

Management and analysis of physics datasets, Part. 1

Ninth Laboratories

Stefano Pavinato
10/1/2019

- 1 Laboratory Introduction
- 2 IPBus (oriented for this course)
- 3 IPBus for these labs
- 4 Homework

- 1 Laboratory Introduction
- 2 IPBus (oriented for this course)
- 3 IPBus for these labs
- 4 Homework

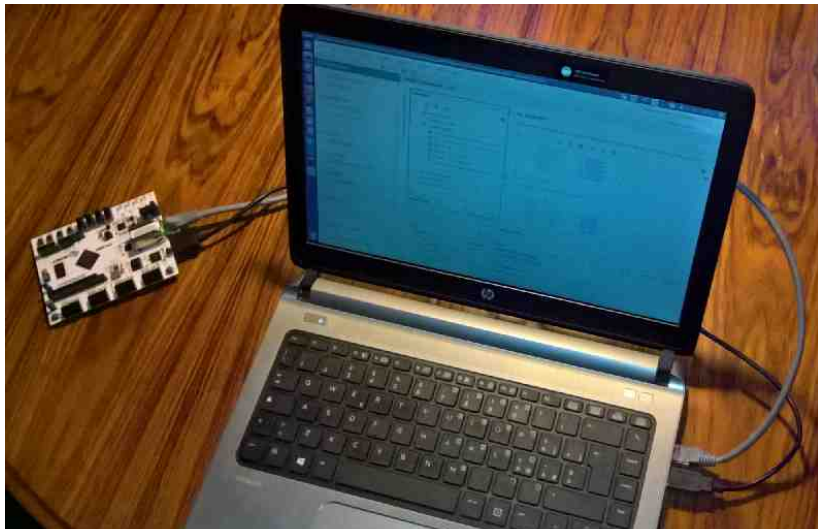
- Integrate the Eighth Laboratory in the IPBus framework.

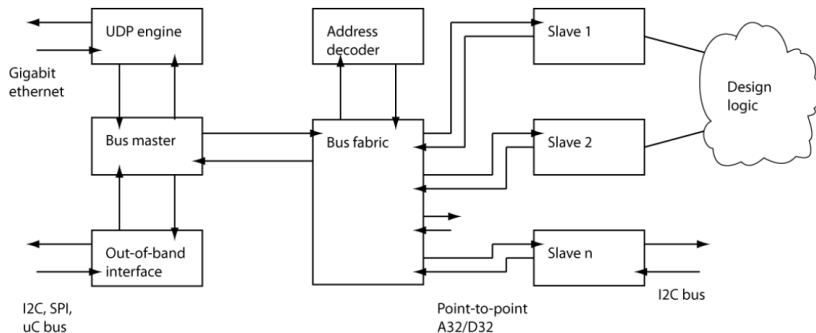
- 1 Laboratory Introduction
- 2 IPBus (oriented for this course)
- 3 IPBus for these labs
- 4 Homework

- The IPbus protocol is a serial packet-based control protocol for reading and modifying memory-mapped resources within FPGA.
- The memory-mapped resources are registers at 32 bit. They are addressed with addresses at 32 bit.
- The IPbus suite of software and firmware implement a high-performance control link for particle physics electronics, based on the IPbus protocol.
- It was developed for the triggering system in the CMS experiment. IPBus is replacing VME control in several large projects.
- Also for data acquisition IPBus is starting to use.

- The IPbus suite (for you) consists of two components:
 - 1 IPbus firmware. It is a module that implements the IPbus protocol within FPGA.
 - 2 uHAL. It is a library that provides a Python API for IPbus reads and writes transactions. In these laboratories ONLY the reads transactions are considered.
- Each IPbus host device (in these laboratories the evaluation board) has an IP address (10.10.10.100) and a port number through which it accepts IPbus control packets.

IPBus - Setup





In uHAL the targets registers layout are specified by XML files, allowing a hierarchical address structure of the memory-mapped resources.

```
<node id="TOP">  
  <node id="A" address="0x000000" mode="block" size="0x100" description="register A" permission="r"/>  
  <node id="B" address="0x000200" description="register B" permission="r"/>  
</node>
```

- **id.** It is the string identifier.
- **mode.** In this labs can be equals to single (single is the default mode if no mode is explicitly declared) or block. Single indicates that the node refers to a single word (i.e. 32 bits) register. Block indicates that the given address is the base address of a block of registers (number indicates from size tag) with a continuous address space (i.e. RAM).

```
<node id="TOP">  
  <node id="A" address="0x000000" mode="block" size="0x100" description="register A" permission="r"/>  
  <node id="B" address="0x000200" description="register B" permission="r"/>  
</node>
```

- **permission.** r indicates a read-only IPbus endpoint (or slave).
- **description.** It is a way of documenting the address table.
The content of the attribute can be accessed by an application.

In order to read the memory block "A" and the memory location "B", you can use the following Python code:

```
a_vector = hw.getNode("A").readBlock(0x100)  
b         = hw.getNode("B").read()
```

Another XML file is used also for configure the link connection.

```
<connections>  
  <connection id="arty7" uri="ipbusudp-2.0://10.10.10.100:50001" address_table="file://arty7_regs.xml"/>  
</connections>
```

- **id**. It is the string identifier.
- **uri**. It is the protocol (ipbusudp-2.0) and location (10.10.10.100) to access a target device in URI format.
- **address_table**. It is the location of the address table file which describes the register space of the FPGA.

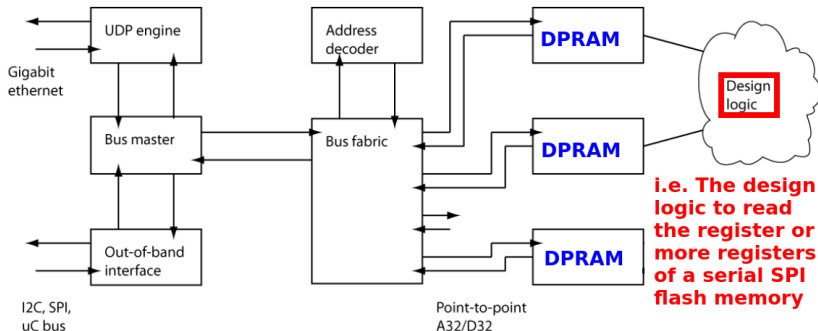
The Python code to use this file is :

```
manager = uhal.ConnectionManager("file://arty7_connection.xml")  
hw = manager.getDevice("arty7")
```

- 1 Laboratory Introduction
- 2 IPBus (oriented for this course)
- 3 IPBus for these labs**
- 4 Homework

What to do?

- 1 You don't have to write Python code (it is given);
- 2 You don't have to write XML files (they are given);
- 3 You have to write only VHDL code. You have to write only the "design logic". The remaining code that implements the IPBus core, the interface from/to the PC and from/to the "design logic" is given.



- DPRAM : Dual Port RAM. It enables independent, simultaneous memory access from two buses with an easy processor arbitration required.
- At this DPRAM block essentially you access from the design logic (i.e. an spi_master) and from the host (i.e. your laptop).
- It is implemented inside the file *ipbus_dpram.vhd*.
- You don't have to write code inside here, but you have to understand how it works, in order to interface it with the design logic. (Not in this lab, but in the next).


```
dpram: ipbus_dpram
  generic map( ADDR_WIDTH => 4)
  port map(
    clk          => ipb_clk,
    rst          => rst_ipb,
    ipb_in       => ipbw(0),
    ipb_out      => ipbr(0),
    rclk         => ipb_clk,
    we           => we_s,
    d            => x"000000" & rxd_s,
    q            => open,
    addr         => addr_s
  );
```

```
type ram_array is array(2 ** ADDR_WIDTH - 1 downto 0) of std_logic_vector(31 downto 0);
shared variable ram: ram_array;
signal sel, rsel: integer; signal ack: std_logic;
begin
```

```
sel <= to_integer(unsigned(ipb_in.ipb_addr(addr_width-1 downto 0)));
```

```
process(clk)
```

```
begin
```

```
    if rising_edge(clk) then
```

```
        ipb_out.ipb_rdata <= ram(sel);
```

```
        if ipb_in.ipb_strobe='1' and ipb_in.ipb_write='1' then
```

```
            ram(sel) := ipb_in.ipb_wdata;
```

```
        end if;
```

```
        ack <= ipb_in.ipb_strobe and not ack;
```

```
    end if;
```

```
end process;
```

```
ipb_out.ipb_ack <= ack;
```

```
ipb_out.ipb_err <= '0';
```

**Code to interface
with the host.
i.e. PC, Laptop ...**

```
rsel <= to_integer(unsigned(addr));
```

```
process(rclk)
```

```
begin
```

```
    if falling_edge(rclk) then
```

```
        q <= ram(rsel);
```

```
        if we = '1' then
```

```
            ram(rsel) := d;
```

```
        end if;
```

```
    end if;
```

```
end process;
```

**Code to interface with
the design logic.
i.e. the code to read the
registers of the serial
SPI flash memory**

- rclk : it is the same clock used in the design logic block, toward you interface;
- we: it is the write enable. If '1' you write the data inside the dual port ram;
- d: the data to write in the dpram. The data must be a vector of 32 bit. Since the data retrieved from the flash memory is a 8 bit data, you have to concatenate it with 24 zero-bits;
- q: the data read from the dpram. It is a 32 bit data.
- addr: it is the address of the dpram. At the memory location with address "addr", you can read or write the data.

```
flash_reader: spi_master_flash
generic map (
    WTIME    => 32,
    TXBITS    => 8*4,
    RXBITS    => 8,
    N_BYTES   => 6
)
port map (
    clock     => clk_base_xc7a_i,
    reset     => vio_rst or rst_ipb,
    txd       => x"03" & x"000000", -- Command & address
    start     => '1',
    ready     => open,
```

```
    miso      => flash_miso_s,
    mosi      => flash_mosi_s,
    sclk      => flash_clk_s,
    cs        => flash_cs_s,
    wr_pr_o   => flash_w_s,
```

SPI signals

```
    rxd_out   => rxd_s,
    we_out    => we_s,
    addr_out  => addr_s
```

**Signals to inter-
face to DPRAM**

```
);
```

- This component, implemented in the *spi_master_flash.vhd* file, read *NBYTES* consecutive registers of the serial SPI flash memory.
- It starts reading the flash memory register with address 0x000000 and then it reads the consecutive five registers.
- When it has read a register from the flash memory, it writes the data stored in the flash memory register in the dpram block at the same address;

The *spi_master_flash.vhd* file can be split in:

- 1 a FSM that manages the interfacing between the dual port RAM and the logic that you implemented in the previous laboratory. It is important you understand how it works the FSM, since you have to modify it for the next laboratories.
- 2 The component *spi_master*, you implemented in the last laboratory.

Running the python script *read_flash_reg.py*, that you can find it in the folder *software*, you should get this screen:

```
@ Address [0x000000] is stored the value : 0x00
@ Address [0x000001] is stored the value : 0x01
@ Address [0x000002] is stored the value : 0x02
@ Address [0x000003] is stored the value : 0x03
@ Address [0x000004] is stored the value : 0x04
@ Address [0x000005] is stored the value : 0x05
```

Before you have to connect your laptop network card to the evaluation board with the ethernet cable given.

- 1 Laboratory Introduction
- 2 IPBus (oriented for this course)
- 3 IPBus for these labs
- 4 Homework

- You download all the folder *Lab9/firmware*, open the project (the file .xpr) and check if the bitstream is generated.
- Unfortunately you have to do this in your personal laptop. You need the license for synthesize the "Tri Mode Ethernet MAC" core. The generation of this license is explained at the last point of the file "20181105 - Installazione_Vivado - Pavinato.pdf".
- Configure your network connection parameters and your operating system as described in the following slides.
- Finish the exercise of the eighth laboratory and integrate it in the IPBus framework. Essentially import your file *spi_master.vhd* in this Vivado project.

- The default IP address value was set to 10.10.10.100.
- You have to change the settings of your laptop network card in order to be in same subnetwork of the IPBus IP address; i.e. 10.10.10.1. Depending on your operating system, you can read these guides:
 - 1 Ubuntu
 - 2 Windows
 - 3 Mac
- If you want to change the IP address value of the evaluation board, you have to modify the field `uri` in the file *arty7_connection.xml* and the value assigned to the signal *s_ip_addr* in the file *top_level.vhd*.
- In order to check if the configuration is ok, text in the terminal "ping 10.10.10.100". The ping must be successful.

- IPBus works only in Linux operating systems (O.S.).
- Depending on which Linux distribution you have, you find the instruction to install the uHAL libraries at the following link:
 - uHAL
- In order to check if the installation of the libraries was successfully, open a terminal and text:
 - python
 - import uhal

If no errors are given (i.e. `ImportError: No module named ...`), the installation is ok.

- Or if you do not have a machine with a Linux operating system you can use a virtual machine.

If you need a virtual machine you can follow these steps to install uHAL:

- Depending on the operating system of your machine you can download the hypervisor you prefer. Choose between:
 - Virtual Box
 - VMware

I suggest VMware.

- Then, depending if you chose Virtual Box or VMware, you can download the Ubuntu image here:
 - Ubuntu image

Download the 64 bit version.

- Finally open the Ubuntu image according to the hypervisor you chose and install uHAL.

- A virtual machine was already implemented. It is based on VMware and it runs an Ubuntu distribution.
- It is about 8 GByte. So I can give you the virtual machine through a USB pendrive.
- It was tested in Linux and Windows operating systems.
- In the desktop there is a folder with the python script to run, in order to communicate, via IPBus, with the evaluation board.

A working bitstream is given since you can check if the software works.

- The generation of the bitstream requires several minutes. The greatest part of this time is for synthesis of the IPBus core. That surely it works.
- To save time you can set as top module the file *top_level_light.vhd*.
- In this top level all the code regarding the IPBus core has been commented.
- Furthermore the code implementing the dual port ram (*dpram.vhd*), has been modified: some lines of code have been commented and a VIO core has been added (*dpram_light.vhd*).




The VIO core is used to set the address of the memory location of interest. Moreover with the same core you can see the data contained in this memory location.

```
p_vio: process(vio_addr)
variable i : integer;
begin
    i := to_integer(unsigned(vio_addr));
    vio_ram <= ram(i);
end process;

vio_dram0: vio_dpram
PORT MAP (
    clk => rclk,
    probe_in0 => vio_ram,
    probe_out0 => vio_addr
);
```

Hmw - Light Version (3)

hw_vio_1

Name	Value	Activity	Direction	VIO
 vio_rst	<input type="text" value="0"/>		Output	hw_vio_2
>  dpram/vio_addr[9:0]	[U] 4		Output	hw_vio_1
>  dpram/vio_ram[31:0]	[H] 0000_0004		Input	hw_vio_1

After setting the address (*vio_addr*) through the VIO core in the *dpram_light.vhd* file, if you read the right value you are sure that also the IPBus readings using the Python script work (after setting the network connection conveniently).

With the light version you can use, as usual, the PC of this Lab room.