CSE 3103: Microprocessor and Microcontroller

Professor Upama Kabir

Computer Science and Engineering, University of Dhaka,

Lecture : ARM Memory System

May 04, 2024

Table of Contents

Processor Memory Model

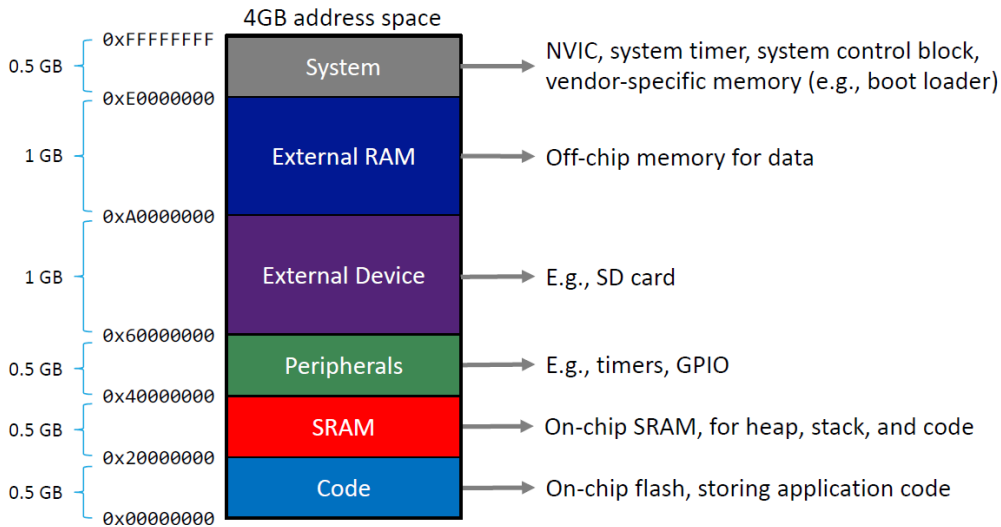


Figure 1

Processor Memory Model

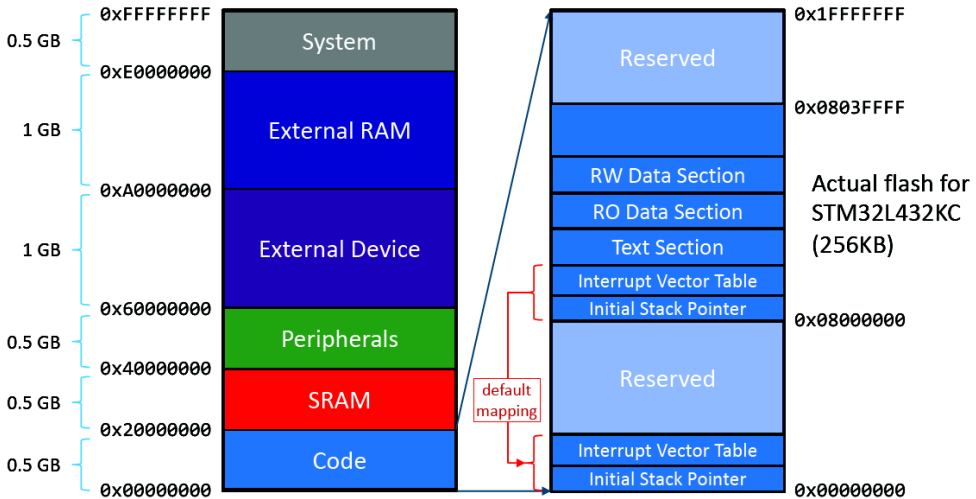
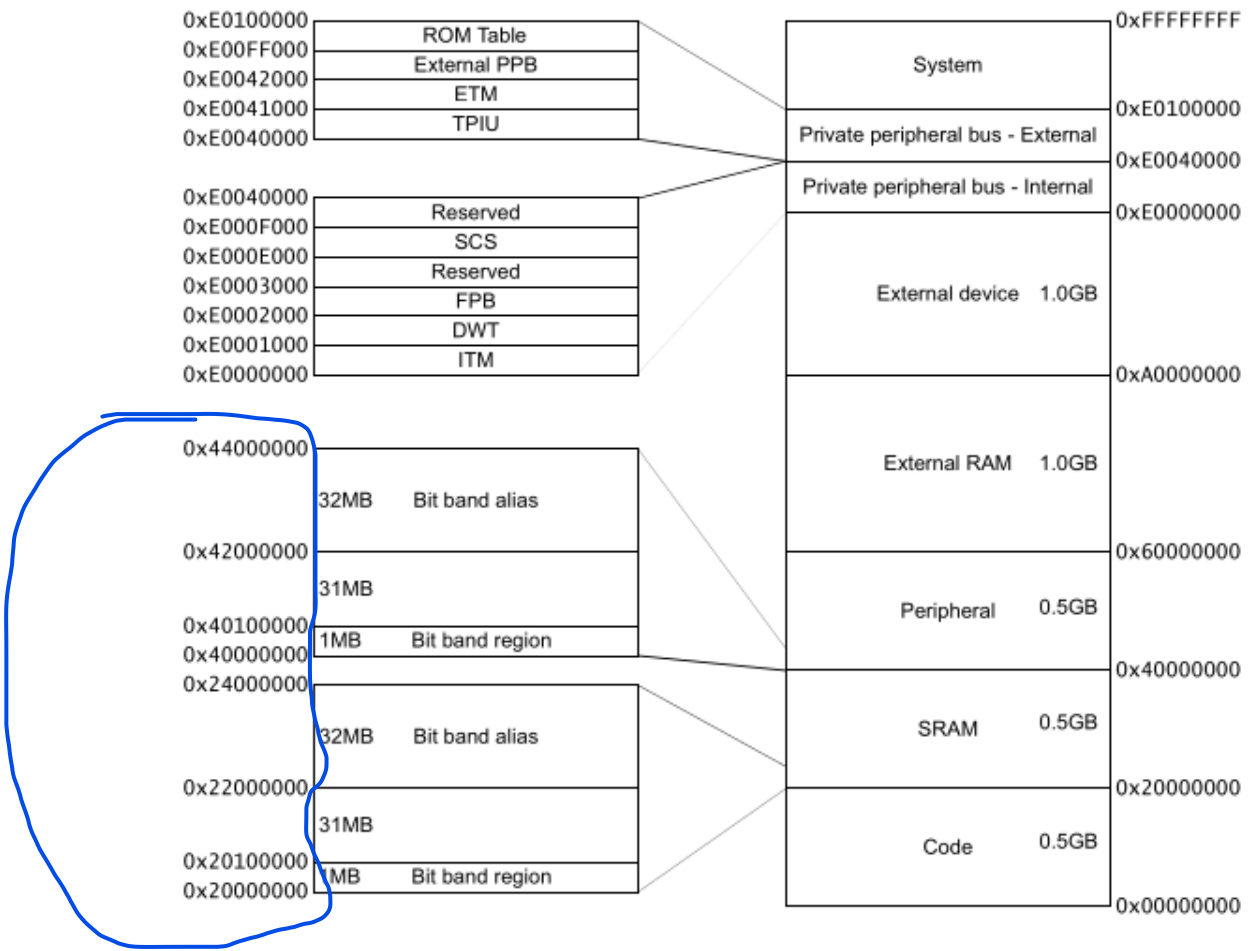


Figure 2

System Address Map



Memory Regions Table

Memory Map	Region
Code	Instruction fetches are performed over the ICode bus. Data accesses are performed over the DCode bus.
SRAM	Instruction fetches and data accesses are performed over the system bus.
SRAM bit-band	Alias region. Data accesses are aliases. Instruction accesses are not aliases.
Peripheral	Instruction fetches and data accesses are performed over the system bus.
Peripheral bit-band	Alias region. Data accesses are aliases. Instruction accesses are not aliases.
External RAM	Instruction fetches and data accesses are performed over the system bus.
External Device	Instruction fetches and data accesses are performed over the system bus.
Private Peripheral Bus	External and internal *Private Peripheral Bus* (PPB) interfaces. This memory region is *Execute Never* (XN), and so instruction fetches are prohibited. An MPU, if present, cannot change this.
System	System segment for vendor system peripherals. This memory region is XN, and so instruction fetches are prohibited. An MPU, if present, cannot change this.

Figure 4

Private Peripheral Bus

The Private Peripheral Bus (PPB) interface provides access to internal and external processor resources.

- The internal Private Peripheral Bus (PPB) interface provides access to:
  - The Instrumentation Trace Macrocell (ITM).
  - The Data Watchpoint and Trace (DWT).
  - The Flashpatch and Breakpoint (FPB).
  - The System Control Space (SCS), including the Memory Protection Unit (MPU) and the Nested Vectored Interrupt Controller (NVIC).
- The external PPB interface provides access to:
  - The Trace Point Interface Unit (TPIU).
  - The Embedded Trace Macrocell (ETM).
  - The ROM table.
  - Implementation-specific areas of the PPB memory map

System Region

The System region of the memory map, starting at 0xE0000000, subdivides as follows:

- The 1MB region at offset +0x00000000 is reserved as a Private Peripheral Bus (PPB).
- The region from offset +0x00100000 is the Vendor system region, Vendor˙SYS.
- System Control Space (SCS): The System Control Space (SCS) is a memory-mapped 4KB address space that provides 32-bit registers for configuration, status reporting and control. The SCS registers divide into the following groups:
  - System control and identification.
  - The CPUID processor identification space.
  - System configuration and status.
  - Fault reporting.
  - A system timer, SysTick.
  - A Nested Vectored Interrupt Controller (NVIC).
  - A Protected Memory System Architecture (PMSA).
  - System debug.

System Region

Table B3-3 SCS address space regions

System Control Space, address range `0xE000E000` **to** `0xE000EFFF`		
Group	**Address range**	**Notes**
System control and ID registers	`0xE000E000-0xE000E00F`	Includes the Interrupt Controller Type and Auxiliary Control registers
	`0xE000ED00-0xE000ED8F`	System Control Block
	`0xE000EDF0-0xE000EEFF`	Debug registers in the SCS
	`0xE000EF00-0xE000EF4F`	Includes the SW Trigger Interrupt Register, see *Software Triggered Interrupt Register, STIR* on page B3-675
	`0xE000EF50-0xE000EF8F`	Cache and branch predictor maintenance, see *Cache and branch predictor maintenance operations* on page B2-633
	`0xE000EF90-0xE000EFCF`	IMPLEMENTATION DEFINED
	`0xE000EFD0-0xE000EFFF`	Microcontroller-specific ID space
SysTick	`0xE000E010-0xE000E0FF`	System Timer, see *The system timer, SysTick* on page B3-676
NVIC	`0xE000E100-0xE000ECFF`	External interrupt controller, see *Nested Vectored Interrupt Controller, NVIC* on page B3-680
MPU	`0xE000ED90-0xE000EDEF`	Memory Protection Unit, see *Protected Memory System Architecture, PMSAv7* on page B3-688

Figure 5

Bit-banding – An Elegant Approach to Setting & Clearing Bits ARM Cortex M4

- The Problem with Read-Modify-Write
  - Typically a CPU core cannot write to individual bits of a register.
  - Instead it must write entire bytes or even words at a time.
  - If a CPU needs to change the value of a bit and can only write a byte at a time, it must first read the current value into a temporary register, modify that value with a logic operation, and then write the final result. This three step process is aptly named Read-Modify-Write.
  - Using Read-Modify-Write operations to set bits works fine when you're doing one thing at a time, but problems can arise when an application is doing multiple things concurrently.
  - For example, what happens if an interrupt occurs between the read and modify operations that changes the value in the register?
  - The new value will get overwritten. This race-condition could lead to undesired behavior.

Bit-banding

Bit Banding is a special feature (optional) of Cortex M3/M4 processors where any bit stored between 0x20000000 and 0x20100000 in SRAM or 0x40000000 and 0x40100000 in the peripheral memory has a corresponding alias address which can be used to reference that bit specifically.

- ARM Cortex-M4 features a 1 MB area in SRAM memory called bit-band region. In this region each bit can be accessed individually.
- To access to bit-band region bits you need to do so via an aliased region, where it maps each bit in that region to an entire word in a second memory region (the Bit-band Alias Region)
- A write to a word in the alias region performs a write to the corresponding bit in the Bit-band region.
- Reading a word in the alias region will return the value of the corresponding bit in the Bit-band region.
- These operations take a single machine instruction thus eliminate race conditions. This is especially useful for interacting with peripheral registers where it is often necessary to set and clear individual bits.

Bit banding

The image below shows a byte in the Bit-band region on the top. The bottom row of bytes represent the bit-band alias region. For demonstration purposes I choose to use 8-bit words. On the Cortex M3 and M4 the alias region would contain 32-bit words.

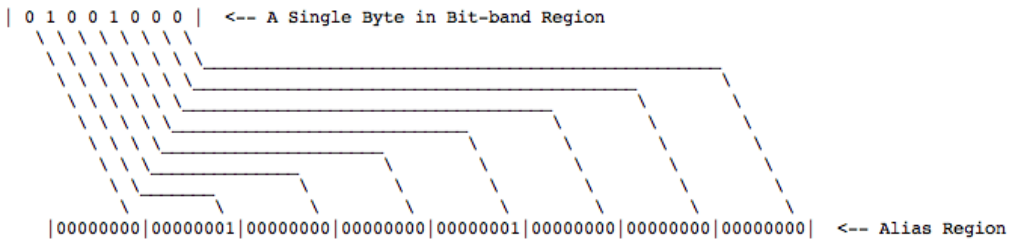


Figure 6

To use this feature, you must first get the address of the word in the alias region that corresponds to the bit you wish to read/write.

Bit banding

First, you need to know the following formula to calculate each bit (from bit-band region) alias address. This formula is adapted from Cortex-M3 technical reference manual:

- bit_word_offset = (byte_offset x 32) + (bit_number $\times$ 4)
- bit_word_addr = bit_band_base + bit_word_offset

Where:

- bit_word_offset is the position of the target bit in the bit-band memory region.
- bit_word_addr is the address of the word in the alias memory region that maps to the targeted bit.
- bit_band_base is the starting address of the alias region.
- byte_offset is the number of the byte in the bit-band region that contains the targeted bit.
- bit_number is the bit position (0-7) of the targeted bit.

Example

- Original start address of the stored variable: 0x20000004
- Base address of that bit band region (in SRAM): 0x20000000
- Base address of that bit band alias region (in SRAM): 0x22000000
- Total offset of stored variable from bit band region: 0x4
- Alias formula: (total offset * 0x20)+(number of bit you want to manipulate *4)= (0x4 * 0x20)+(3*4)=0x14
- Bit band alias memory location: 0x22000000+0x14=0x22000014
- So use 0x22000014 to manipulate the 3rd bit of memory location 0x20000004

Memory Access Permission

- Cortex-M4 memory map has a default configuration for memory access permissions.
- It prevents user programs (non-privileged) from accessing system control memory spaces such as the NVIC.
- The default memory access permission is used when either no MPU is present or the MPU is present but disabled.
- MPU is present and enabled: additional access permission rules defined by the MPU setup will determine whether user accesses are allowed for other memory regions.
- When an unprivileged access is blocked, the fault exception takes place immediately.

Memory Access Permission

Table 6.10 Default Memory Access Permissions

Memory Region	Address	Access in Unprivileged (User) Program
Vendor specific	0xE0100000–0xFFFFFFFF	Full access
ROM Table	0xE00FF000–0xE00FFFFF	Blocked; unprivileged access results in bus fault
External PPB	0xE0042000–0xE00FEFFF	Blocked; unprivileged access results in bus fault
ETM	0xE0041000–0xE0041FFF	Blocked; unprivileged access results in bus fault
TPIU	0xE0040000–0xE0040FFF	Blocked; unprivileged access results in bus fault
Internal PPB	0xE000F000–0xE003FFFF	Blocked; unprivileged access results in bus fault
NVIC	0xE000E000–0xE000EFFF	Blocked; unprivileged access results in bus fault, except Software Trigger Interrupt Register that can be programmed to allow user accesses
FPB	0xE0002000–0xE0003FFF	Blocked; unprivileged access results in bus fault
DWT	0xE0001000–0xE0001FFF	Blocked; unprivileged access results in bus fault
ITM	0xE0000000–0xE0000FFF	Read allows; write ignored except for stimulus ports with unprivileged access enabled (run-time configurable)
External Device	0xA0000000–0xDFFFFFFF	Full access
External RAM	0x60000000–0x9FFFFFFF	Full access
Peripheral	0x40000000–0x5FFFFFFF	Full access
SRAM	0x20000000–0x3FFFFFFF	Full access
Code	0x00000000–0x1FFFFFFF	Full access

Figure 7

System without Embedded OS

The MPU is programmed to have a static configuration. The configuration can be used for functions like:

- Setting a RAM/SRAM region to be read-only to protect important data from accidental corruption
- Making a portion of RAM/SRAM space at the bottom of the stack inaccessible to detect stack overflow
- Setting a RAM/SRAM region to be XN to prevent code injection attacks
- Defining memory attribute settings that can be used by system level cache (level 2) or the memory controllers

System with Embedded OS

The MPU can be programmed at each context switch so that each application task can have a different MPU configuration.

- Define memory access permissions so that stack operations of an application task can only access their own allocated stack space, thus preventing stack corruptions
- Define memory access permissions so that an application task can only have access to a limited set of peripherals
- Define memory access permissions so that an application task can only access its own data, or access its own program data

MPU Regions

- The MPU divides the memory map into a number of regions, and defines the location, size, access permissions, and memory attributes of each region. It supports:
  - independent attribute settings for each region
  - overlapping regions
  - export of memory attributes to the system.
- The memory attributes affect the behavior of memory accesses to the region. The Cortex-M4 MPU defines:
  - eight separate memory regions, 0-7
  - a background region.

MPU Regions

- When memory regions overlap, a memory access is affected by the attributes of the region with the highest number.
- For example, the attributes for region 7 take precedence over the attributes of any region that overlaps region 7.
- The background region has the same memory access attributes as the default memory map, but is accessible from privileged software only.
- The Cortex-M4 MPU memory map is unified. This means instruction accesses and data accesses have same region settings.
- If a program accesses a memory location that is prohibited by the MPU, the processor generates a MemManage fault.

MPU Registers

Name of register	Type	Address
MPU Type Register	Read Only	0xE000ED90
MPU Control Register	Read/Write	0xE000ED94
MPU Region Number register	Read/Write	0xE000ED98
MPU Region Base Address register	Read/Write	0xE000ED9C
MPU Region Attribute and Size register(s)	Read/Write	0xE000EDA0
MPU Alias 1 Region Base Address register	Alias of D9C	0xE000EDA4
MPU Alias 1 Region Attribute and Size register	Alias of DA0	0xE000EDA8
MPU Alias 2 Region Base Address register	Alias of D9C	0xE000EDAC
MPU Alias 2 Region Attribute and Size register	Alias of DA0	0xE000EDB0
MPU Alias 3 Region Base Address register	Alias of D9C	0xE000EDB4
MPU Alias 3 Region Attribute and Size register	Alias of DA0	0xE000EDB8

Figure 8

MPU Type Register

The MPU Type register indicates whether the MPU is present, and if so, how many regions it supports.

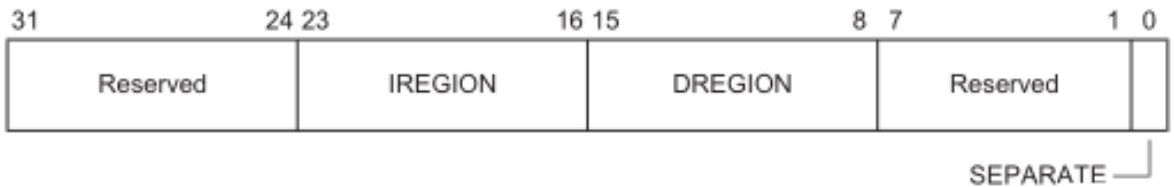


Figure 9

MPU Type Register

Bits	Field	Function
[31:24]	-	Reserved.
[23:16]	IREGION	Because the processor core uses only a unified MPU, IREGION always contains `0x00`.
[15:8]	DREGION	Number of supported MPU regions field. DREGION contains `0x08` if the implementation contains an MPU indicating eight MPU regions, otherwise it contains `0x00`.
[7:0]	-	Reserved.
[0]	SEPARATE	Because the processor core uses only a unified MPU, SEPARATE is always 0.

Figure 10

MPU Control Register

The MPU Control register:

- enables the MPU
- enables the default memory map background region
- enables use of the MPU when in the hard fault, Non-maskable Interrupt (NMI), and FAULTMASK escalated handlers.

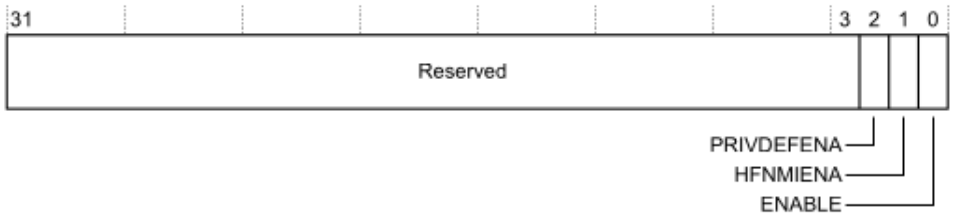


Figure 11

MPU Control Register

Bits	Name	Function
[31:3]	-	Reserved.
[2]	PRIVDEFENA	Enables privileged software access to the default memory map: 0 = If the MPU is enabled, disables use of the default memory map. Any memory access to a location not covered by any enabled region causes a fault. 1 = If the MPU is enabled, enables use of the default memory map as a background region for privileged software accesses. When enabled, the background region acts as if it is region number -1. Any region that is defined and enabled has priority over this default map. If the MPU is disabled, the processor ignores this bit.
[1]	HFNMIENA	Enables the operation of MPU during hard fault, NMI, and FAULTMASK handlers. When the MPU is enabled: 0 = MPU is disabled during hard fault, NMI, and FAULTMASK handlers, regardless of the value of the ENABLE bit 1 = the MPU is enabled during hard fault, NMI, and FAULTMASK handlers. When the MPU is disabled, if this bit is set to 1 the behavior is Unpredictable.
[0]	ENABLE	Enables the MPU: 0 = MPU disabled 1 = MPU enabled.

Figure 12

MPU Region Number Register

The MPU_RNR selects which memory region is referenced by the MPU_RBAR and MPU_RASR registers.

Table 4-41 MPU_RNR bit assignments

Bits	Name	Function
[31:8]	-	Reserved.
[7:0]	REGION	Indicates the MPU region referenced by the MPU_RBAR and MPU_RASR registers. The MPU supports 8 memory regions, so the permitted values of this field are 0-7.

Figure 13

MPU Region Base Address Register

The MPU_RBAR defines the base address of the MPU region selected by the MPU_RNR, and can update the value of the MPU_RNR.

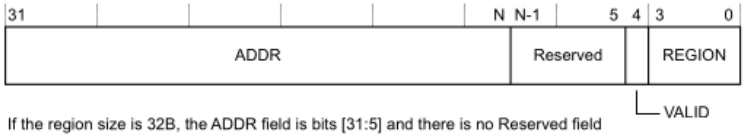


Table 4-42 MPU_RBAR bit assignments

Bits	Name	Function
[31:N]	ADDR	Region base address field. The value of N depends on the region size. For more information see *The ADDR field*.
[(N-1):5]	-	Reserved.
[4]	VALID	MPU Region Number valid bit: Write: 0 = MPU_RNR not changed, and the processor: • updates the base address for the region specified in the MPU_RNR • ignores the value of the REGION field 1 = the processor: • updates the value of the MPU_RNR to the value of the REGION field • updates the base address for the region specified in the REGION field. Always reads as zero.
[3:0]	REGION	MPU region field: For the behavior on writes, see the description of the VALID field. On reads, returns the current region number, as specified by the RNR.

Figure 14

MPU Region Attribute and Size Register

The MPU_RASR defines the region size and memory attributes of the MPU region specified by the MPU_RNR, and enables that region and any subregions.

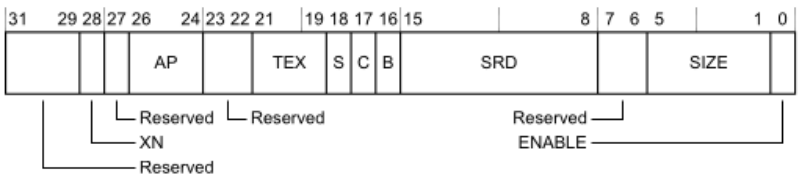


Figure 15

MPU Region Attribute and Size Register

The MPU_RASR defines the region size and memory attributes of the MPU region specified by the MPU_RNR, and enables that region and any subregions.

Bits	Name	Function
[31:29]	-	Reserved.
[28]	XN	Instruction access disable bit: 0 = instruction fetches enabled 1 = instruction fetches disabled.
[27]	-	Reserved.
[26:24]	AP	Access permission field, see Table 4-47 on page 4-44.
[23:22]	-	Reserved.
[21:19, 17, 16]	TEX, C, B	Memory access attributes, see Table 4-45 on page 4-43.
[18]	S	Shareable bit, see Table 4-45 on page 4-43.
[15:8]	SRD	Subregion disable bits. For each bit in this field: 0 = corresponding sub-region is enabled 1 = corresponding sub-region is disabled See *Subregions* on page 4-46 for more information. Region sizes of 128 bytes and less do not support subregions. When writing the attributes for such a region, write the SRD field as `0x00`.
[7:6]	-	Reserved.
[5:1]	SIZE	Specifies the size of the MPU protection region. The minimum permitted value is 3 (`0b00010`). See *SIZE field values* for more information.
[0]	ENABLE	Region enable bit.

Figure 16

Setting up the MPU

Before using the MPU, you need to work out what memory regions the program or application tasks need to (and are allowed to) access:

- Program code for privileged applications including handlers and OS kernel, typically privileged accesses only
- Data memory including stack for privileged applications including handlers and OS kernels, typically privileged accesses only
- Program code for unprivileged applications (application tasks), full access
- Data memory including stack for unprivileged applications (application tasks), full accesses
- Peripherals that are for privileged applications including handlers and OS kernel, privileged accesses only
- Peripherals that can be used by unprivileged applications (application tasks), full accesses
- When defining the address and size of the memory region, be aware that the base address of a region must be aligned to an integer multiple value of the region size.

If the goal for using the MPU is to prevent unprivileged tasks from accessing certain memory regions, the background region feature is very useful as it reduces the setup required. You only need to set up the region setting for unprivileged tasks, and privileged tasks and handlers have full access to other memory spaces using the background region.