# AN4838
# Application note

# Managing memory protection unit (MPU) in STM32 MCUs

## Introduction

This application note describes how to manage the MPU in the STM32 products which is an optional component for the memory protection. Including the MPU in the STM32 microcontrollers 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 that include Cortex®-M0+/M3/M4 and M7 design which support the MPU.

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

- *STM32F7 Series Cortex®-M7 processor programming manual* (PM0253)
- *STM32F3 and STM32F4 Series Cortex®-M4 programming manual* (PM0214)
- *STM32F10xxx/20xxx/21xxx/L1xxxx Cortex®-M3 programming manual* (PM0056)
- *STM32L0 Series Cortex®-M0+ programming manual* (PM0223)

**Table 1. Applicable products**

| Type | Part Number |
|---|---|
| Microcontrollers | STM32F1 Series, STM32F2 Series, STM32F3 Series, STM32F4 Series, STM32F7 Series, STM32L0 Series, STM32L1 Series, STM32L4 Series |

# Contents

1 **Overview** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **5**

1.1 Memory model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

2 **Memory types, registers and attributes** . . . . . . . . . . . . . . . . . . . . . . . . . . **8**

2.1 Memory types . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

2.2 MPU Registers description . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

2.3 Memory attributes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

2.4 Comparison of MPU features between Cortex®-M0+, Cortex®-M3/M4 and Cortex®-M7 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .11

3 **Example for setting up the MPU with cube HAL** . . . . . . . . . . . . . . . . . . **12**

4 **Conclusion** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **14**

5 **Revision history** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **15**

# List of tables

Table 1. Applicable products . . . . . . . . . . 1
Table 2. Region attributes and size in MPU_RASR register. . . . . . . . . . 9
Table 3. Access permission of regions . . . . . . . . . . 9
Table 4. Cache properties and shareability . . . . . . . . . . 10
Table 5. Comparison of MPU features between Cortex®-M0+, Cortex®-M3/M4 and Cortex®-M7 . . . . . . . . . . 11
Table 6. Example of setting up the MPU . . . . . . . . . . 12
Table 7. Document revision history . . . . . . . . . . 15

# List of figures

Figure 1. Example of overlapping regions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
Figure 2. Processor memory map . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

# 1 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.

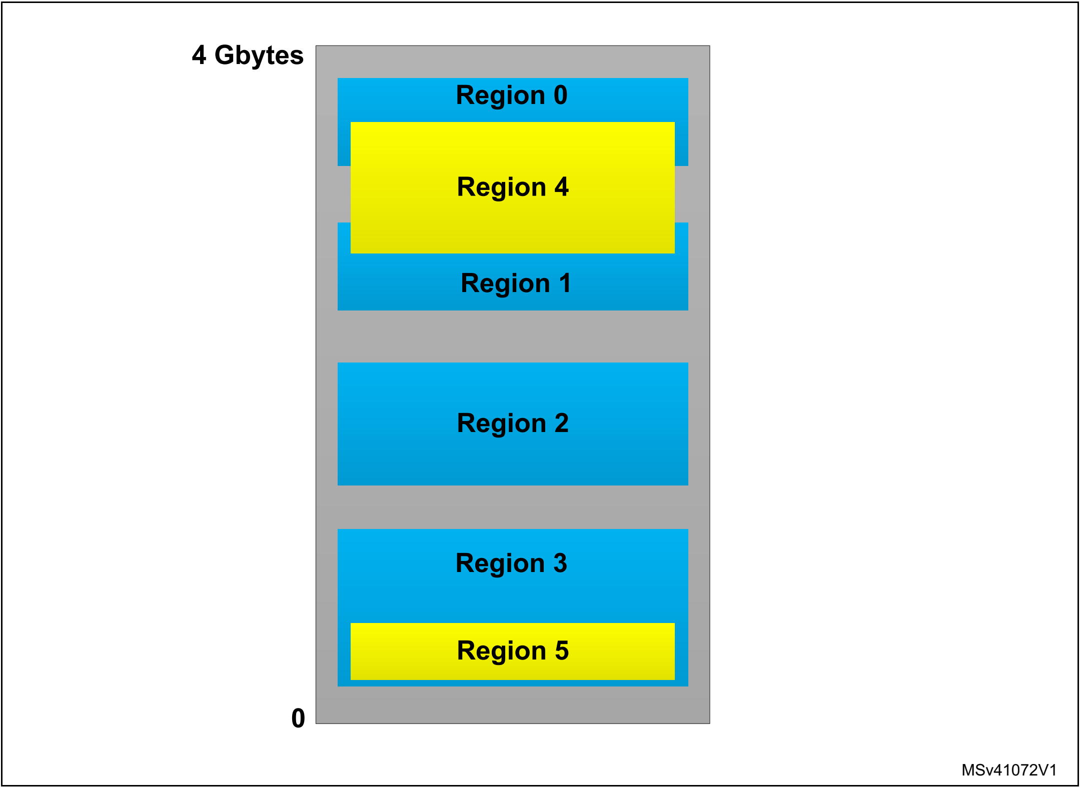
The MPU can be used to protect up to eight memory regions. These, in turn can have eight subregions, if the region is at least 256 bytes. The subregions are always of equal size, and can be enabled or disabled by a subregion number. Because the minimum region size is driven by the cache line length (32 bytes), 8 subregions of 32 bytes corresponds to a 256 bytes size.

The regions are numbered 0-7. In addition, there is another region called the default region with an id of -1. All the 0-7 memory regions take priority over the default region.

The regions can overlap, and can be nested. The region 7 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.

*Figure 1* shows an example with six regions. This example shows the region 4 overlapping the region 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 yellow) 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 will be not writeable.

**Figure 1. Example of overlapping regions**



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

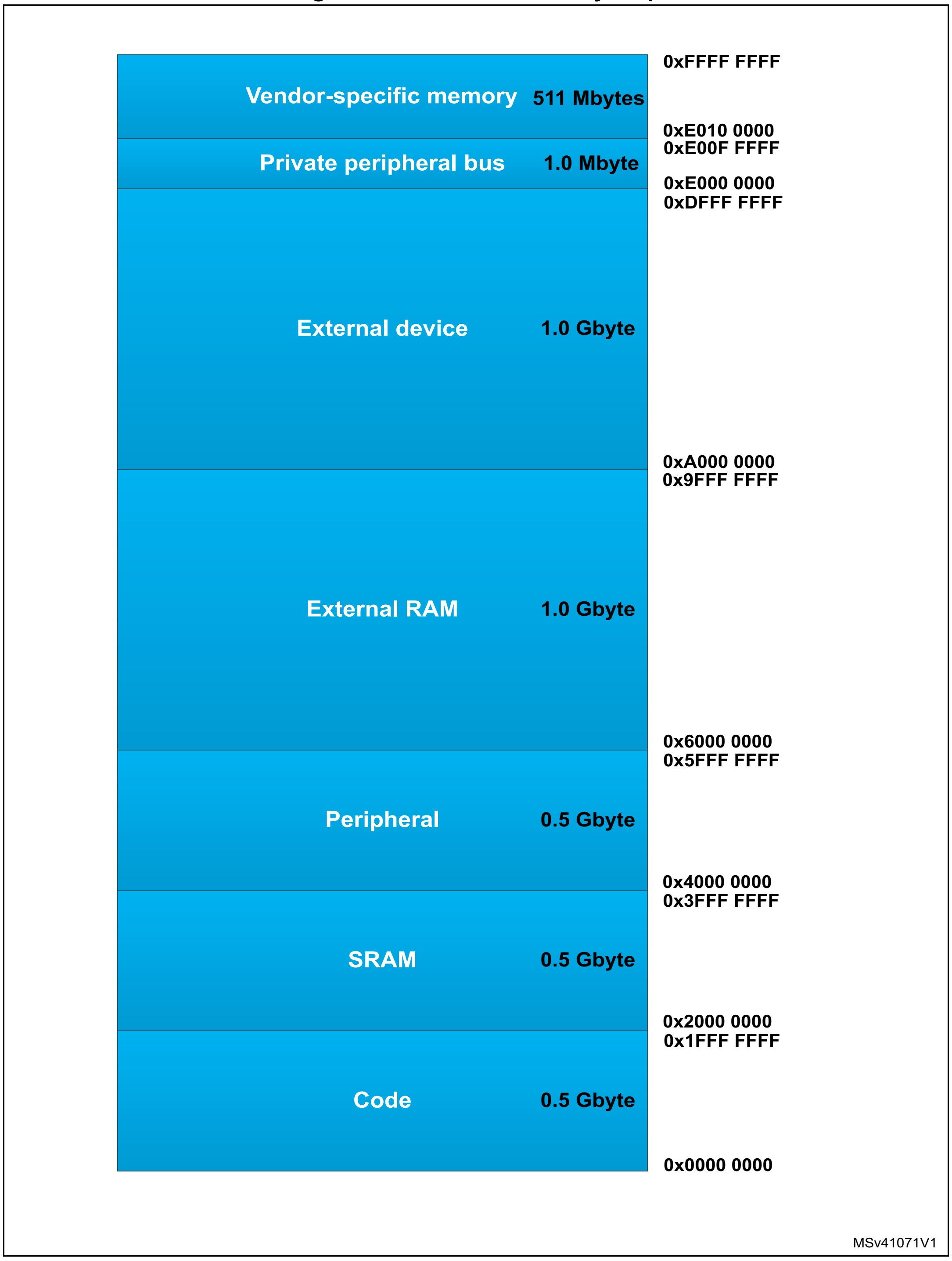
The MPU can be used also to define other memory attributes such as the cacheability, which can be exported to the system level cache unit or the memory controllers. The memory attribute settings can support 2 levels of cache: inner cache and outer cache.

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. The MPU allows to set the cache attributes for level 1 (L1) cache by region (only for the STM32F7 Series which implements a L1-Cache).

## 1.1 Memory model

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

Figure 2. Processor memory map



# 2 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.

## 2.1 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.

## 2.2 MPU Registers description

The MPU registers are located at 0xE000ED90. There are 5 basic MPU registers and a number of alias registers for each of the regions. 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.[a]

MPU_RASR_An: alias n of MPU_RASR, where n is 1 to 3.[a]

For more details about the MPU registers, refer to the programming manuals listed at the introduction section.

a. Cortex®-M0+ does not implement these registers.

## 2.3 Memory attributes

The Region Attributes and Size Register (MPU_RASR) is where all the memory attributes are set. *Table 2* shows a brief description about 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 disable. For each subregion 1=disabled, 0=enabled. |
| 5:1 | SIZE | Specifies the size of the MPU protection region. |

- The XN flag controls the code execution. In order to execute an instruction within the region, there must be read access for the privilege level, and XN must be 0. Otherwise a MemManage fault will be generated.
- The data Access Permission (AP) field defines the AP of memory region. *Table 3* illustrates the access permissions:

**Table 3. Access permission 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 generate 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. 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 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 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.

## 2.4 Comparison of MPU features between Cortex®-M0+, Cortex®-M3/M4 and Cortex®-M7

There are few differences at the MPU level between Cortex®-M0+, Cortex®-M3/M4 and Cortex®-M7, so the user must be aware of them if the MPU configuration software has to be used. *Table 5* illustrates the differences of the MPU features between Cortex®-M0+, Cortex®-M3/M4 and Cortex®-M7.

**Table 5. Comparison of MPU features between Cortex®-M0+, Cortex®-M3/M4 and Cortex®-M7**

| | **Cortex®-M0+** | **Cortex®-M3/M4** | **Cortex®-M7** |
|---|---|---|---|
| Number of regions | 8 | 8 | 8 |
| Unified I and D regions | Yes | Yes | Yes |
| Region address | Yes | Yes | Yes |
| Region size | 256 bytes to 4 Gbytes | 32 bytes to 4 Gbytes | 32 bytes to 4 Gbytes |
| Region memory attributes | S, C, B, XN [1](*) | TEX, S, C, B, XN | TEX, S, C, B, XN |
| Region access permission (AP) | Yes | Yes | Yes |
| Subregion disable | 8 bits | 8 bits | 8 bits |
| MPU bypass for NMI/Hardfault | Yes | Yes | Yes |
| Alias of MPU registers | No | Yes | Yes |
| Fault exception | Hardfault only | Hardfault / MemManage | Hardfault/ MemManage |

1. cortex®-M0+ supports one level of cache policy that's why the TEX field is not available in cortex®-M0+ processor.

# 3 Example for setting up the MPU with cube HAL

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 hard fault handler and NMI.

**Internal SRAM**: 8 Kbytes of internal SRAM will be 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 will be configured as Region1.

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

**Peripheral region**: will be configured as Region2.

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

**Table 6. Example of setting up the MPU**

| Usage | 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<br>C=1, B = 0, TEX = 0, S=1<br>SRD = 0, XN= 0, AP = full access |
| Flash memory | Normal memory | 0x0800 0000 | Region1 | 1 Mbyte | Non-shareable write through, no write allocate<br>C=1, B = 0, TEX = 0, S=0<br>SRD = 0, XN= 0, AP = full access |
| FMC | Normal memory | 0x6000 0000 | Region2 | 512 Mbytes | Shareable, write through, no write allocate<br>C=1, B = 0, TEX = 0, S=1<br>SRD = 0, XN= 0, AP = full access |

- Setting the MPU with cube 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);
}
```

# 4 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 is a description of the different memory attributes, the types and the MPU registers.

It provides also an example for setting up the MPU with the cube HAL to illustrate how to configure the MPU in the STM32 MCUs.

For more details about the MPU registers, refer to the Cortex®-M7/M3/M4/M0+ programming manuals available on ST’s web site.

# 5 Revision history

Table 7. Document revision history

| Date | Revision | Changes |
|---|---|---|
| 24-Mar-2016 | 1 | Initial release. |