

Cortex[™]-M3

Revision: r1p1

Technical Reference Manual



Chapter 9

Memory Protection Unit

This chapter describes the processor *Memory Protection Unit* (MPU). It contains the following sections:

- *About the MPU* on page 9-2
- *MPU programmer's model* on page 9-3
- *Interrupts and updating the MPU* on page 9-19
- *MPU access permissions* on page 9-13
- *MPU aborts* on page 9-15
- *Updating an MPU region* on page 9-16.

9.1 About the MPU

The MPU is a component for memory protection. The processor supports the standard ARMv7 *Protected Memory System Architecture* (PMSAv7) model. The MPU provides full support for:

- protection regions
- overlapping protection regions
- access permissions
- exporting memory attributes to the system.

MPU mismatches and permission violations invoke the programmable-priority MemManage fault handler. For more information, see *Memory Manage Fault Address Register* on page 8-38.

You can use the MPU to:

- enforce privilege rules
- separate processes
- enforce access rules.

9.2 MPU programmer's model

This sections describes the registers that control the MPU. It contains the following:

- *Summary of the MPU registers*
- *Description of the MPU registers.*

9.2.1 Summary of the MPU registers

Table 9-1 provides a summary of the MPU registers.

Table 9-1 MPU registers

Name of register	Type	Address	Reset value	Page
MPU Type Register	Read Only	0xE000ED90	0x00000800	page 9-3
MPU Control Register	Read/Write	0xE000ED94	0x00000000	page 9-4
MPU Region Number register	Read/Write	0xE000ED98	-	page 9-6
MPU Region Base Address register	Read/Write	0xE000ED9C	-	page 9-7
MPU Region Attribute and Size register(s)	Read/Write	0xE000EDA0	-	page 9-8
MPU Alias 1 Region Base Address register	Alias of D9C	0xE000EDA4	-	page 9-11
MPU Alias 1 Region Attribute and Size register	Alias of DA0	0xE000EDA8	-	page 9-11
MPU Alias 2 Region Base Address register	Alias of D9C	0xE000EDAC	-	page 9-11
MPU Alias 2 Region Attribute and Size register	Alias of DA0	0xE000EDB0	-	page 9-11
MPU Alias 3 Region Base Address register	Alias of D9C	0xE000EDB4	-	page 9-11
MPU Alias 3 Region Attribute and Size register	Alias of DA0	0xE000EDB8	-	page 9-11

9.2.2 Description of the MPU registers

This section contains a description of the MPU registers.

MPU Type Register

Use the MPU Type Register to see how many regions the MPU supports. Read bits [15:8] to determine if an MPU is present.

The register address, access type, and Reset state are:

Address 0xE000ED90

Access Read-only

Reset state 0x00000800

Figure 9-1 shows the fields of the MPU Type Register.

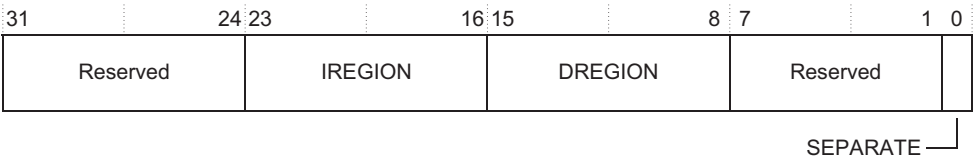


Figure 9-1 MPU Type Register bit assignments

Table 9-2 describes the fields of the MPU Type Register.

Table 9-2 MPU Type Register bit assignments

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.

MPU Control Register

Use the MPU Control Register to:

- enable the MPU
- enable the default memory map (background region)
- enable the MPU when in Hard Fault, *Non-maskable Interrupt* (NMI), and FAULTMASK escalated handlers.

When the MPU is enabled, at least one region of the memory map must be enabled for the MPU to function unless the PRIVDEFENA bit is set. If the PRIVDEFENA bit is set and no regions are enabled, then only privileged code can operate.

When the MPU is disabled, the default address map is used, as if no MPU is present.

When the MPU is enabled, only the system partition and vector table loads are always accessible. Other areas are accessible based on regions and whether PRIVDEFENA is enabled.

Unless HFNMIENA is set, the MPU is not enabled when the exception priority is –1 or –2. These priorities are only possible when in Hard fault, NMI, or when FAULTMASK is enabled. The HFNMIENA bit enables the MPU when operating with these two priorities.

The register address, access type, and Reset state are:

Address 0xE000ED94

Access Read/write

Reset state 0x00000000

Figure 9-2 shows the fields of the MPU Control Register.

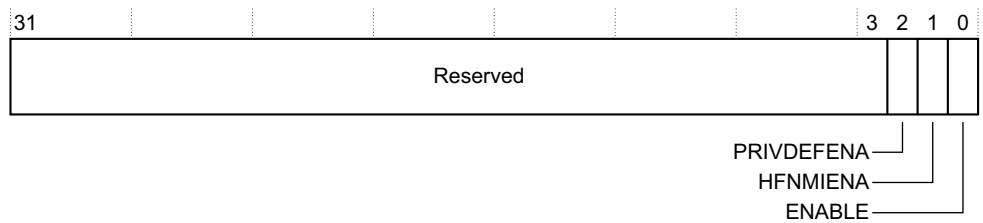


Figure 9-2 MPU Control Register bit assignments

Table 9-3 describes the fields of the MPU Control Register.

Table 9-3 MPU Control Register bit assignments

Bits	Field	Function
[31:2]	-	Reserved.
[2]	PRIVDEFENA	<p>This bit enables the default memory map for privileged access, as a background region, when the MPU is enabled. The background region acts as if it was region number 1 before any settable regions. Any region that is set up overlays this default map, and overrides it.</p> <p>If this bit = 0, the default memory map is disabled, and memory not covered by a region faults.</p> <p>When the MPU is enabled and PRIVDEFENA is enabled, the default memory map is as described in Chapter 4 <i>Memory Map</i>. This applies to memory type, <i>Execute Never</i> (XN), cache and shareable rules. However, this only applies to privileged mode (fetch and data access). User mode code faults unless a region has been set up for its code and data.</p> <p>When the MPU is disabled, the default map acts on both privileged and user mode code. XN and SO rules always apply to the System partition whether this enable is set or not. If the MPU is disabled, this bit is ignored.</p> <p>Reset clears the PRIVDEFENA bit.</p>
[1]	HFNMIENA	<p>This bit enables the MPU when in Hard Fault, NMI, and FAULTMASK escalated handlers. If this bit = 1 and the ENABLE bit = 1, the MPU is enabled when in these handlers. If this bit = 0, the MPU is disabled when in these handlers, regardless of the value of ENABLE. If this bit =1 and ENABLE = 0, behavior is Unpredictable.</p> <p>Reset clears the HFNMIENA bit.</p>
[0]	ENABLE	<p>MPU enable bit:</p> <p>1 = enable MPU</p> <p>0 = disable MPU.</p> <p>Reset clears the ENABLE bit.</p>

MPU Region Number Register

Use the MPU Region Number Register to select which protection region is accessed. Then write to the MPU Region Base Address Register or the MPU Attributes and Size Register to configure the characteristics of the protection region.

The register address, access type, and Reset state are:

- Address 0xE000ED98
- Access Read/write
- Reset state Unpredictable

Figure 9-3 on page 9-7 shows the fields of the MPU Region Number Register.

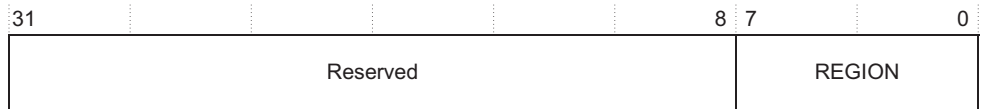
**Figure 9-3 MPU Region Number Register bit assignments**

Table 9-4 describes the fields of the MPU Region Number Register.

Table 9-4 MPU Region Number Register bit assignments

Bits	Field	Function
[31:8]	-	Reserved.
[7:0]	REGION	Region select field. Selects the region to operate on when using the Region Attribute and Size Register and the Region Base Address Register. It must be written first except when the address VALID + REGION fields are written, which overwrites this.

MPU Region Base Address Register

Use the MPU Region Base Address Register to write the base address of a region. The Region Base Address Register also contains a REGION field that you can use to override the REGION field in the MPU Region Number Register, if the VALID bit is set.

The Region Base Address register sets the base for the region. It is aligned by the size. So, a 64-KB sized region must be aligned on a multiple of 64KB, for example, 0x00010000 or 0x00020000.

The region always reads back as the current MPU region number. VALID always reads back as 0. Writing with VALID = 1 and REGION = n changes the region number to n. This is a short-hand way to write the MPU Region Number Register.

This register is Unpredictable if accessed other than as a word.

The register address, access type, and Reset state are:

Address 0xE00ED9C
Access Read/write
Reset state Unpredictable

Figure 9-4 on page 9-8 shows the fields of the MPU Region Base Address Register.

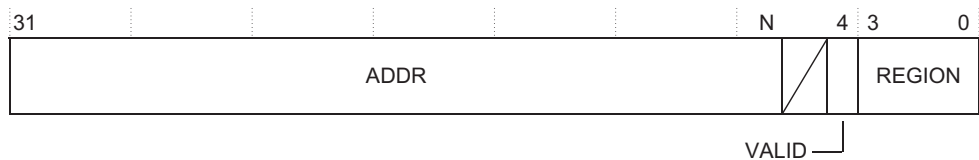


Figure 9-4 MPU Region Base Address Register bit assignments

Table 9-5 describes the fields of the MPU Region Base Address Register.

Table 9-5 MPU Region Base Address Register bit assignments

Bits	Field	Function
[31:N]	ADDR	Region base address field. The value of N depends on the region size, so that the base address is aligned according to an even multiple of size. The power of 2 size specified by the SZENABLE field of the MPU Region Attribute and Size Register defines how many bits of base address are used.
[4]	VALID	MPU Region Number valid bit: 1 = MPU Region Number Register is overwritten by bits 3:0 (the REGION value). 0 = MPU Region Number Register remains unchanged and is interpreted.
[3:0]	REGION	MPU region override field.

MPU Region Attribute and Size Register

Use the MPU Region Attribute and Size Register to control the MPU access permissions. The register is made up of two part registers, each of halfword size. These can be accessed using the individual size, or they can both be simultaneously accessed using a word operation.

The sub-region disable bits are Unpredictable for region sizes of 32 bytes, 64 bytes, and 128 bytes.

The register address, access type, and Reset state are:

Address 0xE000EDA0
Access Read/write
Reset state Unpredictable

Figure 9-5 on page 9-9 shows the fields of the MPU Region Attribute and Size Register.

31	29	28	27	26	24	23	22	21	19	18	17	16	15				8	7	6	5				1	0
Res	XN	Res	AP	Res.	TEX	S	C	B	SRD							Res.	SIZE					EN A			

Figure 9-5 MPU Region Attribute and Size Register bit assignments

Table 9-6 describes the fields of the MPU Region Attribute and Size Register. For more information, see *MPU access permissions* on page 9-13.

Table 9-6 MPU Region Attribute and Size Register bit assignments

Bits	Field	Function	
[31:29]	-	Reserved.	
[28]	XN	Instruction access disable bit: 1 = disable instruction fetches 0 = enable instruction fetches.	
[27]	-	Reserved.	
[26:24]	AP	Data access permission field:	
	Value	Privileged permissions	User permissions
	b000	No access	No access
	b001	Read/write	No access
	b010	Read/write	Read-only
	b011	Read/write	Read/write
	b100	Reserved	Reserved
	b101	Read-only	No access
	b110	Read-only	Read-only
	b111	Read-only.	Read-only.
[23:22]	-	Reserved.	
[21:19]	TEX	Type extension field.	
[18]	S	Shareable bit: 1 = shareable 0 = not shareable.	
[17]	C	Cacheable bit: 1 = cacheable 0 = not cacheable.	

Table 9-6 MPU Region Attribute and Size Register bit assignments (continued)

Bits	Field	Function
[16]	B	Bufferable bit: 1 = bufferable 0 = not bufferable.
[15:8]	SRD	<i>Sub-Region Disable</i> (SRD) field. Setting an SRD bit disables the corresponding sub-region. Regions are split into eight equal-sized sub-regions. Sub-regions are not supported for region sizes of 128 bytes and less. For more information, see <i>Sub-Regions</i> on page 9-12.
[7:6]	-	Reserved.
[5:1]	SIZE	MPU Protection Region Size Field. See Table 9-7.
[0]	ENABLE	Region enable bit.

For information about access permission, see *MPU access permissions* on page 9-13.

Table 9-7 MPU protection region size field

Region	Size
b00000	Reserved
b00001	Reserved
b00010	Reserved
b00011	Reserved
b00100	32B
b00101	64B
b00110	128B
b00111	256B
b01000	512B
b01001	1KB
b01010	2KB
b01011	4KB
b01100	8KB
b01101	16KB

Table 9-7 MPU protection region size field (continued)

Region	Size
b01110	32KB
b01111	64KB
b10000	128KB
b10001	256KB
b10010	512KB
b10011	1MB
b10100	2MB
b10101	4MB
b10110	8MB
b10111	16MB
b11000	32MB
b11001	64MB
b11010	128MB
b11011	256MB
b11100	512MB
b11101	1GB
b11110	2GB
b11111	4GB

9.2.3 Accessing the MPU using the alias registers

You can optimize the loading speed of the MPU registers using register aliasing. There are three sets of *Nested Vectored Interrupt Controller* (NVIC) alias registers. These are described in *NVIC register descriptions* on page 8-7.

The aliases access the registers in exactly the same way, and they exist to enable the use of sequential writes (STM) to update between one and four regions. This is used when disable/change/enable is not required.

You cannot use these aliases to read the contents of the regions because the region number must be written.

An example code sequence for updating four regions is

```
; R1 = 4 region pairs from process control block (8 words)
MOV R0, #NVIC_BASE
ADD R0, #MPU_REG_CTRL
LDM R1, [R2-R9] ; load region information for 4 regions
STM R0, [R2-R9] ; update all 4 regions at once
```

Note

You can normally use the `memcpy()` function in a C/C++ compiler for this sequence. However, you must verify that the compiler uses word transfers.

9.2.4 Sub-Regions

The eight *Sub-Region Disable* (SRD) bits of the Region Attribute and Size Register divide a region into eight equal-sized units based on the region size. This enables selectively disabling some of the 1/8th sub-regions. The least significant bit affects the first 1/8th sub-region, and the most significant bits affects the last 1/8th sub-region. A disabled sub-region enables any other region overlapping that range to be matched instead. If no other region overlaps the sub-region, the default behavior is used, no match – a fault. Sub-regions cannot be used with the three smallest regions of size: 32, 64, and 128. If these sub-regions are used, the results are Unpredictable.

Example of SRD use

Two regions with the same base address overlap. One region is 64KB, and the other is 512KB. The bottom 64KB of the 512KB region is disabled so that the attributes from the 64KB apply. This is achieved by setting SRD for the 512KB region to b11111110.

9.3 MPU access permissions

This section describes the MPU access permissions. The access permission bits, TEX, C, B, AP, and XN, of the Region Access Control Register (see *MPU Region Attribute and Size Register* on page 9-8) control access to the corresponding memory region. If an access is made to an area of memory without the required permissions, then a permission fault is raised.

Table 9-8 describes the TEX, C, and B encoding.

Table 9-8 TEX, C, B encoding

TEX	C	B	Description	Memory type	Region shareability
b000	0	0	Strongly ordered.	Strongly ordered	Shareable
b000	0	1	Shared device.	Device	Shareable
b000	1	0	Outer and inner write-through. No write allocate.	Normal	S
b000	1	1	Outer and inner write-back. No write allocate.	Normal	S
b001	0	0	Outer and inner noncacheable.	Normal	S
b001	0	1	Reserved.	Reserved	Reserved
b001	1	0	Implementation-defined.		
b001	1	1	Outer and inner write-back. Write and read allocate.	Normal	S
b010	0	0	Nonshared device.	Device	Not shareable
b010	0	1	Reserved.	Reserved	Reserved
b010	1	X	Reserved.	Reserved	Reserved
b1BB	A	A	Cached memory BB = outer policy. AA = inner policy.	Normal	S

Note

In Table 9-8, S is the S bit [2] from the MPU Region Attributes and Size Register.

Table 9-9 describes the cache policy for memory attribute encoding.

Table 9-9 Cache policy for memory attribute encoding

Memory attribute encoding (AA and BB)	Cache policy
00	Non-cacheable
01	Write back, write and read allocate
10	Write through, no write allocate
11	Write back, no write allocate

Note

All cache policies presented by **HPROT** and **MEMATTR** relate to an outer cache.

Table 9-10 describes the AP encoding.

Table 9-10 AP encoding

AP[2:0]	Privileged permissions	User permissions	Descriptions
000	No access	No access	All accesses generate a permission fault
001	Read/write	No access	Privileged access only
010	Read/write	Read only	Writes in user mode generate a permission fault
011	Read/write	Read/write	Full access
100	Unpredictable	Unpredictable	Reserved
101	Read only	No access	Privileged read only
110	Read only	Read only	Privileged/user read only
111	Read only	Read only	Privileged/user read only

Table 9-11 describes the XN encoding.

Table 9-11 XN encoding

XN	Description
0	All instruction fetches enabled
1	No instruction fetches enabled

9.4 MPU aborts

For information about MPU aborts, see *Memory Manage Fault Address Register* on page 8-38.

9.5 Updating an MPU region

There are three registers consisting of three memory mapped words that program the MPU regions. These are part registers that you can individually program and access. This means that you can port existing ARMv6, ARMv7, and CP15 code. This replaces MRC and MCR with LDRx and STRx operations.

You can also access these registers as three words, and program them using only two words. Aliases are provided to enable programming a set of regions simultaneously using an STM instruction.

9.5.1 Updating an MPU region using CP15 equivalent code

Using CP15 equivalent code:

```
; R1 = region number
; R2 = size/enable
; R3 = attributes
; R4 = address
MOV R0,#NVIC_BASE
ADD R0,#MPU_REG_CTRL
STR R1,[R0,#0]; region number
STR R4,[R0,#4]; address
STRHR2,[R0,#8]; size and enable
STRHR3,[R0,#10]; attributes
```

Note

If interrupts could pre-empt during this period, this region could affect them. This means that the region must be disabled, written, and then enabled. This is usually not necessary for a context switcher, but would be necessary if updated elsewhere.

```
; R1 = region number
; R2 = size/enable
; R3 = attributes
; R4 = address
MOV R0,#NVIC_BASE
ADD R0,#MPU_REG_CTRL
STR R1,[R0,#0]; region number
BIC R2,R2, #1; disable
STRHR2,[R0,#8]; size and enable
STR R4,[R0,#4]; address
STRHR3,[R0,#10]; attributes
ORR R2,#1 ; enable
STRHR2,[R0,#8]; size and enable
```

DMB/DSB is not necessary because the Private Peripheral Bus is a strongly ordered memory area. However, a DSB is necessary before the effect on the MPU takes place, such as the end of a context switcher.

An ISB is necessary if the code that programs the MPU region or regions is entered using a branch or call. If the code is entered using a return from exception, or by taking an exception, then an ISB is not necessary.

9.5.2 Updating an MPU region using two or three words

You can program directly using two or three words, depending on how the information is divided:

```
; R1 = region number
; R2 = address
; R3 = size, attributes in one
MOV R0,#NVIC_BASE
ADD R0,#MPU_REG_CTRL
STR R1,[R0,#0]; region number
STR R2,[R0,#4]; address
STR R3,[R0,#8]; size, attributes
```

An STM can optimize this:

```
; R1 = region number
; R2 = address
; R3 = size, attributes in one
MOV R0,#NVIC_BASE
ADD R0,#MPU_REG_CTRL
STM R0,{R1-R3}; region number, address, size, and attributes
```

You can do this in two words for pre-packed information. This means that the base address register contains the region number in addition to a region-valid bit. This is useful when the data is statically packed, for example in a boot list or a *Process Control Block* (PCB).

```
; R1 = address and region number in one
; R2 = size and attributes in one
MOV R0,#NVIC_BASE
ADD R0,#MPU_REG_CTRL
STR R1,[R0,#4]; address and region number
STR R2,[R0,#8]; size and attributes
```

An STM can optimize this:

```
; R1 = address and region number in one
; R2 = size and attributes in one
MOV R0,#NVIC_BASE
ADD R0,#MPU_REG_CTRL
```

STM R0,{R1-R2}; address, region number, size

For information about interrupts and updating the MPU, see *Interrupts and updating the MPU* on page 9-19.

9.6 Interrupts and updating the MPU

An MPU region can contain critical data. This is because it takes more than one bus transaction to update. This is normally two words. As a result, it is not thread safe. That is, an interrupt can split the two words, leaving the region with incoherent information. There are two different issues:

- An interrupt can come in that would also update the MPU. This is not only a read-modify-write issue, it also affects cases where the interrupt routine is guaranteed not to modify the same region. This is because the programming relies on the region number being written into a register so that it knows which region to update. So in this case, you must disable interrupts around each update routine.
- An interrupt can come in that would use the region being updated or would be affected because only the base or size fields had been updated. If the new size field is changed, but the base is not, the `base+new_size` might overlap into an area normally handled by another region. In this case, the disable-modify-enable approach is required.

But for standard OS context switch code, which would change user regions, there is no risk, because these regions would be preset to user privilege and a user area address. This means that even an interrupt would cause no side effect. Therefore the disable/enable code is not required nor is interrupt disable.

The most common approach is to only program the MPU from boot code and context switcher. If these are the only two places, and the context switcher is only updating user regions, then disable is not required because the context switcher is already a critical region and the boot code runs with interrupts disabled.