DMA

> **FROM: CS473 Course**

1. Background

Two Traditional Ways to Test Data Status

Polling

By **polling** of status register, the program (CPU) can know when the interface is ready for the transfer. With polling:

- the system is **synchronous**
- But, the program must often test the status register for nothing

Hardware Interrupts

- The synchronization with the information consumer / producer is to be processed by software
- Some instructions needs to be executed to serve the interrupt handler, which means **limited transfer bandwidth**

2. Introduction of DMA

The **DMA controller** performs transfers in place of the processor.

- The DMA controller is a **programmable interface** that must be programmed by the processor before it is operational
- It must have control of the bus:
    - Address
    - Data
    - Control transfers

- Before performing a transfer, an **arbitration** must occur : Only one master can access a slave unit at a given time
- For the DMA to be useful, we need a certain amount of data to transfer

3. Workflow

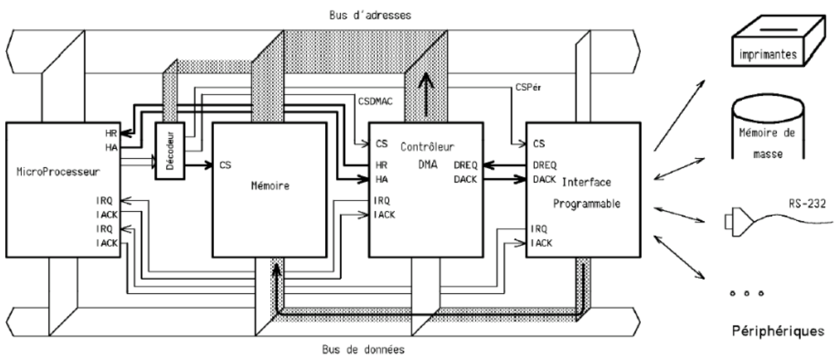
Scenario 1: I/O → Memory

1. I/O → DMA Controller: Transfer Request
2. DMA Controller → CPU: Request of the Bus to the processor
3. I/O→ DMA: Transfer data
4. DMA → Memory: Transfer data
5. End: When a data packet has been transferred, the processor is notified by **interruption** or it can use the **polling of a status register**

Scenario 2: Simple Cycle Transfer

For greater efficiency transfer, access by the interim controller is not always necessary

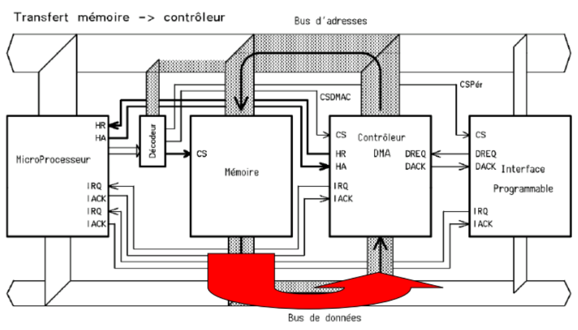
The data directly from I/O → Memory

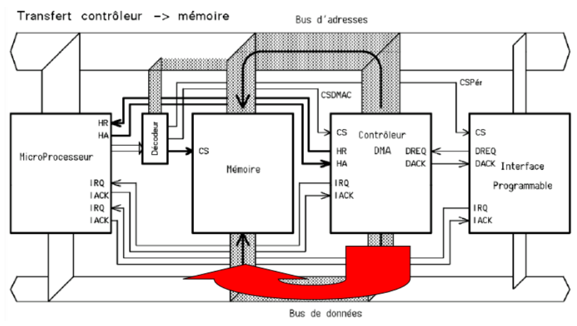


A problem could be arises if the data bus width of memory and Programmable Interface are not the same, then we will need an **alignment drivers**

Scenario 3: Memory to Memory

The DMA unit can be used to transfer data from memory to memory more efficiently than the processor





4. Programmation of DMA

The DMA controller is a programmable interface. It must therefore be initialized prior to use. There are two mainly methods to initialize DMA controller:

1. By direct access to internal DMA **registers** by the processor
2. By **descriptors** automatically loaded **from memory** to the DMA controller by itself

Method 1: Registers

Just normal, no detailed discussion

Method 2: Descriptors

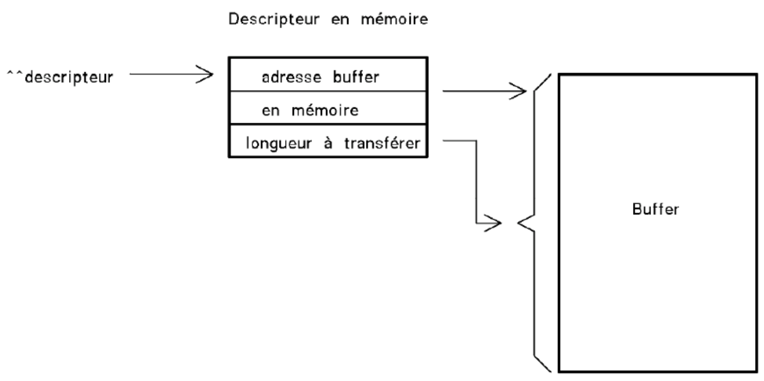
A minimum set of descriptors are available on virtually all controllers DMA:

- Source Address
- Destination Address
- Length of data to transfer / transferred
- Modes of operations
- Status of the controller
- Interrupt control

在基于描述符的DMA操作中，我们可以对一个DMA通道进行编程，在当前的操作序列完成后，自动设置并启动另一次DMA传输。

Base Registers

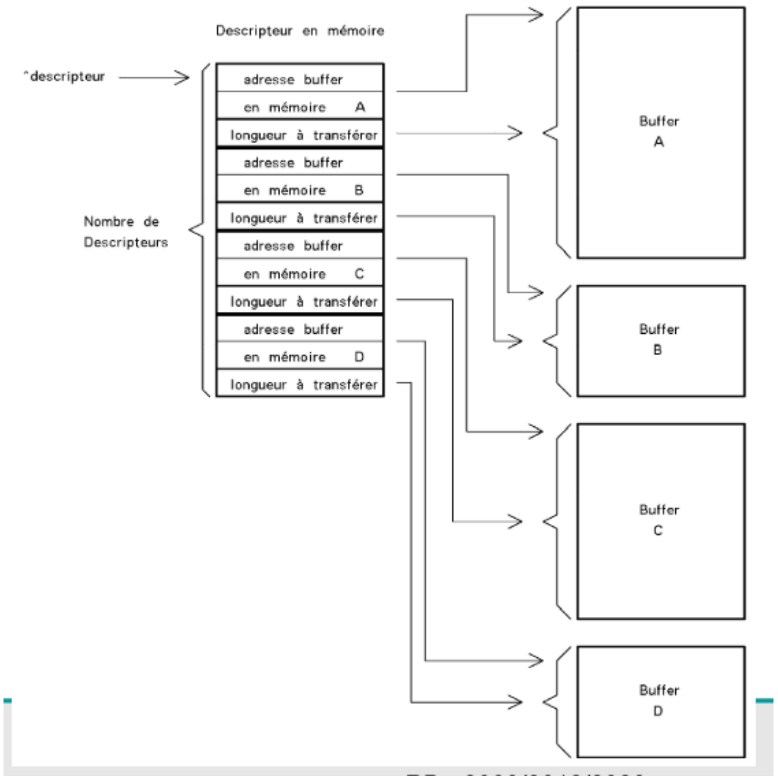
In general, a pointer in the controller point to a descriptor in memory



A single buffer descriptor is limited if new data are received and that the buffer precedent is not yet released

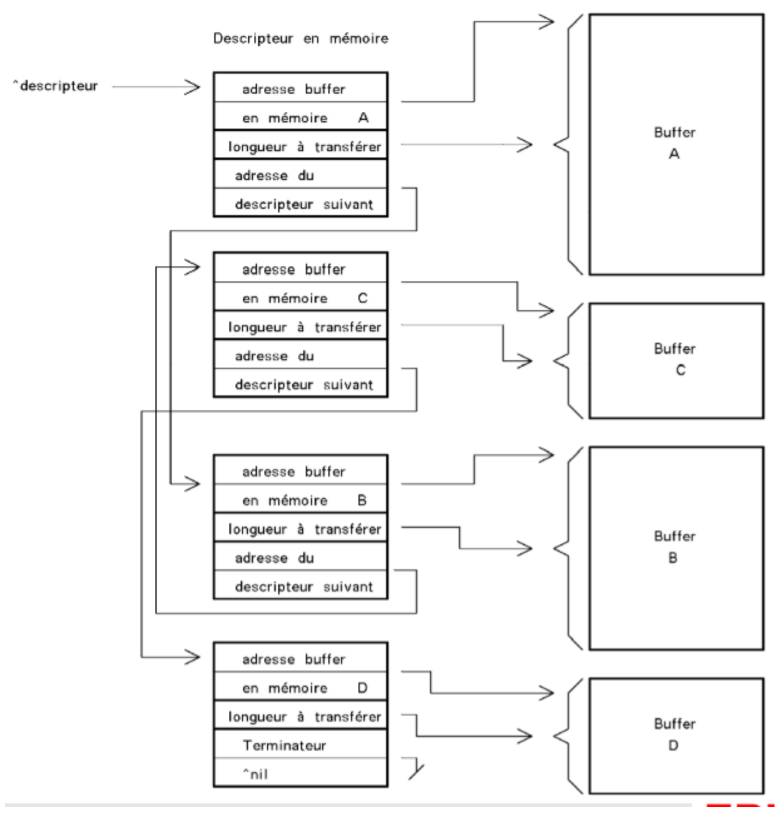
Descriptor Tables

With an array of descriptors, the number of buffer is static. A buffer management is needed to maintain order in the buffer descriptors



Descriptor Linked List

A linked list of descriptors allows greater flexibility in the management of descriptors, and buffer number can easily be dynamic



Example of DMA Descriptors:

> FROM: https://blog.csdn.net/qq_22902757/article/details/104275441

描述符完全是纯软件的概念，它的本质就是我们自己用结构体来实现这个描述符，然后将描述符的首地址写入到 `ETH_DMATDLAR` 寄存器中，STM32就知道这片内存是用来作为发送描述符了。

我们通过定义一个结构体来实现这个发送描述符

```
typedef struct  {
  uint32_t   Status;                /*!< Status */
  uint32_t   ControlBufferSize;     /*!< Control and Buffer1, Buffer2 lengths */
  uint32_t   Buffer1Addr;           /*!< Buffer1 address pointer */
  uint32_t   Buffer2NextDescAddr;   /*!< Buffer2 or next descriptor address pointer */
} ETH_DMADESCTypeDef;
```