Question: LWIP/ V6.5.0. STMH32H735_DISCO (https://github.com/stm32-hotspot/STM32H7-LwIP-Examples) .Rx_PoolSection. (former .RxArraySection) seems not to be guarded by the MPU against cache coherency issues. Is this the right way?

I have compared 2 STMH32H735_DISCO_Eth examples, one built under cube IDE V6.2.1 and a more recent one built with IDE V6.5.0. (thanks to Pavel A. for providing the reference to the new examples on GIT).

The V6.5.0. example moved the Rx_PoolSection/RxArraySection from AHB D2 RAM to AXI D1 RAM. The MPU configuration in the example seems not to protect the Rx_PoolSection/RxArraySection area from cache coherency problems any more. Can such a protection be omitted and if so why?

Analyzing the “OLD” V6.2.1. STMH32H735_DISCO_Eth example:

Ethernetif.c: (LINE 90)

```
#elif defined ( __GNUC__ ) /* GNU Compiler */

ETH_DMADescTypeDef DMARxDscrTab[ETH_RX_DESC_CNT]
__attribute__((section(".RxDecripSection"))); /* Ethernet Rx DMA Descriptors */
ETH_DMADescTypeDef DMATxDscrTab[ETH_TX_DESC_CNT]
__attribute__((section(".TxDecripSection")));   /* Ethernet Tx DMA Descriptors */
uint8_t Rx_Buff[ETH_RX_DESC_CNT][ETH_RX_BUFFER_SIZE]
__attribute__((section(".RxArraySection"))); /* Ethernet Receive Buffers */

#endif
```

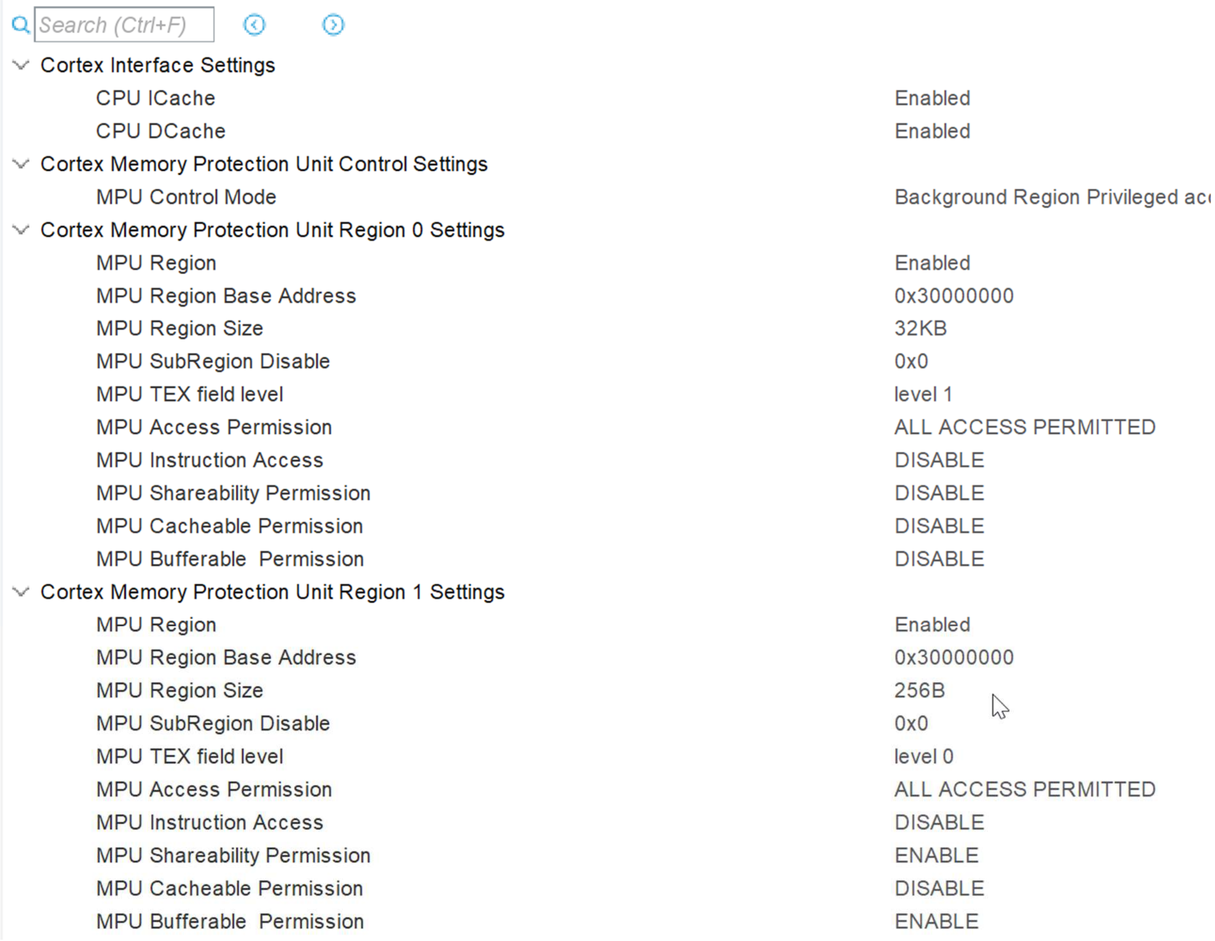
This combined with STM32H735IGKX.FLASH.ld (line 164)

```
  .lwip_sec (NOLOAD) : {
    . = ABSOLUTE(0x30000000);
    *(.RxDecripSection)

    . = ABSOLUTE(0x30000060);
    *(.TxDecripSection)

    . = ABSOLUTE(0x30000200);
    *(.RxArraySection)
  } >RAM_D2
```

Leads to:

.RxArraySection in RAM_D2 @0x30000200.

The section is guarded by the MPU settings:



Cortex Interface Settings	
CPU ICache	Enabled
CPU DCache	Enabled
Cortex Memory Protection Unit Control Settings	
MPU Control Mode	Background Region Privileged ac
Cortex Memory Protection Unit Region 0 Settings	
MPU Region	Enabled
MPU Region Base Address	0x30000000
MPU Region Size	32KB
MPU SubRegion Disable	0x0
MPU TEX field level	level 1
MPU Access Permission	ALL ACCESS PERMITTED
MPU Instruction Access	DISABLE
MPU Shareability Permission	DISABLE
MPU Cacheable Permission	DISABLE
MPU Bufferable Permission	DISABLE
Cortex Memory Protection Unit Region 1 Settings	
MPU Region	Enabled
MPU Region Base Address	0x30000000
MPU Region Size	256B
MPU SubRegion Disable	0x0
MPU TEX field level	level 0
MPU Access Permission	ALL ACCESS PERMITTED
MPU Instruction Access	DISABLE
MPU Shareability Permission	ENABLE
MPU Cacheable Permission	DISABLE
MPU Bufferable Permission	ENABLE

And, given the priority of the MPU region settings (last ref prevails), we end up with Region 0;

Level 1: TEX 1, C=0, B=0, which makes the region where the Rx buffers sit a normally ordered, non cacheable area. This is (IMHO) what it should be as the STM32H7 ETH DMA writes its data telegrams received from the PHY into these buffers from where they are further processed by LWIP.

Analyzing the “NEW” V6.5.0. STMH32H735_DISCO_Eth example:

The V6.5.0. STMH32H735_DISCO_Eth (https://github.com/stm32-hotspot/STM32H7-LwIP-Examples) is different and I did not find a similar mechanism that protects these receive buffers against caching related problems.

Ethernetif.c (line 109)

```
#elif defined ( __GNUC__ ) /* GNU Compiler */

ETH_DMADescTypeDef DMARxDscrTab[ETH_RX_DESC_CNT]
__attribute__((section(".RxDecripSection"))); /* Ethernet Rx DMA Descriptors */
ETH_DMADescTypeDef DMATxDscrTab[ETH_TX_DESC_CNT]
__attribute__((section(".TxDecripSection")));   /* Ethernet Tx DMA Descriptors */

#endif
```

Ethernetif.c (line 144)

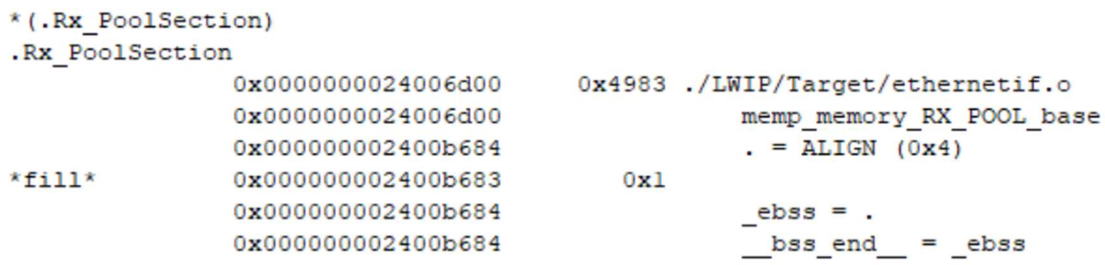
```
#elif defined ( __GNUC__ ) /* GNU Compiler */
__attribute__((section(".Rx_PoolSection"))) extern u8_t
memp_memory_RX_POOL_base[];
#endif
```

STM32H735IGKX_FLASH.LD (line 137)

```
  /* Uninitialized data section */
  . = ALIGN(4);
  .bss :
  {
    /* This is used by the startup in order to initialize the .bss secion */
    _sbss = .;         /* define a global symbol at bss start */
    __bss_start__ = _sbss;
    *(.bss)
    *(.bss*)
    *(COMMON)

    /* ETH_CODE: add placement of RX buffer. STM32H72x/H73x has small D2 RAM, so
we need to put it there.
     * (NOLOAD) attribute used for .bss section to avoid linker warning (.bss
initialized by startup code)
     */
      . = ALIGN(32);
    *(.Rx_PoolSection)

    . = ALIGN(4);
    _ebss = .;         /* define a global symbol at bss end */
    __bss_end__ = _ebss;
  } >RAM_D1
```

There is an .Rx_PoolSection that ends up in RAM_D1:

STM32H735_Disco_Eth.map (line 26999)



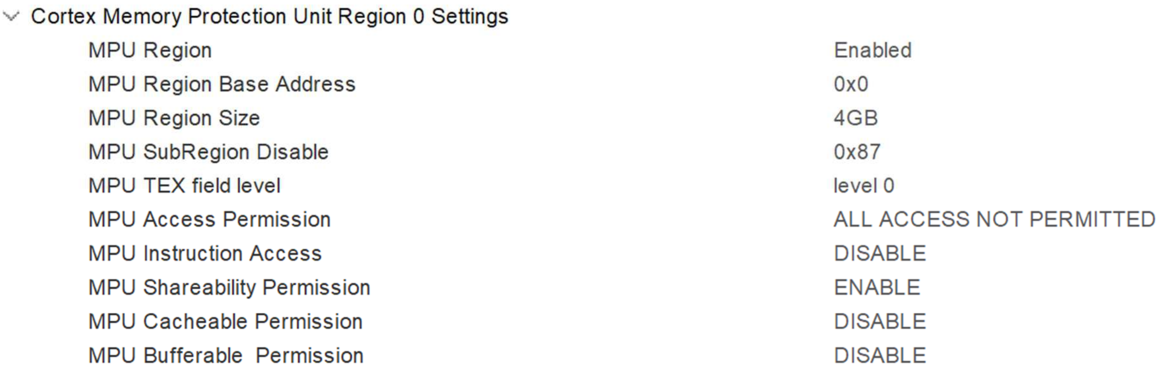
```
*(.Rx_PoolSection)
.Rx_PoolSection
                0x0000000024006d00     0x4983 ./LWIP/Target/ethernetif.o
                0x0000000024006d00                memp_memory_RX_POOL_base
                0x000000002400b684                . = ALIGN (0x4)
*fill*          0x000000002400b683        0x1
                0x000000002400b684                _ebss = .
                0x000000002400b684                __bss_end__ = _ebss
```

According to RM0468 Table 6. “Memory map and default device memory area attributes”

This is AXI SRAM. This makes sense as DMA ETH fills these buffers.

However, and this is my point: I do not see how MPU protects this area from caching issues (or is this not required):

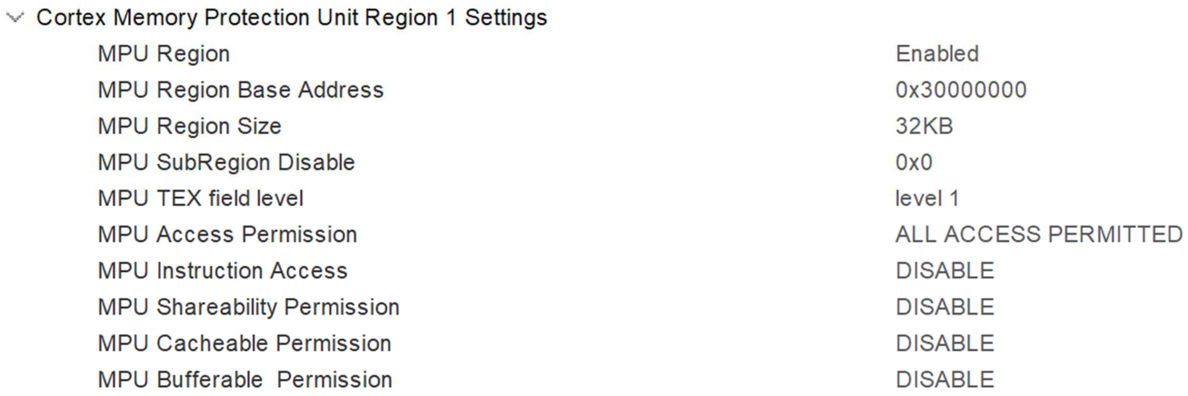
The .IOC mpu configuration:



Cortex Memory Protection Unit Region 0 Settings

MPU Region	Enabled
MPU Region Base Address	0x0
MPU Region Size	4GB
MPU SubRegion Disable	0x87
MPU TEX field level	level 0
MPU Access Permission	ALL ACCESS NOT PERMITTED
MPU Instruction Access	DISABLE
MPU Shareability Permission	ENABLE
MPU Cacheable Permission	DISABLE
MPU Bufferable Permission	DISABLE

Region 0, as I understand it, blocks the memory space related to External RAM and External device RAM. (SubRegion spec 0x87 excludes the lower 1.5GB and the top 0.5GB (code + SRAM + Peripheral, vendor specific memory)). Hence the bottom 1.5GB and the top 0.5GB are available for use.

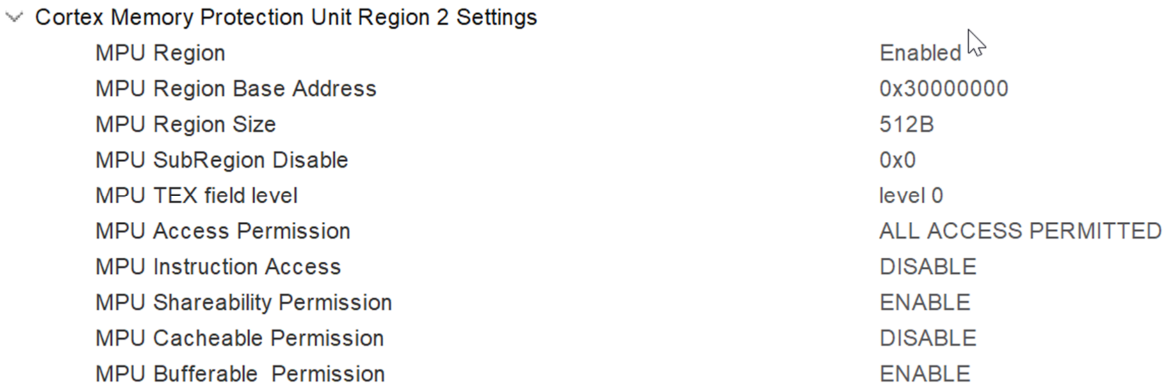


Cortex Memory Protection Unit Region 1 Settings

MPU Region	Enabled
MPU Region Base Address	0x30000000
MPU Region Size	32KB
MPU SubRegion Disable	0x0
MPU TEX field level	level 1
MPU Access Permission	ALL ACCESS PERMITTED
MPU Instruction Access	DISABLE
MPU Shareability Permission	DISABLE
MPU Cacheable Permission	DISABLE
MPU Bufferable Permission	DISABLE

Region 1: protects 32 KB above 0x3000 0000, in the same sense as mentioned above.

Level 1: This is the same as in the first example, however the RX buffers have moved to another area 0x24006d00 and are no longer located at 0x3000 0200 as in the LWIP example mentioned in the first part of this post.



Cortex Memory Protection Unit Region 2 Settings

MPU Region	Enabled
MPU Region Base Address	0x30000000
MPU Region Size	512B
MPU SubRegion Disable	0x0
MPU TEX field level	level 0
MPU Access Permission	ALL ACCESS PERMITTED
MPU Instruction Access	DISABLE
MPU Shareability Permission	ENABLE
MPU Cacheable Permission	DISABLE
MPU Bufferable Permission	ENABLE

Region 2: Is there to protect the descriptors, related to this part. This part seems fine to me.

So in essence, if the DMA updates the receive buffers, I see no mechanism that prevents the MCU from using its cache with possibly outdated data (dating from before the update by the DMA) instead of what is in the AXI ram. Possibly I did overlook something?