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

Die Memory Protection Unit oder kurz MPU ist ein ungeliebtes Kind, obwohl sie sehr wichtig und hilfreich ist. Der Grund für die Missgunst liegt in der Natur der Sache, die MPU ist nicht konstruktiv, sondern deckt Fehlverhalten der Software auf und verhindert Schlimmeres. Sie sorgt nicht dafür, dass SW plötzlich funktioniert, sondern eher im Gegenteil. Es bedarf genauer Planung und komplizierter Konfiguration, um SW unter Einsatz der MPU wieder zum Laufen zu bringen, die ohne MPU lief. Also warum soll ich mir das antun? Ganz einfach, weil die MPU die SW sicherer und stabiler macht und weil die MPU in sicherheitsrelevanten Anwendungen Pflicht ist. Damit die komplizierte Konfiguration und deren Kontrolle einfach funktioniert, werden im Folgenden zwei kostenfreie Tools zur Vereinfachung vorgestellt. Einmal ein Excel basierter graphischer MPU Code Generator und ein C-Code Modul, MPU-Info, das die Einstellungen der MPU ausgibt, da viele Debugger dies leider nicht können. Mit der Einführung der Armv8-M Architektur wurde auch eine neue Protected Memory Software Architecture PMSAv8 eingeführt, die dazu führt, dass die MPU der v8-M nicht rückwärtskompatibel zur v7-M ist. Die unterschiedlichen MPUs machen die Sache natürlich zusätzlich komplexer, … ungeliebtes Kind eben. Keine Angst, die beiden Tools erleichtern das Leben, da wir nicht in die einzelnen Registerdetails abtauchen müssen

Memory Protection Unit

Die MPU ist eine im Arm Core integrierte Hardware Komponente, die im Bussystem zwischen CPU und Memorycontroller sitz und den Zugriff überwacht. Wann immer der Core eine Adresse liest oder schreibt wird dieser Zugriff durch die MPU kontrolliert. Das gilt für Zugriffe auf Flash, RAM, Peripherie, intern wie extern und selbst für das Tightly Coupled Memory. Die Cache-Controller werden sogar über die MPU orchestriert, d.h. über die MPU wird festgelegt, welche Speicherbereiche gecached werden und welche Cache Policy jeweils angewendet wird. Die MPU wird über Regionen programmiert.

Regionen der MPU

Regionen sind Speicherbereiche für welche Zugriffsrechte und Speicherverhalten festgelegt werden. Sollte ein Zugriff des Cores nicht durch eine passende Region erlaubt sein, dann wird ein Memory Fault ausgelöst. Das bedeutet, der Prozessor führt eine Exception, ähnlich eines Interrupts aus, in der eine Fehlerbehandlung ausgeführt und i.d.R. auch ein Reset ausgelöst wird. Da die Zugriffsrechte nicht nur Lesen und Schreiben unterscheiden, sondern auch getrennt für den privilegierten und unprivilegierten Zugriff festgelegt werden, lassen sich so sehr praktische Szenarien realisieren, in denen z.B. verhindert

werden kann, dass Applikationen auf OS-Daten zugreifen, oder ein Stacküberlauf abgefangen wird.

MPU Konfiguration

Um die MPU an den jeweiligen Anwendungsfall anzupassen, müssen also diese Regionen programmiert werden. Dazu stehen für jede Region zwei 32-bit Register zur Verfügung. In einem 3. Register, dem Region-Number-Register wird festgelegt welche Region aktuell konfiguriert werden kann. Dazu werden die beiden Register der ausgewählten Region immer an das gleiche Adressenpaar gemappt. In einem 4. nur lesbaren Register, dem MPU-Type-Register lässt sich auslesen, wie viele Regionen ein Controller besitz. Das sind in der v7-M Architektur 0, 8 oder für manche Cortex-M7 auch mal 16 Regionen. In der v8-M Architektur können das 0, 4, 8, 12 oder 16 Regionen sein. Zusätzlich hat ein v8-M Controller mit Trust-Zone auch zwei MPUs und damit zwei Sätze an Registern, einmal für den Secure und einmal für den Non-Secure Bereich. Die verschiedenen Bits innerhalb der Register sind sehr unterschiedlich zwischen der v7-M und v8-M Architektur und auch nicht zueinander kompatibel. Diese Details würden den Rahmen eines Blog-Beitrags sprengen. Das eingangs erwähnte Excel Tool zur Konfiguration der MPU Register erledigt diese Detailarbeit für uns. Wir können also bequem über die graphische Oberfläche des Excel Tools die Regionen konfigurieren und erhalten als Ergebnis C-Code, der die Regionen mit den gewählten Werten besetzt. Da in der v8-M Architektur die 2 x 32-Bits der Register pro Region nicht ausreichend waren, da die flexiblere Definition der Regionenadressen zu viel Platz braucht, wurden zusätzliche Attribution-Indirection-Register MAIR hinzugefügt. Das sind 8x 8-Bit Register, mit Index 0-7, die in zwei 32-Bit Register zusammengefasst sind. Jede Region beinhaltet in 3 Bits einen solchen Index und „zeigt“ damit auf eines der Attribution Indirections. Aber keine Sorge, auch diese Konfiguration wird über das Excel Tool übernommen.
Dadurch sind zwei unterschiedliche Excel Tools erforderlich, eines für die

- v7-M, also Cortex-M0+, M3. M4. M7 und ein zweites für die
- v8-M, also Cortex-M23, M33, M35P, M52, M55 und M85.

EXCEL MPU Configurator

Über die Links

- www.tnms.de/mpu oder
- www.tnms.de und dem Menueintrag Downloads->MPU Configurator

stehen die beiden Versionen des Excel Tools kostenfrei zum Download bereit. Eingabewerte zur Konfiguration der MPU werden in dem Excel Sheet „CONFIG“ eingegeben. Excel erlaubt die Eingabe von Werten für 16 Regionen, auch wenn nur weniger auf dem Chip realisiert sind. Gemäß Arm Spezifikation ist das Verhalten der MPU undefiniert, wenn Werte für nicht vorhandene Regionen konfiguriert werden. Daher wird in dem generierten Macro **`MPU_CONFIG_REGION(num)`** zur Laufzeit überprüft, ob die jeweilige Region existiert und nur dann werden die eingegebenen Werte auch in die MPU programmiert, dadurch wird ein potentiell undefinierter Zustand vermieden.

```
#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));\
  }
```

Bild 1: MPU_CONFIG_REGION Macro in Sheet CODE bzw. armv7m_mpu_config.h

In Bild 1 ist der MPU_CONFIG_REGION Macro abgebildet, der für jede Region aufgerufen wird, wenn das Define CONFIG_REGn auf 1 gesetzt ist und die MPU mindestens n Regionen besitzt. Das Define wird über die in Bild2 dargestellte graphische Oberfläche durch die erste Checkbox „Configure Region“ für die jeweiligen Region gesetzt, in Bild2 in Zelle C7 grün bzw. in Zelle C26 rot hinterlegt.

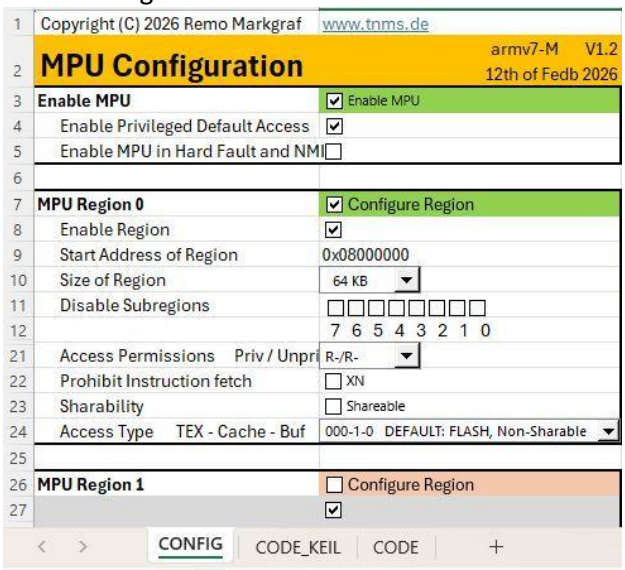

Bild 2: Graphische Oberfläche für armv7m_MPU

Als letzter Eintrag einer Region ist ein Wert für „TEX Cache Buf“ einzugeben, der zur Steuerung eines potentiell vorhandenen Cache benötigt wird. Da in der armv7-M Architektur ein Cache nur für den M7 vorgesehen ist müssen Sie sich bei Projekten für den M0+, M3, M4 nicht mit diesen komplexeren Einstellungen herumärgern. Die von Arm für einfachere Fälle veröffentlichten Defaults sind über die oberen Einträge das Pulldown-Menus einfach einstellbar, um die Konfiguration zu erleichtern (siehe Bild 3).

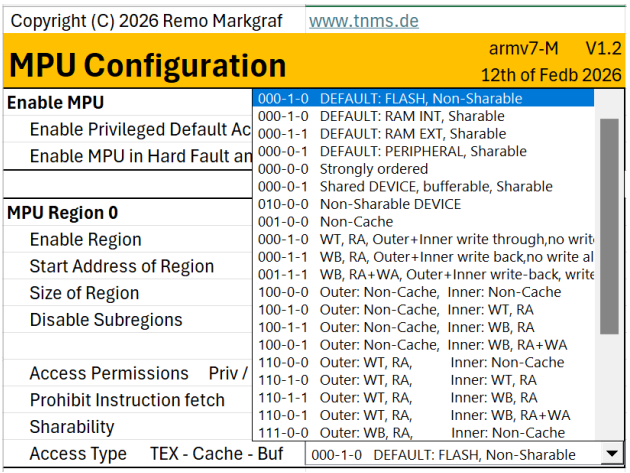

Bild 3: Pulldown für TEX-C-B mit Defaults

```
// 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)
```

Bild 4: In Sheet CODE gesetzte Defines für Region 0

In Bild 4 sind die Defines einer Region abgebildet, deren Werte durch die graphische Oberfläche gesetzt werden. Die Werte werden in dem bereits erwähnten MPU_CONFIG_REGION Macro verwendet und die MPU dadurch konfiguriert. Die MPU_CONFIG_REGION Macros werden von der in Bild 5 dargestellten inline Funktion MPU_Init() aufgerufen.

```
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)
```

Bild 5: In Sheet CODE generierte Funktion MPU_Init()

Das Sheet CODE_KEIL enthält die gleichen Defines und Macros, nur zusätzlich Kommentarzeilen, welche den in Keil µVision im Editor integrierten graphischen Editor steuern. Damit ist es möglich ohne das Excel Tool im Projekt die Werte der Defines über den Editor graphisch zu editieren. Die vielen Kommentarzeilen machen den Code sehr unübersichtlich und daher gibt es das Sheet CODE, das Sie verwenden können, wenn Sie nicht mit Keil µVision arbeiten. Bild 6 zeigt beispielhaft, wie der graphischen Editor für armv7m_mpu_config_keil.h in Keil µVision aussieht.

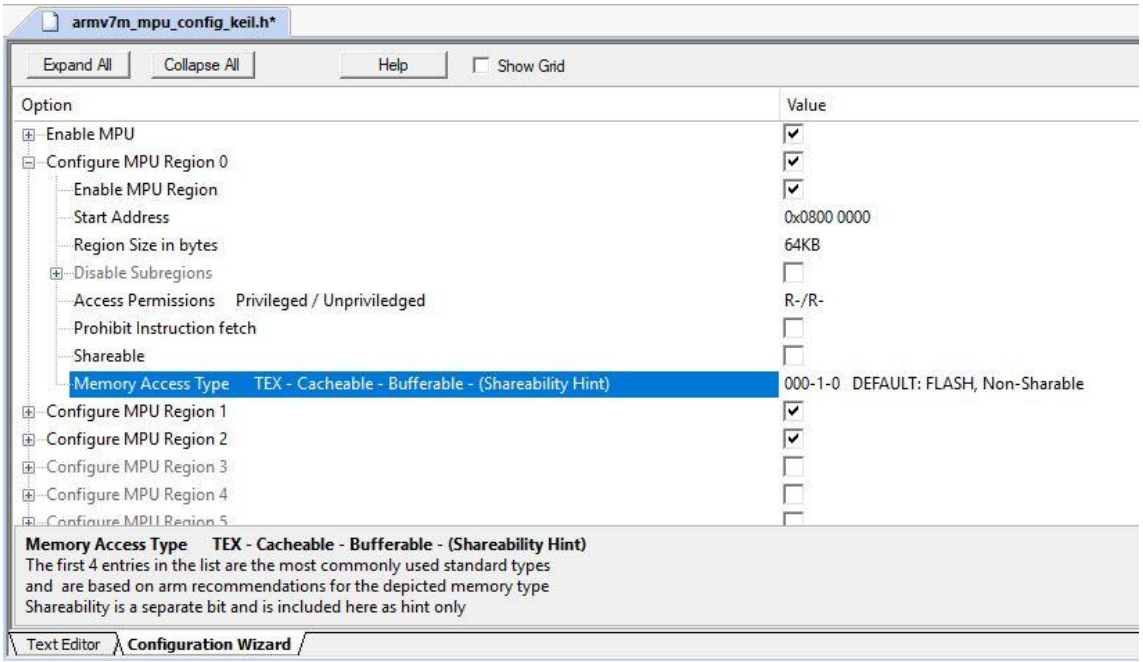

Bild 6: armv7m_mpu_config_keil.h in Keil µVision

Um den Excel Configurator zu nutzen geben Sie also die Werte für die benötigten Regionen in Excel ein, kopieren den gesamten Inhalt des Sheets CODE oder CODE_Keil in den Editor Ihrer Entwicklungsumgebung und nennen die Datei z.B. armv7m_mpu_config.h. Diese inkludieren Sie dann in Ihrem main.c und rufen innerhalb der main() die Funktion MPU_Init() auf. Ein Beispiel für eine solche main() Funktion ist in Bild 7 dargestellt

Armv8-M MPU Configurator

Der Armv8-M MPU Configurator (siehe Bild 8) unterscheidet sich gegenüber der v7-M in:

- Zusätzliches Sheet CONFIG_NS für die Non-Secure MPU
- Geänderte Regionen Parameter
- Graphische Einstellungen für den Attribution Indirection Bereich unterhalb der Regionen

```
1       #include "board.h"                //depends on your environment
2       #include "armv7m_mpu_config.h"    //resp. "armv8m_mpu_config.h"
3       #include "armv7m_mpu_info.h"      //resp. "armv8m_mpu_info.h"
4
5       int main(void)
6       {
7               //
8               // initialize board
9               //
10
11              // check working printout
12              const char* str="Hello World :-)\r\n";
13              while(*str != '\0')
14                      putchar(*str++);
15
16              MPU_Init();
17              MPU_Info();
18
19              // loop forever
20              while( 1)
21              {
22              }
23      }
```

Bild 7: Code Beispiel main.c

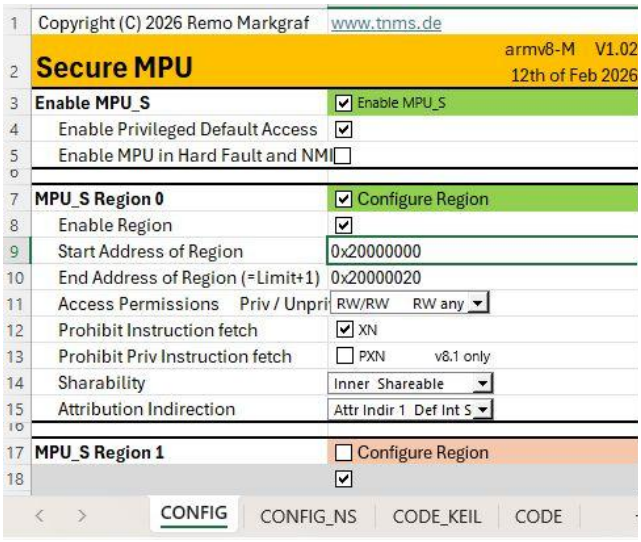

Bild 8: Graphische Oberfläche für armv8m_MPU

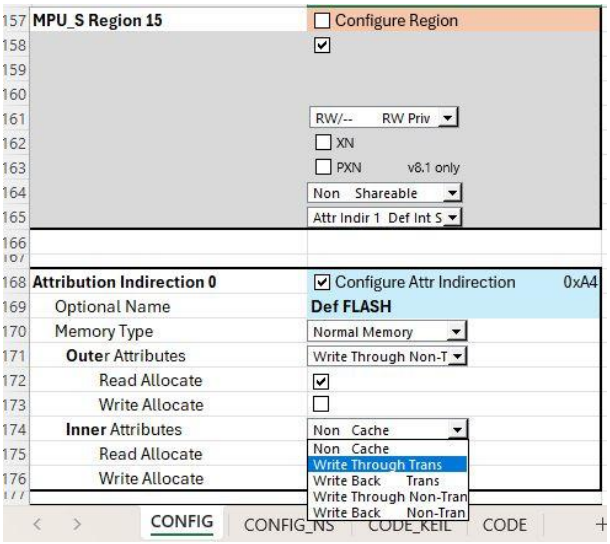

Bild 9: MAIR Bereich unter den Regionen

MPU Info

Wer viel macht, macht viele Fehler, daher ist es unerlässlich zu kontrollieren welche Einstellungen Sie der MPU mitgegeben haben. Dazu eignet sich sehr gut ein Debugger, wenn er denn die MPU Konfiguration halbwegs vernünftig anzeigen kann. Leider ist das bei den meisten Debuggern nicht der Fall. In der Not ist eine einfache Ausgabe der Einstellungen über ein Terminalprogramm oder die Debug Console ein sehr hilfreiches Mittel. Bild 10 zeigt die beispielhafte Ausgabe aller relevanten Einstellungen einer v7-M MPU mit 8 Regionen. Der nötige Code zur Ausgabe steht ebenfalls kostenfrei zum Download bereit und besteht aus den beiden Sourcen

- `armv7m_mpu_info.h` und
- `armv7m_mpu_info.c`
  für die v7-M Architektur und
- `armv8m_mpu_info.h` und
- `armv8m_mpu_info.c`
  für die v8-M Architektur.

Die Sourcen stehen auf der gleichen Seite wie das Excel Tool zum Download bereit:

www.tnms.de/mpu

In der Abbildung der main.c Source in Bild 7 ist der Aufruf der MPU_Info() Funktion abgebildet. Wie in Bild 11 dargestellt werden im Falle der armv-8m sowohl die Secure als auch die Non-Secure Einstellungen ausgegeben, sofern eine TrustZone vorhanden ist und MPU-Info() aus der Secure Welt heraus aufgerufen wird. Ansonsten wird nur eine MPU ausgegeben.

Zur Ausgabe wird die sprintf() Funktion verwendet und der gefüllte Stringbuffer zeichenweise über den Macro PRINTCHR ausgegeben. Zur leichten Migration ist oben in der mpu_info.c Source die Zeile
`#define PRINTCHR(c) putchar(c)`
enhalten und kann zur Anpassung an die jeweilige Umgebung leicht angepasst werden.

```
*** 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
```

Bild 10: Ausgabe der MPU_Info() Funktion der 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
```

Bild 11: Ausgabe der MPU_Info() Funktion der Armv8-M M23, M33, M35P, M52, M55, M85

Viel Spaß mit den Cortex-Mx-ern 😊 .

Hallo, ich heiße Remo Markgraf und biete intensive Trainings und Consulting in Projekten rund um Cortex-M, Embedded Software Entwicklung und Test, Test-Driven Development und agile Entwicklung von Embedded Systemen. Training und Consulting ist auf Deutsch und Englisch möglich. Ich verfüge über Erfahrung und Kenntnisse in verschiedensten Bereichen der Software Entwicklung, Test Entwurf und Umsetzung, System Architekturen, Projekt-, Produkt-, Lifecycle- und Business-Management und natürlich Live und Online Schulungen. www.tnms.de