**DMA is not working on STM32H7 devices, or the transmitted/received data are corrupted. Polling and interrupt based methods for the same peripheral configuration are working.**

Answer

The problem is related two things: memory layout on STM32H7 and internal data cache (D-Cache) of the Cortex-M7 core.

In summary these can be the possible issues:

* Memory placed in DTCM RAM for D1/D2 peripherals. Unfortunately this memory is used as default in some projects including examples.
* Memory not placed in D3 SRAM4 for D3 peripherals.
* D-Cache enabled for DMA buffers, different content in cache and in SRAM memory.
* Starting the DMA just after writing the data to TX buffer, without placing \_\_DSB() instruction between.

For Ethernet related problems, please see separate FAQ: [FAQ: Ethernet not working on STM32H7x3](https://community.st.com/s/article/FAQ-Ethernet-not-working-on-STM32H7x3)

**1. Explanation: memory layout**

The STM32H7 device consists of three bus matrix domains (D1, D2 and D3) as seen on the picture below. The D1 and D2 are connected through bus bridges, both can also access data in D3 domain. However there is no connection from D3 domain to D1 or D2 domain.

The **DMA1** and **DMA2** controllers are located in D2 domain and can access almost all memories with exception of ITCM and DTCM RAM (located at 0x20000000). This DMA is used in most cases.

**BDMA** controller is located in D3 domain and can access only SRAM4 and backup SRAM in the D3 domain.

**MDMA** controller is located in D1 domain and can access all memories, including ITCM/DTCM. This controller is mainly used for handling D1 peripherals and memory to memory transfers.

Diagram, schematic

Description automatically generated

From performance point of view it is better to put DMA buffers inside D2 domain (SRAM1, SRAM2 and SRAM3), since the D2-to-D1 bridge can add additional delay.

**2. Explanation: handling DMA buffers with D-Cache enabled**

The Cortex-M7 contains two internal caches, I-Cache for loading instructions and D-Cache for data. The D-Cache can affect the functionality of DMA transfers, since it will hold the new data in the internal cache and don't write them to the SRAM memory. However the DMA controller loads the data from SRAM memory and not D-Cache.

In case the DMA transfer is started just after writing the data to the tx\_buffer in the code, it can happen that the tx\_buffer data will be still in write-buffer inside the CPU, while the DMA is already started. Solution can be to set the tx\_buffer as device type and force CPU to order the memory operations, or add \_\_DSB() instruction before starting the DMA.

There are several ways how to keep manage DMA buffers with D-Cache:

* Disable D-Cache globally. It is the most simple solution, but not effective one, since you can loose great part of performance. Can be useful for debugging, to analyze if the problem is related to D-Cache.
* Disable D-Cache for part of the memory. This can be done by configuring the memory protection unit (MPU). The downside is that the MPU regions have certain alignment restrictions and you need to place the DMA buffers to specific parts of memory. Each toolchain (GCC, IAR, KEIL) needs to be configured in different way.
  + Note that MPU regions can overlap and the higher region number has priority. Together with subregion disable bits, this can be useful to soften the alignment and size restrictions.
  + Note that Device and Strongly ordered memory types not allow unaligned access to the memory.
* Configure part of memory as write-through. Can be used only for TX DMA. Similar to the previous option.
* Use cache maintenance operations. It is possible to write data stored in cache back to memory ("clean" operation) for specific address range, and also discard data stored in cache ("invalidate" operation).
  + The downside is that these operations work withe cache-line size which is 32 bytes, so you can't clean or invalidate single byte from the cache. This can lead to errors when RX buffer "shares" the cache line with other data or TX buffer (please see the picture below).
  + Beware that with uninitialized D-Cache, the maintenance operations "clean" or "clean and invalidate" can lead to BusFault exception. This is caused by uninitialized ECC (error correction code) after power-on reset. If you have project with a lot of maintenance operations and want to disable D-Cache temporarily, you can use SCB\_InvalidateDCache function, which will clean the cache and set correct ECC, without enabling it.

Diagram

Description automatically generated

Below are the possible MPU configurations. Green are configurations suitable for DMA buffers, blue is suitable only for TX-only DMA buffer and red are forbidden. Other configurations are not suitable for DMA buffers and will require cache maintenance oprations:

| **TEX** | **Cacheable** | **Bufferable** | **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 |

**3. Solution example 1: Simple placement of all memory to D1 domain**

***D-Cache must be disabled globally for this solution to work.***

**GCC (Atollic TrueStudio/System Workbench for STM32/Eclipse)**

Replace DTCMRAM with RAM\_D1 for section placement in linkerscript (.ld file extension). E.g. like this:

.data :

{

  ... /\* Keep same \*/

} >RAM\_D1 AT> FLASH

this should be done also for .bss and .\_user\_heap\_stack sections.

In some linkerscripts, the initial stack is defined separately. So you either need to update it with the section, or define it inside the section like:

.\_user\_heap\_stack :

{

. = ALIGN(8);

PROVIDE ( end = . );

PROVIDE ( \_end = . );

. = . + \_Min\_Heap\_Size;

. = . + \_Min\_Stack\_Size;

\_estack = .; /\* <<<< line added \*/

. = ALIGN(8);

} >RAM\_D1

and remove the original \_estack definition.

**IAR (in project settings):**

Graphical user interface, application

Description automatically generated

**For Keil:**

Graphical user interface

Description automatically generated

**4. Solution example 2: Placing buffers in separated memory part**

***D-Cache must be disabled via MPU for that particular memory region, where DMA buffer is placed. Please note that MPU region size must be in power of two. Also the regions start address must have same alignment as size. E.g. if the regions is 512 bytes, the start address must be aligned to 512 bytes (9 LSBs must be zero).***

NOTE: IAR compiler and Keil compiler version <= 5 allow placing variables at absolute address in code using compiler specific extensions.

**C code:**

Define placement macro:

#if defined( \_\_ICCARM\_\_ )

  #define DMA\_BUFFER \

      \_Pragma("location=\".dma\_buffer\"")

#else

  #define DMA\_BUFFER \

      \_\_attribute\_\_((section(".dma\_buffer")))

#endif

Specify DMA buffers in code:

DMA\_BUFFER uint8\_t rx\_buffer[256];

**GCC linkerscript (\*.ld file)**

Place section to D2 RAM (you can also specify your own memory regions in linkerscript file):

.dma\_buffer : /\* Space before ':' is critical \*/

{

  \*(.dma\_buffer)

} >RAM\_D2

This is without default value initialization. Otherwise you need to place special symbols and add your own initialization code.

**IAR linker file (\*.icf file)**

define region D2\_SRAM2\_region   = mem:[from 0x30020000 to 0x3003FFFF];

place in D2\_SRAM2\_region { section .dma\_buffer};

initialize by copy { section .dma\_buffer}; /\* optional initialization of default values \*/

**Keil scatter file (\*.sct file)**

LR\_IROM1 0x08000000 0x00200000  {    ; load region size\_region

  ER\_IROM1 0x08000000 0x00200000  {  ; load address = execution address

   \*.o (RESET, +First)

   \*(InRoot$$Sections)

   .ANY (+RO)

  }

  RW\_IRAM2 0x24000000 0x00080000  {  ; RW data

   .ANY (+RW +ZI)

  }

  ; Added new region

  DMA\_BUFFER 0x30040000 0x200 {

  \*(.dma\_buffer)

  }

}

Generation of scatter file should be disabled in Keil:

Graphical user interface, text, application

Description automatically generated

**5. Solution example 3: Use Cache maintenance functions**

Transmitting data:

#define TX\_LENGTH  (16)

uint8\_t tx\_buffer[TX\_LENGTH];

/\* Write data \*/

tx\_buffer[0] = 0x0;

tx\_buffer[1] = 0x1;

/\* Clean D-cache \*/

/\* Make sure the address is 32-byte aligned and add 32-bytes to length, in case it overlaps cacheline \*/

SCB\_CleanDCache\_by\_Addr((uint32\_t\*)(((uint32\_t)tx\_buffer) & ~(uint32\_t)0x1F), TX\_LENGTH+32);

/\* Start DMA transfer \*/

HAL\_UART\_Transmit\_DMA(&huart1, tx\_buffer, TX\_LENGTH);

Receiving data:

#define RX\_LENGTH  (16)

uint8\_t rx\_buffer[RX\_LENGTH];

/\* Invalidate D-cache before reception \*/

/\* Make sure the address is 32-byte aligned and add 32-bytes to length, in case it overlaps cacheline \*/

SCB\_InvalidateDCache\_by\_Addr((uint32\_t\*)(((uint32\_t)rx\_buffer) & ~(uint32\_t)0x1F), RX\_LENGTH+32);

/\* Start DMA transfer \*/

HAL\_UART\_Receive\_DMA(&huart1, rx\_buffer, RX\_LENGTH);

/\* No access to rx\_buffer should be made before DMA transfer is completed \*/

Please note that in case of reception there can be problem if rx\_buffer is not aligned to the size of cache-line (32-bytes), because during the invalidate operation another data sharing the same cache-line(s)  with rx\_buffer can be lost.

**6. References**

* "AN4838: Managing memory protection unit (MPU) in STM32 MCUs"

<https://www.st.com/content/ccc/resource/technical/document/application_note/group0/bc/2d/f7/bd/fb/3f/48/47/DM00272912/files/DM00272912.pdf/jcr:content/translations/en.DM00272912.pdf>

* "AN4839: Level 1 cache on STM32F7 Series and STM32H7 Series":

<https://www.st.com/content/ccc/resource/technical/document/application_note/group0/08/dd/25/9c/4d/83/43/12/DM00272913/files/DM00272913.pdf/jcr:content/translations/en.DM00272913.pdf>

* "AN4296: Overview and tips for using STM32F303/328/334/358xx CCM RAM with IAR EWARM, Keil MDK-ARM and GNU-based toolchains":

<https://www.st.com/content/ccc/resource/technical/document/application_note/bb/09/ca/83/14/e9/44/c5/DM00083249.pdf/files/DM00083249.pdf/jcr:content/translations/en.DM00083249.pdf>

* "AN4891: STM32H7x3 system architecture and performance software expansion for STM32Cube":

<https://www.st.com/content/ccc/resource/technical/document/application_note/group0/0d/b5/e7/b7/47/0c/4a/ae/DM00306681/files/DM00306681.pdf/jcr:content/translations/en.DM00306681.pdf>