DMA
Direct Memory Access

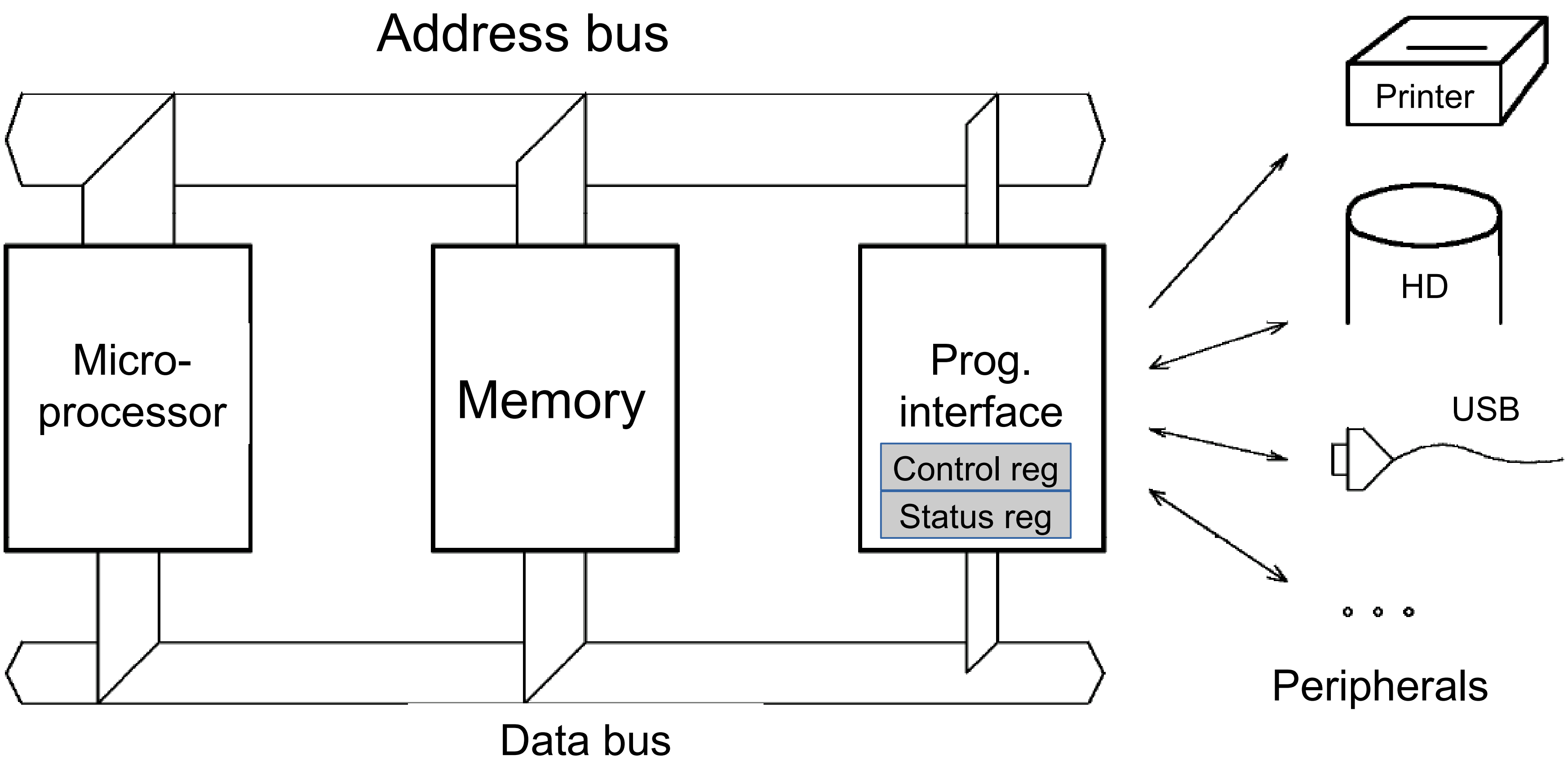
Andres Upegui, René Beuchat
andres.upegui@hesge.ch

Contents

Peripheral access

- In a computer system, peripheral access through programmable interfaces can be performed by processor transfer instructions.
- **Control** registers, present in the peripheral, allow the processor to indicate the actions to be performed. For instance, a start signal may launch a procedure.
- **Status** registers allow to check the current state of peripheral. For instance, verify whether a data transfer can be done or not.

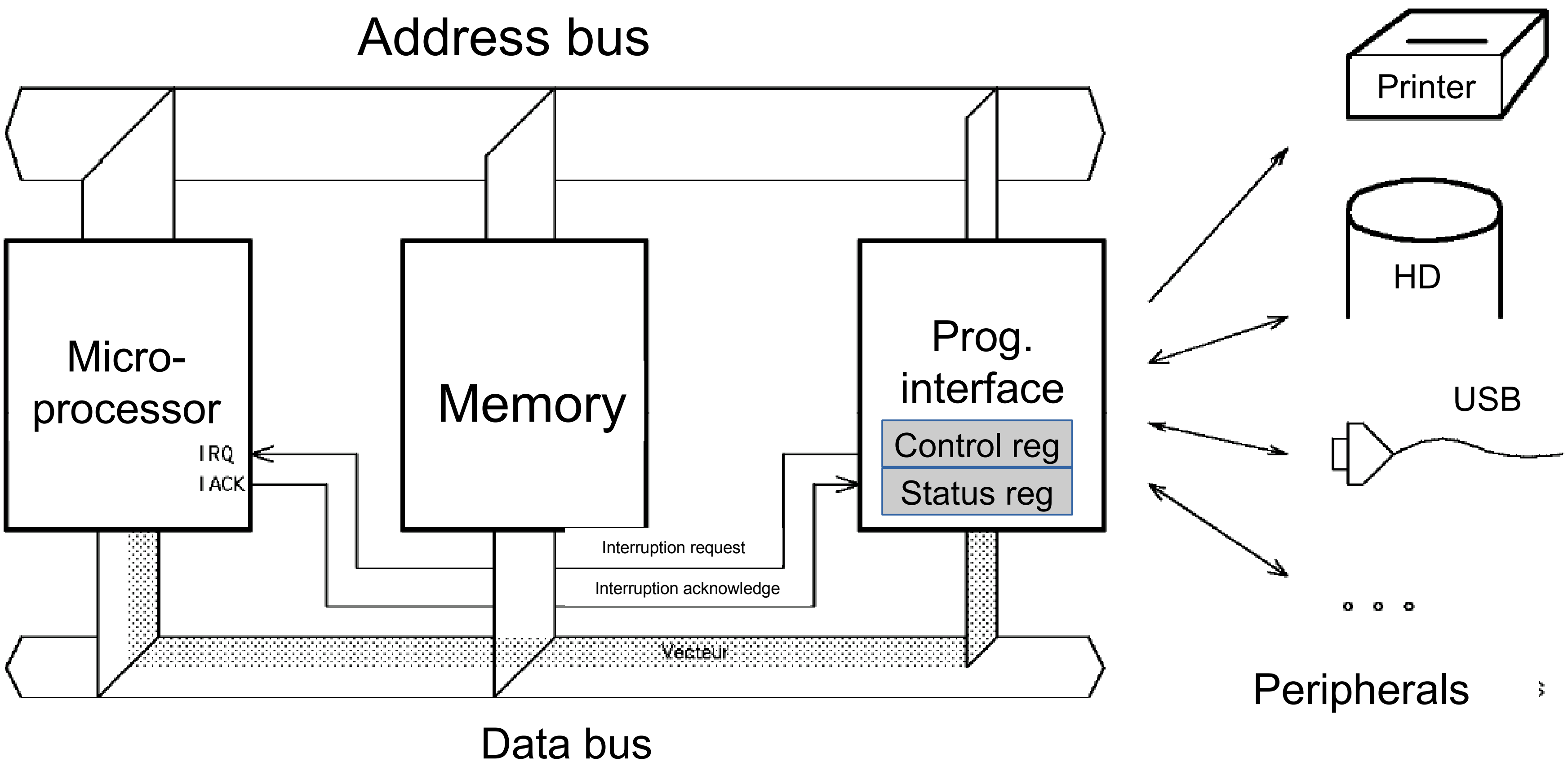
Polling


Address bus
Micro-
processor
Memory
Prog.
interface
Control reg
Status reg
Data bus
Printer
HD
USB
Peripherals

Interruption

- If we want the processor not to loose time polling unnecessarily the different interfaces, hardware interrupts can indicate the processor when a certain condition is met in order to allow the processor to execute a special function called **interrupt handler** or **ISR** for interruption service routine.
- The synchronization with the information consumer / producer is to be processed by software (message, semaphore, FIFO, etc.…)

Interruption



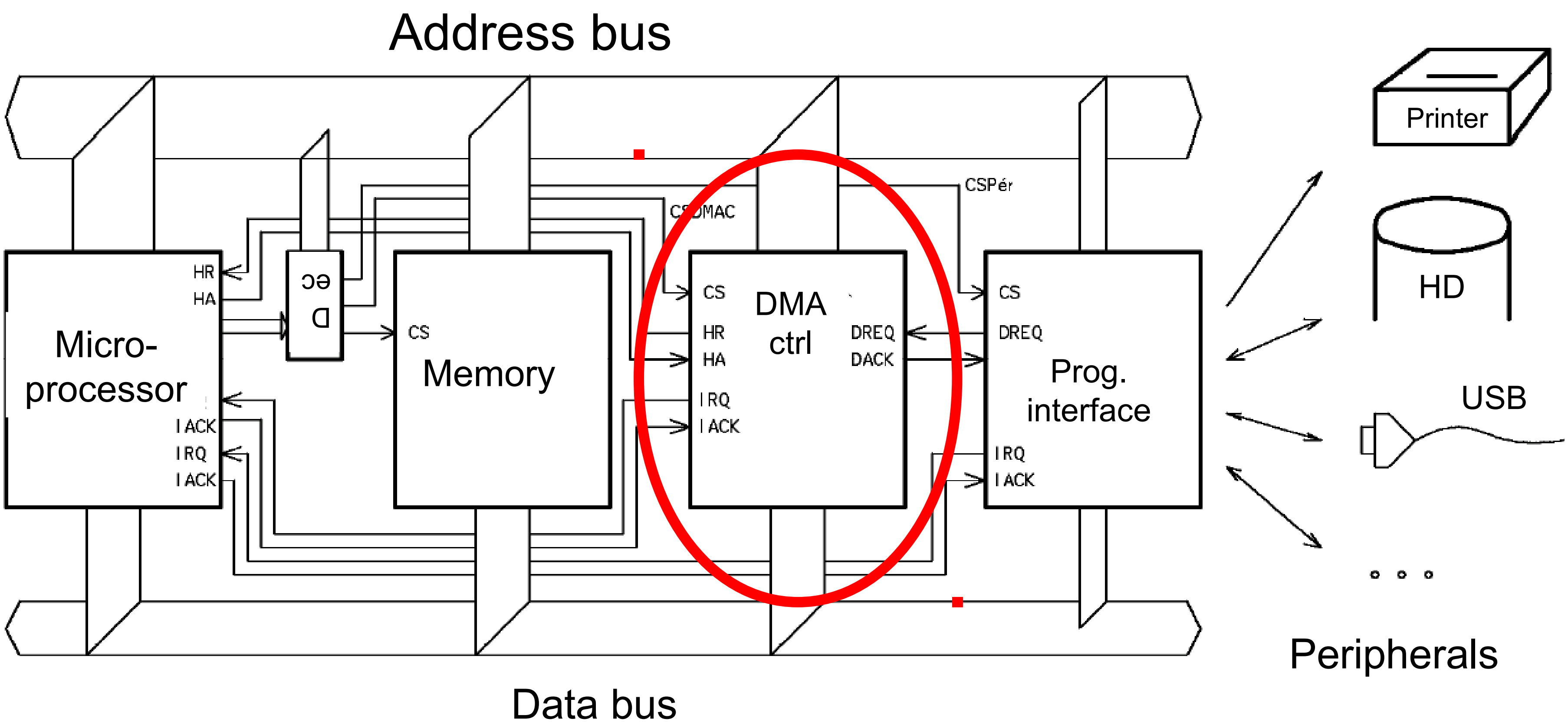
Interruption

- Interruptions need **specialized hardware** in the processor and in the programmable interface. This hardware depends on the processor used (interrupt vector, way to access the interrupt handler function, etc…)
- Some signals are necessary as **IRQ** (*Interrupt Request*) (at least) and sometimes ***Interrupt Acknowledge***
- Some instructions need to be executed to serve the interrupt handler (context saving and switching, request testing, programmable interface servicing and acknowledge)
  → **Limited transfer bandwidth**

DMA – Direct Memory Access

- For systems where the transfer rate between the I / O and memory is high, the polling or interruptions are unusable. A more efficient system is needed → **DMA**
- The transfer is carried out by a specialized unit: the **DMA controller**

DMA



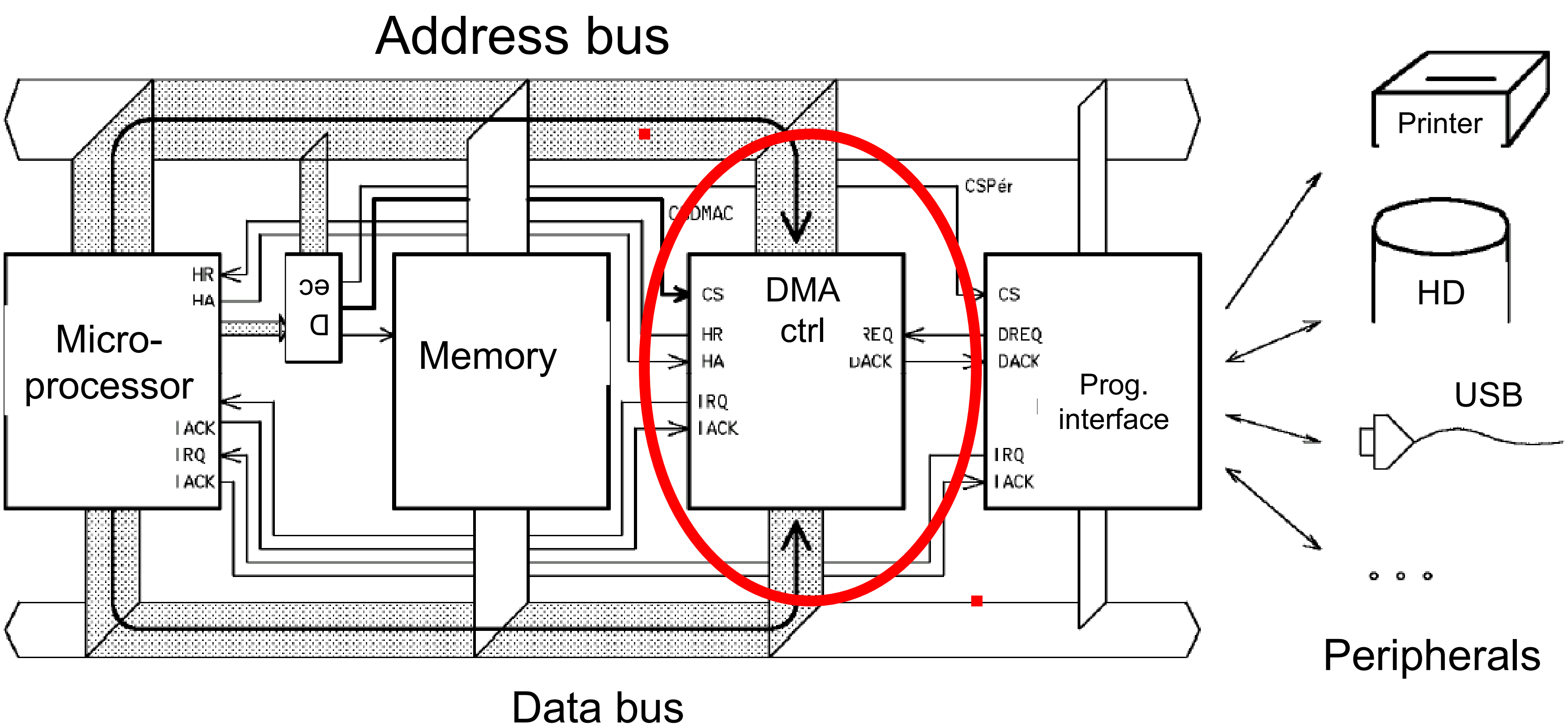
Address bus
Micro-
processor
HR
HA
I ACK
IRQ
I ACK
Dec
CS
Memory
CSPér
CSDMAC
CS
HR
HA
IRQ
I ACK
DMA
ctrl
DREQ
DACK
CS
DREQ
Prog.
interface
IRQ
I ACK
Printer
HD
USB
Data bus
Peripherals

DMA

- The DMA controller performs transfers **instead of the processor**
- DMA must have control of the bus as a **Master** including Address bus, Data bus and Control signals
- The DMA controller is a programmable interface that **must be programmed by the processor** before it is operational

DMA Controller: a programmable interface

DMA controller programming :



DMA : end of transfer(s)

- When a data packet has been transferred, the processor is notified by interruption.
- The status register can also give additional information about the transfer completion.
- For the DMA to be useful, we need a **certain amount** of data to transfer, not only one byte, as we need to initialize the DMA controller before using it.

Type of transfers

The DMA unit can be used to transfer data more efficiently than the processor. Two main types of transfers:

Memory to memory:

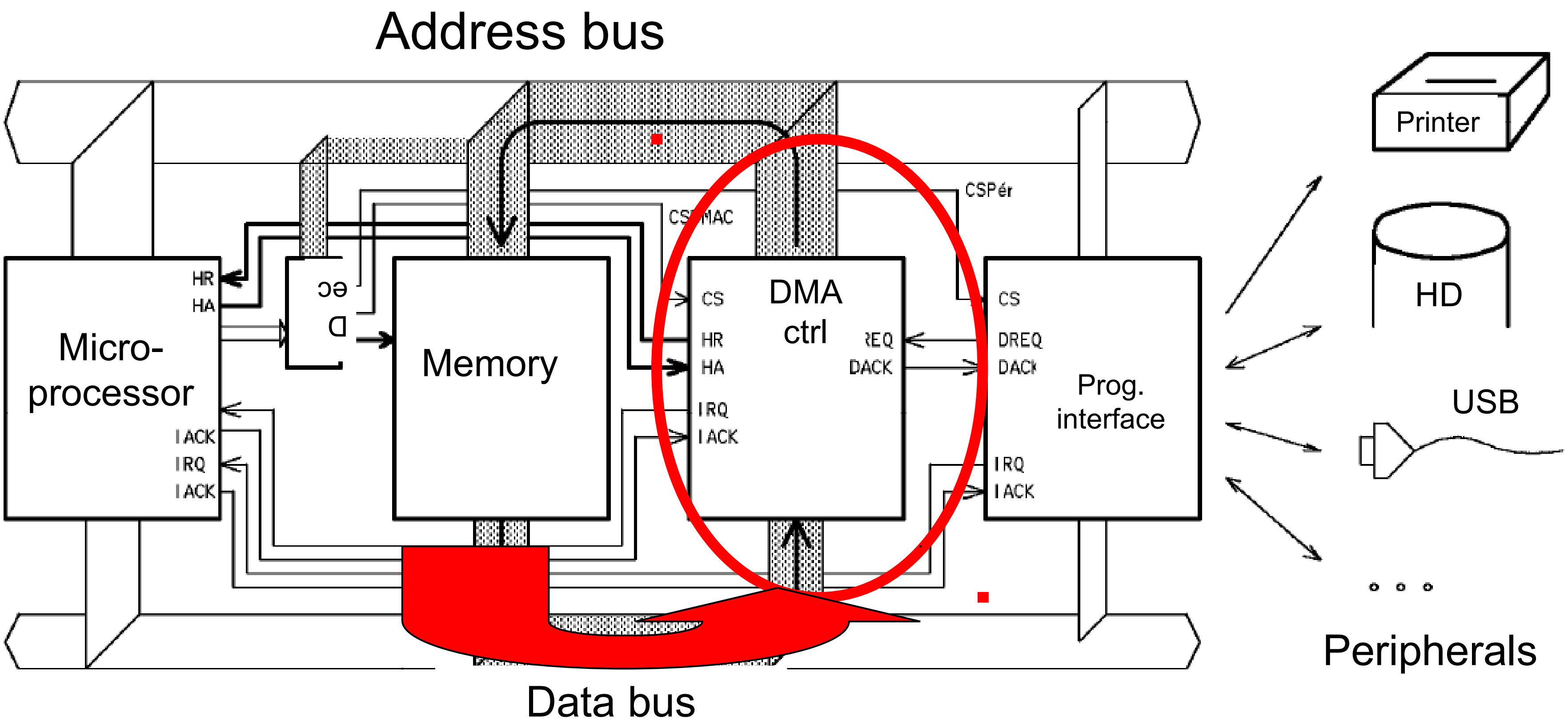
- Memory address must be automatically incremented for both : source and destination

Interface to memory (and viceversa):

- Memory address is automatically incremented, while interface pointer is typically fixed to an address pointing to a data FIFO in the interface.

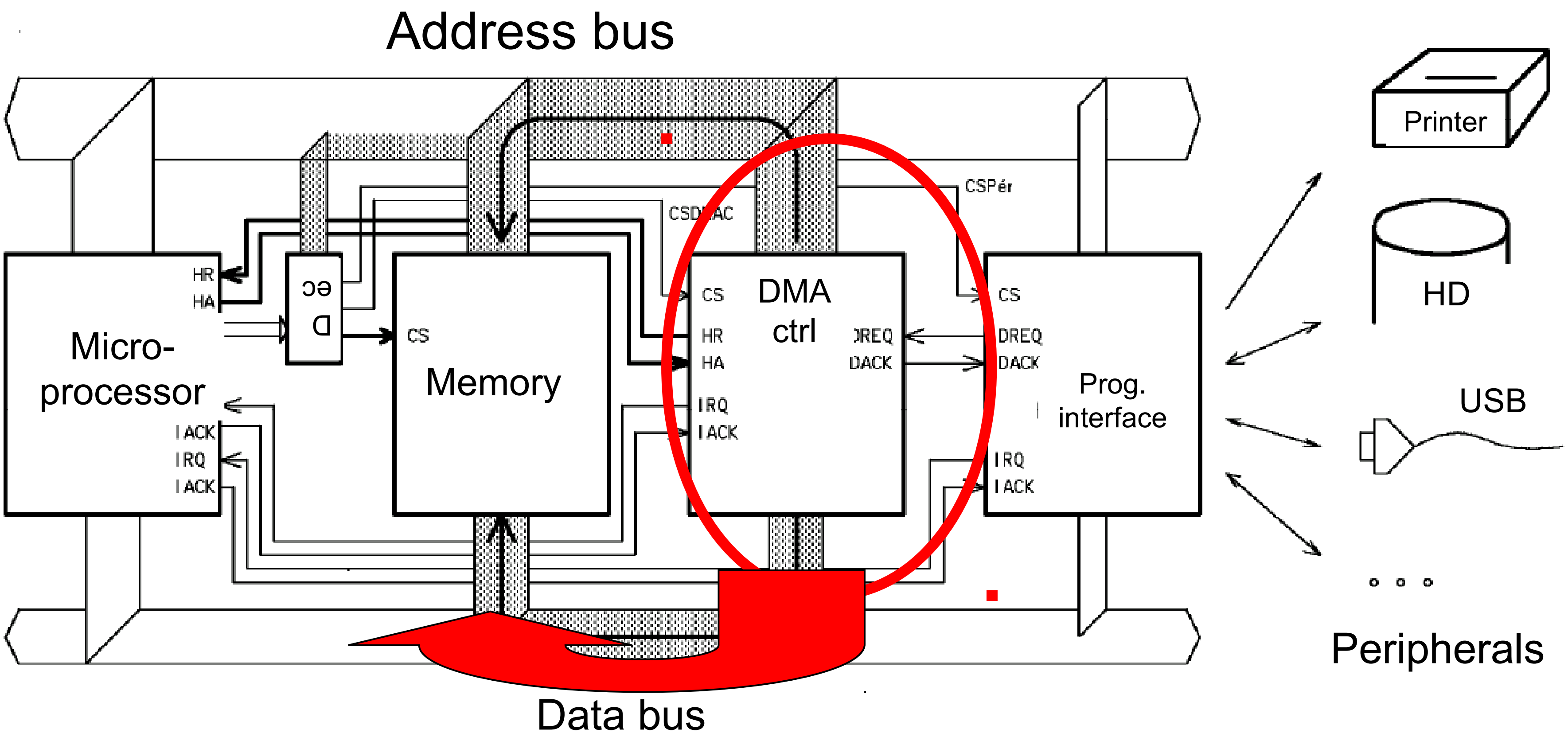
Transfers memory to memory (1)

Step 1: memory read



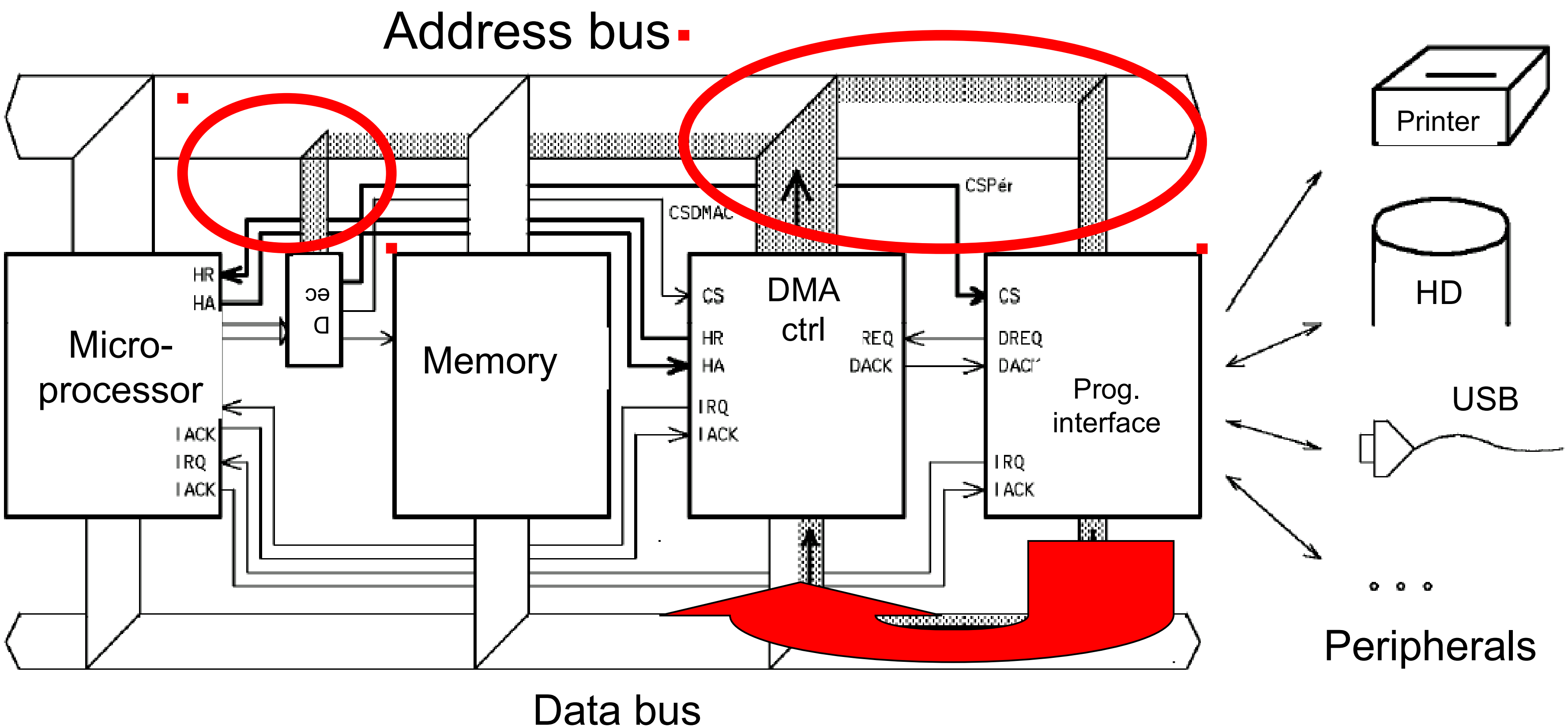
Transfers memory to memory (2)

Step 2: memory write



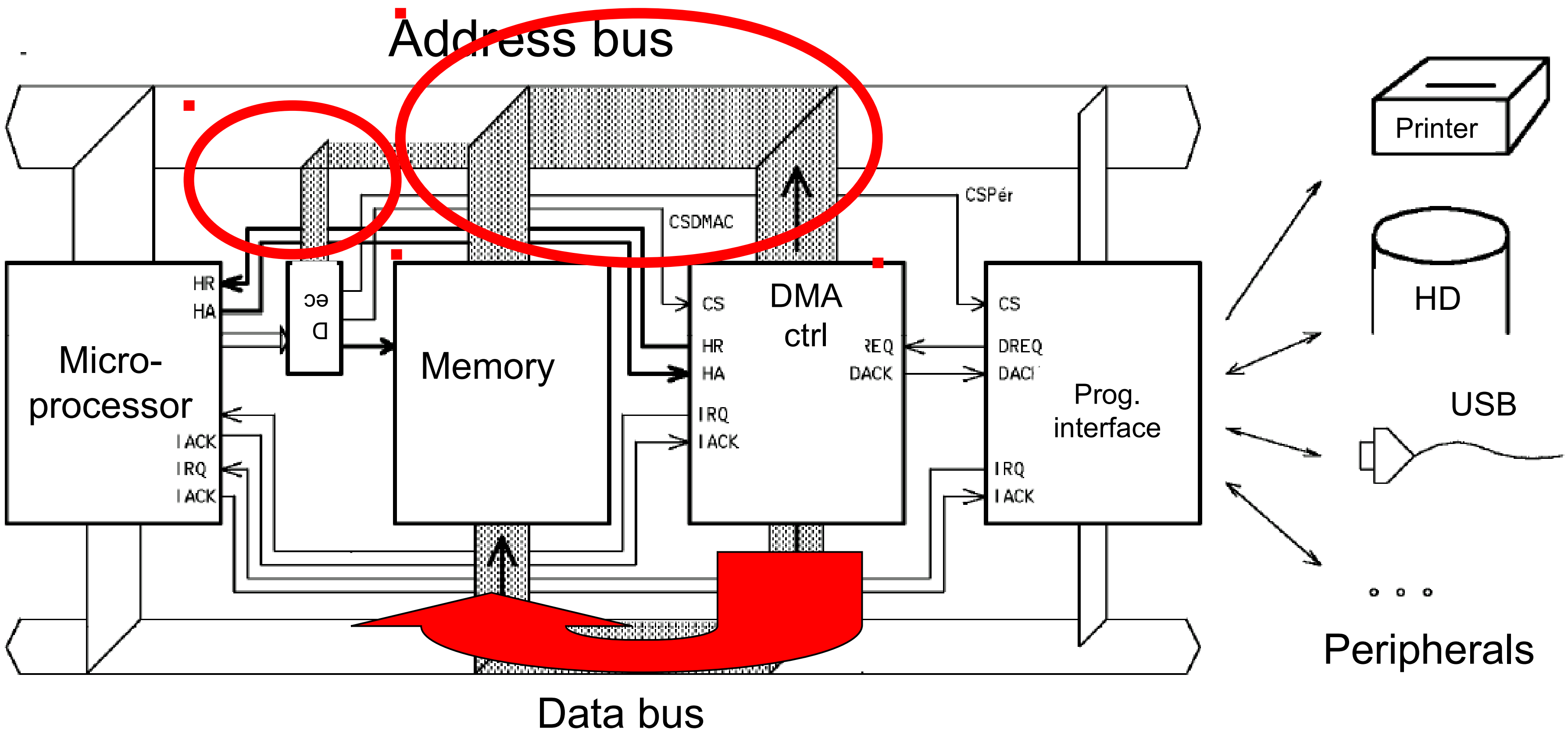
Transfer I/O to Memory (1)

Step 1: Transfer Interface Prog. → Ctrl DMA



Transfer I/O to Memory (2)

Step 2: Transfer Ctrl DMA → Memory



DMA Configuration

- The DMA controller is a programmable interface. It must therefore be initialized prior to use.
- Several methods are possible depending on the circuit used:
  - By direct access to internal DMA registers by the processor
  - By descriptors automatically loaded from memory to the DMA controller by itself

DMA Configuration (2)

- A minimum set of descriptors are available on virtually all DMA controllers :
  - Source Address
  - Destination Address
  - Length of data to transfer
  - Modes of operation
  - Status of the controller
  - Interrupt control

Base Registers

Exemple of a DMA register model :

Status Register
Control Register
Error Register
Interruption Vector
Source address
Destination address
Transfer size
Transferred data

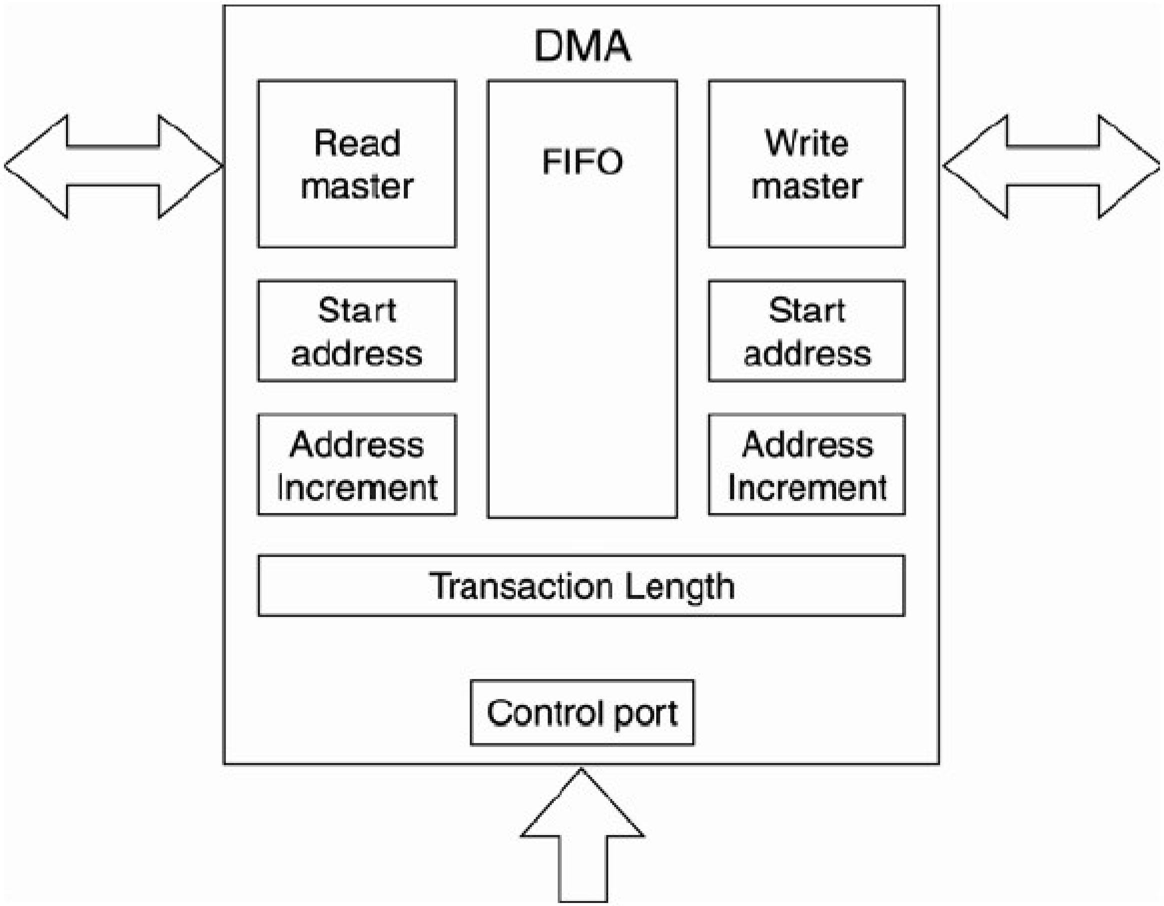
DMA on SoPC (FPGAs)

Several options:

- You can write your own DMA core in HDL (VHDL / Verilog). This allows you to manipulate data in a more application oriented manner. (FFT, images, ...)
- You can write your core embedded in an interface, in order to directly read/write from/to memory to the interface. (Camera, LCD, ...)
- There are synthesizable IP to integrate on programmable logic for implementing standard DMA transfers
  Example: DMA for Avalon bus

DMA for Avalon

- DMA unit architecture
- Read bus
- Internal FIFO
- Write bus
- Programmable control unit



DMA registers

- Seen by the processor (NIOS) as 8 * 32 bits registers

A2..A0	Register Name	R/W	Description/Register Bits											
			31	. . .	9	8	7	6	5	4	3	2	1	0
0	status[1]	RW								len	weop	reop	busy	done
1	readaddress	RW	Read master start address											
2	writeaddress	RW	Write master start address											
3	length	RW	Length in bytes											
4	reserved1	–	Reserved											
5	reserved2	–	Reserved											
6	control	RW			wcon	rcon	leen	ween	reen	i_en	go	word	hw	byte
7	reserved3	–	Reserved											

Status Register

- Information on controller status
- A write access clear len, weop, reop, and done bits

Bit Number	Bit Name	Description
0	done	A DMA transfer is completed.
1	busy	A DMA transfer is in progress.
2	reop	Read end of packet occurred.
3	weop	Write end of packet occurred.
4	len	A DMA transfer is completed and the requested number of bytes are transferred.

- ***done*** is activated at the end of the transfer
- Bits ***len***, ***weop***, and ***reop*** allow to know the cause of the transfer end.
- When ***done*** is deactivated by a write to this register, the interrupt request is deactivated too

Control Registers

- *Readaddress*, *writeaddress*, *length* specify the source, destination addresses and the length of the transfer
- *length* defines the number of bytes
- Width of registers is specified at the DMA unit creation.
- The *Control* register

Control Register

- The *Control* register specify modes and enabling functions

Bit Number	Bit Name	Description
0	byte	Byte (8-bit) transfer.
1	hw	Half-word (16-bit) transfer.
2	word	Word (32-bit) transfer.
3	go	Enable DMA.
4	i_en	Enable interrupt.
5	reen	Enable read end of packet.
6	ween	Enable write end of packet.
7	leen	End DMA transfer when `length` register reaches 0.
8	rcon	Read from a fixed address.
9	wcon	Write to a fixed address.

Control

- *rcon* et *wcon* specified if the read or write address is fixed ('1') or to increment ('0')
- depending the transfer width and *con* specified, the addresses are incremented by 0, 1, 2 or 4

Bit Name	Transfer Width	Increment
byte	byte	1
hw	half-word	2
word	word	4

DMA programming

1) Clear mode

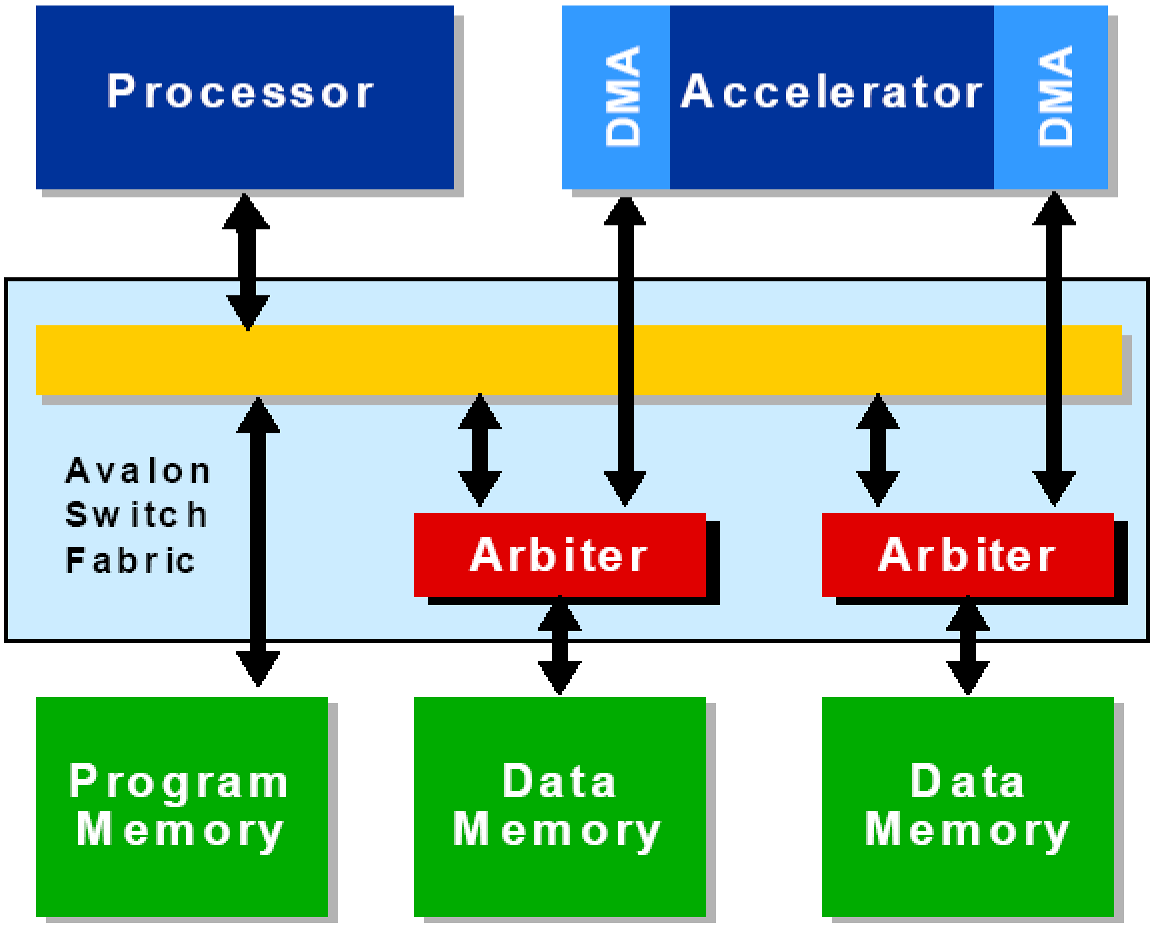
2) Set up everything except the go-bar

```
dma->np_dma_status = 0;
dma->np_dma_read_address = (int)source_address;
dma->np_dma_write_address = (int)destination_address;
dma->np_dma_length = transfer_count * bytes_per_transfer;
```

3) Construct the control word... to start

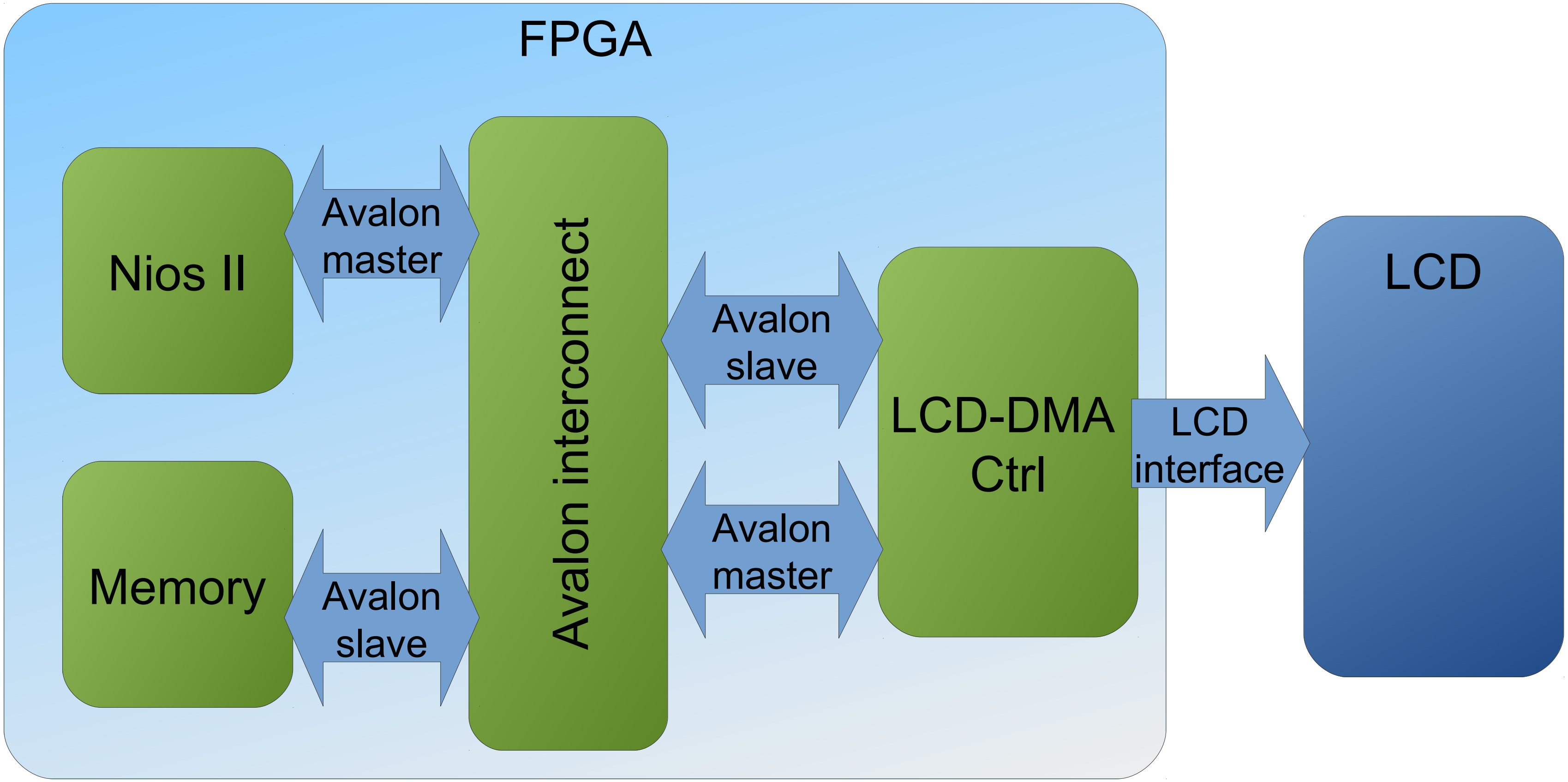
4) Wait until it's all done… Through an interruption!!

Another approach: Hardware accelerator



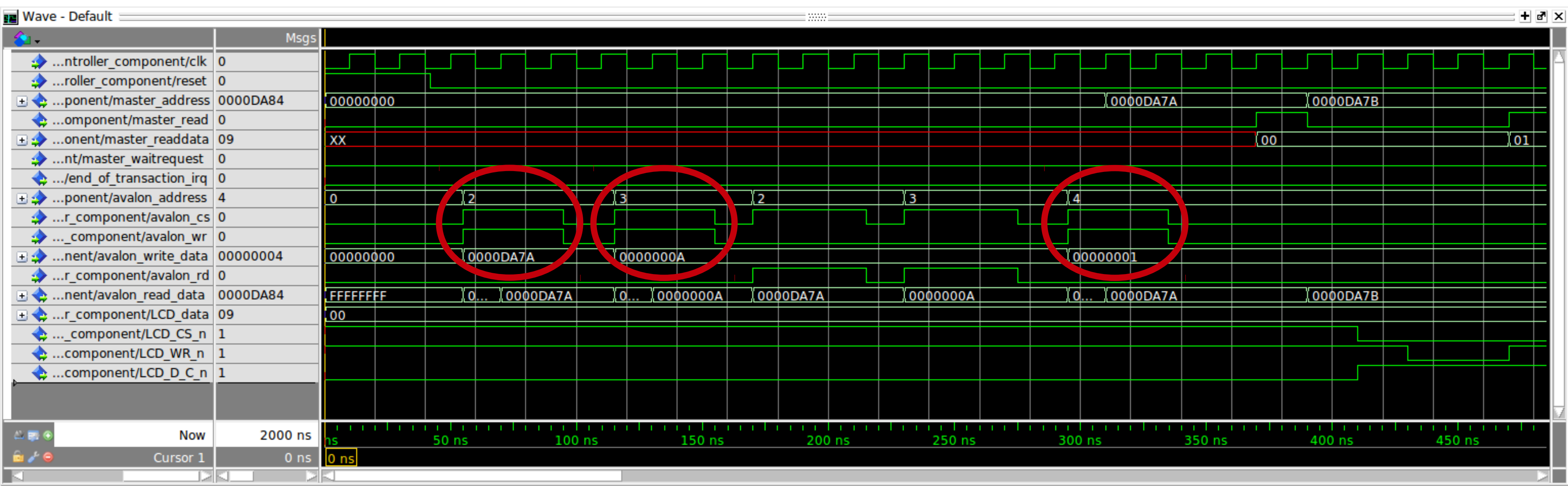
- A hardware accelerator is a master unit with at least 2 DMA channels :
  - One (or more) to read data
  - One (or more) to write result
- To build a DMA unit, a **master Avalon** unit has to be designed
- It has to provide the address of the data to access and to generate the data transfers
- The Avalon ***WaitRequest*** signal is mandatory to synchronize the end of the transfer cycle.

Example: LCD-DMA Controller - Schematic



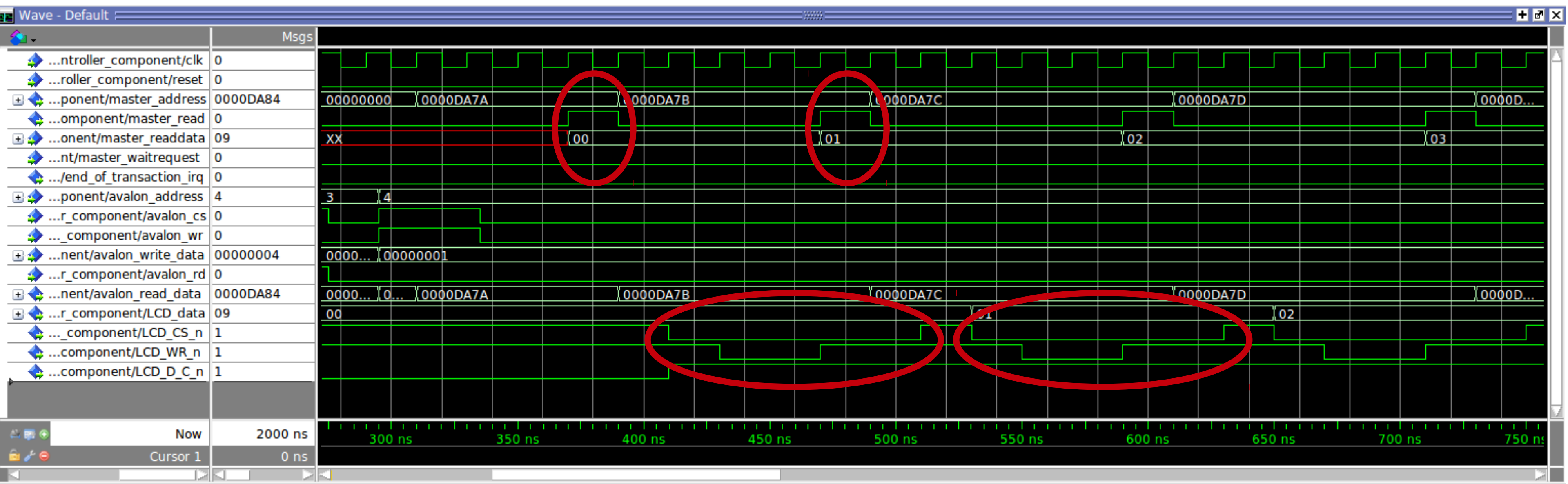
Example: LCD-DMA Controller – Configuration of tranfer

Initially the DMA transfer must be **configured** through the avalon slave interface. The value 0xDA7A is written to the register address 2 for indicating the source address. The value 0xA is writen to the address 3 for indicating the number of transfers to execute. Then, both addresses are read for verification purposes. Finally, the value 0x1 is written to the register 4 (control register) for starting the transfer.



Example: LCD-DMA Controller - Transfer

After starting the **transfer** a series of 10 consecutive reads are performed from the address 0xDA7A, address is further auto-incremented for each consecutive address. Each value read through the avalon master interface is further presented in the LCD data output and the corresponding output LCD WR n and LCD CS n signals are generated. NOTE : this example performs transfers on 8 bits, addresses are thus increased by 1. 16-bit transfers require an increase of 2.



Example: LCD-DMA Controller – Completion of transfer

At the **end** of a complete transfer, an irq output signal is generated by the DMA. The irq must be maintained until an acknowledge is performed from the processor. The acknowledge is completed by writing a 1 to the third bit of the control register (writing the value 4 to register 4)