

Managing memory protection unit in 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 Cortex®-M0+/M3/M4 and M7 design which 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 *Cortex®-M0+ for STM32L0, STM32G0, STM32WL and STM32WB Series* (PM0223)
- Programming manual *STM32 Cortex®-M4 MCUs and MPUs* (PM0214)
- Programming manual *STM32 Cortex®-M33 MCUs* (PM0264)

Table 1. Applicable products

Type	Product series
Microcontrollers	<ul style="list-style-type: none"> • STM32F1 Series, STM32F2 Series, STM32F3 Series, STM32F4 Series, STM32F7 Series • STM32G0 Series, STM32G4 Series • STM32H7 Series • STM32L0 Series, STM32L1 Series, STM32L4 Series, STM32L4+ Series, STM32L5 Series • STM32U5 Series • STM32WB 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 sixteen memory regions. In Armv6 and Armv7 architecture (Cortex-M0+, M3, M4, and M7, this 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 1](#) 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 corresponds to a 256-byte size.

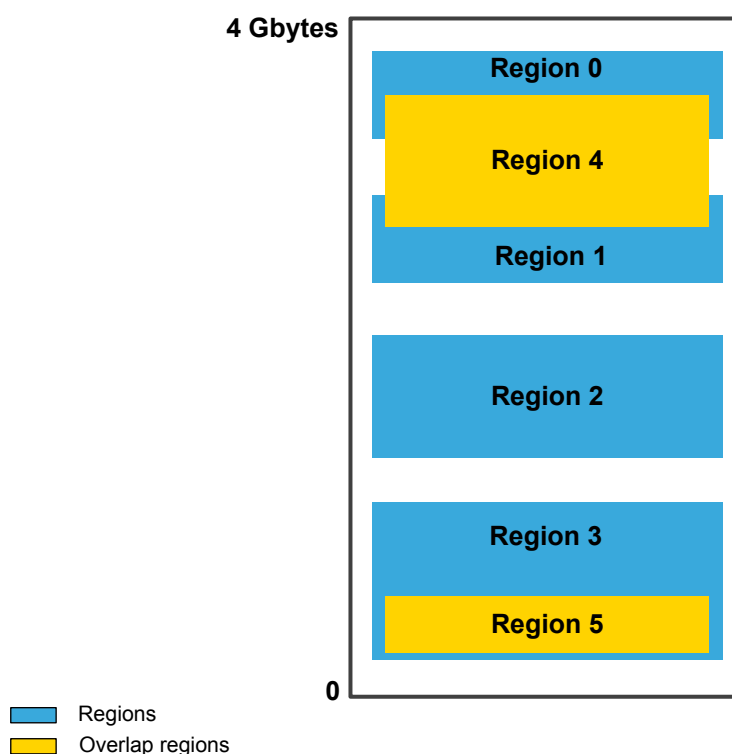
The regions are numbered 0 to 15. In addition, there is a region called the default region with an id of -1. All the 0-15 memory regions take priority over the default region.

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) the regions are defined using a base and a limit address offering flexibility and simplicity to the developer on the way to organise them. Additionally the Cortex-M33 does 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 (Cortex-M33), 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.

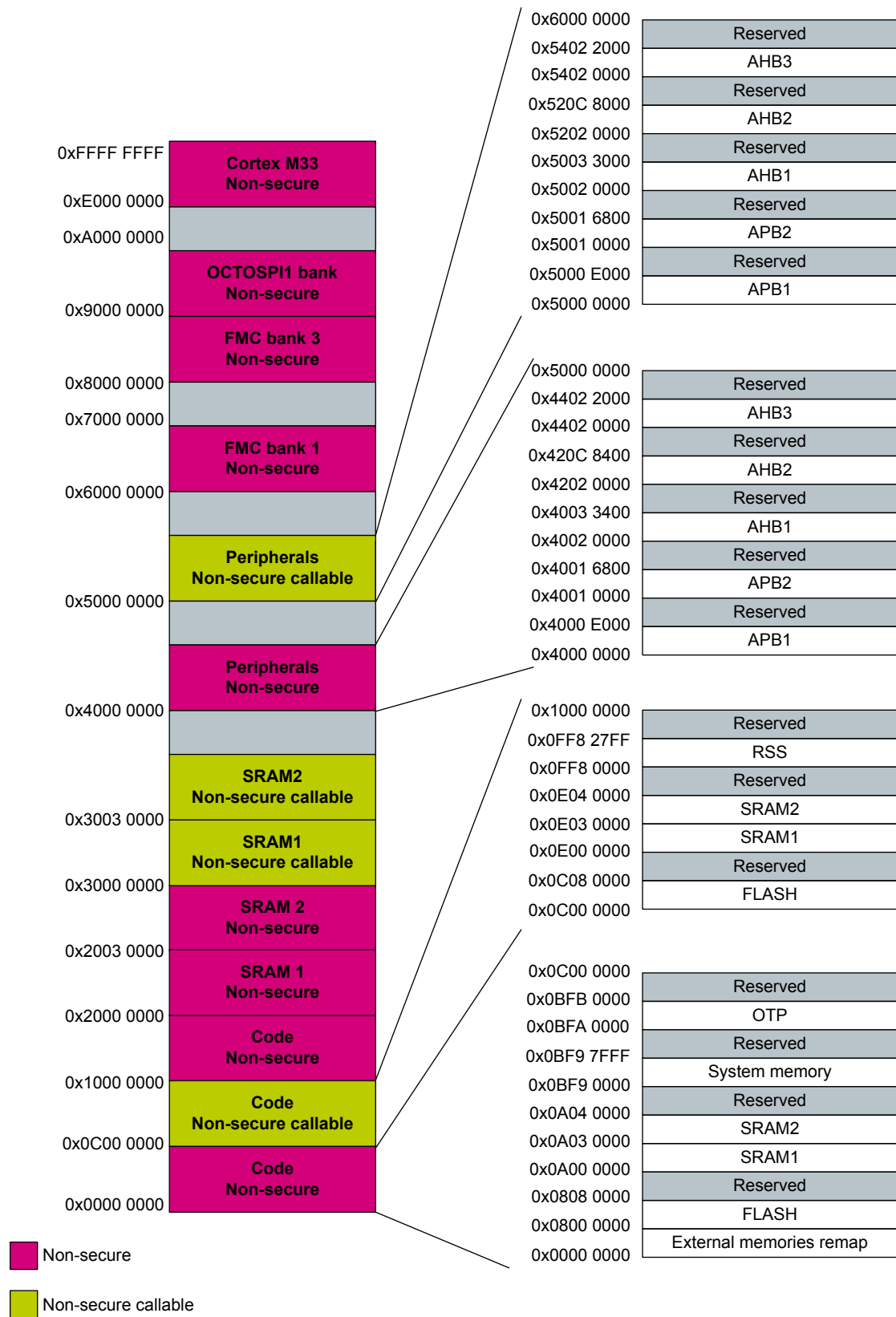
The cache control is done globally by the cache control register, but the MPU can specify the cache policy and whether the region is cacheable or not.

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

Vendor-specific memory	511 Mbytes	0xFFFF FFFF
Private peripheral bus	1.0 Mbyte	0xE010 0000 0xE00F FFFF 0xE000 0000 0xDFFF FFFF
External device	1.0 Gbyte	
External RAM	1.0 Gbyte	0xA000 0000 0x9FFF FFFF
Peripheral	0.5 Gbyte	0x6000 0000 0x5FFF FFFF
SRAM	0.5 Gbyte	0x4000 0000 0x3FFF FFFF
Code	0.5 Gbyte	0x2000 0000 0x1FFF FFFF
		0x0000 0000

Figure 3. Cortex-M33 processor memory map


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 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.

3.3 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 2. 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 3. 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 4. 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 which 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 memory types, registers and attributes

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.

4.1 Memory types and attributes

In Armv8-M architecture, 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)

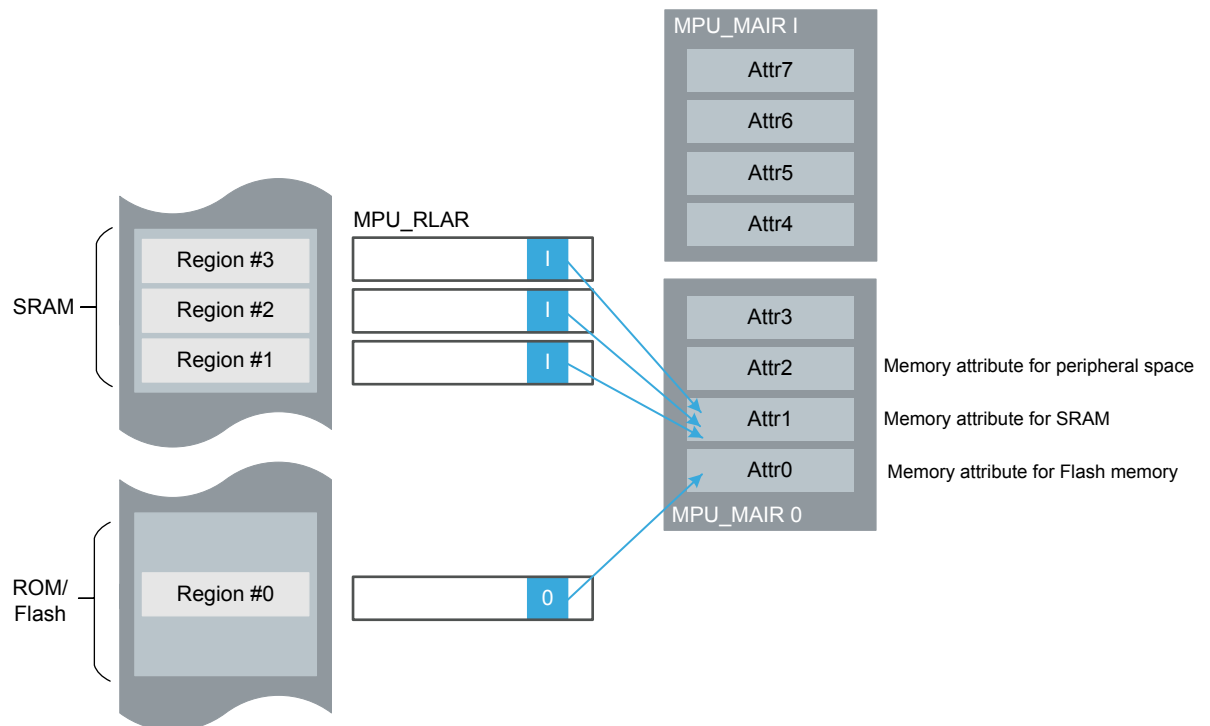
Only four combinations of these attributes are valid :

- 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

4.2 Attribute indirection

The attribute indirection mechanism allows multiple MPU regions to share a set of memory attributes. For example, in the following figure, MPU regions 1, 2 and 3 are all assigned to SRAM, so 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 MPU registers are different from previous Cortex cores, offering more flexibilities and compatibilities with Arm TrustZone. Consequently, the programming approach used in previous products cannot be applied for this 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 5. Cortex-M33 MPU registers

Secure address	NS address alias	Register	Description
0xE000ED90	0xE002ED90	MPU_TYPE	MPU Type Register
0xE000ED94	0xE002ED94	MPU_CTRL	MPU Control Register
0xE000ED98	0xE002ED98	MPU_RNR	MPU Region Number Register
0xE000ED9C	0xE002ED9C	MPU_RBAR	MPU Region Base Address Register
0xE000EDA0	0xE002EDA0	MPU_RLAR	MPU Region Base Limit Register
0xE000EDA4	0xE002EDA4	MPU_RBAR_A1	MPU Region Base Address Register Alias 1
0xE000EDAC	0xE002EDAC	MPU_RBAR_A2	MPU Region Base Address Register Alias 2
0xE000EDB4	0xE002EDB4	MPU_RBAR_A3	MPU Region Base Address Register Alias 3
0xE000EDA8	0xE002EDA8	MPU_RLAR_A1	MPU Region Limit Address Register Alias 1
0xE000EDB0	0xE002EDB0	MPU_RLAR_A2	MPU Region Limit Address Register Alias 2
0xE000EDB8	0xE002EDB8	MPU_RLAR_A3	MPU Region Limit Address Register Alias 3
0xE000EDC0	0xE002EDC0	MPU_MAIR0	MPU Memory Attribute Indirection Register 0
0xE000EDC4	0xE002EDC4	MPU_MAIR1	MPU Memory Attribute Indirection Register 1

5 MPU features comparison between Cortex cores

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

Table 6. Comparison of MPU features between Cortex cores

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

1. For STM32H7 Series devices.

2. For STM32F7 Series devices.

3. Cortex-M0+ supports one level of cache policy. That is why the TEX field is not available in Cortex-M0+ processor.

4. Early write acknowledge (similar to bufferable)

5. Gathering

6. Reordering

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 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: non-shareable memory, write through with no write allocate, full access permission and code execution enabled

Peripheral region: is configured as Region2

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

Table 7. 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 = 0 SRD = 0, XN = 0, AP = full access
FMC	Normal memory	0x6000 0000	Region2	512 Mbytes	Shareable, write through, no write allocate C = 1, B = 0, TEX = 0, S = 1 SRD = 0, XN = 0, 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.IsShareable = MPU_ACCESS_NOT_SHAREABLE;
    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 = 0x60000000;
    MPU_InitStruct.Size = MPU_REGION_SIZE_512MB;
    MPU_InitStruct.IsShareable = MPU_ACCESS_SHAREABLE;
    MPU_InitStruct.Number = MPU_REGION_NUMBER2;
    HAL_MPU_ConfigRegion(&MPU_InitStruct);
    /* Enable MPU */
    HAL_MPU_Enable(MPU_PRIVILEGED_DEFAULT);
}
```

7 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 registers, refer to the Cortex core programming manuals.

Revision history

Table 8. 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: <ul style="list-style-type: none"> 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 : <ul style="list-style-type: none"> title of the document Section Introduction Section 2 Overview Section 2.1 Memory model
20-Sep-2021	5	Updated: <ul style="list-style-type: none"> Table 1. Applicable products

Contents

1	General information	2
2	Overview	3
2.1	Memory model	4
3	Cortex-M0+/M3/M4/M7 memory types, registers and attributes	6
3.1	Memory types	6
3.3	Memory attributes	6
3.4	Cortex-M7 constraint speculative prefetch	8
4	Cortex-M33 memory types, registers and attributes	9
4.1	Memory types and attributes	9
4.2	Attribute indirection	10
4.3	MPU registers	11
5	MPU features comparison between Cortex cores	12
6	MPU setting example with STM32Cube HAL on Armv6 and Armv7 architectures	13
7	Conclusion	15
	Revision history	16
	List of tables	18
	List of figures	19

List of tables

Table 1.	Applicable products	1
Table 2.	Region attributes and size in MPU_RASR register	6
Table 3.	Access permissions of regions.	7
Table 4.	Cache properties and shareability	7
Table 5.	Cortex-M33 MPU registers	11
Table 6.	Comparison of MPU features between Cortex cores.	12
Table 7.	Example of setting up the MPU	13
Table 8.	Document revision history	16

List of figures

Figure 1.	Example of overlapping regions	3
Figure 2.	Cortex-M0+/M3/M4/M7 processor memory map	4
Figure 3.	Cortex-M33 processor memory map	5
Figure 4.	Attribute indirection example	10

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, please refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2021 STMicroelectronics – All rights reserved