Introduction to memory protection unit management on STM32 MCUs

Introduction

This application note describes how to manage the memory protection unit (MPU) in the STM32 products.

The MPU is an optional component for the memory protection. Including the MPU in the STM32 microcontrollers (MCUs) makes them more robust and reliable. The MPU must be programmed and enabled before using it. If the MPU is not enabled, there is no change in the memory system behavior.

This application note concerns all the STM32 products listed in Table 1 that include the Cortex®-M0+/M3/M4/M7 and Cortex®-M33/M55 design that supports the MPU.

For more details about the MPU, refer to the following documents available on www.st.com

- Programming manual *STM32F7 series and STM32H7 series Cortex®-M7 processor* (PM0253)
- Programming manual *STM32F10xxx/20xxx/21xxx/L1xxxx Cortex®-M3* (PM0056)
- Programming manual *STM32 Cortex®-M0+ MCUs programming manual* (PM0223)
- Programming manual *STM32 Cortex®-M4 MCUs and MPUs* (PM0214)
- Programming manual *STM32 Cortex®-M33 MCUs* (PM0264)
- Programming manual *STM32 Cortex®-M55 MCUs* (PM0273)

Table 1. Applicable products

Type	Product series
Microcontrollers	• STM32C0 series • STM32F1 series, STM32F2 series, STM32F3 series, STM32F4 series, STM32F7 series • STM32G0 series, STM32G4 series • STM32H5 series, STM32H7 series • STM32L0 series, STM32L1 series, STM32L4 series, STM32L4+ series, STM32L5 series • STM32U0 series, STM32U3 series, STM32U5 series • STM32WB series, STM32WB0 series • STM32N6 series

1 General information

This application note applies to STM32 microcontrollers Arm®-based devices.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

2 Overview

The MPU can be used to make an embedded system more robust and more secure by:

- prohibiting the user applications from corrupting data used by critical tasks (such as the operating system kernel)
- defining the SRAM memory region as a non-executable (execute never XN) to prevent code injection attacks
- changing the memory access attributes

The MPU can be used to protect up to 16 memory regions. In Armv6 and Armv7 architecture (Cortex-M0+, M3, M4, and M7, these regions in turn can have eight subregions, if the region is at least 256 bytes. The exact amount of regions protected can vary between core and devices in the STM32, refer to Table 6 for more details. The subregions are always of equal size, and can be enabled or disabled by a subregion number. Because the minimum region size is driven by the cache line length (32 bytes), eight subregions of 32 bytes correspond to a 256-byte size.

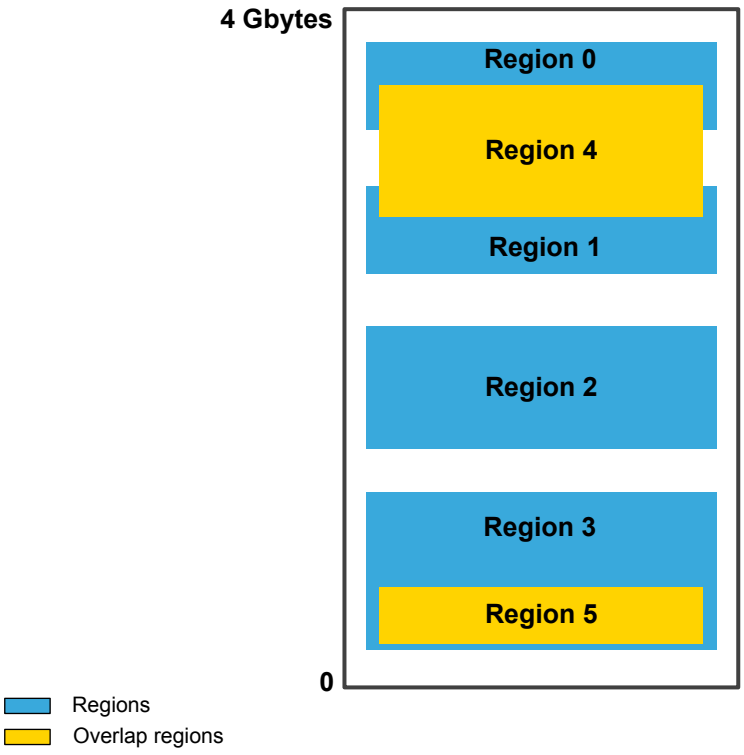
The regions are numbered 0 to 15. In addition, there is a region called the default region with an id of -1. All the 0-15 memory regions take priority over the default region.

The regions can overlap, and can be nested. The region 15 has the highest priority and the region 0 has the lowest one and this governs how overlapping the regions behave. The priorities are fixed, and cannot be changed.

In Armv8 architecture (Cortex-M33 and M55) the regions are defined using a base and a limit address offering flexibility and simplicity to the developer on the way to organize them. Additionally, Cortex-M33 and M55 do not include subregions as the region size is now more flexible.

The figure below shows an example with six regions. This example shows the region 4 overlapping the regions 0 and 1. The region 5 is enclosed completely within the region 3. Since the priority is in an ascending order, the overlap regions (in orange) have the priority. So, if the region 0 is writeable and the region 4 is not, an address falling in the overlap between 0 and 4 is not writeable.

Figure 1. Example of overlapping regions



Caution: *In Armv8 architecture, regions are now not allowed to overlap. As the MPU region definition is much more flexible, overlapping MPU regions is not necessary.*

The MPU is unified, meaning that there are not separate regions for the data and the instructions.

The MPU can be used also to define other memory attributes such as the cacheability, which can be exported to the system level cache unit, or to the memory controllers. The memory attribute settings in Arm® architecture can support two levels of cache: inner cache and outer cache. For the STM32F7 and STM32H7 series, only one level of cache (L1-cache) is supported.

The cache control is done globally by the cache control register, but the MPU can specify the cache policy and whether the region is cacheable or not.

2.1 Memory model

In STM32 products, the processor has a fixed default memory map that provides up to 4 Gbytes of addressable memory.

Figure 2. Cortex-M0+/M3/M4/M7 processor memory map

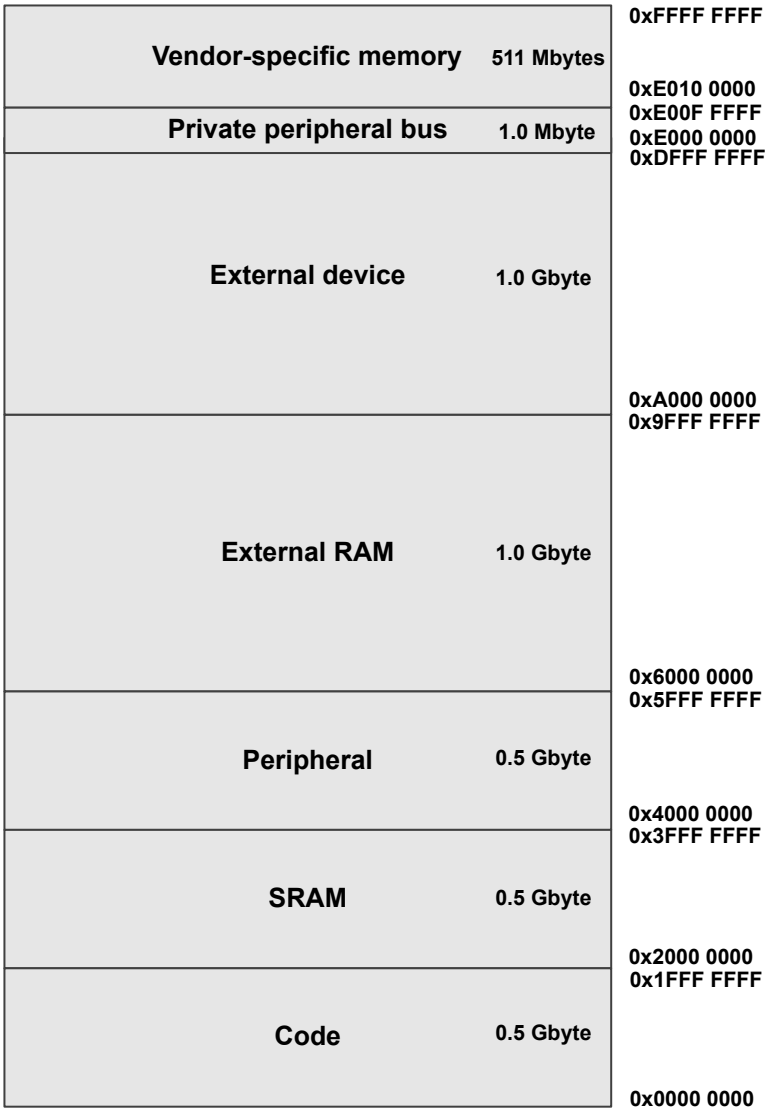


Figure 3. **Cortex-M33 processor memory map**

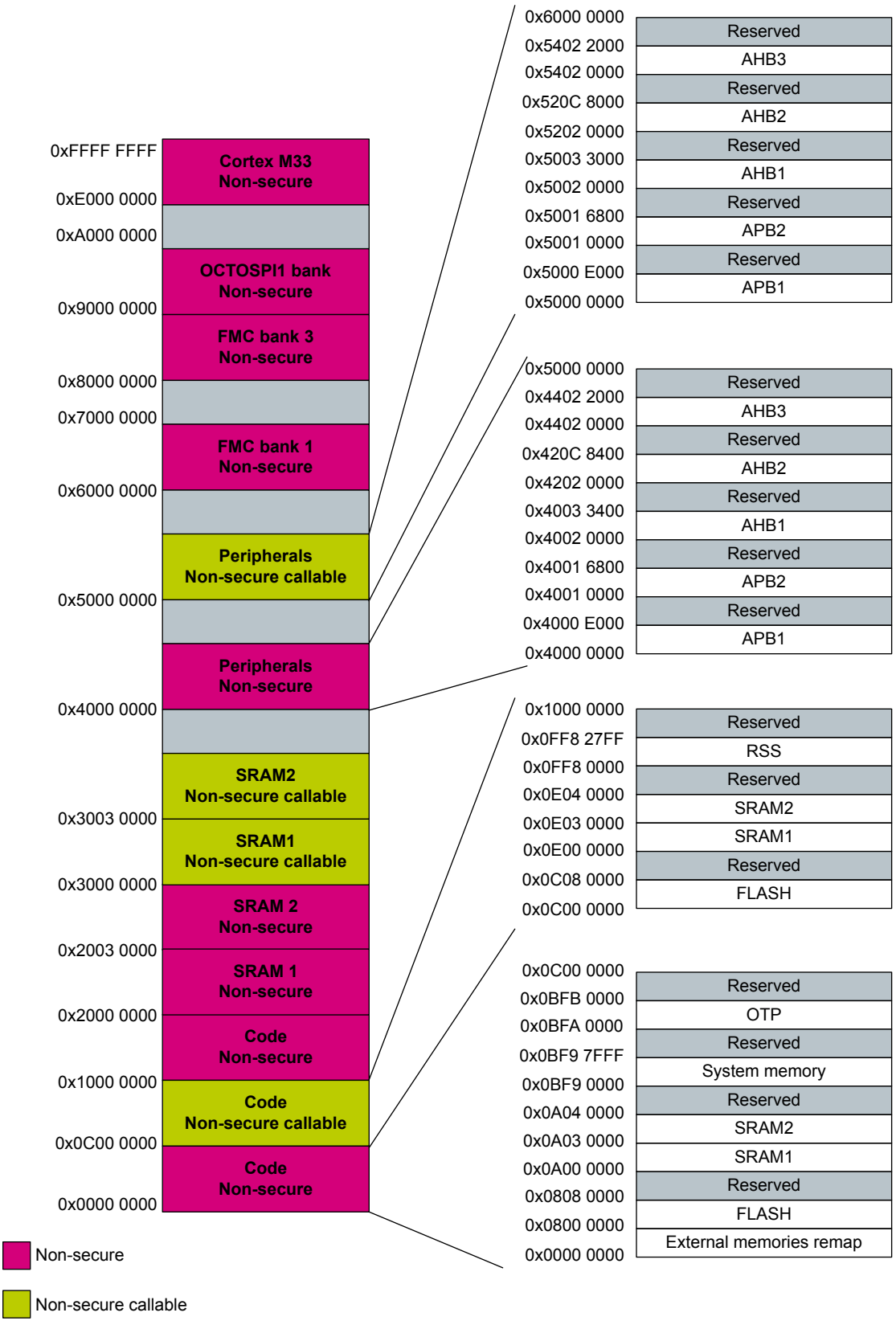


Table 2. Default memory map for the Cortex-M55 processor

Address range (inclusive)	Region	Interface
0x00000000-0x1FFFFFFF	Code	All accesses are performed on the instruction tightly coupled memory (ITCM) or manager-AXI (M-AXI) interface.
0x20000000-0x3FFFFFFF	SRAM	All accesses are performed on the data tightly coupled memory (DTCM) or M-AXI interface.
0x40000000-0x5FFFFFFF	Peripheral	Data accesses are performed on peripheral AHB (P-AHB) or M-AXI interface. Instruction accesses are performed on M-AXI.
0x60000000-0x9FFFFFFF	External RAM	All accesses are performed on the M-AXI interface
0xA0000000-0xDFFFFFFF	External device	All accesses are performed on the M-AXI interface.
0xE0000000-0xE00FFFFF	Private peripheral bus (PPB)	Instruction fetches are not supported. Reserved for system control and debug. Data accesses are either performed internally or on external private peripheral bus (EPPB).
0xE0100000-0xFFFFFFFF	Vendor_SYS	Instruction fetches are not supported. 0xE0100000-0xEFFFFFFF is reserved. Vendor resources start at 0xF0000000. Data accesses are performed on P-AHB interface.

3 Cortex-M0+/M3/M4/M7 memory types, registers and attributes

The memory map and the programming of the MPU split the memory map into regions. Each region has a defined memory type, and memory attributes. The memory type and attributes determine the behavior of accesses to the region.

3.1 Memory types

There are three common memory types:

- Normal memory: allows the load and store of bytes, half-words, and words to be arranged by the CPU in an efficient manner (the compiler is not aware of memory region types). For the normal memory region, the load/store is not necessarily performed by the CPU in the order listed in the program.
- Device memory: within the device region, the loads and stores are done strictly in order. This is to ensure that the registers are set in the proper order.
- Strongly ordered memory: everything is always done in the programmatically listed order, where the CPU waits the end of load/store instruction execution (effective bus access) before executing the next instruction in the program stream. This can cause a performance hit.

3.2 MPU register description

The MPU registers are located at 0xE000 ED90. There are five basic MPU registers and a number of alias registers for region. The following are used to set up regions in the MPU:

- MPU_TYPE: read-only register used to detect the MPU presence.
- MPU_CTRL: control register.
- MPU_RNR: region number, used to determine which region operations are applied to.
- MPU_RBAR: region base address.
- MPU_RASR: region attributes and size.
- MPU_RBAR_An: alias n of MPU_RBAR, where n is 1 to 3.
- MPU_RASR_An: alias n of MPU_RASR, where n is 1 to 3.

Note: The Cortex-M0+ does not implement the MPU_RBAR_An and MPU_RASR_An registers.

For more details about the MPU registers, refer to the programming manuals listed in Introduction.

3.3 Memory attributes

The region attributes and size register (MPU_RASR) are where all the memory attributes are set. The table shows a brief description of the region attributes and size in the MPU_RASR register.

Table 3. Region attributes and size in MPU_RASR register

Bits	Name	Description
28	XN	Execute never
26:24	AP	Data access permission field (RO, RW, or No access)
21:19	TEX	Type extension field
18	S	Shareable
17	C	Cacheable
16	B	Bufferable
15:8	SRD	Subregion disabled. For each subregion 1 = disabled, 0 = enabled.
5:1	SIZE	Specifies the size of the MPU protection region.

Parameters of the previous table are detailed below:

- The XN flag controls the code execution. In order to execute an instruction within the region, there must be read access for the privileged level, and XN must be 0. Otherwise, a MemManage fault is generated.

- The data access permission (AP) field defines the AP of memory region. The table below illustrates the access permissions:

Table 4. Access permissions of regions

AP[2:0]	Privileged permissions	Unprivileged permissions	Description
000	No access	No access	All accesses generate a permission fault
001	RW	No access	Access from a privileged software only
010	RW	RO	Written by an unprivileged software generates a permission fault
011	RW	RW	Full access
100	Unpredictable	Unpredictable	Reserved
101	RO	No access	Read by a privileged software only
110	RO	RO	Read only, by privileged or unprivileged software
111	RO	RO	Read only, by privileged or unprivileged software

- The S field is for a shareable memory region: the memory system provides data synchronization between bus masters in a system with multiple bus masters, for example, a processor with a DMA controller. A strongly-ordered memory is always shareable. If multiple bus masters can access a non-shareable memory region, the software must ensure the data coherency between the bus masters. The STM32F7 series and STM32H7 series do not support hardware coherency. The S field is equivalent to non-cacheable memory.
- The TEX, C and B bits are used to define cache properties for the region, and to some extent, its shareability. They are encoded as per the following table.

Table 5. Cache properties and shareability

TEX	C	B	Memory type	Description	Shareable
000	0	0	Strongly ordered	Strongly ordered	Yes
000	0	1	Device	Shared device	Yes
000	1	0	Normal	Write through, no write allocate	S bit
000	1	1	Normal	Write-back, no write allocate	S bit
001	0	0	Normal	Non-cacheable	S bit
001	0	1	Reserved	Reserved	Reserved
001	1	0	Undefined	Undefined	Undefined
001	1	1	Normal	Write-back, write and read allocate	S bit
010	0	0	Device	Non-shareable device	No
010	0	1	Reserved	Reserved	Reserved

- The subregion disable bits (SRD) flag whether a particular subregion is enabled or disabled. Disabling a subregion means that another region overlapping the disabled range matches instead. If no other enabled region overlaps the disabled subregion, the MPU issues a fault.

For the products that implement a cache (only for STM32F7 series and STM32H7 series that implement L1-cache) the additional memory attributes include:

- Cacheable/non-cacheable: means that the dedicated region can be cached or not.
- Write through with no write allocate: on hits, it writes to the cache and the main memory. On misses it updates the block in the main memory not bringing that block to the cache.
- Write-back with no write allocate: on hits, it writes to the cache setting dirty bit for the block. The main memory is not updated. On misses, it updates the block in the main memory not bringing that block to the cache.
- Write-back with write and read allocate: on hits it writes to the cache setting dirty bit for the block, the main memory is not updated. On misses it updates the block in the main memory and brings the block to the cache.

Note: For Cortex-M7, TCMs memories always behave as non-cacheable, non-shared normal memories, irrespective of the memory type attributes defined in the MPU for a memory region containing addresses held in the TCM.

Otherwise, the access permissions associated with an MPU region in the TCM address space are treated in the same way as addresses outside the TCM address space.

3.4 Cortex-M7 constraint speculative prefetch

The Cortex-M7 implements the speculative prefetch feature, which allows speculative accesses to normal memory locations (for example: FMC, Quad-SPI devices). When a speculative prefetch happens, it may impact memories or devices that are sensitive to multiple accesses (such as FIFOs, LCD controller). It may also disturb the traffic generated by another masters such as LCD-TFT or DMA2D with higher bandwidth consumption when a speculative prefetch happens. In order to protect normal memories from a speculative prefetch, it is recommended to change memory attributes from normal to a strongly ordered or to device memory thanks to the MPU. For more details about configuring memory attributes, refer to Section 6: MPU setting example with STM32Cube HAL on Armv6 and Armv7 architectures.

4 Cortex-M33/M55 memory types, registers and attributes

Cortex-M33 and Cortex-M55 are respectively based on the Armv8-M and Armv8.1-M architectures (Cortex®-M55 processor supports Arm Protected Memory System Architecture (PMSA)). In both cases, the MPU is a component that is primarily used for memory region protection.

Although the concepts for the MPU operations are similar, the MPU in the Armv8-M architecture has a different programmers' model to the MPU in previous versions of the M-profile Arm® architecture.

It is important to realize that all MPU registers are banked. If Arm TrustZone® is enabled, there is a set of MPU registers for the secure state, and a mirror set for the non-secure state. When accessing the MPU address between 0xE000 ED90 and 0xE000 EDC4, the type of MPU registers accessed is determined by the current state of the processor.

Non-secure code can access non-secure MPU registers and secure code can access secure MPU registers. Secure code can access non-secure MPU registers at their aliased address.

Secure access sees secure MPU registers, non-secure access sees non-secure MPU registers. Secure software can also access non-secure MPU registers using the alias address.

Note: See Arm®-M Architecture reference manual for more information about the memory model.

4.1 Memory types and attributes

In Armv8-M and Armv8.1-M architectures, memory types are divided into:

- normal memory
- device memory

Note: The strongly ordered (SO) device memory type in Armv6-M and Armv7-M is now a subset of the device memory type.

A normal memory type is intended to be used for MPU regions that are used to access general instruction or data memory. Normal memory allows the processor to perform some memory access optimizations, such as access reordering or merging. Normal memory also allows memory to be cached and is suitable for holding executable code. Normal memory must not be used to access peripheral MMIO registers. The device memory type is intended for that use. A normal memory definition remains mostly unchanged from the Armv7-M architecture.

A normal memory has the following attributes:

- cacheability: memories cacheable or non-cacheable
- shareability: normal memory shareable or non-shareable
- execute never: memories marked as executable or execute never (XN)

A device memory must be used for memory regions that cover peripheral control registers. Some of the optimizations that are allowed to normal memory, such as access merging or repeating, are unsafe to a peripheral register.

A device memory has the following attributes:

- G or nG: gathering or non-gathering. (multiple accesses to a device can be merged into a single transaction except for operations with memory ordering semantics, for example, memory barrier instructions, load acquire/store release).
- R or nR: reordering
- E or nE: early write acknowledge (similar to bufferable)

Only four combinations of these attributes are valid:

- device-nGnRnE: equivalent to Armv7-M strongly ordered memory type
- device-nGnRE: equivalent to Armv7-M device memory
- device-nGRE: new to Armv8-M
- device-GRE: new to Armv8-M

4.1.1 Cortex®-M55 access privilege level for device and normal memory

The AMBA® 5 AXI, AMBA® 5 AHB, and AMBA® 4 APB protocols have signals that can report the privilege level of an access request to the system.

The Cortex-M55 processor supports these signals across the manager AXI (M-AXI), the peripheral AHB (P-AHB), and the external private peripheral bus (EPPB) interfaces for device memory. It also supports privilege reporting for normal memory on P-AHB. However, accesses to normal memory on M-AXI can be buffered and cached, allowing memory read and write requests, as well as instruction fetches from both privileged and unprivileged software, to be merged. For these transactions initiated by loads and stores, the AXI signals ARPROT[0] and AWPROT[0] are always 1 indicating a privileged access. Access permission to a region of memory can always be restricted to software running in privileged mode by using the MPU.

The instruction tightly coupled memory (ITCM) and data tightly coupled memory (DTCM) interfaces provide the following signals: ITCMPRIV, D0TCMPRIV, D1TCMPRIV, D2TCMPRIV, and D3TCMPRIV. These signals indicate the privilege of all memory accesses except the ones initiated by loads and stores.

4.2 Attribute indirection

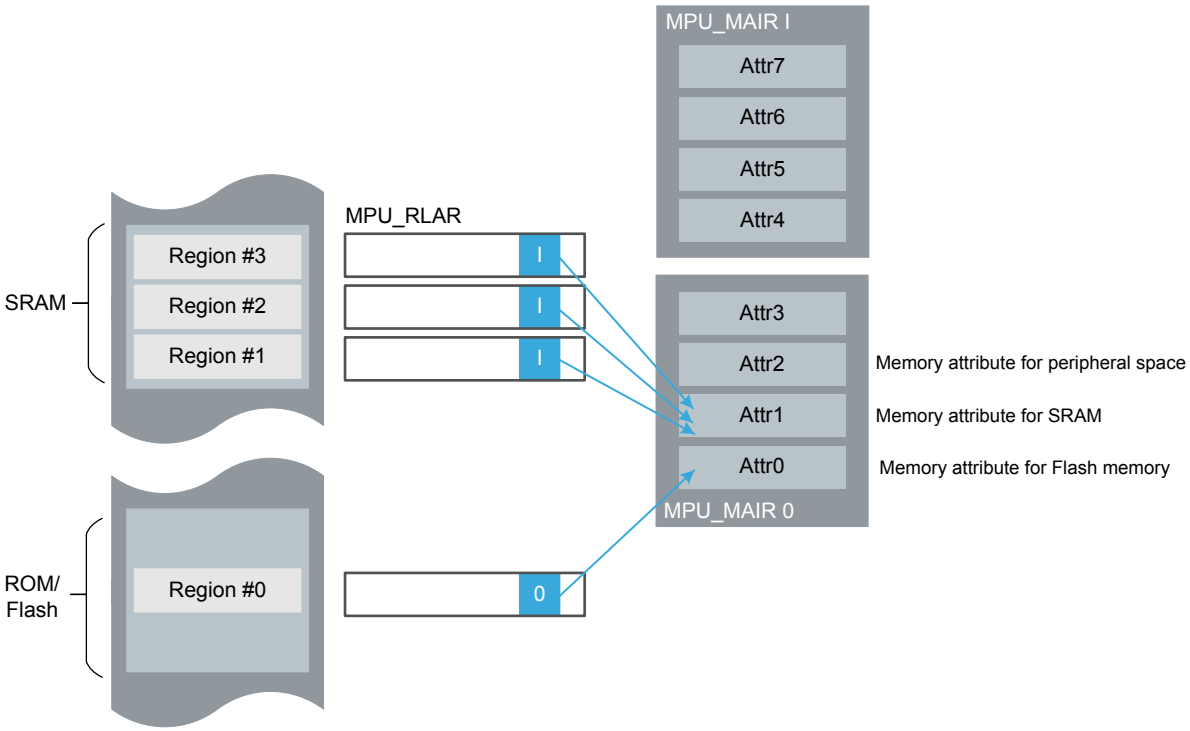
The attribute indirection mechanism allows multiple MPU regions to share a set of memory attributes.

The MPU can be configured to support 0, 4, 8, 12, or 16 memory regions (security extension is included in the Cortex-M55 processor. Memory protection can be duplicated between secure and nonsecure MPU (MPU_S and MPU_NS).)

The number of regions in the secure and nonsecure MPU can be configured independently, and each can be programmed to protect memory for the associated security state.

The following figure shows that MPU regions 1, 2, and 3 are all assigned to SRAM. Therefore, they can share cache-related memory attributes.

Figure 4. Attribute indirection example



At the same time, regions 1, 2, and 3 can still have their own access permission, XN, and shareability attributes. This is required as each region can be used differently in the application.

4.3 MPU registers

The Cortex-M33/M55 MPU registers are different from previous Cortex® cores, offering more flexibility and compatibility with Arm® TrustZone®. Consequently, the programming approach used in previous products cannot be applied for these ones. The introduction of the MPU Region Base Limit Register, for example, allows the user to easily define start and end of their protected regions.

Table 6. MPU register summary

AdressName	Type	Reset value	Description
0xE000ED90	MPU_TYPE(1)	RO	0x0000xx00
0xE000ED94	MPU_CTRL	RW	0x00000000 MPU Type Register
0xE000ED98	MPU_RNR	RW	0x000000XX MPU Control Register
0xE000ED9C	MPU_RBAR	RW	Unknown MPU Region Number Register
0xE000EDA0	MPU_RLAR	RW	Unknown, bit [0] resets to 0 MPU Region Base Address Register
0xE000EDA4	MPU_RBAR_A1	RW	Unknown MPU Region Limit Address Register
0xE000EDA8	MPU_RLAR_A1	RW	Unknown MPU Region Base Address Register Alias 1
0xE000EDAC	MPU_RBAR_A2	RW	Unknown MPU Region Limit Address Register Alias 1
0xE000EDB0	MPU_RLAR_A2	RW	Unknown MPU Region Base Address Register Alias 2
0xE000EDB4	MPU_RBAR_A3	RW	Unknown MPU Region Limit Address Register Alias 2
0xE000EDB8	MPU_RLAR_A3	RW	Unknown Address Register Alias 3
0xE000EDC0	MPU_MAIR0	RW	Unknown. MPU Memory Attribute Indirection Register 0
0xE000EDC4	MPU_MAIR1	RW	Unknown MPU Memory Attribute Indirection Register 1

1. *MPU_TYPE[15:8] depends on the number of MPU regions configured. This value can be 0, 4, 8, 12, or 16.*

5 MPU features comparison between Cortex® cores

There are few MPU differences between Cortex-M0+, Cortex-M3/M4, Cortex-M7, Cortex-M33 and Cortex-M55.The user must be aware of them if the MPU configuration software has to be used. The table below illustrates these differences.

Table 7. Comparison of MPU features between Cortex cores

Features	Cortex-M0+	Cortex-M3/M4	Cortex-M7	Cortex-M33	Cortex-M55
Number of regions	8	8	8/16 (1)(2)	8 MPU_S / 8 MPU_NS(3)	16 MPU_S / 16 MPU_NS
Region address	Yes	Yes	Yes	Yes	Yes
Region size	256 bytes to 4 Gbytes	32 bytes to 4 Gbytes	32 bytes to 4 Gbytes	32 bytes to 4 Gbytes	32 bytes to 4 Gbytes
Region memory attributes	S, C, B, XN(4)	TEX, S, C, B, XN	TEX, S, C, B, XN	S,C, E (5),G (6), R (7), XN	S,C, E(5) ,G (6), R (7), XN, PXN(8)
Region access permission (AP)	Yes	Yes	Yes	Yes (privileged or not)	Yes (privileged or not)
Subregion disable	8 bits	8 bits	8 bits	NA	NA
MPU bypass for NMI/HardFault	Yes	Yes	Yes	Yes	Yes
Alias of MPU registers	No	Yes	Yes	Yes	Yes
Fault exception	HardFault only	HardFault/ MemManage	HardFault/ MemManage	HardFault/ MemManage	HardFault/ MemManage

1. *For STM32H7 series devices.*
2. *For STM32F7 series devices.*
3. *For STM32H5/STM32U3: 12 MPU_S / 8 MPU_NS .*
4. *Cortex-M0+ supports one level of cache policy. That is why the TEX field is not available in Cortex-M0+ processor.*
5. *Early write acknowledge (similar to bufferable)*
6. *Gathering*
7. *Reordering*
8. *Privileged eXecute Never attribute in Armv8.1-M architecture.*

6 MPU setting example with STM32Cube HAL on Armv6 and Armv7 architectures

The table below describes an example of setting up the MPU with the following memory regions: Internal SRAM, flash memory and peripherals. The default memory map is used for privileged accesses as a background region, the MPU is not enabled for the HardFault handler and NMI.

Internal SRAM: 8 Kbytes of internal SRAM is configured as Region0.

Memory attributes: shareable and cacheable memory, write through with no write allocate, full access permission and code execution enabled.

Flash memory: the whole Flash memory is configured as Region.

Memory attributes: shareable and cacheable memory, write through with no write allocate, full access permission and code execution enabled.

Peripheral region: is configured as Region2.

Memory attributes: cached and shared device, full access permission and execute never.

Table 8. Example of setting up the MPU

Configuration	Memory type	Base address	Region number	Memory size	Memory attributes
Internal SRAM	Normal memory	0x2000 0000	Region0	8 Kbytes	Shareable, write through, no write allocate C = 1, B = 0, TEX = 0, S = 1 SRD = 0, XN = 0, AP = full access
Flash memory	Normal memory	0x0800 0000	Region1	1 Mbyte	Non-shareable write through, no write allocate C = 1, B = 0, TEX = 0, S = 1 SRD = 0, XN = 0, AP = full access
FMC	Device memory	0x4000 0000	Region2	512 Mbytes	Shareable, write through, no write allocate C = 1, B = 0, TEX = 0, S = 1 SRD = 0, XN = 1, AP = full access

Setting the MPU with STM32Cube HAL

```
void MPU_RegionConfig(void)
{
MPU_Region_InitTypeDef MPU_InitStruct;
/* Disable MPU */
HAL_MPU_Disable();
/* Configure RAM region as Region N°0, 8kB of size and R/W region */
MPU_InitStruct.Enable = MPU_REGION_ENABLE;
MPU_InitStruct.BaseAddress = 0x20000000;
MPU_InitStruct.Size = MPU_REGION_SIZE_8KB;
MPU_InitStruct.AccessPermission = MPU_REGION_FULL_ACCESS;
MPU_InitStruct.IsBufferable = MPU_ACCESS_NOT_BUFFERABLE;
MPU_InitStruct.IsCacheable = MPU_ACCESS_CACHEABLE;
MPU_InitStruct.IsShareable = MPU_ACCESS_SHAREABLE;
MPU_InitStruct.Number = MPU_REGION_NUMBER0;
MPU_InitStruct.TypeExtField = MPU_TEX_LEVEL0;
MPU_InitStruct.SubRegionDisable = 0x00;
MPU_InitStruct.DisableExec = MPU_INSTRUCTION_ACCESS_ENABLE;
HAL_MPU_ConfigRegion(&MPU_InitStruct);
/* Configure FLASH region as REGION N°1, 1MB of size and R/W region */
MPU_InitStruct.BaseAddress = 0x08000000;
MPU_InitStruct.Size = MPU_REGION_SIZE_1MB;
MPU_InitStruct.Number = MPU_REGION_NUMBER1;
HAL_MPU_ConfigRegion(&MPU_InitStruct);
/* Configure FMC region as REGION N°2, 0.5GB of size, R/W region */
MPU_InitStruct.BaseAddress = 0x40000000;
MPU_InitStruct.Size = MPU_REGION_SIZE_512MB;
MPU_InitStruct.IsShareable = MPU_ACCESS_SHAREABLE;
MPU_InitStruct.Number = MPU_REGION_NUMBER2;
```

```
MPU_InitStruct.DisableExec = MPU_INSTRUCTION_ACCESS_DISABLE;
HAL_MPU_ConfigRegion(&MPU_InitStruct);
/* Enable MPU */
HAL_MPU_Enable(MPU_PRIVILEGED_DEFAULT);
}
```

7 MPU setting example with STM32Cube HAL on Armv8-M architecture

The table below describes an example of setting up the MPU in Armv8-M architecture with the following memory regions: internal SRAM, internal flash memory and peripherals.

Internal SRAM

- Region number 0 with a size of 256 KB
- Memory attributes: normal, non-cacheable, full access permission, code execution enabled

Internal flash memory

- Region number 1 with a size of 2 MB
- Memory attributes: normal, write through and no write allocate, full access permission, code execution enabled

Peripherals

- Region number 2 with a size of 512 MB
- Memory attributes: device nGnRE, full access permission, code execution disabled

Table 9. Example of setting up the MPU with Armv8-M

Configuration	Memory type	Base address	Region number	Memory size	Memory attributes
Internal SRAM	Normal memory	0x2000 0000	Region0	256 Kbytes	Memory attribute = normal, non-cacheable. XN = 0 AP = full access S = 0
Flash memory	Normal memory	0x0800 0000	Region1	2 Mbyte	Memory attribute = normal, write through and no write allocate. XN = 0 AP = full access S = 0
Peripherals	Device	0x4000 0000	Region2	512 Mbytes	Memory attribute = device nGnRE XN = 1 AP = full access S = 0

Setting the MPU with STM32Cube HAL

```
void MPU_Config(void)
{
MPU_Region_InitTypeDef MPU_InitStruct;
MPU_Attributes_InitTypeDef MPU_AttributesInit;

/* Disable MPU */
HAL_MPU_Disable();

/* Configure RAM region as Region Number 0, 256KB of size */
MPU_AttributesInit.Number = MPU_ATTRIBUTES_NUMBER0;
MPU_AttributesInit.Attributes = INNER_OUTER(MPU_NOT_CACHEABLE);
HAL_MPU_ConfigMemoryAttributes(&MPU_AttributesInit);

MPU_InitStruct.Enable = MPU_REGION_ENABLE;
MPU_InitStruct.BaseAddress = 0x20000000;
```

```
MPU_InitStruct.LimitAddress = 0x2003FFFF;
MPU_InitStruct.AccessPermission = MPU_REGION_ALL_RW;
MPU_InitStruct.AttributesIndex = MPU_ATTRIBUTES_NUMBER0;
MPU_InitStruct.IsShareable = MPU_ACCESS_NOT_SHAREABLE;
MPU_InitStruct.Number = MPU_REGION_NUMBER0;
MPU_InitStruct.DisableExec = MPU_INSTRUCTION_ACCESS_ENABLE;
HAL_MPU_ConfigRegion(&MPU_InitStruct);


/* Configure FLASH region as REGION Number 1, 2M of size */
MPU_AttributesInit.Number = MPU_ATTRIBUTES_NUMBER1;
MPU_AttributesInit.Attributes = INNER_OUTER(MPU_WRITE_THROUGH|MPU_R_ALLOCATE);
HAL_MPU_ConfigMemoryAttributes(&MPU_AttributesInit);

MPU_InitStruct.Enable = MPU_REGION_ENABLE;
MPU_InitStruct.BaseAddress = 0x08000000;
MPU_InitStruct.LimitAddress = 0x081FFFFF;
MPU_InitStruct.AccessPermission = MPU_REGION_ALL_RW;
MPU_InitStruct.AttributesIndex = MPU_ATTRIBUTES_NUMBER1;
MPU_InitStruct.IsShareable = MPU_ACCESS_NOT_SHAREABLE;
MPU_InitStruct.Number = MPU_REGION_NUMBER1;
MPU_InitStruct.DisableExec = MPU_INSTRUCTION_ACCESS_ENABLE;
HAL_MPU_ConfigRegion(&MPU_InitStruct);

/* Configure Peripheral region as REGION Number 2, 512MB of size, Execute Never region */
MPU_AttributesInit.Number = MPU_ATTRIBUTES_NUMBER2;
MPU_AttributesInit.Attributes = MPU_DEVICE_nGnRE;
HAL_MPU_ConfigMemoryAttributes(&MPU_AttributesInit);

MPU_InitStruct.BaseAddress = 0x40000000;
MPU_InitStruct.LimitAddress = 0x5FFFFFFF;
MPU_InitStruct.AccessPermission = MPU_REGION_ALL_RW;
MPU_InitStruct.AttributesIndex = MPU_ATTRIBUTES_NUMBER2;
MPU_InitStruct.IsShareable = MPU_ACCESS_NOT_SHAREABLE;
MPU_InitStruct.Number = MPU_REGION_NUMBER2;
MPU_InitStruct.DisableExec = MPU_INSTRUCTION_ACCESS_DISABLE;

/* Enable MPU */
HAL_MPU_Enable(MPU_PRIVILEGED_DEFAULT);
}
```

8 Conclusion

Using the MPU in the STM32 microcontrollers makes them robust, reliable and in some cases more secure by preventing the application tasks from accessing or corrupting the stack and data memory used by the other tasks.

This application note describes the different memory attributes, the types, and the MPU registers. It provides also an example for setting up the MPU with the STM32Cube HAL to illustrate how to configure the MPU in the STM32 MCUs.

For more details about the MPU register, refer to the Cortex® core programming manuals.

Revision history

Table 10. **Document revision history**

Date	Revision	Changes
24-Mar-2016	1	Initial release.
04-May-2018	2	Added STM32H7 series in the whole document. Updated Figure 1. Example of overlapping regions.. Added Section 1 General information Added Section 3.4 Cortex-M7 constraint speculative prefetch.
17-Jul-2019	3	Updated Introduction adding STM32G0 series, STM32G4 series, STM32L4+ series, STM32L5 series and STM32WB series.
10-Feb-2020	4	Added: • PM0214 in Section Introduction • Section 4 Memory types, registers and attributes of the CM33 • Section 5 Comparison of MPU features between Cortex-M0+, Cortex-M3/M4, Cortex-M7, and Cortex-M33 Updated: • title of the document • Section Introduction • Section 2 Overview • Section 2.1 Memory model
20-Sep-2021	5	Updated the Applicable products table.
07-Feb-2023	6	Added the STM32C0 series and STM32H5 series in Table 1. Applicable products. Updated the Section 6: MPU setting example with STM32Cube HAL on Armv6 and Armv7 architectures. Updated the whole document with minor changes. Updated the document title.
04-Mar-2024	7	Added the STM32U0 series in Table 1. Applicable products.
04-Apr-2024	8	Added: • STM32WB0 series in Table 1. Applicable products Section 3.2: MPU register description
15-Jan-2025	9	Added STM32N6 series. Updated: • Section 2.1: Memory model • Section 4.2: Attribute indirection • Section 4.3: MPU registers Added Section 4.1.1: Cortex®-M55 access privilege level for device and normal memory.
25-Feb-2025	10	Added STM32U3 series. Updated: • Section Introduction • Section 2: Overview • Section 4.3: MPU registers • Section 4: Cortex-M33/M55 memory types, registers and attributes • Section 5: MPU features comparison between Cortex® cores Added Section 7: MPU setting example with STM32Cube HAL on Armv8-M architecture.

Contents

List of tables

List of figures