# Unit-4

# Memory System

# Overview

- Memory System
- Memory Map
- Elements of a Microcontroller
- *Program Memory, Boot Loader, and Memory Remapping*
- *Little Endian and Big Endian Memory Support*
- *Data Alignment*
- *Memory Attributes*
- *Memory Types*
- *Memory Attribute Map*

# Memory System

- The Cortex-M0 processor has a **32-bit system bus** interface with 32-bit address lines (4 GB of address space).
- The system bus is based on a bus protocol called **AHB-Lite** (Advanced Highperformance Bus), which is a protocol defined in the Advanced Microcontroller Bus Architecture (AMBA) standard.
- A secondary bus segment can also be found for slower devices including peripherals.
- In ARM microcontrollers, **the peripheral bus system** is normally based on the Advanced Peripheral Bus **(APB)** protocol.
- The APB is connected to the AHB-Lite via a bus bridge and may run at a different clock speed compared to the AHB system bus.
- The data path on the APB is also 32-bit, but the address lines are often less than 32-bit as the peripheral address space is relatively small (Figure 7.1).

# Memory System

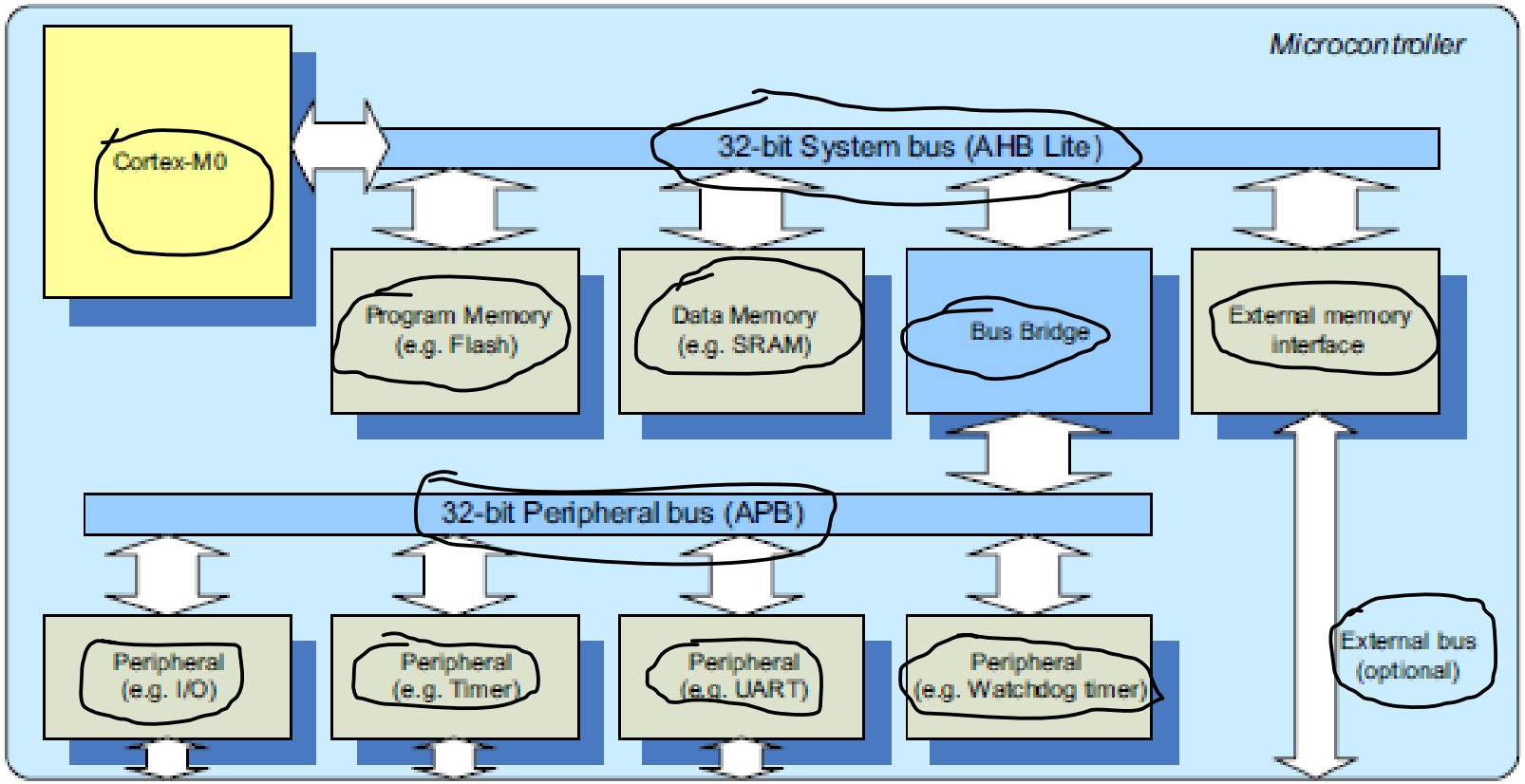


**Figure 7.1:**
Separation of system and peripheral bus in typical 32-bit microcontrollers.

# Memory System

- Because of the separation of main system bus and peripheral bus, and **in some cases with separated clock frequency controls, an application might need to initialize some clock control hardware in the microcontroller before accessing the peripherals**.
- In some cases, there can be multiple peripheral bus segments in a microcontroller running at different clock frequencies.
- Besides allowing some part of the system to run at a slower speed, the separation of bus segments also provides the **possibility of power reduction by allowing the clock to a peripheral system to be stopped.**

# Memory System

- Depending on the microcontroller design, **some high-speed peripherals might be connected to the AHB-Lite system bus** instead of the APB.
- This is because the AHB-Lite protocol requires fewer clock cycles for each transfer when compared to the APB.

- The bus protocol behavior affects the system operation and the programmer's view on the memory system in a number of ways.

# Memory Map

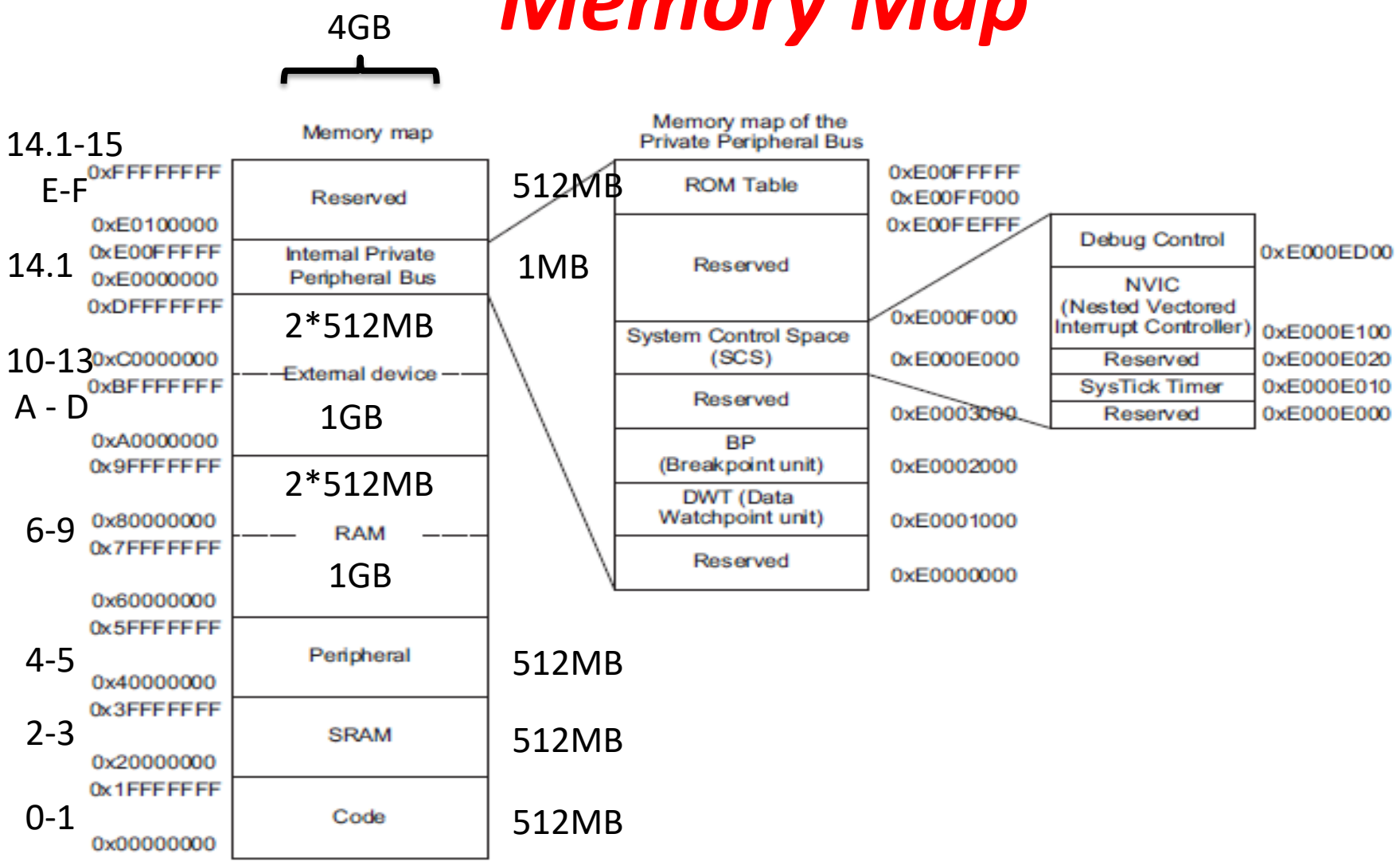


**Figure 7.2:**
Architecturally defined memory map of the Cortex-M0 processor.

# *Memory Map*

- The 4GB memory space of the Cortex-M0 processor is architecturally divided into a number of regions (Figure 7.2). Each region has its recommended usage, and the memory access behavior could depend on which memory region you are accessing to.

# Memory Map

- ***Code Region (0x00000000- 0x1FFFFFFF)*** The size of the code region is 512 MB. It is primarily used to store **program code**, including the **exception vector table**, which is a part of the program image. It can also be used for data memory (connection to RAM).
- ***SRAM Region (0x20000000 -0x3FFFFFFF)*** The SRAM region is the located in the next 512 MB of the memory map. It is primarily used **to store data, including stack**. It can also be used to store program code. For example, in some cases you might want to copy program code from slow external memory to the SRAM and execute it from there. if the SRAM memory range is from 0x20000000 to 0x20007FFF, we can start the stack pointer at 0x20008000.
- In this case, the first stack PUSH will take place at address 0x20007FFC, the top word of the SRAM.
- ***Peripheral Region (0x40000000 - 0x5FFFFFFF)*** The peripheral region also has the size of 512 MB. It is primarily used **for peripherals and can also be used for data storage**. However, program execution is not allowed in the peripheral region. The peripherals connected to this memory region can be either the AHB-Lite peripheral or APB peripherals (via a bus bridge).

# Memory Map

- ***RAM Region (0x60000000-0x9FFFFFFF)*** The RAM region consists of two 512 MB blocks, which results in total of 1 GB of space. Both 512 MB memory blocks are primarily **used to stored data**, and in most cases the RAM region can be used as a 1GB continuous memory space. The RAM region can also be used for **program code execution**.
- ***Device Region (0xA0000000 - 0xDFFFFFFF)*** The external device region consists of two 512 MB memory blocks, which results in a total of 1 GB of space. Both 512MBmemory blocks are primarily **used for peripherals and I/O usage**. The device region **does not allow program execution**, but it can be used for general data storage.
- **Internal Private Peripheral Bus (PPB) (0xE0000000 - 0xE00FFFFF)** The internal PPB memory space is allocated for peripherals inside the processor**, such as the interrupt controller NVIC, as well as the debug components**. The internal PPB memory space is 1 MB in size, and program execution is not allowed in this memory range.
- **Reserved Memory Space (0xE0100000 - 0xFFFFFFFF)** The last section of the memory map is a 511 MB reserved memory space. This may be reserved in some microcontrollers for vendor-specific usages.

# Elements for a Microcontroller

- Flash memory (for program code)
- Internal SRAM (for data)
- Internal peripherals
- External memory interface (for external memories as well as external peripherals (optional))
- Interfaces for other external peripherals (optional)

# *Program Memory, Boot Loader, and Memory Remapping*

- The program memory of the Cortex-M0 is implemented with on chip flash memory.
- When the Cortex-M0 processor comes out of reset, it accesses the vector table in address zero for initial MSP value and reset vector value, and it then starts the program execution from the reset vector.
- To ensure that the system works correctly, a valid vector table and a valid program memory must be available in the system to prevent the processor from executing rogue program code.

# Boot Loader

- **boot loader**, a small program located on the microcontroller chip that executes after the processor powers up and branches to the user application in the flash memory only if the flash is programmed.
- When a boot loader is present, it is common for the microcontroller vendor to implement a memory map-switching feature called **"remap"** on the system bus.
- The switching of the memory map is controlled by a hardware register, which is programmed when the boot loader is executed. There are various types of remap arrangements.

# Two types of Remap

- One common remap arrangements to allow the boot loader to be mapped to the start of the memory during the power-up phase using address alias, as shown in Figure 7.4.
- The boot loader might also support additional features like hardware initialization (clock and PLL setup), supporting of multiple boot configurations, firmware protection, or even flash erase utilities.
- The memory remap feature is implemented on the system bus and is not a part of the Cortex-M0 processor, therefore different microcontrollers from different vendors have different implementations.

# Boot Loader Remapping

## *Memory Remapping*

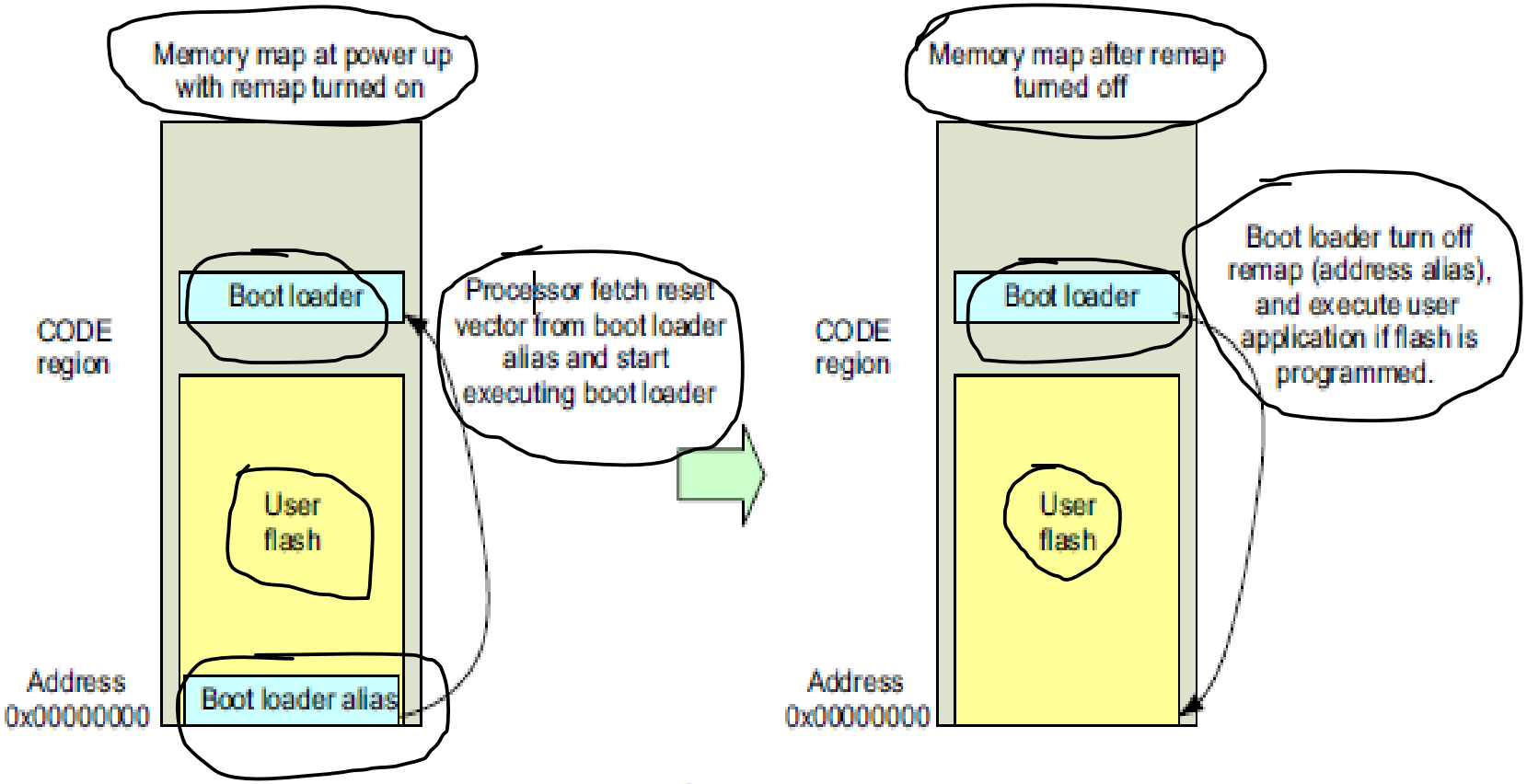


**Figure 7.4:**
An example of a memory-remap implementation with the boot loader.

# SRAM Remap

- Another common type of remap feature implemented on some ARM microcontrollers allows an SRAM block to be remapped to address 0x0 (Figure 7.5).
- Normally nonvolatile memory used on microcontrollers like flash memory is slower than SRAM. When the microcontroller is running at a high clock rate, wait states would be required if the program is executed from theflash memory.
- By allowing an SRAM memory block to be remapped to address 0x0, then the program can be copied to SRAM and executed at maximum speed.
- This also avoids wait states in vector table fetch, which affects interrupt latency.

# SRAM Remapping

# *Memory Remapping*

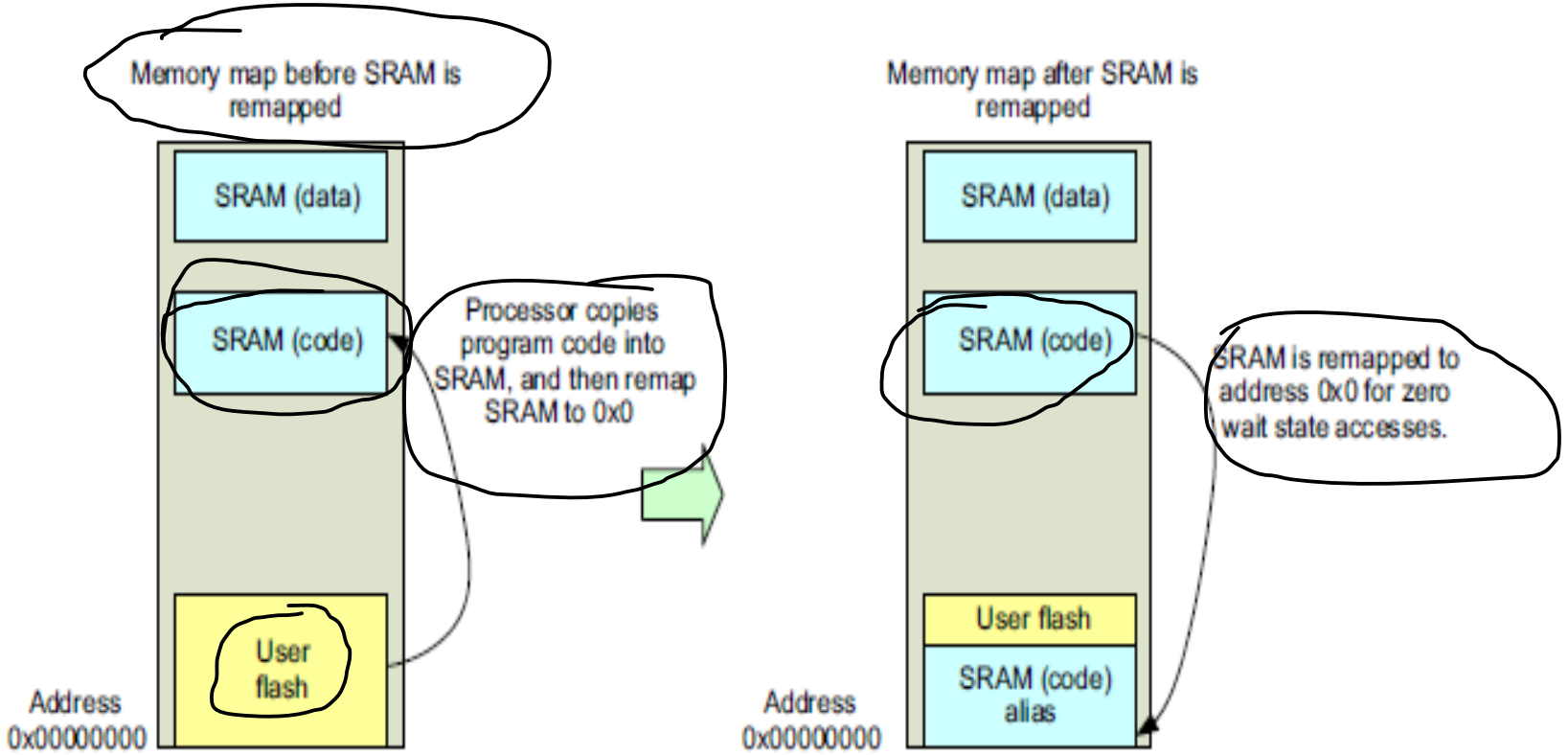


**Figure 7.5:**
A different example of memory-remap implementation—SRAM for fast program accesses.

# Data Memory

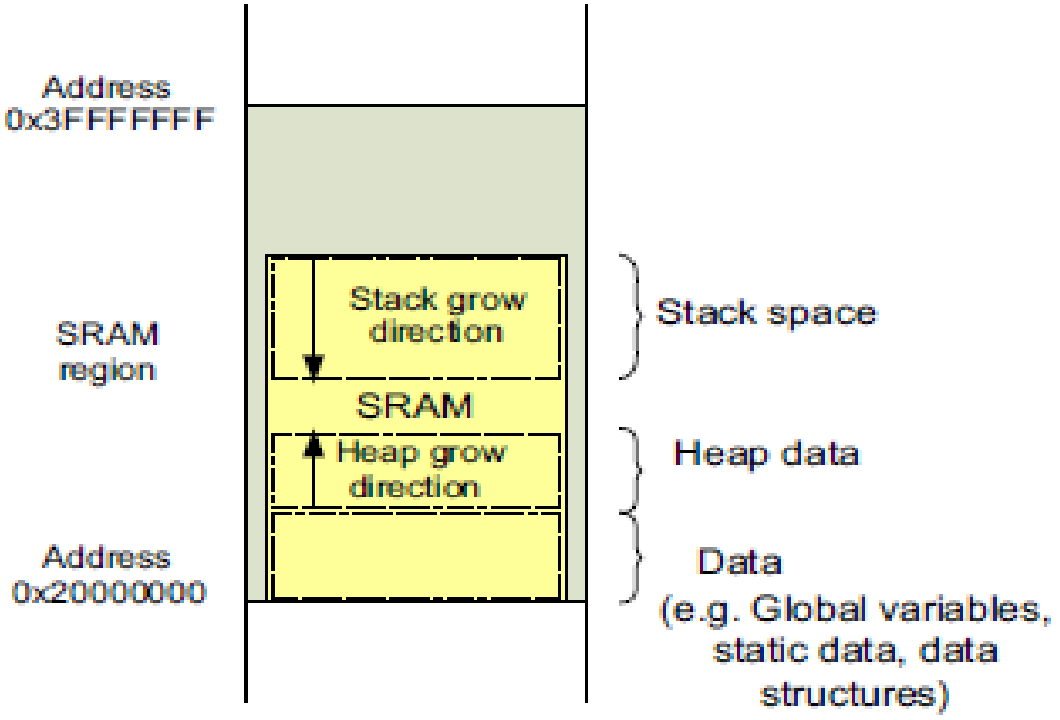


**Figure 7.6:**
An example of common SRAM usage.

# *Little Endian Memory Support*

Lowest byte in LSB

High ……. Low

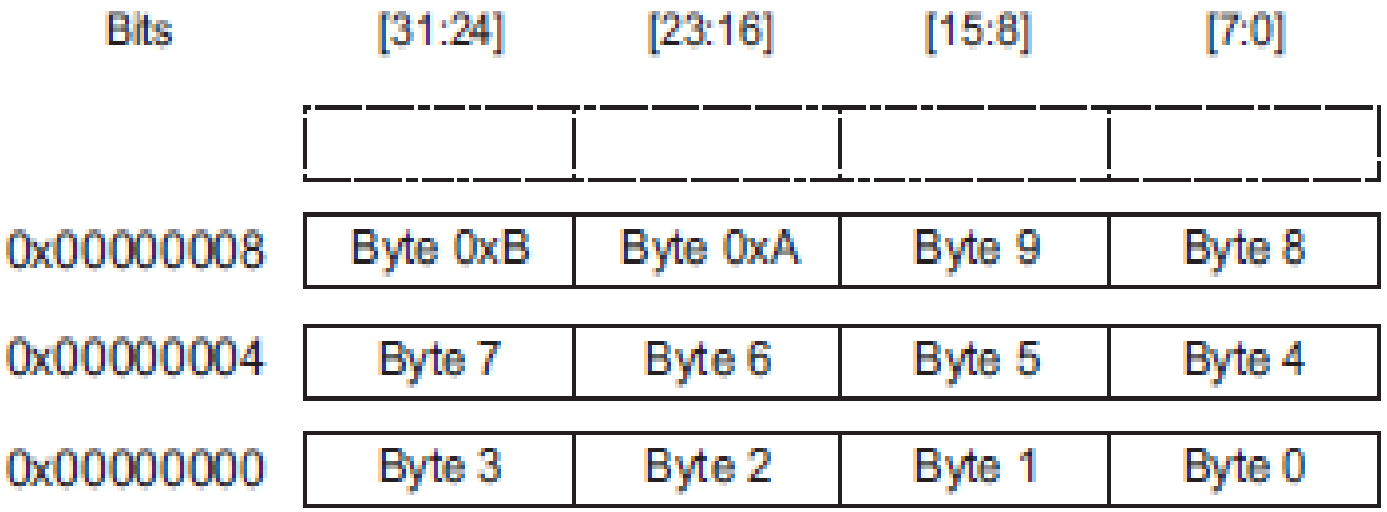


**Figure 7.7:**
Little endian 32-bit memory.

# *Big Endian Memory Support*

Lowest byte in MSB

Low …….. High

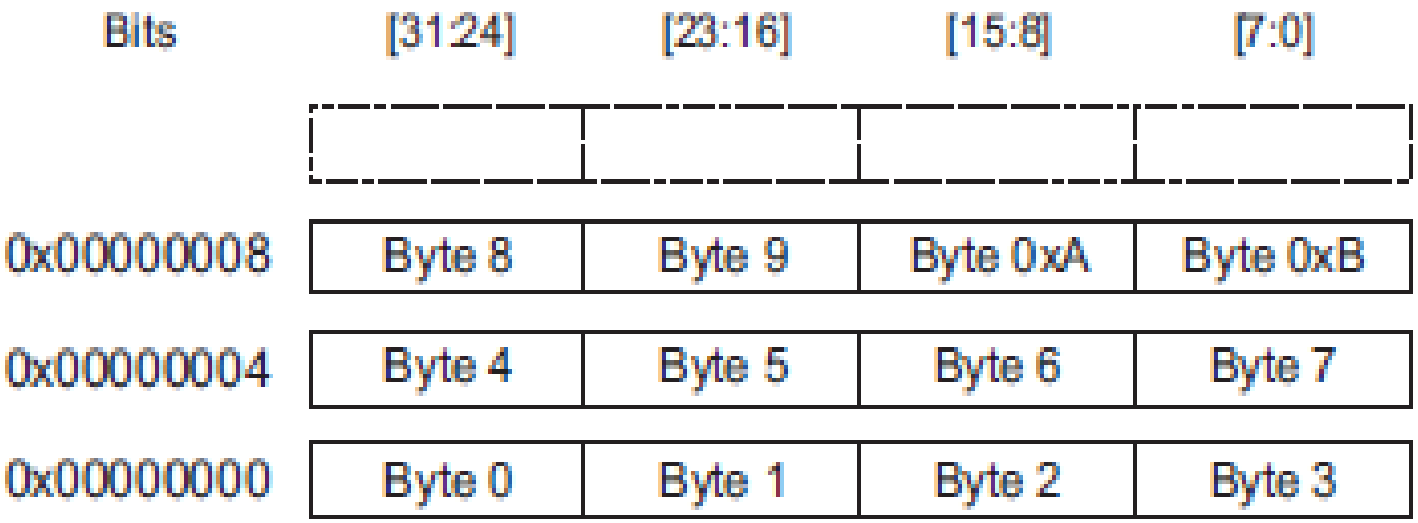


**Figure 7.8:**
Big endian 32-bit memory.

# Data Access in *Little Endian System*

| Address | Size | Bits 31-24 | Bits 23-16 | Bits 15-8 | Bits 7-0 |
|---|---|---|---|---|---|
| 0x00000000 | Word | Data[31:24] | Data[23:16] | Data[15:8] | Data[7:0] |
| 0x00000000 | Half word | | | Data[15:8] | Data[7:0] |
| 0x00000002 | Half word | Data[15:8] | Data[7:0] | | |
| 0x00000000 | Byte | | | | Data[7:0] |
| 0x00000001 | Byte | | | Data[7:0] | |
| 0x00000002 | Byte | | Data[7:0] | | |
| 0x00000003 | Byte | Data[7:0] | | | |

**Figure 7.9:**
Data access in little endian systems.

# Data Access in *Big Endian System*

| Address | Size | Bits 31-24 | Bits 23-16 | Bits 15-8 | Bits 7-0 |
|---|---|---|---|---|---|
| 0x00000000 | Word | Data[7:0] | Data[15:8] | Data[23:16] | Data[31:24] |
| 0x00000000 | Half word | Data[7:0] | Data[15:8] | | |
| 0x00000002 | Half word | | | Data[7:0] | Data[15:8] |
| 0x00000000 | Byte | Data[7:0] | | | |
| 0x00000001 | Byte | | Data[7:0] | | |
| 0x00000002 | Byte | | | Data[7:0] | |
| 0x00000003 | Byte | | | | Data[7:0] |

**Figure 7.10:**
Data access in big endian system.

# *Data Alignment*

- The Thumb instruction set supported by the Cortex-M0 processor can only generate aligned transfers
- It means that the transfer address must be a multiple of the transfer size.
- For example, a **word size transfer** can only access addresses like **0x0, 0x4, 0x8, 0xC**, and so forth. Similarly, a **half-word transfer** can only access addresses like **0x0, 0x2, 0x4**, and so forth.
- All byte data accesses are aligned.
- If the program attempts to generate an unaligned transfer, this will result in a fault exception and cause the hard fault handler to be executed.

# *Memory Attributes*

- **Executable.** The **Executable** attribute defines whether program execution is allowed in that memory region. If a memory region is define as nonexecutable, in ARM documentation it is marked as **eXecute Never (XN).**
- **Bufferable.** When a data write is carried out to a bufferable memory region, the write transfer can be buffered, which means the processor can continue to execute the next instruction without waiting for the current write transfer to complete.

# Memory Attributes

- **Cacheable.** If a cache device is present on the system, it can keep a local copy of the data during a data transfer and reuse it the next time the same memory location is accessed to speed up the system. The cache device can be a cache memory unit or a small buffer in a memory controller.
- **Shareable.** The shareable attribute defines whether more than one processor can access a shareable memory region. If a memory region is shareable, the memory system needs to ensure coherency between memory accesses by multiple processors in this region.

# Memory Types

- **Normal memory.** Normal memories can be shareable or nonshareable and cacheable or noncacheable.
- **Device memory.** Device memories are noncacheable. They can be shareable or nonshareable.
- **Strongly-ordered (SO) memory.** A memory region that is nonbufferable, noncacheable, and transfers to/from a strongly ordered region takes effect immediately.

# Memory Attribute Map

Table 7.3: Memory Attribute Map

| Address | Region | Memory Type | Cache | XN | Descriptions |
|---|---|---|---|---|---|
| 0x00000000–0x1FFFFFFF | CODE | Normal | WT | — | Memory for program code including vector table |
| 0x20000000–0x3FFFFFFF | SRAM | Normal | WBWA | — | SRAM, typically used for data and stack memory |
| 0x40000000–0x5FFFFFFF | Peripheral | Device | — | XN | Typically used for on-chip devices |
| 0x60000000–0x7FFFFFFF | RAM | Normal | WBWA | — | Normal memory with Write Back, Write Allocate cache attributes |
| 0x80000000–0x9FFFFFFF | RAM | Normal | WT | — | Normal memory with Write Through cache attributes |
| 0xA0000000–0xBFFFFFFF | Device | Device, shareable | — | XN | Shareable device memory |
| 0xC0000000–0xDFFFFFFF | Device | Device | — | XN | Nonshareable device memory |
| 0xE0000000–0xE00FFFFF | PPB | Strongly ordered, shareable | — | XN | Internal private peripheral bus |
| 0xE0100000–0xFFFFFFFF | Reserved | Reserved | — | — | Reserved (vendor-specific usage) |