Laboratorio di Architetture e Programmazione dei Sistemi Elettronici Industriali

Prof. Luca Benini <luca.benini@unibo.it>

Simone Benatti <simone.benatti@unibo.it>

Filippo Casamassima<filippo.casamassima@unibo.it>

#7 DMA

DMA

- Direct memory access (DMA) is used in order to provide **high-speed data transfer between peripherals and memory** as well as **memory to memory**. Data can be quickly moved by DMA without any CPU actions. <u>**This keeps CPU resources free for other operations.**</u>

- The two DMA controllers have 12 channels in total (7 for DMA1 and 5 for DMA2), **each dedicated to managing memory access requests** from one or more peripherals. It has an arbiter for handling the priority between DMA requests.

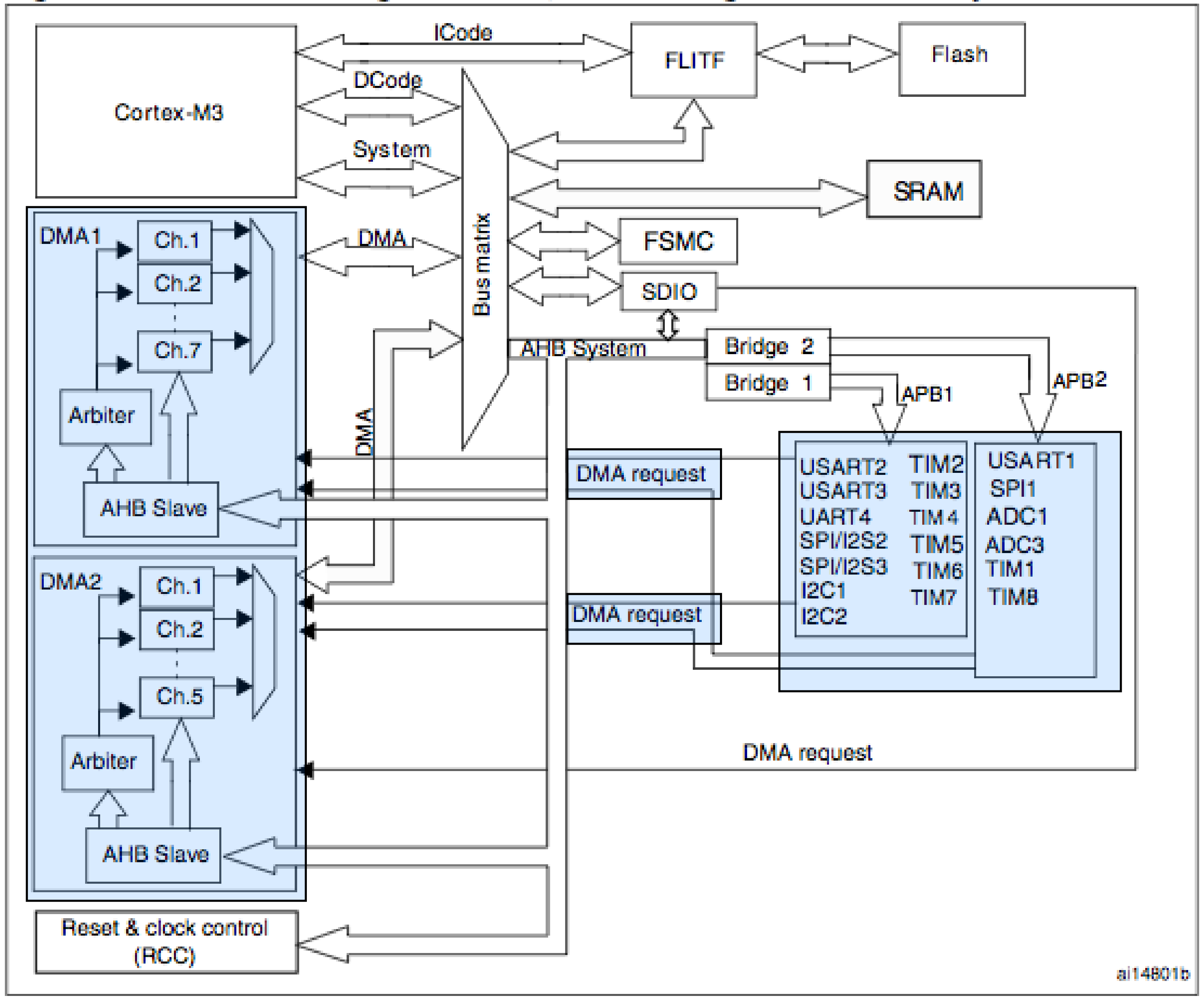
- DMA main features:
  - Each of the 12 channels is connected to dedicated hardware DMA requests, software trigger is also supported on each channel. This **configuration is done by software**.
  - Priorities between requests from channels of one DMA are software programmable (**4 levels** consisting of very high, high, medium, low) or hardware in case of equality (request 1 has priority over request 2, etc.)
  - **Independent source and destination transfer size** (byte, half word, word), emulating packing and unpacking. Source/destination addresses must be aligned on the data size.
  - **3 event flags** (<u>DMA Half Transfer, DMA Transfer complete and DMA Transfer Error</u>) logically ORed together in a single interrupt request for each channel
  - **Memory-to-memory transfer**
  - **Peripheral-to-memory and memory-to-peripheral, and peripheral-to-peripheral transfers**

DMA: **DMA transactions**

- After an event, the **peripheral sends a request signal to the DMA Controller**. The DMA controller serves the request depending on the channel priorities.

- The DMA channels can also work without being triggered by a request from a peripheral. This mode is called **Memory to Memory mode**.

Figure 23. DMA block diagram in low-, medium- high- and XL-density devices

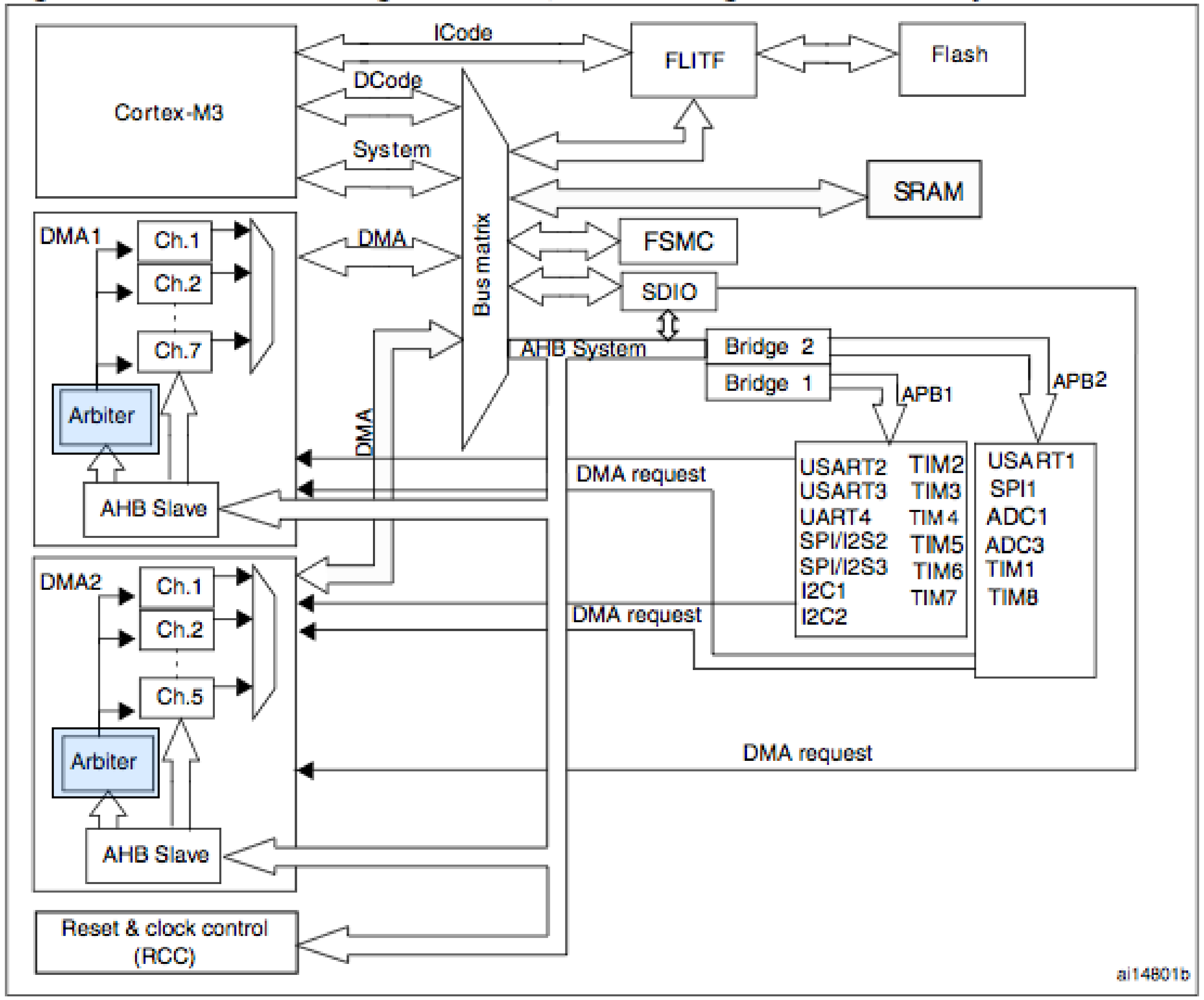


DMA: **Arbiter**

- The arbiter **manages the channel requests** based on their priority and launches the peripheral/memory access sequences.

- The priorities are managed in two stages:
  - **Software**
  - **Hardware** (if 2 requests have the same software priority level, the channel with the lowest number will get priority versus the channel with the highest number.)

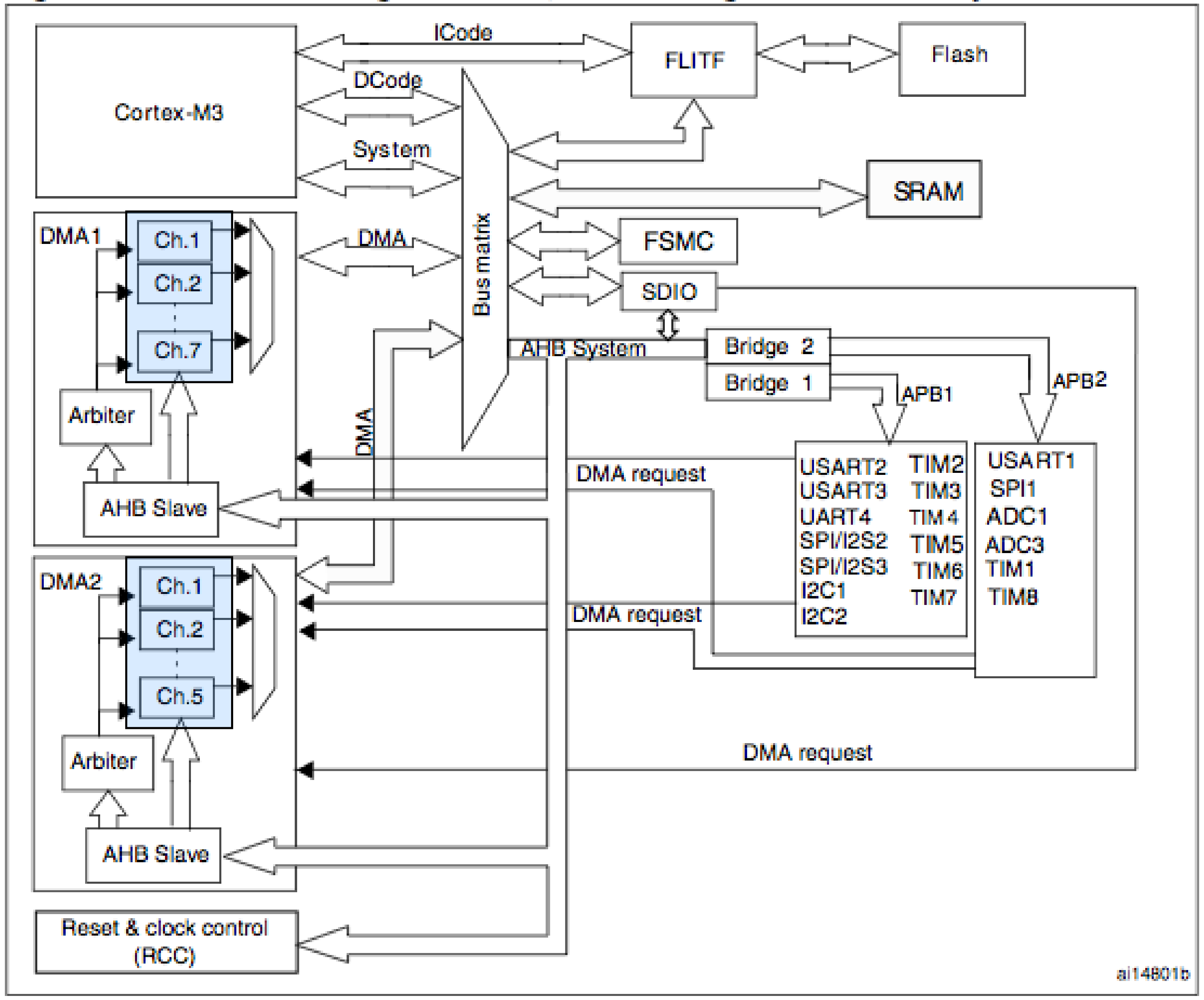
Figure 23. DMA block diagram in low-, medium- high- and XL-density devices



DMA: **DMA channels**

- Each channel can handle DMA transfer **between a peripheral register** located at a fixed address **and a memory address**.
- Peripheral and memory pointers can optionally be **automatically post-incremented** after each transaction.
- If incremented mode is enabled, the **address of the next transfer will be the address of the previous one incremented by 1, 2 or 4** depending on the chosen data size.

Figure 23. DMA block diagram in low-, medium- high- and XL-density devices

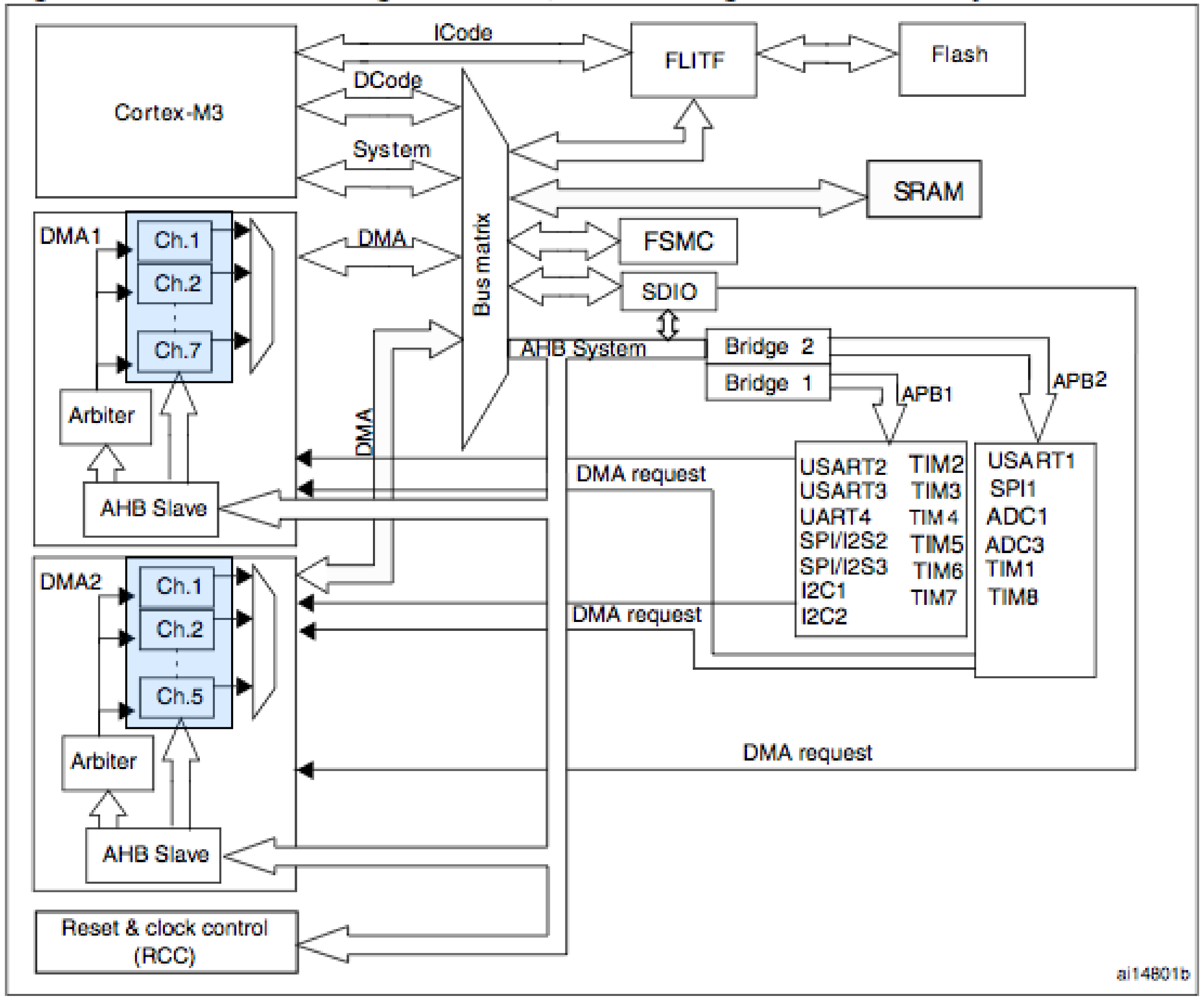


DMA: **Circular mode**

- If the channel is configured in **noncircular mode**, no DMA request is served after the last transfer (that is once the number of data items to be transferred has reached zero).

- **Circular mode** is available to handle **circular buffers and continuous data** flows (e.g. ADC scan mode). When circular mode is activated, the number of data to be transferred is automatically reloaded with the initial value programmed during the channel configuration phase, and the DMA requests continue to be served.

Figure 23. DMA block diagram in low-, medium- high- and XL-density devices



DMA: **Request mapping**

- The requests from the peripherals are simply logically ORed before entering the DMAx. The peripheral DMA requests can be independently activated/de-activated by **programming the DMA control bit** in the registers of the corresponding peripheral.

Figure 24. DMA1 request mapping

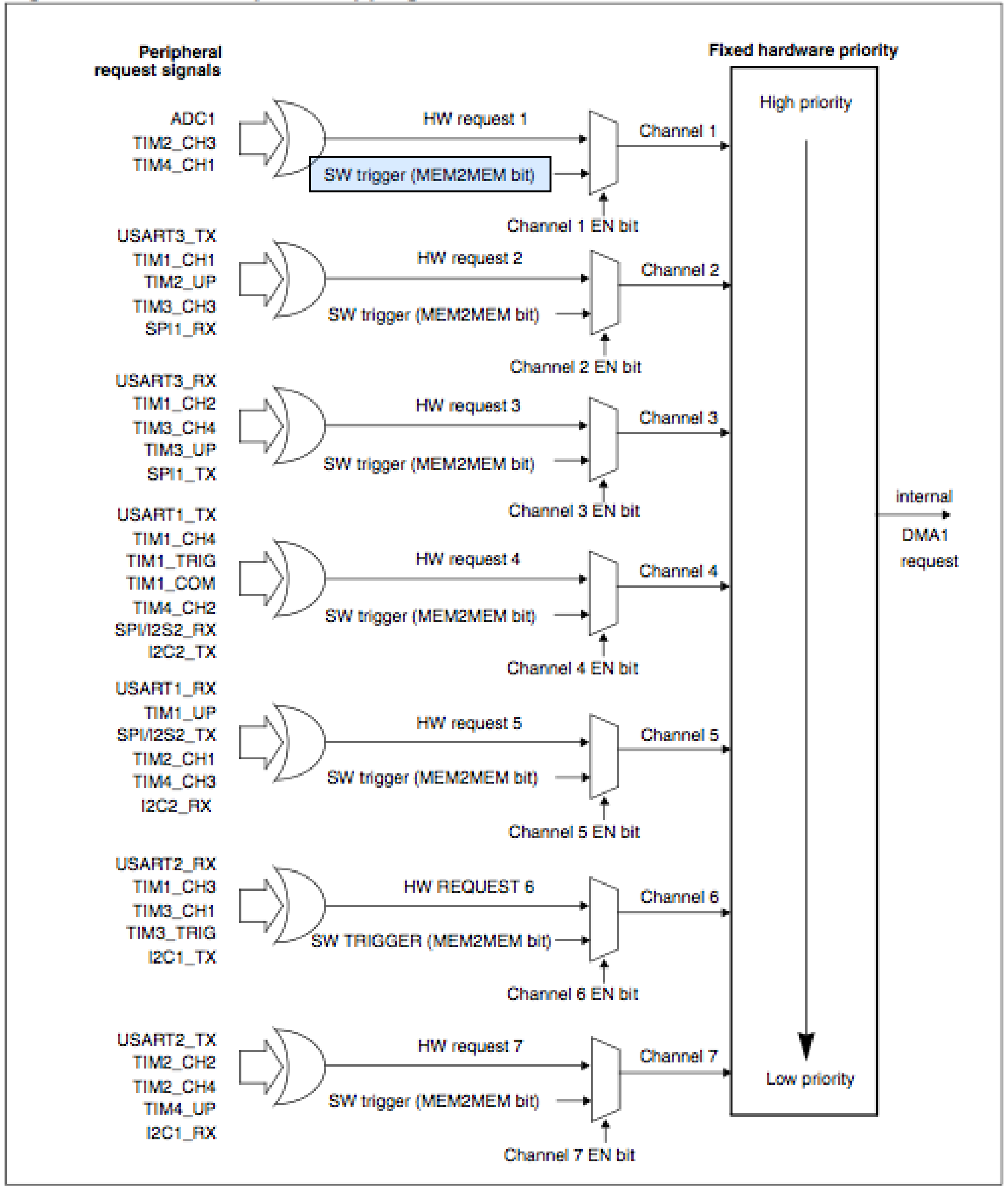
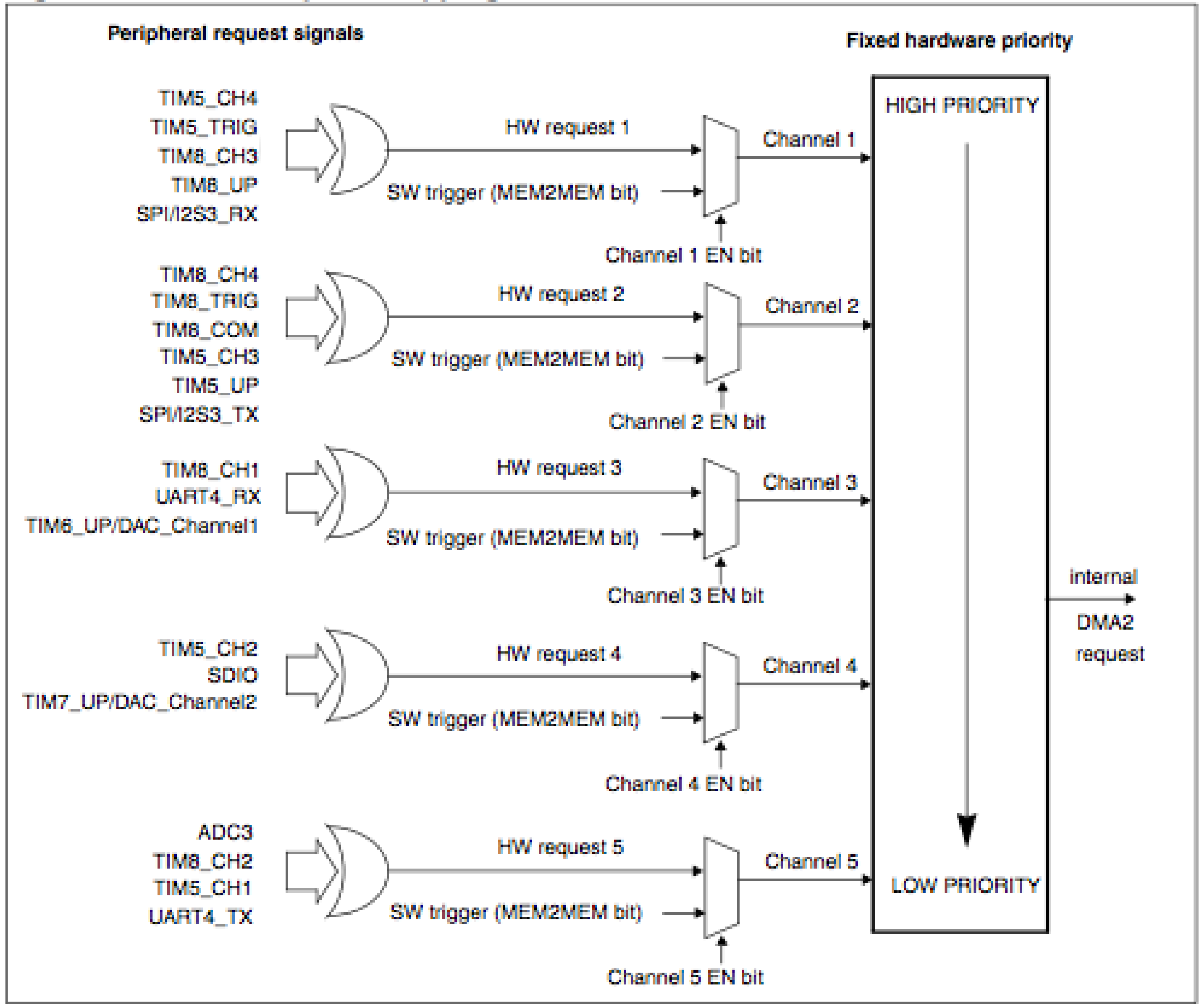


Figure 25. DMA2 request mapping



DMA (**what**)

- We want to use a **DMA** channel to **transfer a word data buffer from FLASH memory to embedded SRAM memory.**

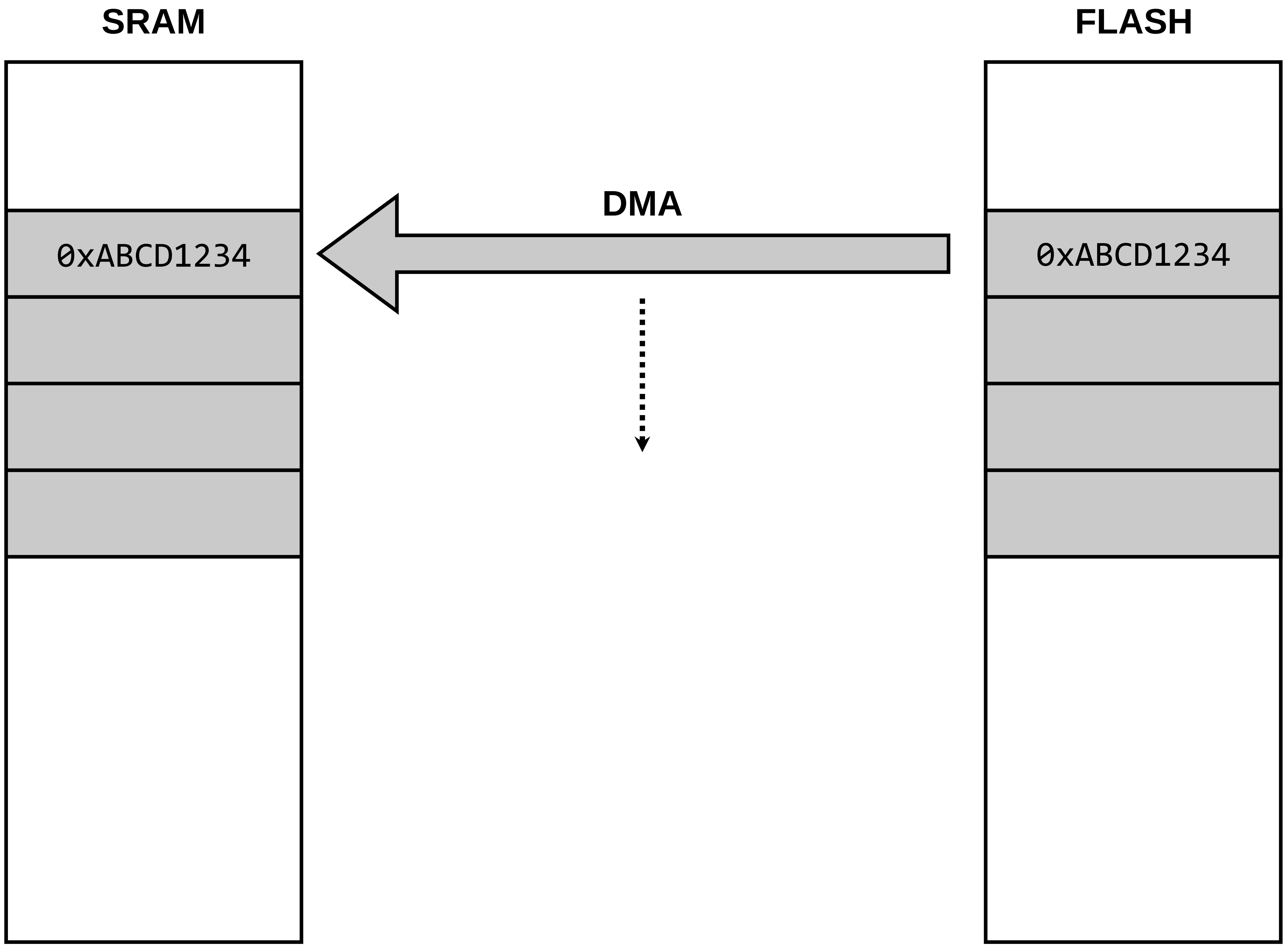
  Enable the clock for AHB BUS

  Configure NVIC

  Configure and enable DMA

  - DMA1 Channel6 is configured to transfer the contents of a 32-word data buffer stored in Flash memory to the reception buffer declared in RAM.
  - The start of transfer is triggered by software. DMA1 Channel6 **memory-to-memory transfer** is **enabled**. Source and destination **addresses incrementing** is also **enabled**.
  - The transfer is **started** by setting the Channel **enable bit** for DMA1 Channel6.
  - At the end of the transfer a Transfer Complete **interrupt is generated** since it is enabled. Once interrupt is generated, the **remaining data** to be transferred is read which must be equal to 0.
  - The Transfer Complete Interrupt **pending bit** is then **cleared**.

DMA (**what**)



DMA (**how**)

Enable the clock for AHB BUS

```
void RCC_Configuration(void)
{
    /* Enable peripheral clocks */
    /* Enable DMA1 clock */
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
}
```

Configure NVIC

```
void NVIC_Configuration(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;
    /* Enable DMA1 channel6 IRQ Channel */
    NVIC_InitStructure.NVIC_IRQChannel = DMA1_Channel6_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

DMA (**how**)

Configure and enable DMA

```
typedef enum {FAILED = 0, PASSED = !FAILED} TestStatus;
#define BufferSize  32
DMA_InitTypeDef  DMA_InitStructure;
```

As usual a **struct** is used for the configuration

DMA (**how**)

Configure and enable DMA

```
typedef enum {FAILED = 0, PASSED = !FAILED} TestStatus;
#define BufferSize  32
DMA_InitTypeDef  DMA_InitStructure;
uint16_t CurrDataCounterBegin = 0, CurrDataCounterEnd = 0x01;
TestStatus TransferStatus = FAILED;
```

Data **counters** (to track the process down)

DMA (**how**)

Configure and enable DMA

```
typedef enum {FAILED = 0, PASSED = !FAILED} TestStatus;
#define BufferSize  32
DMA_InitTypeDef  DMA_InitStructure;
uint16_t CurrDataCounterBegin = 0, CurrDataCounterEnd = 0x01;
TestStatus TransferStatus = FAILED;
const uint32_t SRC_Const_Buffer[BufferSize]= {
      0x01020304,0x05060708,0x090A0B0C,0x0D0E0F10,
      0x11121314,0x15161718,0x191A1B1C,0x1D1E1F20,
      0x21222324,0x25262728,0x292A2B2C,0x2D2E2F30,
      0x31323334,0x35363738,0x393A3B3C,0x3D3E3F40,
      0x41424344,0x45464748,0x494A4B4C,0x4D4E4F50,
      0x51525354,0x55565758,0x595A5B5C,0x5D5E5F60,
      0x61626364,0x65666768,0x696A6B6C,0x6D6E6F70,
      0x71727374,0x75767778,0x797A7B7C,0x7D7E7F80
};
uint32_t DST_Buffer[BufferSize];
```

Source and destinations buffers

DMA (**how**)

Configure and enable DMA

```
typedef enum {FAILED = 0, PASSED = !FAILED} TestStatus;
#define BufferSize  32
DMA_InitTypeDef  DMA_InitStructure;
uint16_t CurrDataCounterBegin = 0, CurrDataCounterEnd = 0x01;
TestStatus TransferStatus = FAILED;
const uint32_t SRC_Const_Buffer[BufferSize]= {
     0x01020304,0x05060708,0x090A0B0C,0x0D0E0F10,
     0x11121314,0x15161718,0x191A1B1C,0x1D1E1F20,
     0x21222324,0x25262728,0x292A2B2C,0x2D2E2F30,
     0x31323334,0x35363738,0x393A3B3C,0x3D3E3F40,
     0x41424344,0x45464748,0x494A4B4C,0x4D4E4F50,
     0x51525354,0x55565758,0x595A5B5C,0x5D5E5F60,
     0x61626364,0x65666768,0x696A6B6C,0x6D6E6F70,
     0x71727374,0x75767778,0x797A7B7C,0x7D7E7F80
};
uint32_t DST_Buffer[BufferSize];

...

/* DMA1 channel6 configuration */
DMA_DeInit(DMA1_Channel6);
DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)SRC_Const_Buffer;
DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)DST_Buffer;
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
DMA_InitStructure.DMA_BufferSize = BufferSize;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Enable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
DMA_InitStructure.DMA_M2M = DMA_M2M_Enable;
DMA_Init(DMA1_Channel6, &DMA_InitStructure);
```

Specifies the **peripheral** base (source) address for DMA1 Channel6

DMA (**how**)

Configure and enable DMA

```c
typedef enum {FAILED = 0, PASSED = !FAILED} TestStatus;
#define BufferSize  32
DMA_InitTypeDef  DMA_InitStructure;
uint16_t CurrDataCounterBegin = 0, CurrDataCounterEnd = 0x01;
TestStatus TransferStatus = FAILED;
const uint32_t SRC_Const_Buffer[BufferSize]= {
     0x01020304,0x05060708,0x090A0B0C,0x0D0E0F10,
     0x11121314,0x15161718,0x191A1B1C,0x1D1E1F20,
     0x21222324,0x25262728,0x292A2B2C,0x2D2E2F30,
     0x31323334,0x35363738,0x393A3B3C,0x3D3E3F40,
     0x41424344,0x45464748,0x494A4B4C,0x4D4E4F50,
     0x51525354,0x55565758,0x595A5B5C,0x5D5E5F60,
     0x61626364,0x65666768,0x696A6B6C,0x6D6E6F70,
     0x71727374,0x75767778,0x797A7B7C,0x7D7E7F80
};
uint32_t DST_Buffer[BufferSize];

...

/* DMA1 channel6 configuration */
DMA_DeInit(DMA1_Channel6);
DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)SRC_Const_Buffer;
DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)DST_Buffer;
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
DMA_InitStructure.DMA_BufferSize = BufferSize;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Enable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
DMA_InitStructure.DMA_M2M = DMA_M2M_Enable;
DMA_Init(DMA1_Channel6, &DMA_InitStructure);
```

Specifies the **memory** base address for DMA1 Channel6

DMA (**how**)

Configure and enable DMA

```
typedef enum {FAILED = 0, PASSED = !FAILED} TestStatus;
#define BufferSize  32
DMA_InitTypeDef  DMA_InitStructure;
uint16_t CurrDataCounterBegin = 0, CurrDataCounterEnd = 0x01;
TestStatus TransferStatus = FAILED;
const uint32_t SRC_Const_Buffer[BufferSize]= {
     0x01020304,0x05060708,0x090A0B0C,0x0D0E0F10,
     0x11121314,0x15161718,0x191A1B1C,0x1D1E1F20,
     0x21222324,0x25262728,0x292A2B2C,0x2D2E2F30,
     0x31323334,0x35363738,0x393A3B3C,0x3D3E3F40,
     0x41424344,0x45464748,0x494A4B4C,0x4D4E4F50,
     0x51525354,0x55565758,0x595A5B5C,0x5D5E5F60,
     0x61626364,0x65666768,0x696A6B6C,0x6D6E6F70,
     0x71727374,0x75767778,0x797A7B7C,0x7D7E7F80
};
uint32_t DST_Buffer[BufferSize];

...

/* DMA1 channel6 configuration */
DMA_DeInit(DMA1_Channel6);
DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)SRC_Const_Buffer;
DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)DST_Buffer;
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
DMA_InitStructure.DMA_BufferSize = BufferSize;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Enable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
DMA_InitStructure.DMA_M2M = DMA_M2M_Enable;
DMA_Init(DMA1_Channel6, &DMA_InitStructure);
```

Specifies if the peripheral is the source or destination

DMA (**how**)

Configure and enable DMA

```
typedef enum {FAILED = 0, PASSED = !FAILED} TestStatus;
#define BufferSize  32
DMA_InitTypeDef  DMA_InitStructure;
uint16_t CurrDataCounterBegin = 0, CurrDataCounterEnd = 0x01;
TestStatus TransferStatus = FAILED;
const uint32_t SRC_Const_Buffer[BufferSize]= {
     0x01020304,0x05060708,0x090A0B0C,0x0D0E0F10,
     0x11121314,0x15161718,0x191A1B1C,0x1D1E1F20,
     0x21222324,0x25262728,0x292A2B2C,0x2D2E2F30,
     0x31323334,0x35363738,0x393A3B3C,0x3D3E3F40,
     0x41424344,0x45464748,0x494A4B4C,0x4D4E4F50,
     0x51525354,0x55565758,0x595A5B5C,0x5D5E5F60,
     0x61626364,0x65666768,0x696A6B6C,0x6D6E6F70,
     0x71727374,0x75767778,0x797A7B7C,0x7D7E7F80
};
uint32_t DST_Buffer[BufferSize];

...

/* DMA1 channel6 configuration */
DMA_DeInit(DMA1_Channel6);
DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)SRC_Const_Buffer;
DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)DST_Buffer;
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
DMA_InitStructure.DMA_BufferSize = BufferSize;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Enable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
DMA_InitStructure.DMA_M2M = DMA_M2M_Enable;
DMA_Init(DMA1_Channel6, &DMA_InitStructure);
```

Specifies the **buffer size**, in **data unit**, of the specified channel

DMA (**how**)

Configure and enable DMA

```
typedef enum {FAILED = 0, PASSED = !FAILED} TestStatus;
#define BufferSize  32
DMA_InitTypeDef  DMA_InitStructure;
uint16_t CurrDataCounterBegin = 0, CurrDataCounterEnd = 0x01;
TestStatus TransferStatus = FAILED;
const uint32_t SRC_Const_Buffer[BufferSize]= {
      0x01020304,0x05060708,0x090A0B0C,0x0D0E0F10,
      0x11121314,0x15161718,0x191A1B1C,0x1D1E1F20,
      0x21222324,0x25262728,0x292A2B2C,0x2D2E2F30,
      0x31323334,0x35363738,0x393A3B3C,0x3D3E3F40,
      0x41424344,0x45464748,0x494A4B4C,0x4D4E4F50,
      0x51525354,0x55565758,0x595A5B5C,0x5D5E5F60,
      0x61626364,0x65666768,0x696A6B6C,0x6D6E6F70,
      0x71727374,0x75767778,0x797A7B7C,0x7D7E7F80
};
uint32_t DST_Buffer[BufferSize];

...

/* DMA1 channel6 configuration */
DMA_DeInit(DMA1_Channel6);
DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)SRC_Const_Buffer;
DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)DST_Buffer;
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
DMA_InitStructure.DMA_BufferSize = BufferSize;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Enable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
DMA_InitStructure.DMA_M2M = DMA_M2M_Enable;
DMA_Init(DMA1_Channel6, &DMA_InitStructure);
```

Specifies whether the **Peripheral** address register is **incremented** or **not**.

DMA (**how**)

Configure and enable DMA

```
typedef enum {FAILED = 0, PASSED = !FAILED} TestStatus;
#define BufferSize  32
DMA_InitTypeDef  DMA_InitStructure;
uint16_t CurrDataCounterBegin = 0, CurrDataCounterEnd = 0x01;
TestStatus TransferStatus = FAILED;
const uint32_t SRC_Const_Buffer[BufferSize]= {
      0x01020304,0x05060708,0x090A0B0C,0x0D0E0F10,
      0x11121314,0x15161718,0x191A1B1C,0x1D1E1F20,
      0x21222324,0x25262728,0x292A2B2C,0x2D2E2F30,
      0x31323334,0x35363738,0x393A3B3C,0x3D3E3F40,
      0x41424344,0x45464748,0x494A4B4C,0x4D4E4F50,
      0x51525354,0x55565758,0x595A5B5C,0x5D5E5F60,
      0x61626364,0x65666768,0x696A6B6C,0x6D6E6F70,
      0x71727374,0x75767778,0x797A7B7C,0x7D7E7F80
};
uint32_t DST_Buffer[BufferSize];

...

/* DMA1 channel6 configuration */
DMA_DeInit(DMA1_Channel6);
DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)SRC_Const_Buffer;
DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)DST_Buffer;
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
DMA_InitStructure.DMA_BufferSize = BufferSize;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Enable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
DMA_InitStructure.DMA_M2M = DMA_M2M_Enable;
DMA_Init(DMA1_Channel6, &DMA_InitStructure);
```

Specifies whether the **Memory** address register is **incremented** or **not**.

DMA (**how**)

Configure and enable DMA

```
typedef enum {FAILED = 0, PASSED = !FAILED} TestStatus;
#define BufferSize  32
DMA_InitTypeDef  DMA_InitStructure;
uint16_t CurrDataCounterBegin = 0, CurrDataCounterEnd = 0x01;
TestStatus TransferStatus = FAILED;
const uint32_t SRC_Const_Buffer[BufferSize]= {
    0x01020304,0x05060708,0x090A0B0C,0x0D0E0F10,
    0x11121314,0x15161718,0x191A1B1C,0x1D1E1F20,
    0x21222324,0x25262728,0x292A2B2C,0x2D2E2F30,
    0x31323334,0x35363738,0x393A3B3C,0x3D3E3F40,
    0x41424344,0x45464748,0x494A4B4C,0x4D4E4F50,
    0x51525354,0x55565758,0x595A5B5C,0x5D5E5F60,
    0x61626364,0x65666768,0x696A6B6C,0x6D6E6F70,
    0x71727374,0x75767778,0x797A7B7C,0x7D7E7F80
};
uint32_t DST_Buffer[BufferSize];

...

/* DMA1 channel6 configuration */
DMA_DeInit(DMA1_Channel6);
DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)SRC_Const_Buffer;
DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)DST_Buffer;
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
DMA_InitStructure.DMA_BufferSize = BufferSize;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Enable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
DMA_InitStructure.DMA_M2M = DMA_M2M_Enable;
DMA_Init(DMA1_Channel6, &DMA_InitStructure);
```

Specifies the **Peripheral** data width

DMA (**how**)

Configure and enable DMA

```
typedef enum {FAILED = 0, PASSED = !FAILED} TestStatus;
#define BufferSize  32
DMA_InitTypeDef  DMA_InitStructure;
uint16_t CurrDataCounterBegin = 0, CurrDataCounterEnd = 0x01;
TestStatus TransferStatus = FAILED;
const uint32_t SRC_Const_Buffer[BufferSize]= {
     0x01020304,0x05060708,0x090A0B0C,0x0D0E0F10,
     0x11121314,0x15161718,0x191A1B1C,0x1D1E1F20,
     0x21222324,0x25262728,0x292A2B2C,0x2D2E2F30,
     0x31323334,0x35363738,0x393A3B3C,0x3D3E3F40,
     0x41424344,0x45464748,0x494A4B4C,0x4D4E4F50,
     0x51525354,0x55565758,0x595A5B5C,0x5D5E5F60,
     0x61626364,0x65666768,0x696A6B6C,0x6D6E6F70,
     0x71727374,0x75767778,0x797A7B7C,0x7D7E7F80
};
uint32_t DST_Buffer[BufferSize];

...

/* DMA1 channel6 configuration */
DMA_DeInit(DMA1_Channel6);
DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)SRC_Const_Buffer;
DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)DST_Buffer;
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
DMA_InitStructure.DMA_BufferSize = BufferSize;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Enable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
DMA_InitStructure.DMA_M2M = DMA_M2M_Enable;
DMA_Init(DMA1_Channel6, &DMA_InitStructure);
```

Specifies the **Memory** data width

DMA (**how**)

Configure and enable DMA

```
typedef enum {FAILED = 0, PASSED = !FAILED} TestStatus;
#define BufferSize  32
DMA_InitTypeDef  DMA_InitStructure;
uint16_t CurrDataCounterBegin = 0, CurrDataCounterEnd = 0x01;
TestStatus TransferStatus = FAILED;
const uint32_t SRC_Const_Buffer[BufferSize]= {
     0x01020304,0x05060708,0x090A0B0C,0x0D0E0F10,
     0x11121314,0x15161718,0x191A1B1C,0x1D1E1F20,
     0x21222324,0x25262728,0x292A2B2C,0x2D2E2F30,
     0x31323334,0x35363738,0x393A3B3C,0x3D3E3F40,
     0x41424344,0x45464748,0x494A4B4C,0x4D4E4F50,
     0x51525354,0x55565758,0x595A5B5C,0x5D5E5F60,
     0x61626364,0x65666768,0x696A6B6C,0x6D6E6F70,
     0x71727374,0x75767778,0x797A7B7C,0x7D7E7F80
};
uint32_t DST_Buffer[BufferSize];

...

/* DMA1 channel6 configuration */
DMA_DeInit(DMA1_Channel6);
DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)SRC_Const_Buffer;
DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)DST_Buffer;
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
DMA_InitStructure.DMA_BufferSize = BufferSize;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Enable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
DMA_InitStructure.DMA_M2M = DMA_M2M_Enable;
DMA_Init(DMA1_Channel6, &DMA_InitStructure);
```

Normal mode: not a circular buffer

DMA (**how**)

Configure and enable DMA

```
typedef enum {FAILED = 0, PASSED = !FAILED} TestStatus;
#define BufferSize  32
DMA_InitTypeDef  DMA_InitStructure;
uint16_t CurrDataCounterBegin = 0, CurrDataCounterEnd = 0x01;
TestStatus TransferStatus = FAILED;
const uint32_t SRC_Const_Buffer[BufferSize]= {
     0x01020304,0x05060708,0x090A0B0C,0x0D0E0F10,
     0x11121314,0x15161718,0x191A1B1C,0x1D1E1F20,
     0x21222324,0x25262728,0x292A2B2C,0x2D2E2F30,
     0x31323334,0x35363738,0x393A3B3C,0x3D3E3F40,
     0x41424344,0x45464748,0x494A4B4C,0x4D4E4F50,
     0x51525354,0x55565758,0x595A5B5C,0x5D5E5F60,
     0x61626364,0x65666768,0x696A6B6C,0x6D6E6F70,
     0x71727374,0x75767778,0x797A7B7C,0x7D7E7F80
};
uint32_t DST_Buffer[BufferSize];

...

/* DMA1 channel6 configuration */
DMA_DeInit(DMA1_Channel6);
DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)SRC_Const_Buffer;
DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)DST_Buffer;
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
DMA_InitStructure.DMA_BufferSize = BufferSize;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Enable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
DMA_InitStructure.DMA_M2M = DMA_M2M_Enable;
DMA_Init(DMA1_Channel6, &DMA_InitStructure);
```

High Priority

DMA (**how**)

Configure and enable DMA

```
typedef enum {FAILED = 0, PASSED = !FAILED} TestStatus;
#define BufferSize  32
DMA_InitTypeDef  DMA_InitStructure;
uint16_t CurrDataCounterBegin = 0, CurrDataCounterEnd = 0x01;
TestStatus TransferStatus = FAILED;
const uint32_t SRC_Const_Buffer[BufferSize]= {
      0x01020304,0x05060708,0x090A0B0C,0x0D0E0F10,
      0x11121314,0x15161718,0x191A1B1C,0x1D1E1F20,
      0x21222324,0x25262728,0x292A2B2C,0x2D2E2F30,
      0x31323334,0x35363738,0x393A3B3C,0x3D3E3F40,
      0x41424344,0x45464748,0x494A4B4C,0x4D4E4F50,
      0x51525354,0x55565758,0x595A5B5C,0x5D5E5F60,
      0x61626364,0x65666768,0x696A6B6C,0x6D6E6F70,
      0x71727374,0x75767778,0x797A7B7C,0x7D7E7F80
};
uint32_t DST_Buffer[BufferSize];

...

/* DMA1 channel6 configuration */
DMA_DeInit(DMA1_Channel6);
DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)SRC_Const_Buffer;
DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)DST_Buffer;
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
DMA_InitStructure.DMA_BufferSize = BufferSize;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Enable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
DMA_InitStructure.DMA_M2M = DMA_M2M_Enable;
DMA_Init(DMA1_Channel6, &DMA_InitStructure);
```

Memory-to-memory transfer

DMA (**how**)

Configure and enable DMA

```
typedef enum {FAILED = 0, PASSED = !FAILED} TestStatus;
#define BufferSize  32
DMA_InitTypeDef  DMA_InitStructure;
uint16_t CurrDataCounterBegin = 0, CurrDataCounterEnd = 0x01;
TestStatus TransferStatus = FAILED;
const uint32_t SRC_Const_Buffer[BufferSize]= {
     0x01020304,0x05060708,0x090A0B0C,0x0D0E0F10,
     0x11121314,0x15161718,0x191A1B1C,0x1D1E1F20,
     0x21222324,0x25262728,0x292A2B2C,0x2D2E2F30,
     0x31323334,0x35363738,0x393A3B3C,0x3D3E3F40,
     0x41424344,0x45464748,0x494A4B4C,0x4D4E4F50,
     0x51525354,0x55565758,0x595A5B5C,0x5D5E5F60,
     0x61626364,0x65666768,0x696A6B6C,0x6D6E6F70,
     0x71727374,0x75767778,0x797A7B7C,0x7D7E7F80
};
uint32_t DST_Buffer[BufferSize];

…

/* DMA1 channel6 configuration */
DMA_DeInit(DMA1_Channel6);
DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)SRC_Const_Buffer;
DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)DST_Buffer;
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
DMA_InitStructure.DMA_BufferSize = BufferSize;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Enable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
DMA_InitStructure.DMA_M2M = DMA_M2M_Enable;
DMA_Init(DMA1_Channel6, &DMA_InitStructure);
```

Init DMA1 Channel6

DMA (**how**)

Configure and enable DMA

```
typedef enum {FAILED = 0, PASSED = !FAILED} TestStatus;
#define BufferSize  32
DMA_InitTypeDef  DMA_InitStructure;
uint16_t CurrDataCounterBegin = 0, CurrDataCounterEnd = 0x01;
TestStatus TransferStatus = FAILED;
const uint32_t SRC_Const_Buffer[BufferSize]= {
      0x01020304,0x05060708,0x090A0B0C,0x0D0E0F10,
      0x11121314,0x15161718,0x191A1B1C,0x1D1E1F20,
      0x21222324,0x25262728,0x292A2B2C,0x2D2E2F30,
      0x31323334,0x35363738,0x393A3B3C,0x3D3E3F40,
      0x41424344,0x45464748,0x494A4B4C,0x4D4E4F50,
      0x51525354,0x55565758,0x595A5B5C,0x5D5E5F60,
      0x61626364,0x65666768,0x696A6B6C,0x6D6E6F70,
      0x71727374,0x75767778,0x797A7B7C,0x7D7E7F80
};
uint32_t DST_Buffer[BufferSize];

...

/* DMA1 channel6 configuration */
DMA_DeInit(DMA1_Channel6);
DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)SRC_Const_Buffer;
DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)DST_Buffer;
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
DMA_InitStructure.DMA_BufferSize = BufferSize;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Enable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
DMA_InitStructure.DMA_M2M = DMA_M2M_Enable;
DMA_Init(DMA1_Channel6, &DMA_InitStructure);

/* Enable DMA1 Channel6 Transfer Complete interrupt */
DMA_ITConfig(DMA1_Channel6, DMA_IT_TC, ENABLE);
```

Interrupt (to detect when the transfer is complete)

DMA (**how**)

Configure and enable DMA

```
/* Get Current Data Counter value before transfer begins */
CurrDataCounterBegin = DMA_GetCurrDataCounter(DMA1_Channel6);
/* Enable DMA1 Channel6 transfer */
DMA_Cmd(DMA1_Channel6, ENABLE);
```

Enable DMA (to start transfer)

DMA (**how**)

Configure and enable DMA

```
/* Get Current Data Counter value before transfer begins */
CurrDataCounterBegin = DMA_GetCurrDataCounter(DMA1_Channel6);
/* Enable DMA1 Channel6 transfer */
DMA_Cmd(DMA1_Channel6, ENABLE);
/* Wait the end of transmission */
while (CurrDataCounterEnd != 0) { }
      while (1) { }
}
```

Wait the end of transmission

DMA (**how**)

Configure and enable DMA

```
/* Get Current Data Counter value before transfer begins */
CurrDataCounterBegin = DMA_GetCurrDataCounter(DMA1_Channel6);
/* Enable DMA1 Channel6 transfer */
DMA_Cmd(DMA1_Channel6, ENABLE);
/* Wait the end of transmission */
while (CurrDataCounterEnd != 0)  {  }
      while (1)  {  }
}
```

stm32f10x_it.c

```
extern __IO uint16_t CurrDataCounterEnd;

...

void DMA1_Channel6_IRQHandler(void){
      /* Test on DMA1 Channel6 Transfer Complete interrupt */
      if(DMA_GetITStatus(DMA1_IT_TC6))  {
            /* Get Current Data Counter value after complete transfer */
            CurrDataCounterEnd = DMA_GetCurrDataCounter(DMA1_Channel6);
            /* Clear DMA1 Channel6 Half Transfer, Transfer Complete and Global interrupt pending bits */
            DMA_ClearITPendingBit(DMA1_IT_GL6);
     }
}
```

Manage the **interrupt**

DMA (**code**)

stm32f10x_conf.h

```
…
#include “stm32f10x_dma.h”
…
```

main.c

```
include "stm32f10x.h"
typedef enum {FAILED = 0, PASSED = !FAILED} TestStatus;
#define BufferSize  32
DMA_InitTypeDef  DMA_InitStructure;
uint16_t CurrDataCounterBegin = 0, CurrDataCounterEnd = 0x01;
TestStatus TransferStatus = FAILED;
const uint32_t SRC_Const_Buffer[BufferSize]= {
    0x01020304,0x05060708,0x090A0B0C,0x0D0E0F10,
    0x11121314,0x15161718,0x191A1B1C,0x1D1E1F20,
    0x21222324,0x25262728,0x292A2B2C,0x2D2E2F30,
    0x31323334,0x35363738,0x393A3B3C,0x3D3E3F40,
    0x41424344,0x45464748,0x494A4B4C,0x4D4E4F50,
    0x51525354,0x55565758,0x595A5B5C,0x5D5E5F60,
    0x61626364,0x65666768,0x696A6B6C,0x6D6E6F70,
    0x71727374,0x75767778,0x797A7B7C,0x7D7E7F80
};
uint32_t DST_Buffer[BufferSize];

void RCC_Configuration(void);
void NVIC_Configuration(void);

int main(void)
{
    /* System Clocks Configuration */
    RCC_Configuration();
    /* NVIC configuration */
    NVIC_Configuration();
```

DMA (**code**)

```
    /* DMA1 channel6 configuration */

      DMA_DeInit(DMA1_Channel6);
      DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)SRC_Const_Buffer;
      DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)DST_Buffer;
      DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
      DMA_InitStructure.DMA_BufferSize = BufferSize;
      DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Enable;
      DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
      DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
      DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
      DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
      DMA_InitStructure.DMA_Priority = DMA_Priority_High;
      DMA_InitStructure.DMA_M2M = DMA_M2M_Enable;
      DMA_Init(DMA1_Channel6, &DMA_InitStructure);
      /* Enable DMA1 Channel6 Transfer Complete interrupt */
      DMA_ITConfig(DMA1_Channel6, DMA_IT_TC, ENABLE);
      /* Get Current Data Counter value before transfer begins */
      CurrDataCounterBegin = DMA_GetCurrDataCounter(DMA1_Channel6);
      /* Enable DMA1 Channel6 transfer */
      DMA_Cmd(DMA1_Channel6, ENABLE);
      /* Wait the end of transmission */
      while (CurrDataCounterEnd != 0)  {  }
            while (1)  {  }
      }

  void RCC_Configuration(void){
      /* Enable DMA1 clock */
      RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
  }
```

DMA (code)

```
void NVIC_Configuration(void)
{
      NVIC_InitTypeDef NVIC_InitStructure;
      /* Enable DMA1 channel6 IRQ Channel */
      NVIC_InitStructure.NVIC_IRQChannel = DMA1_Channel6_IRQn;
      NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
      NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
      NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
      NVIC_Init(&NVIC_InitStructure);
}

void assert_failed(uint8_t* file, uint32_t line){   while (1)  {  }}
```

stm32f10x_it.c

```
extern uint16_t CurrDataCounterEnd;

…

void DMA1_Channel6_IRQHandler(void){
       /* Test on DMA1 Channel6 Transfer Complete interrupt */
       if(DMA_GetITStatus(DMA1_IT_TC6))  {
              /* Get Current Data Counter value after complete transfer */
              CurrDataCounterEnd = DMA_GetCurrDataCounter(DMA1_Channel6);
              /* Clear DMA1 Channel6 Half Transfer, Transfer Complete and Global interrupt pending bits */
              DMA_ClearITPendingBit(DMA1_IT_GL6);
      }
}
```

DMA (**exercises and questions**)

1. **Check the correctness of the written data using the debugger** (memory dump)

- **Check the correctness of the data using code**
- **Create a 100byte char buffer in FLASH and use the DMA to transfer it to SRAM memory**
- **Rewrite the example code not using the DMA to perform the data transfer**
- **Measure the time needed to perform the data transfer in exercise 4 and in the example code. What do you notice? Why?**
- **In the example code** SRC_Const_Buffer **is said to be in FLASH whereas** DST_Buffer **in embedded SRAM. Why?**