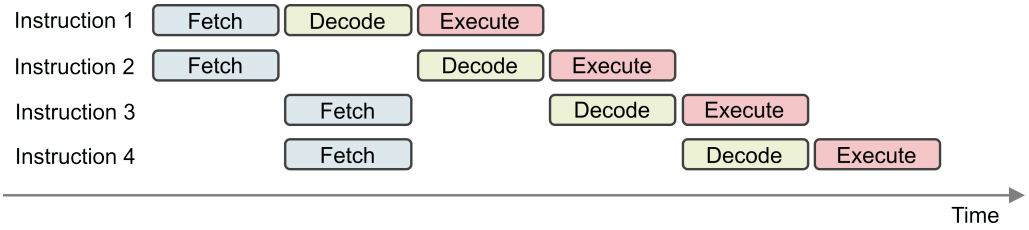


Figure 1.13: Block diagram of the Cortex-M4 processor.

1. Processor core which contains :
   - The ALU, data path, and some control logic.
   - Sixteen 32-bit registers (Register bank) for both general and special usage. General-purpose register are used to store and process temporary data within the processor core.
2. Processor pipeline stages
   - Three-stage pipeline: fetch, decode, and execution
   - Some instructions may take multiple cycles to execute, in which case the pipeline will be stalled.
   - Speculatively prefetches instructions from branch target addresses.
   - Up to two instructions can be fetched in one transfer (16-bit instructions).



3. Nested Vectored Interrupt Controller (NVIC): It allows the processor to respond to external events (interrupts) or internal events (exceptions) in a controlled manner. it has:
    - Up to 240 interrupt request signals and a Non-Maskable Interrupt (NMI).
    - It can automatically handles nested interrupts, such as comparing priorities between interrupt requests and the current priority level
4. Wakeup Interrupt Controller (WIC)
    - For low-power applications, the microcontroller can enter sleep mode by shutting down most of the components.
    - When an interrupt request is detected, the WIC can inform the power management unit to power up the system.
5. Memory Protection Unit (MPU): The MPU is an optional component used to protect memory content, e.g., make some memory regions read-only or preventing user applications from accessing privileged application data.
6. Bus interconnect
    - It allows data transfer to take place on different buses simultaneously.
    - It provides data transfer management, e.g. write buffer, bit-oriented operations (bit-band)
    - May include bus bridges (e.g. AHB-to-APB bus bridge) to connect different buses into a network using a single global memory space.
    - Includes the internal bus system, the data path in the processor core, and the AHB LITE interface unit.
7. Debug subsystem
    - It handles debug control, program breakpoints, and data watchpoints.
    - When a debug event occurs, it can put the processor core in a halted state, so developers can analyse the status of the processor, such as register values and flags, at that point.
8. Floating Point Unit (FPU): It is an optional block, it is dedicated hardware designed to accelerate floating-point arithmetic operations.
9. Memory Protection Unit (MPU): Optional provided for memory protection.
10. Bus interfaces : comprises of three Advanced High-performance Bus-Lite (AHB-Lite) interfaces and Private Peripheral Bus (PPB).
11. Low-cost debug solution features debug access to all memory and registers in the system, Serial Wire Debug Port (SW-DP) or Serial Wire JTAG Debug Port (SWJ-DP) debug access, or both.
12. Set of components to support software debug operations (the debug features are optional).
13. Optionally Cortex-M4 offers Flash Patch and Breakpoint (FPB) unit for implementing breakpoints and code patches.
14. The internal bus interconnect: is needed to route transfers from the processor and the debugger to various parts of the design.

1.3.10 Memory system

The Cortex M4 processor does not have program memory, SRAM, or cache. Instead, it comes with a generic on-chip bus interface, so microcontroller vendors will need to add the following items to the memory system:

- Program memory, typically flash
- Data memory, typically SRAM
- Peripherals

Based on the Advanced Microcontroller Bus Architecture (AMBA) standard, which includes a collection of several bus protocol specifications. The primary bus interface protocol is AHB Lite (Advanced High-performance Bus), utilized in program memory and system bus interfaces. Another bus protocol used is the Advanced Peripheral Bus (APB) interface. Therefore, silicon designers can freely employ these protocol standards to connect memories and peripherals.

1.4 Arm Cortex-M4 processor registers

The Arm Cortex-M4 processor registers are divided into two parts (See Figure 1.14):

1.4.1 Register bank

The register bank contains sixteen 32-bit registers, including 13 general-purpose (R0-R12), a Stack Pointer (SP, R13), a Link Register (LR, R14), a Program Counter (PC, R15). The R0–R12 general purpose registers can be divided into high and low registers:

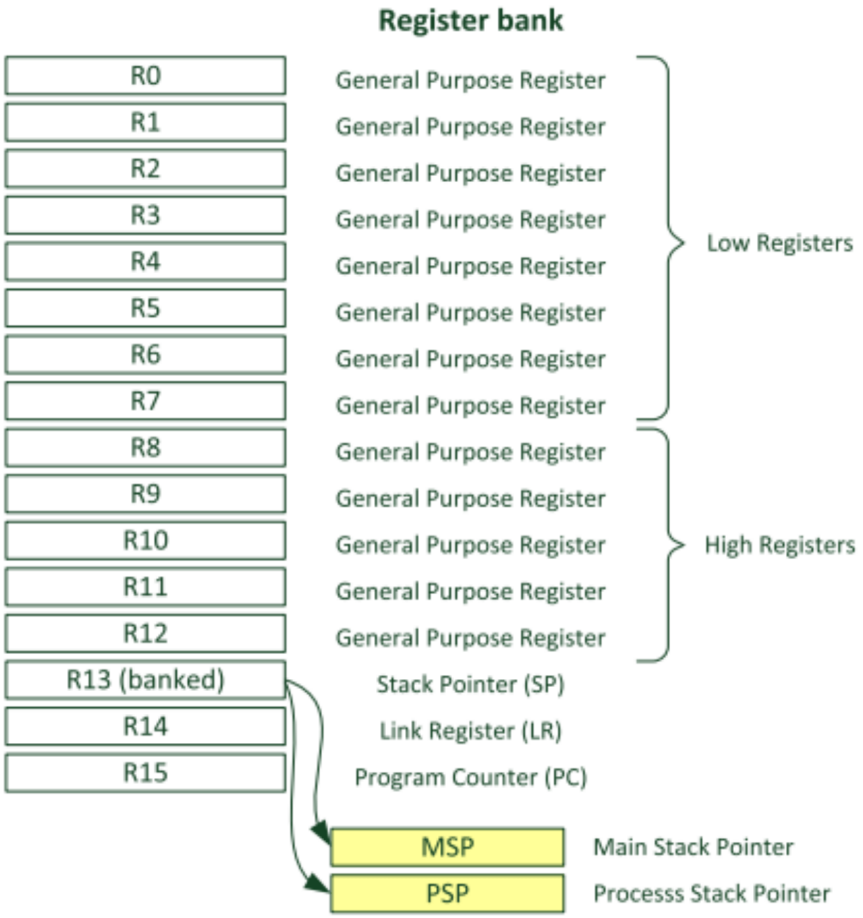


Figure 1.14: Registers of the Register Bank of ARM Cortex-M4 Processor.

- **Low registers:** Registers R0-R7 that are accessible by all instructions that specify a general-purpose register.
- **High registers:** Registers R8-R12 that are accessible by all 32-bit instructions that specify a general-purpose register and not accessible by most 16-bit instructions.
- **Stack Pointer:** R13 is used as the SP, it has the following functions:
  - Records the current address of the stack.
  - Used for saving the context of a program while switching between tasks.
  - Cortex-M4 has two SPs: Main SP (MSP), used in applications that require privileged access e.g. OS kernel, and Process SP (PSP), used in base-level application code (when not running an exception handler)
- **Program Counter:** R15 is used as the PC, it has the following functions:
  - Records the address of the current instruction code .
  - Automatically incremented by four at each operation (for 32-bit instruction code (Bit [0] is always 0)), except branching operations.
  - A branching operation, such as function calls, will change the PC to a specific address, while saving the current PC to the Link Register (LR).

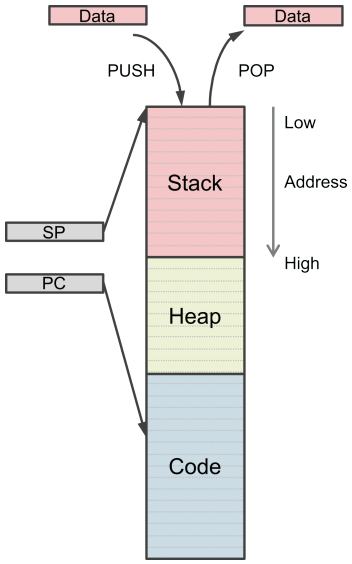


Figure 1.15: PC and SP Usage in Cortex-M4 Processor.

- **Link Register:** R14 is used as the LR, it has the following functions:
  - The LR is used to store the return address of a subroutine or a function call. It receives the return address from PC when a Branch and Link (BL) or Branch and Link with Exchange (BLX) instruction is executed.
  - The program counter (PC) will load the value from LR after a function is finished.
  - A branching operation, such as function calls, will change the PC to a specific address, while saving the current PC to the LR.

1.4.2 Special registers

Figure 1.17 shows the five special registers of ARM cortex M4 processor:

- **xPSR, combined Program Status Register:** it provides information about program

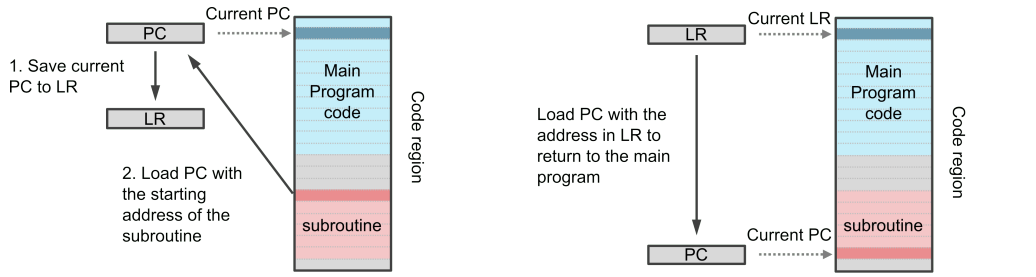


Figure 1.16: Link Register Usage in Cortex-M4 Processor.

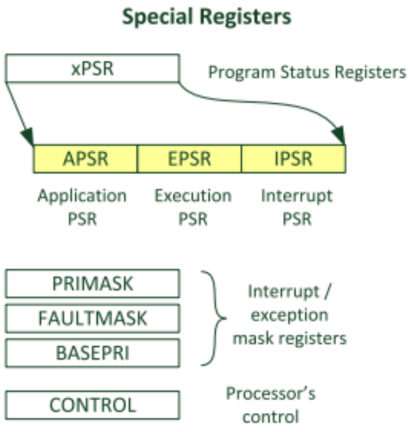


Figure 1.17: Special Registers of ARM Cortex-M3 Processor.

execution and ALU flags, it combines three registers that are mutually exclusive bitfields in the 32-bit PSR register (See Figure 1.18):

- The Application Program Status Register (APSR): it contains the current state of the condition flags from previous instruction executions.
- Interrupt Program Status Register (IPSR): it contains the exception type number of the current Interrupt Service Routine (ISR).
- The Execution Program Status Register (EPSR): it contains the Thumb state bit, and the execution state bits for either the If-Then (IT) instruction or the Interruptible-Continuable Instruction (ICI) field for an interrupted load multiple or store multiple instruction.

Figure 1.18: Bit Configuration of the Program Status Register.

The configuration of the various utilized bits is as follows:

- **APSR**
  - N: negative flag – set to one if the result from ALU is negative
  - Z: zero flag – set to one if the result from ALU is zero
  - C: carry flag – set to one if an unsigned overflow occurs
  - V: overflow flag – set to one if a signed overflow occurs
  - Q: sticky saturation flag – set to one if saturation has occurred in saturating arithmetic instructions, or overflow has occurred in certain multiply instructions
- **IPSR**
  - ISR number – current executing interrupt service routine number
- **EPSR**
  - T: Thumb state – always one since Cortex-M4 only supports the Thumb state (more on processor states in the next module)
  - ICI/IT: Interrupt-Continuable Instruction (ICI) bit, IF-THEN instruction status bit
- **Exception mask registers:** they disable the handling of exceptions by the processor. Disable exceptions where they might impact on timing critical tasks (see Figure 1.19).
  - **1-bit PRIMASK (Priority Mask Register):**
    If set to one, will block all the interrupts apart from non-maskable interrupt (NMI) and the hard fault exception.
  - **1-bit FAULTMASK (Fault Mask Register) :**
    If set to one, it will block all interrupts, including the HardFault handler, except for NMI, allowing the NMI exception handler to be executed in this state.
  - **1-bit BASEPRI (Base Priority Mask Register) :**
    It is used to set a priority threshold, and disable all interrupts with higher priority value than threshold when it is set to one.

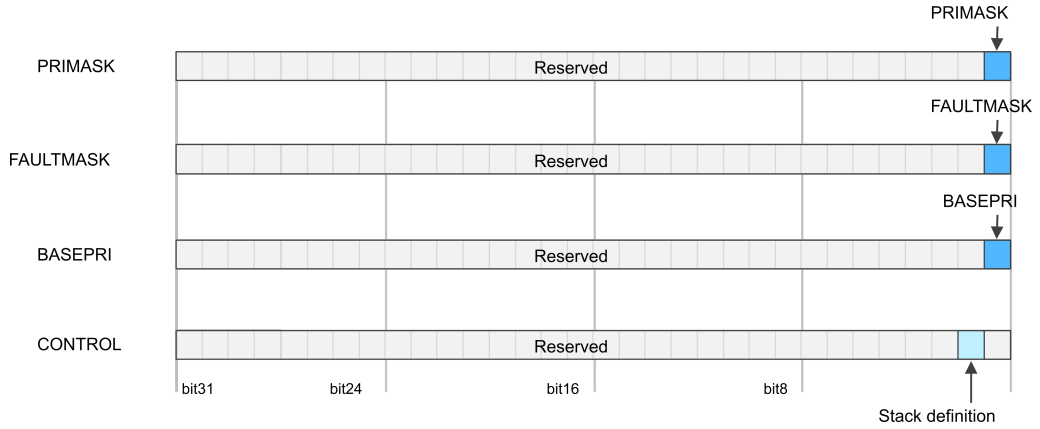


Figure 1.19: Bit Configuration of the special Registers.

- **CONTROL: special register**
  - **1-bit stack definition**
    - Set to one to use the process stack pointer (PSP)
    - Clear to zero to use the main stack pointer (MSP)

1.5 Arm Cortex-M4 memory map

The Arm Cortex-M4 memory map describes the organization of the processor's address space. The address space contains data sources and sinks such as internal and external RAM and ROM as well as buses, peripherals and further parts and equipment.

The total memory space of a 32bit system, like the Cortex-M4, is $2^{32} = 4$GB. It is split and separated into regions with different functionality which can be individually changed, apart from some fixed assigned addresses.

Figure 1.20 shows the memory regions of Armv7-M architecture. Each region has a defined memory type, and some regions have additional memory attributes. The memory type and attributes determine the behavior of accesses to the region. The only real requirement for data memory is that the memory must be byte addressable. The address range and properties of these regions are as follows:

- **Code region**[0x00000000- 0x1FFFFFFF]: it is the executable region for the program core. Data can also be put here. The memory is on-chip FLASH memory.
- **SRAM region**[0x20000000- 0x3FFFFFFF]: it is primarily used to store data, such as heaps and stacks, can also be used for program code. This region includes the bit band and bit band alias areas.
- **Peripheral region**[0x40000000- 0x5FFFFFFF]: it is used for peripherals attached to the core on the address and data bus such as Advanced High-performance Bus (AHB) or Advanced Peripheral Bus (APB) peripherals. This region includes bit band and bit band alias areas.
- **External RAM region** [0x60000000 - 0x9FFFFFFF]: it is primarily used to store large data blocks or memory caches. it is used for off-chip memory, slower than on-chip SRAM region.
- **External device region** [0xA0000000 - 0xDFFFFFFF]: it is primarily used to map to external devices , off-chip devices, such as SD card.

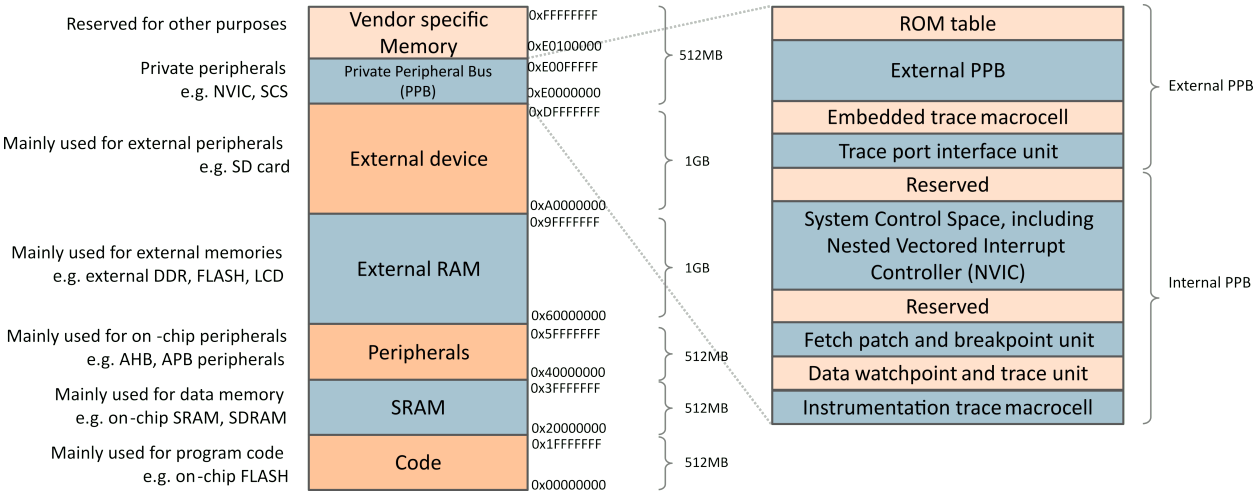


Figure 1.20: Arm Cortex-M4 memory map.

- **Private Peripheral Bus region** [0xE0000000 - 0xE00FFFFF]: it provides access to internal and external processor resources.

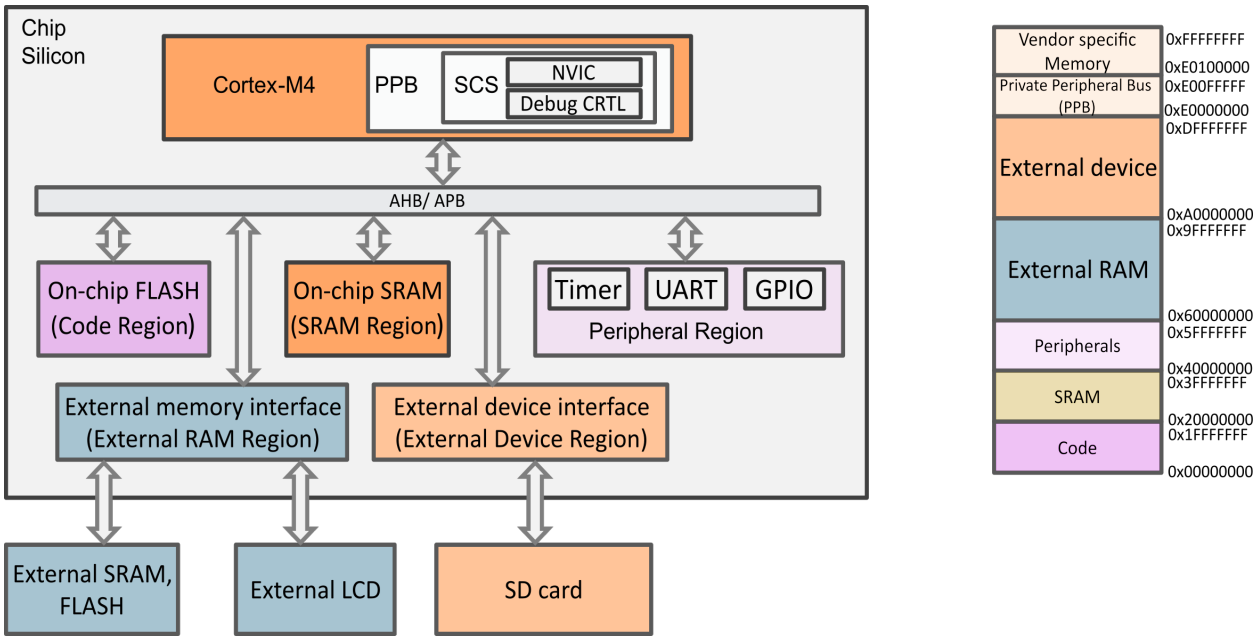


Figure 1.21: Example of Arm Cortex-M4 memory map.

In Figure 1.21, a structural example of the memory map is given. Beginning at the top, the processor internal elements connected to the PPB, such as debug ctrl and NVIC, are implemented. All additional components are connected to the processor via the AHB (Advanced High-Performance Bus), including the on-chip peripherals such as the timer, general purpose input/output ports or communication systems like UART. Directly attached to the high-performance bus are the chip internal memory components for the ROM (flash) and RAM. However, external memory and peripheral components share a

common interface to the AHB respectively between several components.

1.6 Memory endianness

Endianness refers to the order in which bytes are stored in a memory. There are two main types of endian architectures: **Big-endian** and **Little-endian**. Their difference can be compared to the distinction between English and Arabic writing. In English, writing progresses from left to right, while in Arabic, it proceeds from right to left.

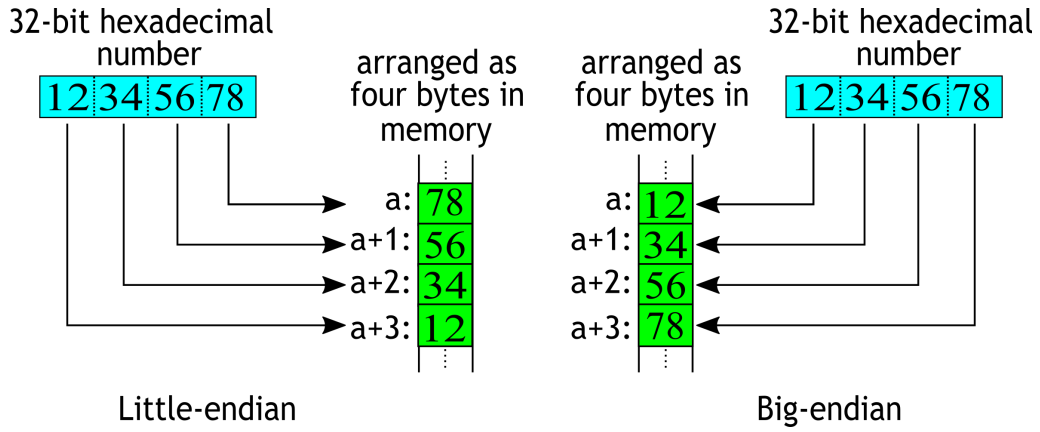


Figure 1.22: Principle of Little-endian and Big-endian Byte Ordering

Similarly, when a word-size data is stored in memory, if the rightmost (least significant) byte is stored at the lowest memory address, then it is a Little-Endian memory system. If the leftmost (most significant) byte is stored first, it is a Big-Endian memory system, as shown in Figure 1.22.

The Cortex-M4 processor supports both little-endian and big-endian memory systems. The endianness of the memory system is determined at system reset. Once set, the endianness cannot be changed until the next system reset.

1.7 Bit-Band Alias Region and address

The bit-band alias region is a memory region that provides a one-to-one mapping between individual bits in the alias region and corresponding bits in the actual memory. This implies that within the bit-band region, each word is represented by an LSB of 32 words in the bit-band alias address range. Therefore, by using bit-band operations, we can directly write '0' or '1' to the alias address, allowing us to set/clear the corresponding bit in the actual memory.

SRAM and Peripheral regions include bit-band and bit-band alias areas as shown in Figure 1.23.

SRAM region

For the SRAM region, 32MB memory space (0x22000000 – 0x23FFFFFF) is used as the bit-band alias region for 1MB data (0x20000000 – 0x200FFFFF).

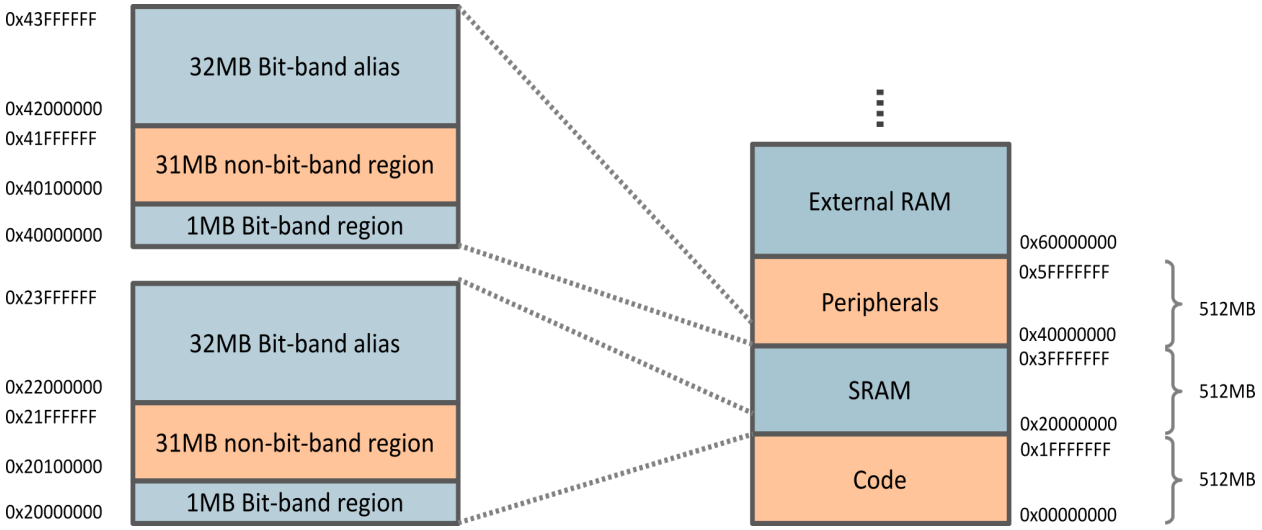


Figure 1.23: Bit-Band Alias Region for SRAM and Peripherals Regions

Peripherals region

For the peripherals region, 32MB memory space (0x42000000 – 0x43FFFFFF) is used as the bit-band alias region for 1MB data (0x40000000 – 0x400FFFFF).

Bit-band address

The bit-band address is calculated using the following formula:

$$\textit{bit-band-address} = \textit{bit-band-basis-address} + (\textit{byte-offset} \times 32) + (\textit{bit-number} \times 4)$$

Where :

- bit-band-basis-address : is the starting address of the bit-band region.
- byte-offset: is the byte offset of the target word in memory (it is set to zero when you are working with a single variable, it changes with an array or a structure)
- bit-number: is the position of the target bit within the word.

Example:

In this example, we want to calculate the bit-band alias address of the fourth bit (bit[3]) of the data at address 0x20000000.

bit-band-address = 0x20000000 + (0 x 32) + (3 x 4) = 0x2000000C

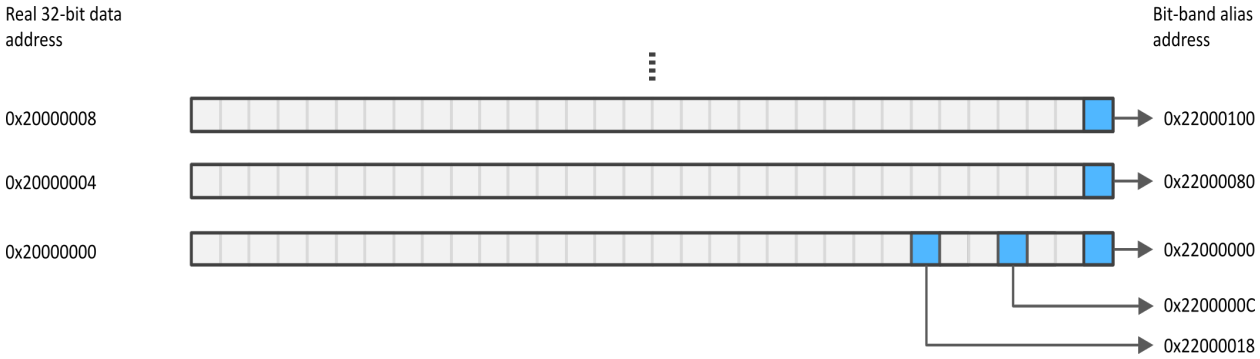
1.7.1 Bit-band operations

To change a single bit of a 32-bit data, we can use the standard operation based on the read-modify-write procedure or the alternative method introduced by Arm called bit-band operation:

read-modify-write operation

In this operation, three steps should be performed:

1. Read the value of 32-bit data



2. Modify a single bit of the 32-bit value with AND, OR, XOR instructions (keep other bits unchanged)
3. Write the value back to the address

Bit-band operation

Using bit-band operation, directly write a single bit (0 or 1) to the "bit-band alias address" of the data.

Example 1:

To set the fourth bit (bit[3]) of the 32-bit data at 0x20000000 address to '1', we will write '1' to address 0x2200000C, which is the alias address of the fourth bit of the 32-bit data at 0x20000000.

```
;Bit-band Operation

LDR R1, =0x2200000C ;Setup address
MOV R0, #1                ;Load data
STR R0, [R1]              ;Write
```

Example 2:

In this example, we demonstrate read and write operations to/from the bit-band alias region:

1. Store the value 0x3355AACC at memory address 0x20000000.
2. Read from memory address 0x22000008. This read access is remapped to access 0x20000000. The returned value is 1 (bit[2], the third bit, of 0x3355AACC), it is like reading of bit[2] from memory address 0x20000000.
3. Write 0x0 to to memory address 0x22000008. This write access is remapped into a READ-MODIFY-WRITE to 0x20000000. The value 0x3355AACC is read from memory, bit 2 is cleared, and resulting in 0x3355AAC8, which is then written back to memory address 0x20000000.
4. Now read 0x20000000. That gives you a return value of 0x3355AAC8 (bit[2] cleared).