Memory Protection Unit of the Cortex-M

© Remo Markgraf, info@tnms.de, www.tnms.de

V1.1, 24.02.2026

Deutsche Version: www.tnms.de/memory-protection-unit-des-cortex-m
English Version: www.tnms.de/memory-protection-unit-of-the-cortex-m

Intro

The Memory Protection Unit, or MPU for short, is an unloved child, although it is very important and helpful. The reason for the resentment lies in the nature of the matter, the MPU is not constructive, but uncovers software malfunction and prevents worse. It does not ensure that SW suddenly works, but rather the contrary, you spend hard work to get SW running under MPU control, that worked nicely without the MPU. So why should I do this? Because the MPU makes the SW safer and more stable and because the MPU is mandatory in safety-relevant applications. In order to simply the complex configuration, two free tools are presented here. The first is an Excel-based graphical MPU code generator and the second is a piece of C-code, MPU info, which outputs the settings of the MPU. Unfortunately many debuggers are not capable of doing this properly. With the introduction of the Armv8-M architecture, a new Protected Memory Software Architecture PMSAv8 has also been introduced, which means that the MPU of the v8-M is not backwards compatible with the v7-M. The different MPUs make the matter even more complex, ... unloved child. Don’t worry, the two tools make life easier, as we don’t have to dive into the individual register details.

Memory Protection Unit

The MPU is a hardware component integrated in the Arm Core that is located in the bus system between the CPU and memory controller and monitors access. Whenever the core reads or writes an address, this access is controlled by the MPU. This applies to accesses to flash, RAM, peripherals, both internally and externally, and even to the tightly coupled memory. The cache controllers are even orchestrated by the MPU, i.e. the MPU determines which memory areas are cached and which cache policy is applied. The MPU is programmed by configuring regions.

MPU Regions

Regions are storage areas for which access rights and storage attributes may be defined. If access is not permitted by a suitable region, a memory fault is triggered. This means that the processor executes an exception, similar to an interrupt, in which an error handling is executed and usually also a reset is triggered. Since the access rights not only distinguish between reading and writing, but also between privileged and unprivileged access, several scenarios can be realized in which, for example, applications can be prevented from accessing OS data or a stack overflow may be detected.

MPU Configuration

In order to adapt the MPU to the respective application, the regions must be programmed. For this purpose, two 32-bit registers are available for each region. In a 3rd register, the Region Number register is used to determine which region can currently be configured. For this purpose, the two registers of the selected region are mapped to a fixed pair of addresses. In a 4th, read only register, the MPU-Type register, The number of implemented regions may be read out. In v7-M architecture there are 0, 8 or for some Cortex-M7 optional 16 regions. In the v8-M architecture, this can be 0, 4, 8, 12 or 16 regions. In addition, a v8-M controller with trust zone has two MPUs and thus two sets of registers, one set for the Secure and one for the Non-Secure world. The bits within the registers differ substantially between the v7-M and v8-M architecture and are not compatible to each other. Those details go beyond the scope of a blog post. The Excel tool mentioned at the beginning does the detailed work for us. So we can conveniently configure the regions via the graphical interface of the Excel tool and receive as a result C-code, which populates the regions with the selected values. Since the 2 x 32-bits of the registers per region were not sufficient in the v8-M architecture, since the more flexible definition of the region addresses take too much space, additional attribution-indirection registers MAIR were added. This are 8x 8-bit registers, with index 0-7, which are combined into two 32-bit registers. Each region contains such an index in 3 bits and thus points to one of the attribution indirections. But don't worry, this configuration is also handled via the Excel tool.

There are two different Excel tools, for

- v7-M, i.e. Cortex-M0+, M3, M4, M7 and
- v8-M, i.e. Cortex-M23, M33, M35P, M52, M55 and M85.

EXCEL MPU Configurator

Via the links

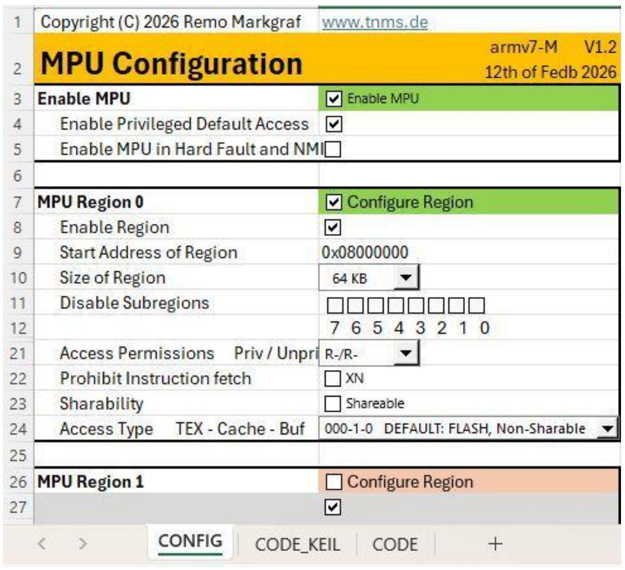
- www.tnms.de/mpu or
- www.tnms.de and the Menu Entry Downloads->MPU Configurator

the two versions of the Excel tool are available for download free of charge.
Input values for the configuration of the MPU are entered in the Excel sheet "CONFIG". Excel allows the input of values for 16 regions, even if only less are realized on the chip. According to Arm Specification, the behaviour of the MPU is undefined when values are configured for non-existent regions. Therefore, in the generated macro MPU_CONFIG_REGION(num), it is checked at runtime whether the respective region exists and only then the entered values are also programmed into the MPU, thus avoiding a potentially undefined state.

```
#define MPU_CONFIG_REGION(num)\
  if( num < ((MPU->TYPE & MPU_TYPE_DREGION_Msk)>>MPU_TYPE_DREGION_Pos)) \
  {\
    MPU->RNR  = (num & MPU_RNR_REGION_Msk);\
    MPU->RBAR =  ARM_MPU_RBAR(num,BASE_REG##num);\
    MPU->RASR = (ARM_MPU_RASR_EX(XN_REG##num, PERM_REG##num, TEXSCB_REG##num,\
    SRD_REG##num,SIZE_REG##num)&~((ENABLE_REG##num)?0:MPU_RASR_ENABLE_Msk));\
  }
```

Figure 1: MPU_CONFIG_REGION Macro in Sheet CODE for armv7m_mpu_config.h

Figure 1 shows the MPU_CONFIG_REGION Macro, which is called for each region when the defined CONFIG_REGn is set to 1 and the MPU has at least n regions. The definition is set via the graphical user interface shown in Figure 2 by the first checkbox "Configure Region" displayed in Figure 2 in cell C7 in green and in cell C26 in red.
..

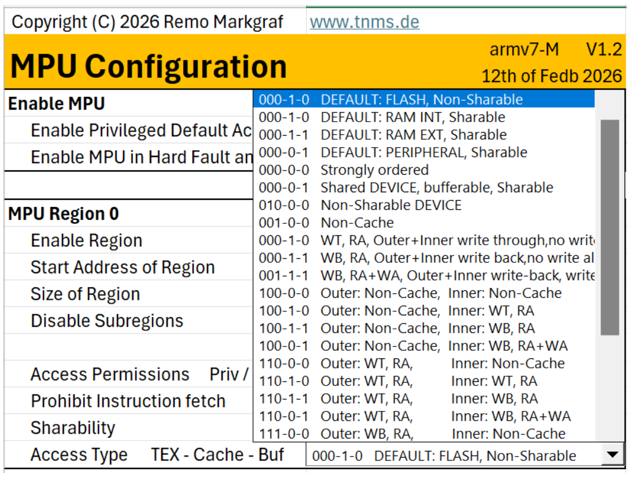


1	Copyright (C) 2026 Remo Markgraf	www.tnms.de
2	**MPU Configuration**	armv7-M V1.2 12th of Fedb 2026
3	**Enable MPU**	☑ Enable MPU
4	Enable Privileged Default Access	☑
5	Enable MPU in Hard Fault and NMI	☐
6		
7	**MPU Region 0**	☑ Configure Region
8	Enable Region	☑
9	Start Address of Region	0x08000000
10	Size of Region	64 KB
11	Disable Subregions	☐☐☐☐☐☐☐☐
12		7 6 5 4 3 2 1 0
21	Access Permissions Priv / Unpri	R-/R-
22	Prohibit Instruction fetch	☐ XN
23	Sharability	☐ Shareable
24	Access Type TEX - Cache - Buf	000-1-0 DEFAULT: FLASH, Non-Sharable
25		
26	**MPU Region 1**	☐ Configure Region
27		☑

CONFIG | CODE_KEIL | CODE | +

Figure 2: Graphical User Interface armv7m_MPU

The last entry of a region is to enter a value for "TEX Cache Buf" that is required to control a potentially existing cache. Since in the armv7-M architecture a cache is only provided for the M7, you do not have to worry about these more complex settings in projects of the M0+, M3, M4. The defaults published by Arm for simpler cases are selectable via the top entries to ease the configuration (see Figure 3).



Copyright (C) 2026 Remo Markgraf www.tnms.de

MPU Configuration armv7-M V1.2 12th of Fedb 2026

Enable MPU	
Enable Privileged Default Ac	
Enable MPU in Hard Fault an	
MPU Region 0	
Enable Region	
Start Address of Region	
Size of Region	
Disable Subregions	
Access Permissions Priv /	
Prohibit Instruction fetch	
Sharability	
Access Type TEX - Cache - Buf	000-1-0 DEFAULT: FLASH, Non-Sharable

```
000-1-0  DEFAULT: FLASH, Non-Sharable
000-1-0  DEFAULT: RAM INT, Sharable
000-1-1  DEFAULT: RAM EXT, Sharable
000-0-1  DEFAULT: PERIPHERAL, Sharable
000-0-0  Strongly ordered
000-0-1  Shared DEVICE, bufferable, Sharable
010-0-0  Non-Sharable DEVICE
001-0-0  Non-Cache
000-1-0  WT, RA, Outer+Inner write through,no writ
000-1-1  WB, RA, Outer+Inner write back,no write al
001-1-1  WB, RA+WA, Outer+Inner write-back, write
100-0-0  Outer: Non-Cache,  Inner: Non-Cache
100-1-0  Outer: Non-Cache,  Inner: WT, RA
100-1-1  Outer: Non-Cache,  Inner: WB, RA
100-0-1  Outer: Non-Cache,  Inner: WB, RA+WA
110-0-0  Outer: WT, RA,     Inner: Non-Cache
110-1-0  Outer: WT, RA,     Inner: WT, RA
110-1-1  Outer: WT, RA,     Inner: WB, RA
110-0-1  Outer: WT, RA,     Inner: WB, RA+WA
111-0-0  Outer: WB, RA,     Inner: Non-Cache
```

Figure 3: Pulldown for TEX-C-B with Defaults

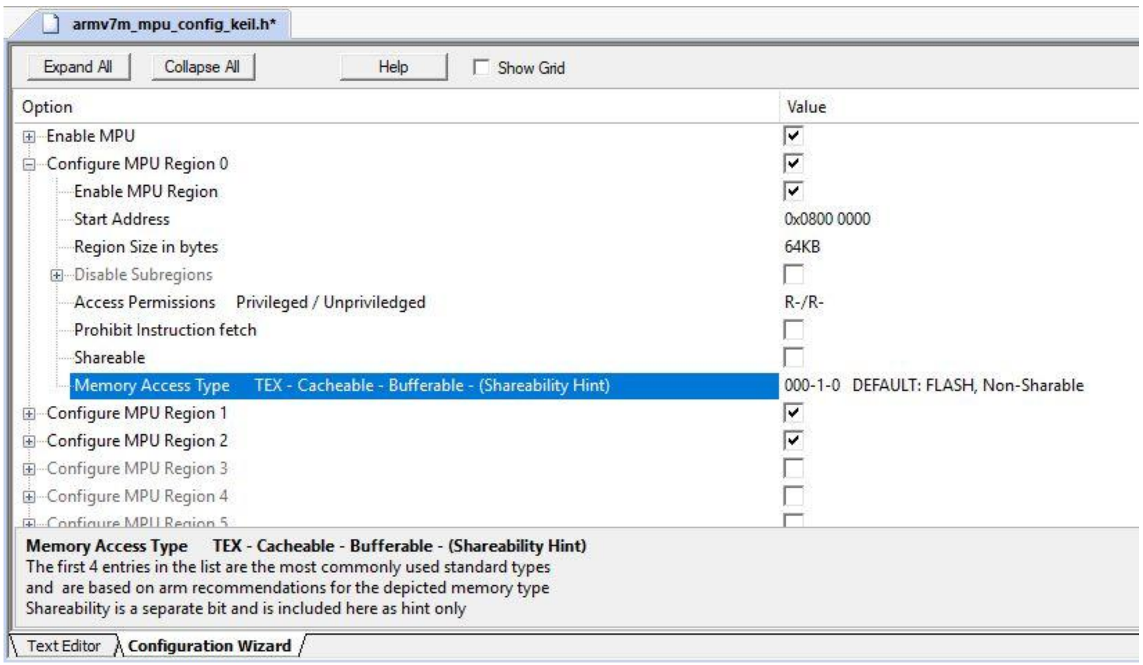
```
// MPU Region 0 ********************************
#define CONFIG_REG0  1
#define ENABLE_REG0  1
#define BASE_REG0    0x08000000
#define SIZE_REG0    15
#define SRD0_REG0    0
#define SRD1_REG0    0
#define SRD2_REG0    0
#define SRD3_REG0    0
#define SRD4_REG0    0
#define SRD5_REG0    0
#define SRD6_REG0    0
#define SRD7_REG0    0
#define SRD_REG0     (SRD0_REG0 | SRD1_REG0 | SRD2_REG0 | SRD3_REG0 |\
                      SRD4_REG0 | SRD5_REG0 | SRD6_REG0 | SRD7_REG0)
#define PERM_REG0    6
#define XN_REG0      0
#define SHARE_REG0   (0<<MPU_RASR_S_Pos)
#define ACCESS_REG0  0x020000
#define TEXSCB_REG0  (ACCESS_REG0 | SHARE_REG0)
```
Figure 4: In Sheet CODE implemented Defines for Region 0

Figure 4 depicts the definitions of a region with values set by the graphical user interface. The values are used in the previously MPU_CONFIG_REGION Macro and the MPU is configured accordingly. The MPU_CONFIG_REGION Macros are called by the inline function MPU_Init() shown in Figure 5.

```
static inline void MPU_Init(void)
{
#if defined(__MPU_PRESENT) && (__MPU_PRESENT == 1U)
  #if defined (CONFIG_REG0) && (CONFIG_REG0 == 1U)
    MPU_CONFIG_REGION(0);
  #endif
  …
  #if defined (CONFIG_REG15) && (CONFIG_REG15 == 1U)
    MPU_CONFIG_REGION(15);
  #endif
  #if defined (MPU_INIT_CTRL_ENABLE) && (MPU_INIT_CTRL_ENABLE== 1U)
    ARM_MPU_Enable( MPU_INIT_CTRL_BITS );
  #endif
#endif //__MPU_PRESENT
}   //MPU_Init (void)
```

Figure 5: In Sheet CODE generated function MPU_Init()

The CODE_KEIL sheet contains the same defines and macros, only additional comment lines are added, which control the graphical editor integrated in Keil µVision. Thus, it is possible to graphically edit the values of the Defines via the editor without the Excel tool in the project. The many comment lines make the code very confusing and therefore there is the sheet CODE that you can use if you do not work with Keil µVision. Figure 6 shows an example of how the graphical editor of armv7m_mpu_config_keil.h looks like in Keil µVision.



Option	Value
Enable MPU	✓
Configure MPU Region 0	✓
Enable MPU Region	✓
Start Address	0x0800 0000
Region Size in bytes	64KB
Disable Subregions	☐
Access Permissions Privileged / Unpriviledged	R-/R-
Prohibit Instruction fetch	☐
Shareable	☐
Memory Access Type TEX - Cacheable - Bufferable - (Shareability Hint)	000-1-0 DEFAULT: FLASH, Non-Sharable
Configure MPU Region 1	✓
Configure MPU Region 2	✓
Configure MPU Region 3	☐
Configure MPU Region 4	☐
Configure MPU Region 5	☐

Memory Access Type TEX - Cacheable - Bufferable - (Shareability Hint)
The first 4 entries in the list are the most commonly used standard types
and are based on arm recommendations for the depicted memory type
Shareability is a separate bit and is included here as hint only

Figure 6: armv7m_mpu_config_keil.h in Keil µVision

To use the Excel Configurator, enter the values for the required regions in Excel, copy the entire content of the CODE or CODE_Keil sheet to the editor of your development environment and name the file, for example, armv7m_mpu_config.h. You then include it in your main.c and call the MPU_Init() function within main(). An example of such a main() function is shown in Figure 7.

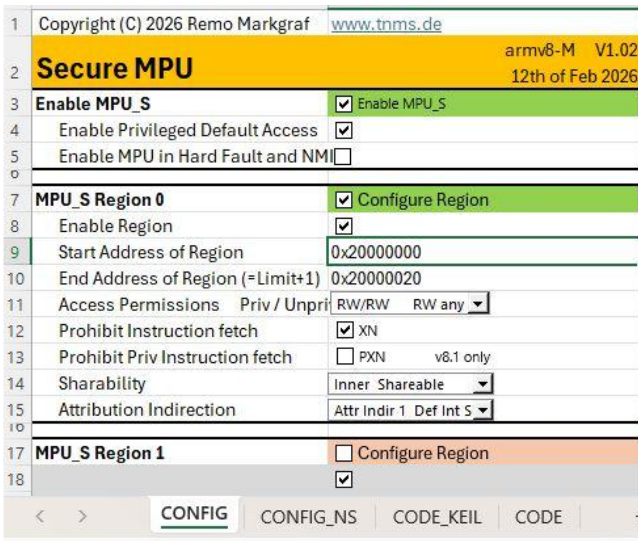
Armv8-M MPU Configurator

The Armv8-M MPU Configurator (see Figure 8) differs from the v7-M in:

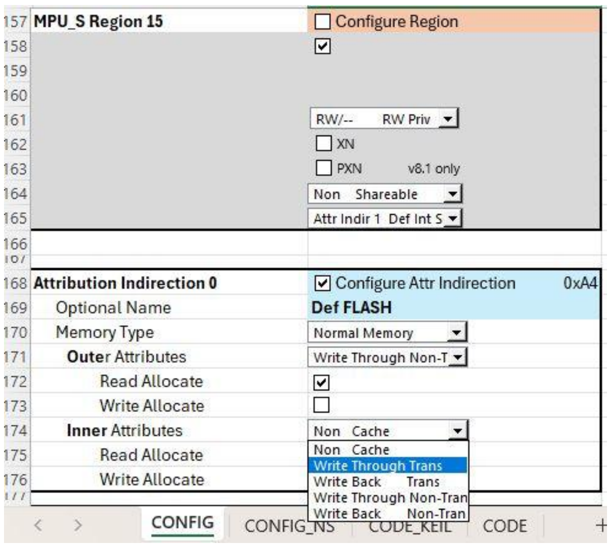
- Additional sheet CONFIG_NS for the non-secure MPU
- Changed Regions Parameters
- Graphical settings for the Attribution Indirection Register below the regions

```
#include "board.h"              //depends on your environment
#include "armv7m_mpu_config.h"  //resp. "armv8m_mpu_config.h"
#include "armv7m_mpu_info.h"    //resp. "armv8m_mpu_info.h"

int main(void)
{
        //
        // initialize board
        //

        // check working printout
        const char* str="Hello World :-)\r\n";
        while(*str != '\0')
                putchar(*str++);

        MPU_Init();
        MPU_Info();

        // loop forever
        while( 1)
        {
        }
}
```

Figure 7: Code Example for main.c



Copyright (C) 2026 Remo Markgraf	www.tnms.de
Secure MPU	armv8-M V1.02 12th of Feb 2026
Enable MPU_S	✓ Enable MPU_S
Enable Privileged Default Access	✓
Enable MPU in Hard Fault and NMI	☐
MPU_S Region 0	✓ Configure Region
Enable Region	✓
Start Address of Region	0x20000000
End Address of Region (=Limit+1)	0x20000020
Access Permissions Priv / Unpri	RW/RW RW any
Prohibit Instruction fetch	✓ XN
Prohibit Priv Instruction fetch	☐ PXN v8.1 only
Sharability	Inner Shareable
Attribution Indirection	Attr Indir 1 Def Int S
MPU_S Region 1	☐ Configure Region
	✓



MPU_S Region 15	☐ Configure Region	
	✓	
	RW/-- RW Priv	
	☐ XN	
	☐ PXN v8.1 only	
	Non Shareable	
	Attr Indir 1 Def Int S	
Attribution Indirection 0	✓ Configure Attr Indirection	0xA4
Optional Name	**Def FLASH**	
Memory Type	Normal Memory	
Outer Attributes	Write Through Non-T	
Read Allocate	✓	
Write Allocate	☐	
Inner Attributes	Non Cache	
Read Allocate		
Write Allocate		

Figure 8: GUI for armv8m_MPU

Figure 9: MAIR Register settings below Regions

MPU Info

Those who do a lot make many mistakes, so it is essential to control what settings you have given to the MPU. A debugger is very suitable for this if it can display the MPU configuration properly. Unfortunately, this is not the case with most debuggers. A simple output of the settings via a terminal program or the debug console is a very helpful tool in this case. Figure 10 shows the exemplary output of relevant settings of a v7-M MPU with 8 regions. The necessary code for the MPU_Info() is also available for download free of charge and consists of the sources

for v7-M architecture

- armv7m_mpu_info.h and
- armv7m_mpu_info.c

for the v8-M architecture

- armv8m_mpu_info.h and
- armv8m_mpu_info.c

The sources are available for download on the same page as the Excel tool:

www.tnms.de/mpu

The call of the MPU_Info() function is displayed as part of the main.c Source in Figure 7. In Figure 11, in the case of the armv-8m, both the Secure and the Non-Secure settings are output if a TrustZone is present and MPU info() is called from the Secure world. Otherwise, only one MPU is output.

For output, the sprintf() function is used and the filled string buffer is output character by character via the Macro PRINTCHR. For easy migration, the line is located at the top of the mpu_info.c source

#define PRINTCHR(c) putchar(c)

and can be easily adapted to the respective environment for adaptation.

```
*** MPU_Info START
      MPU is present
  _________________________
  |                       |
  |       MPU             |
  |_______________________|
  | MPU Settings
  |   is enabled
  |   is NOT active during HF & NMI
  |   is configured with a default Region when privileged
  | MPU Regions
  |   supports 8 Regions
  |   may overlap and need to be aligned to their size
  |   have priorities, 0 is lowest, 7 is highest priority
  |   No_E_Start______Size__XN_SRD______Pr/Un_TEX___S_C_B
  |    0 E 0x08000000  64KB -- -------- R-/R- 0b000 - C - Flash
  |    1 E 0x20000000 512B  XN -------- RW/RW 0b000 S C - Internal RAM
  |    2 E 0x48028000   2KB XN DD-DDD-D RW/RW 0b000 S - B Peripheral
  |    3 - 0x00000000   0    -- -------- --/-- 0b000 - - -
  |    4 - 0x00000000   0    -- -------- --/-- 0b000 - - -
  |    5 - 0x00000000   0    -- -------- --/-- 0b000 - - -
  |    6 - 0x00000000   0    -- -------- --/-- 0b000 - - -
  |    7 - 0x00000000   0    -- -------- --/-- 0b000 - - -
  |________________________
--- MPU_Info END
```

Figure 10: Example Output of MPU_Info() for Armv7-M M0+, M3, M4, M7

```
*** MPU_Info START
      MPU is present
  _________________________
  |                        |
  | SECURE MPU_S           |
  |________________________|
  | MPU_S Settings
  |   is enabled
  |   is NOT active during HF & NMI
  |   is configured with a default Region when privileged
  | MPU_S Regions
  |   supports 8 Regions
  |   may NOT overlap and need to be aligned to 32 byte
  |   have NO priorities
  |   No_E_Start______Limit______XN_PXN_Pr/Un_Share_A
  |    0 E 0x20000000 0x2000001F XN --- RW/RW Inner 1
  |    1 - 0x00000000 0x0000001F -- --- RW/-- none  0
  |    2 - 0x00000000 0x0000001F -- --- RW/-- none  0
  |    3 - 0x00000000 0x0000001F -- --- RW/-- none  0
  |    4 - 0x00000000 0x0000001F -- --- RW/-- none  0
  |    5 - 0x00000000 0x0000001F -- --- RW/-- none  0
  |    6 - 0x00000000 0x0000001F -- --- RW/-- none  0
  |    7 - 0x00000000 0x0000001F -- --- RW/-- none  0
  | MPU_S Attribution Indirection
  |   No_Val__Type___OuterCache_InnerCache
  |    0 0xAA Normal WT_nT_RA   WT_nT_RA
  |    1 0xAA Normal WT_nT_RA   WT_nT_RA
  |    2 0x00 Device nGnRnE
  |    3 0x00 Device nGnRnE
  |    4 0x00 Device nGnRnE
  |    5 0x00 Device nGnRnE
  |    6 0x00 Device nGnRnE
  |    7 0x00 Device nGnRnE
  |_________________________
  _________________________
  |                        |
  | NON-SECURE MPU_NS      |
  |________________________|
  | MPU_NS Settings
  |   is NOT enabled
  |   is NOT active during HF & NMI
  |   is configured with NO default Region when privileged
  | MPU_NS Regions
  |   supports 8 Regions
  |   may NOT overlap and need to be aligned to 32 byte
  |   have NO priorities
  |   No_E_Start______Limit______XN_PXN_Pr/Un_Share_A
  |    0 - 0x00000000 0x0000001F -- --- RW/-- none  0
  |    1 - 0x00000000 0x0000001F -- --- RW/-- none  0
  |    2 - 0x00000000 0x0000001F -- --- RW/-- none  0
  |    3 - 0x00000000 0x0000001F -- --- RW/-- none  0
  |    4 - 0x00000000 0x0000001F -- --- RW/-- none  0
  |    5 - 0x00000000 0x0000001F -- --- RW/-- none  0
  |    6 - 0x00000000 0x0000001F -- --- RW/-- none  0
  |    7 - 0x00000000 0x0000001F -- --- RW/-- none  0
  | MPU_NS Attribution Indirection
  |   No_Val__Type___OuterCache_InnerCache
  |    0 0xAA Normal WT_nT_RA   WT_nT_RA
  |    1 0x00 Device nGnRnE
  |    2 0x00 Device nGnRnE
  |    3 0x00 Device nGnRnE
  |    4 0x00 Device nGnRnE
  |    5 0x00 Device nGnRnE
  |    6 0x00 Device nGnRnE
  |    7 0x00 Device nGnRnE
  |_________________________
--- MPU_Info END
```

Figure 11: Example Output of MPU_Info() for Armv8-M M23, M33, M35P, M52, M55, M85

Hi, my name is Remo Markgraf and I train and consult extensively in projects related to Cortex-M, embedded software development and testing, test-driven development and agile development for embedded systems. Training and Consulting is provided in German and English language. My experience covers numerous areas of software development, test engineering, system architecture, project, product, lifecycle and business management and of course Live und Online Trainings www.tnms.de