# CPU-FPGA communication

# FPGA as real-time Xenomai/Linux co-processor - Redpitaya/Zynq presentation, communications bus and CPU-FPGA communication

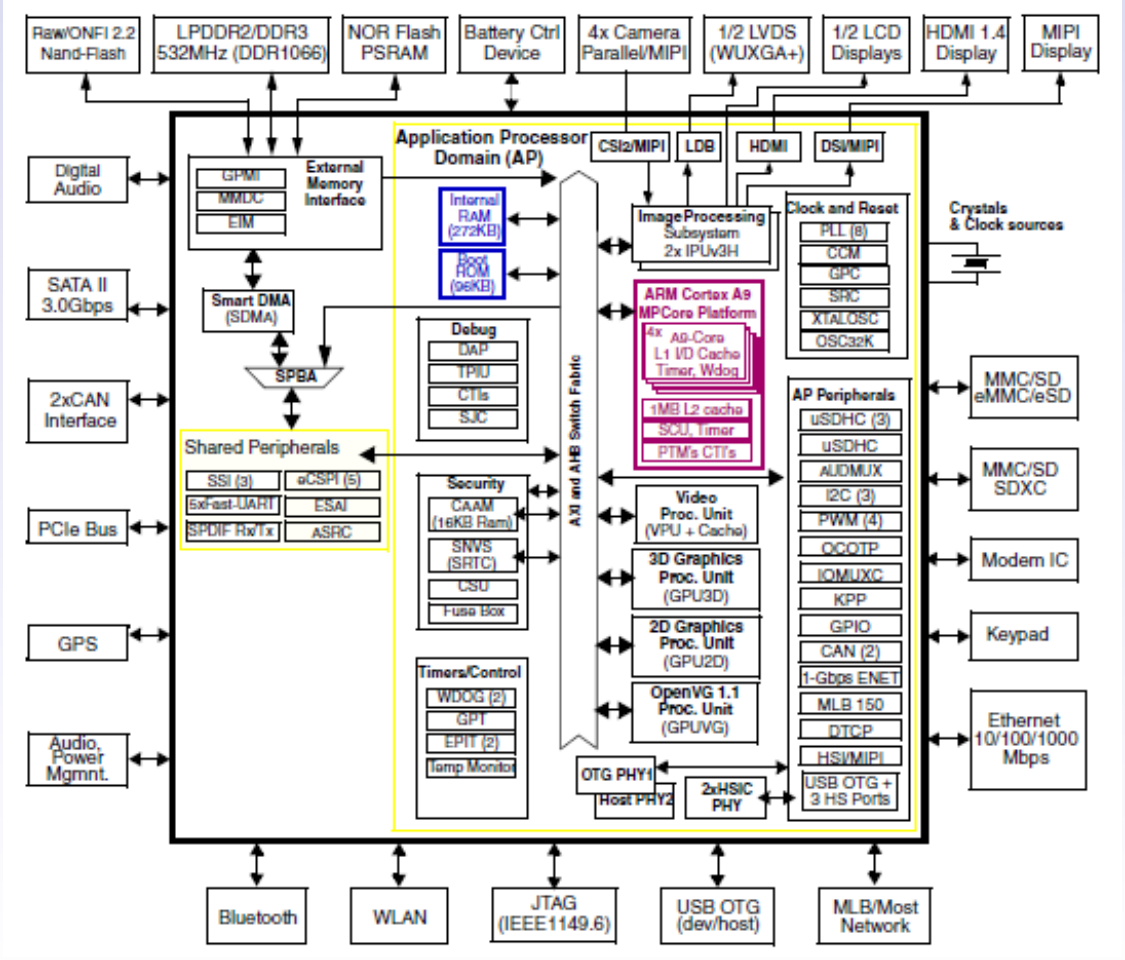
**Gwenhaël GOAVEC-MEROU**

September 12, 2022

Slides available at
`http://www.trabucayre.com/enseignement/presentation_zynq.pdf`

# Motivation

- appropriate the principle of communication between a processor and hardware peripherals
- is not just limited to CPU/FPGA communication → also true between a core and the hardware peripherals of a processor.



# Communication busses

Already seen: serial protocols (SPI, I2C).
Topic : parallel protocols.
Used for:

- to communicate with RAM chip, mass storage (NAND, NOR, (e)MMC, etc.);
- communication between a core and hardware controlers;
- to communicate with FPGAs

Usually composed of:

- an address bus;
- one or two data bus (full duplex or master→slave and slave→master);
- control signals (read/write request, data valid, ack, interrupts)

# Operating principle

- one master, n slaves;
- shared memory used to communicate between CPU and peripherals;
- memory is sliced between peripherals: each of them have a dedicated area and a sub-address map.

Warning: Memory access are aligned according to the size (8bits, 16, 32 or 64)

```
// ok: writing a short (16bits) at an address multiple of 2 (octets)
pos = 0x02; /* 1 << 1 */
*(unsigned short*)(ptr_fpga+pos) = 0x01;

// fail writing a short (16bits) at an address multiple of 1 (octet)
pos = 0x01; /* 1 << 0 */
*(unsigned short*)(ptr_fpga+pos) = 0x01;
```

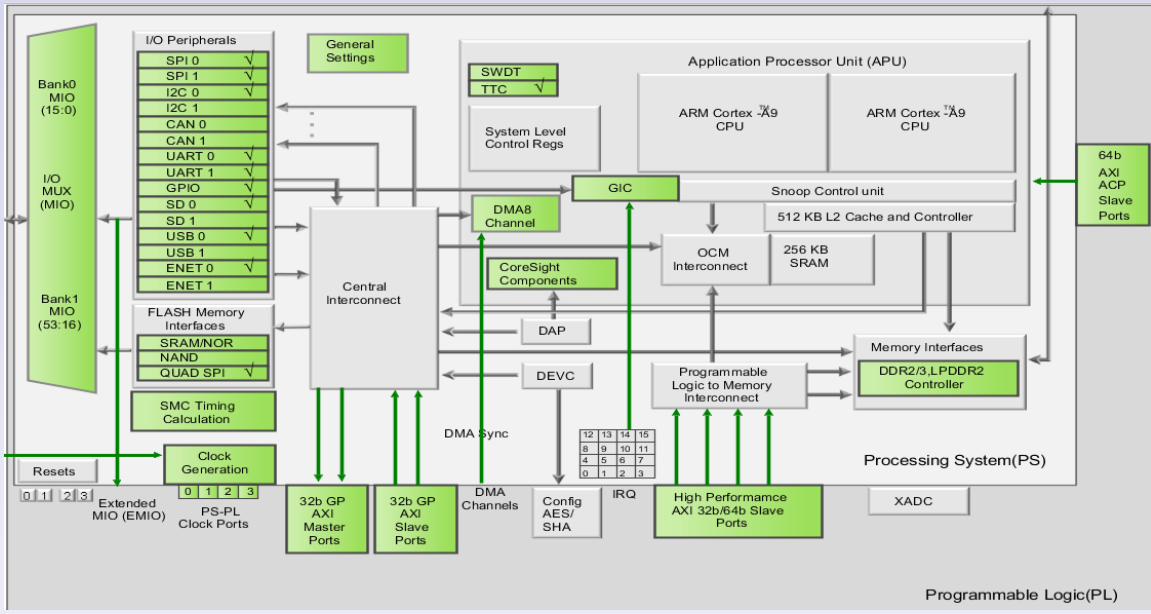
A bus with n bytes does not necessarily imply accesses on n bytes: ex: SPI controller on STM32F3/4: different behavior depending on the type of access and the size of the data to be transmitted.

# Hardware environmentriel

The hardware used: a Redpitaya card (based on Zynq).

The same package contains a SoC (PS) and an FPGA (PL).
Features:

- a dual core Cortex A9 processor;
- an FPGA (equivalent to artix7).



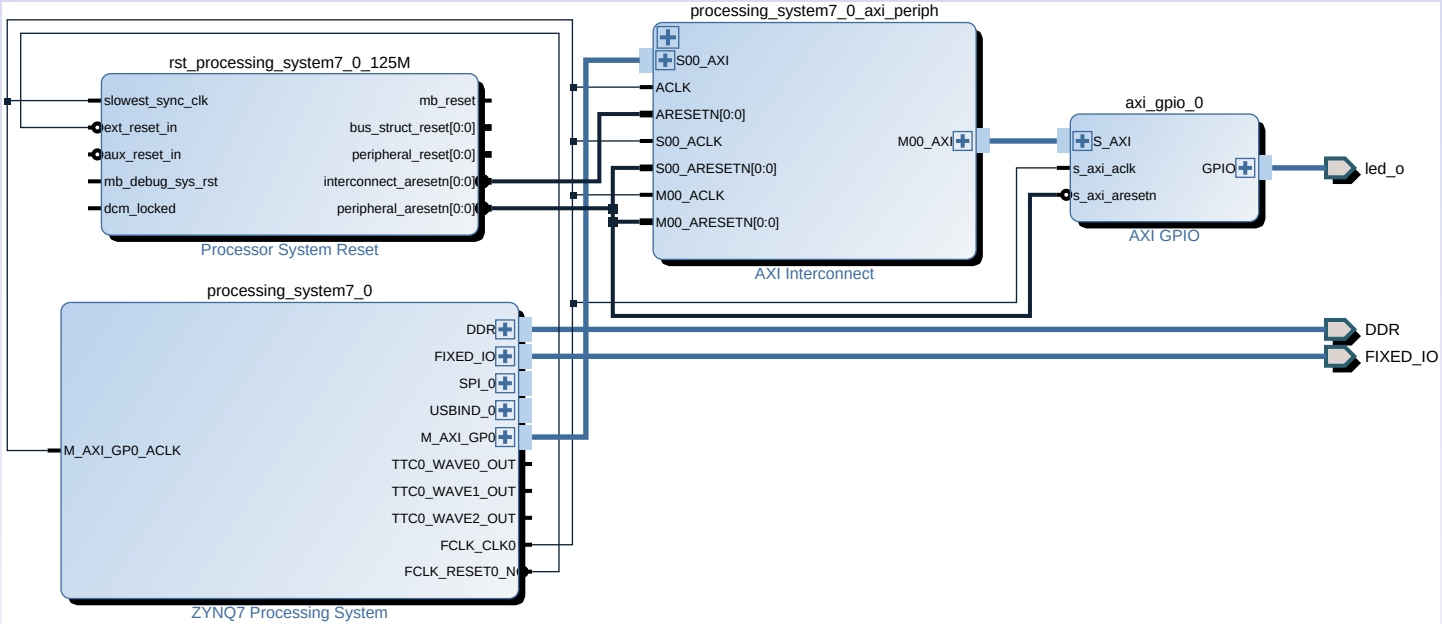

Communications: *AXI slave* and *AXI High-Performance*

- 32 data bits;
- 32 address bits;
- DMA (Direct Memory Access) with AXI HP.
- interrupts;

Methodology also valid for the **wishbone** and **AVALON** busses.

# Reminder/acquired

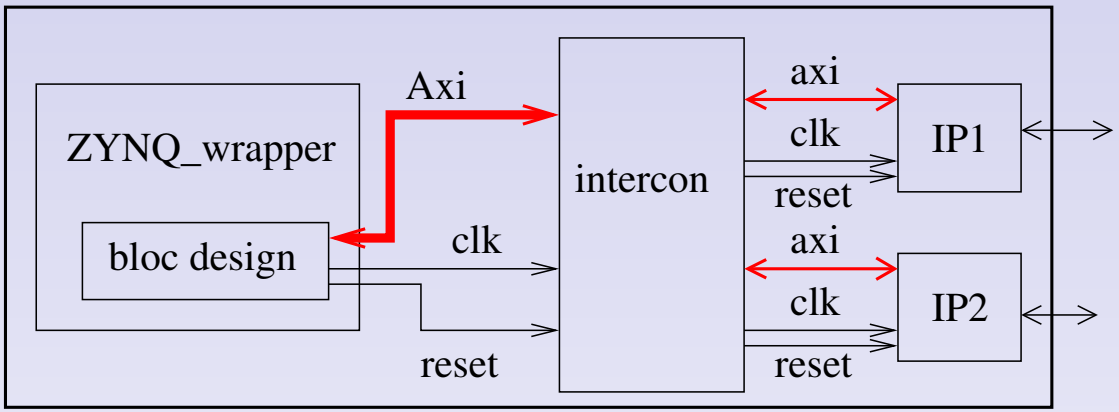
Gateware with an `axi_gpio` block targeted for the `FPGA` and access from CPU.




Reminder :

- `Zynq Processing System` : `Zynq PS` interface / wrapper;
- `Intercon` : address decoder / multiplexer;
- `axi_gpio` : `AXI` slave IP shipped with `Xilinx` tools.

# Gateware architecture




Internal structure:

- a `wrapper` for the block design;
- `AXI` bus, `clock` and `reset` signals are exported;
- address decoder to communicate with several blocks (IPs): intercon

$\Rightarrow$ simplify/accelerate `IP`'s development phase.

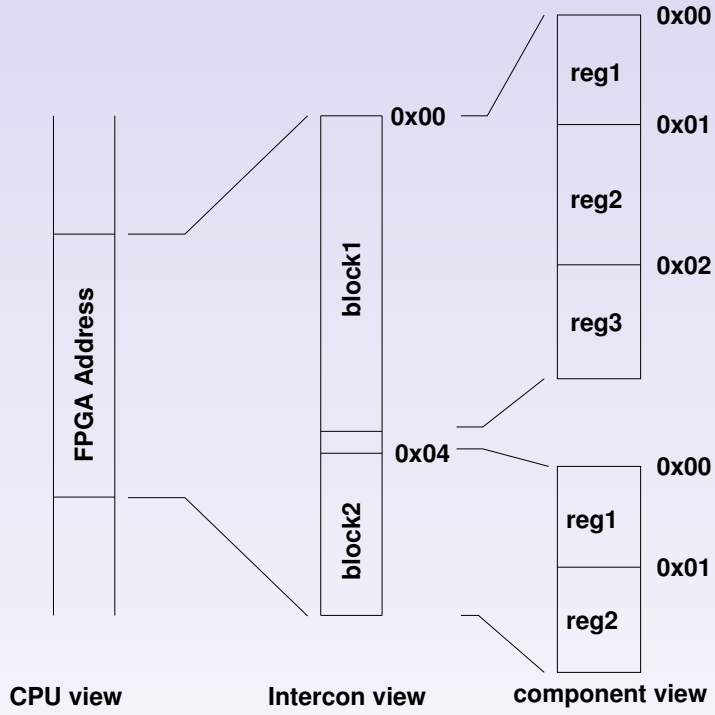
Structure automatically generate by *Peripheral On Demand* software. [1] (Not covered in this lab).

[1] `http://github.com/martoni/periphondemand`

# Address slicing and abstraction

Principle :

- an address range (1 Gb) is dedicated to the FPGA access from the CPU (absolute address);
- this range is sliced into sub-addresses to access individually each blocks (`intercon`: become relative address for IPs by removing absolute part);
- this sub-space is also splited for registers access (blocks specifics. Absolute address and intercon part are lost)



# AXI4 lite bus presentation

clk and reset : s00_axi_aclk and s00_axi_reset

Write:

Address/controls :

- s00_axi_awaddr
- s00_axi_awvalid
- s00_axi_awready
- s00_axi_bresp
- s00_axi_bvalid
- s00_axi_bready

Data:

- s00_axi_wdata
- s00_axi_wvalid
- s00_axi_wready
- s00_axi_wstrb

Read:

Address/controls:

- s00_axi_araddr
- s00_axi_arvalid
- s00_axi_arready

Data:

- s00_axi_rdata
- s00_axi_rvalid
- s00_axi_rready
- s00_axi_rresp

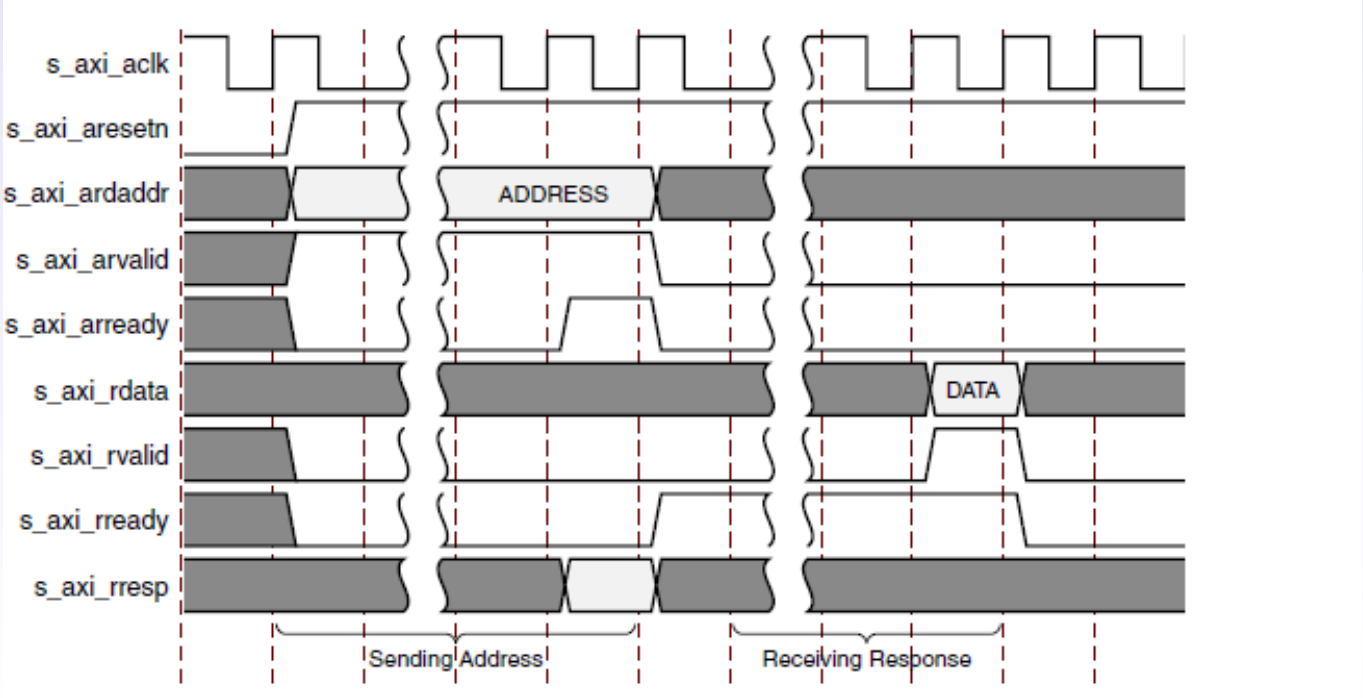
# AXI4 lite bus presentation: read

Address :

- s00_axi_araddr
- s00_axi_arvalid
- s00_axi_arready

Data:

- s00_axi_rdata
- s00_axi_rvalid
- s00_axi_rready
- s00_axi_rresp



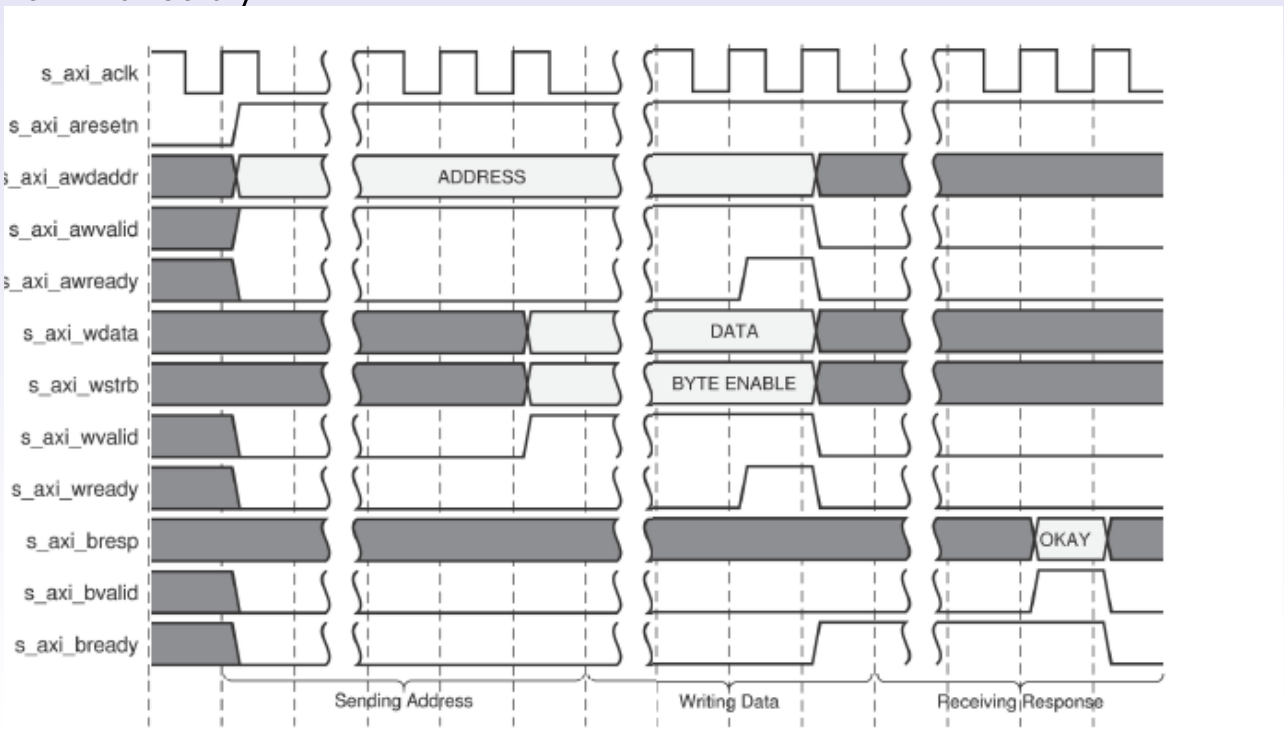
# AXI4 lite bus presentation: write

Addrese/controls :

- s00_axi_awaddr
- s00_axi_awvalid
- s00_axi_awready
- s00_axi_bresp
- s00_axi_bvalid
- s00_axi_bready

Data:

- s00_axi_wdata
- s00_axi_wvalid
- s00_axi_wready
- s00_axi_wstrb



# Communication management

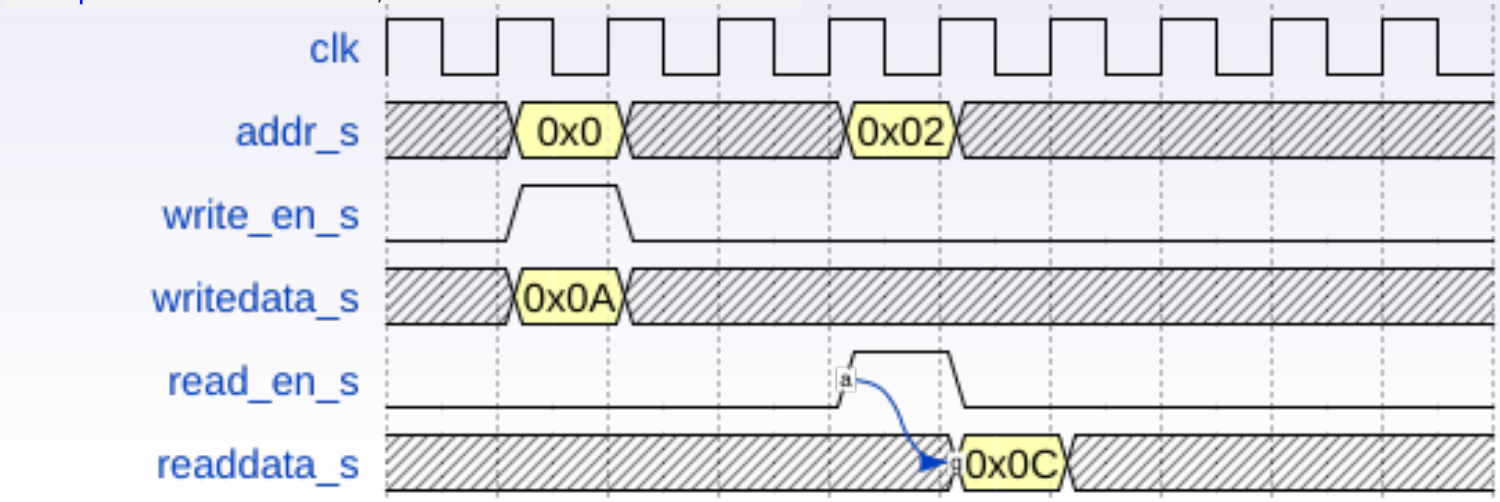
Using a `wrapper` to hide the complex part.

## read handling:

```
read_bloc : process(clk, reset)
begin
  if reset = '1' then
    readdata_s <= (others => '0');
  elsif rising_edge(clk) then
    readdata_s <= readdata_s;
    if read_en_s = '1' then
      case addr_s is
      when REG_ID =>
        readdata_s <= std_logic_vector(→
              ↪to_unsigned(id,→
              ↪C_S00_AXI_DATA_WIDTH));
      when REG_RESULT =>
        readdata_s <= result_s;
      when others =>
        readdata_s <= (others => '0');
      end case;
    end if;
  end if;
end process read_bloc;
```

## write handling:

```
write_bloc : process(clk, reset)
begin
   if reset = '1' then
     op1_s <= (others => '0');
     op2_s <= (others => '0');
   elsif rising_edge(clk) then
     op1_s <= op1_s;
     op2_s <= op2_s;
    if write_en_s = '1' then
      case addr_s is
      when REG_OP1 =>
        op1_s <= s00_axi_wdata;
      when REG_OP2 =>
        op2_s <= s00_axi_wdata;
      when others =>
      end case;
      end if;
   end if;
end process write_bloc;
```



# FPGA's tools

bitstream synthesis + PnR: **Vivado**

```
source /opt/Xilinx/Vivado/VERSION/settings64.sh
```

To launch the tools:

```
vivado&
```

**Warning:** `Vivado` is shipped/installed with a set of libraries also available on host system → may produces conflicts with host ones, and some tools may fails to works correctly.

`bit.bin` convertion:

A `bif` file:

```
all:
{
        proj_dir/objs/proj_name.runs/impl_1/proj_name.bit
}
```

Generation: bootgen –w –image fichier.bif –arch zynq –process_bitstream bin

Next: `bit.bin` must be copied into a shared directory.

On the board:

- bit.bin copy (or move) into `/lib/firmware`
- load: echo "proj_name.bit.bin" > /sys/class/fpga_manager/fpga0/firmware

FPGA access from a terminal : **devmem**

```
devmem 0x43C00000 32 ⟶ read a 32bits @0x43C00000 (reg 0)
devmem 0x43C00004 32 10 ⟶ writes 10 @ 0x43C00004 (reg 1)
```

# Communication from userspace

```c
#define PAGE_SIZE 4096

#define REG_ID      (0<<2)
#define REG_START   (2<<2)

/* adresse physique du FPGA */
#define FPGA_BASE_ADDR 0x43C00000

[...]
void *ptr_fpga;
/* required to have a memory access */
int fd = open("/dev/mem", O_RDWR|O_SYNC);
if (fd < 0) {
  return EXIT_FAILURE;
}

/* obtaining a pointer to the virtual memory page (physical area projection)
 */
ptr_fpga = mmap(0, PAGE_SIZE, PROT_READ|PROT_WRITE, MAP_SHARED,
  fd, FPGA_BASE_ADDR);
if (ptr_fpga == MAP_FAILED) {
  return -2;
}

/* read */
value = *(unsigned int*)(ptr_fpga + REG_ID);

/* write */
*(unsigned int*)(ptr_fpga + REG_START) = 0x01;
```

# Lab topic

1. communication discovery: read and write access to registers
2. evolution to add a new register;
3. access to a RAM to read its contents;
4. Xenomai's applications reusing;
5. period counter creation.

**FPGA use (hardware real-time) to qualify Xenomai (software real-time latences).**

Suject available at
`http://www.trabucayre.com/enseignement/tp_fpga.pdf`
Sources available at
`http://www.trabucayre.com/enseignement/tp_fpga_sources.tgz`