UNIVERSITÀ DEGLI STUDI DELL'AQUILA

Embedded Systems (Dr. L. Pomante)

A.A. 2017/2018

HomeLab ARTY FPGA

(VHDL's Exercises and Introduction to Vivado Design Suite)

Author

Eng. Giacomo valente (giacomo.valente@graduate.univaq.it)

Version 3.0: 05/10/2017

Index

Before start: preparation

All the illustrated examples have been tested using VirtualBox on Windows 7 operating system. VirtualBox is a program that, starting from a host operating system and a guest one, makes possible to run the latter on the former by emulating it. In the case of test, Xubuntu 16.04 64 bits operating system has been run as guest on the top of Windows 7 host. What is required to the host is that it is able to run VirtualBox: if the host were a different operating system with respect to Windows 7 (e.g. Windows 8/8.1/10, Ubuntu, etc.), it would be necessary only to install VirtualBox on these other hosts and run the guest on top.

Components

- ARTY Development Board
- USB A to Micro-B Cable

Required software

- Virtual Box [link]
- 30 day evaluation license for Vivado: in order to obtain this license, connect to Xilinx web site, make a subscription and download the license file.
- *puTTY* [download], or any terminal emulator with serial console functionality (e.g. *HyperTerminal*, *minicom*)

Time and feedbacks

In order to perform the homelab, one week is allowed. The board are provided on Friday during the Embedded Systems lesson, and requests will be served in a First input/First Output fashion. Any feedback about problems or suggestions would be highly appreciated.

Virtual Machine

Install Virtual Box on the host PC (download the latest version).

Import the Virtual Machine image (obtained from the Professor) on Virtual Box and launch the booting. On the guest operating system, there are:

- Vivado Design Suite 2017.2
- Board description files to use ARTY development board
- Driver to communicate with ARTY development board

ARTY

Take a deep look to Reference Manual (RM) [link] and verify, before the beginning of the homelab, that the board is properly configured. This means that the *jumper* configuration must be checked: a jumper serves as connection between two points. In general, it can allow certain voltages to be applied in some parts of the circuit, so take care when selecting jumper configurations: they can damage the board if not correctly set. The rightness of the jumpers position can be checked on RM: select them in order to use the board with JTAG connection.

Communication test between Board and PC

Connect the board to the PC through USB cable, and it will be automatically powered-on.

Connect the ARTY board to the Virtual Machine by selecting it among the devices:

Open a shell on Xubuntu and check the connection of the board to the virtual machine by using:

djtgcfg enum

Now, check the connection of the board to Vivado:

- Set environment variables with the command:

  source /opt/Xilinx/Vivado/2017.2/settings64.sh

- run Vivado with the command:

  vivado

- Select *Open Hardware Manager* from the icons.

Select the link *Open Target* and click to *Auto-Connect*. The ARTY board has to be recognized by the environment without errors.

Further investigations

- Standard IEEE 1149.1: Standard Test Access Port and Boundary-Scan Architecture (JTAG)
- ARTY board, documentation [link]

Section 1: Simulation and Synthesis of simple combinatorial circuits

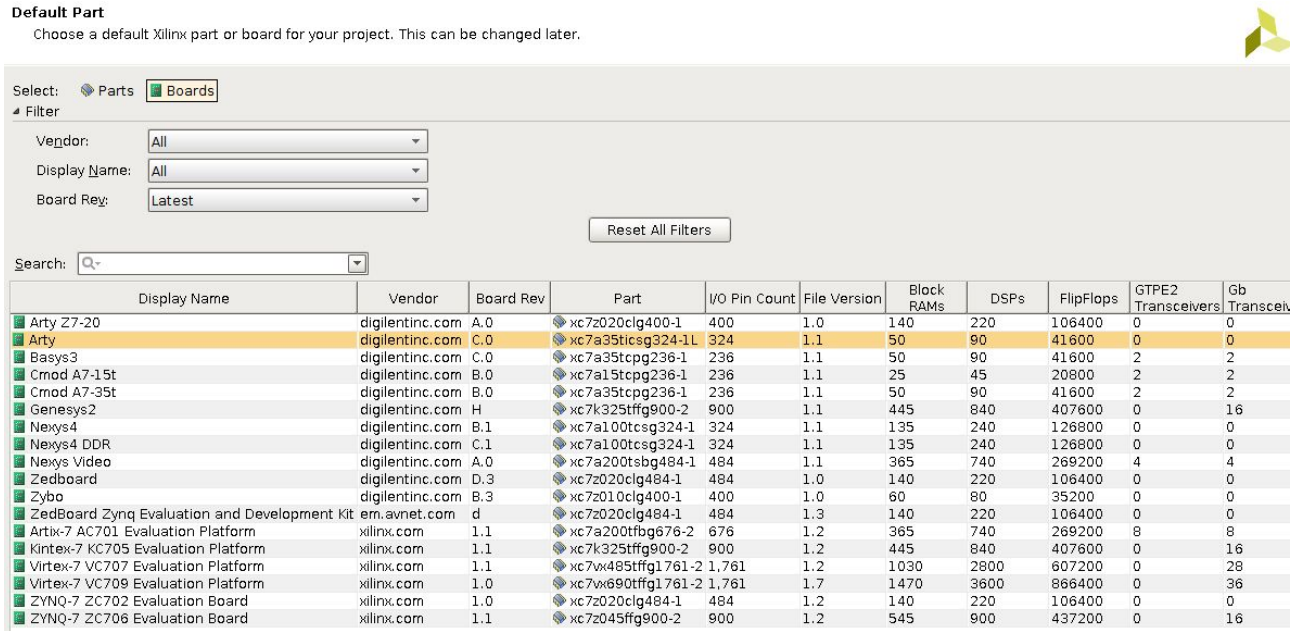
This section refers on how to use Vivado Design Suite for simulation and synthesis of a simple digital circuit, the half adder.

1.1 – VHDL Description of Half Adder

The creation of a new project in Vivado and description of the half-adder come through the following steps:

- Exit from Vivado and re-launch it from linux shell
- Select *Create New Project* from icons
- Click *Next* and gives project name and project location (pay attention to select destination folders without spaces in the folder name)
- Click *Next* and select RTL Project
- Click *Next* and Select VHDL Language and Mixed simulator. Do not add sources and click *Next*. Do not add Existing IP and click *Next*. Do not add constraints and click *next*.

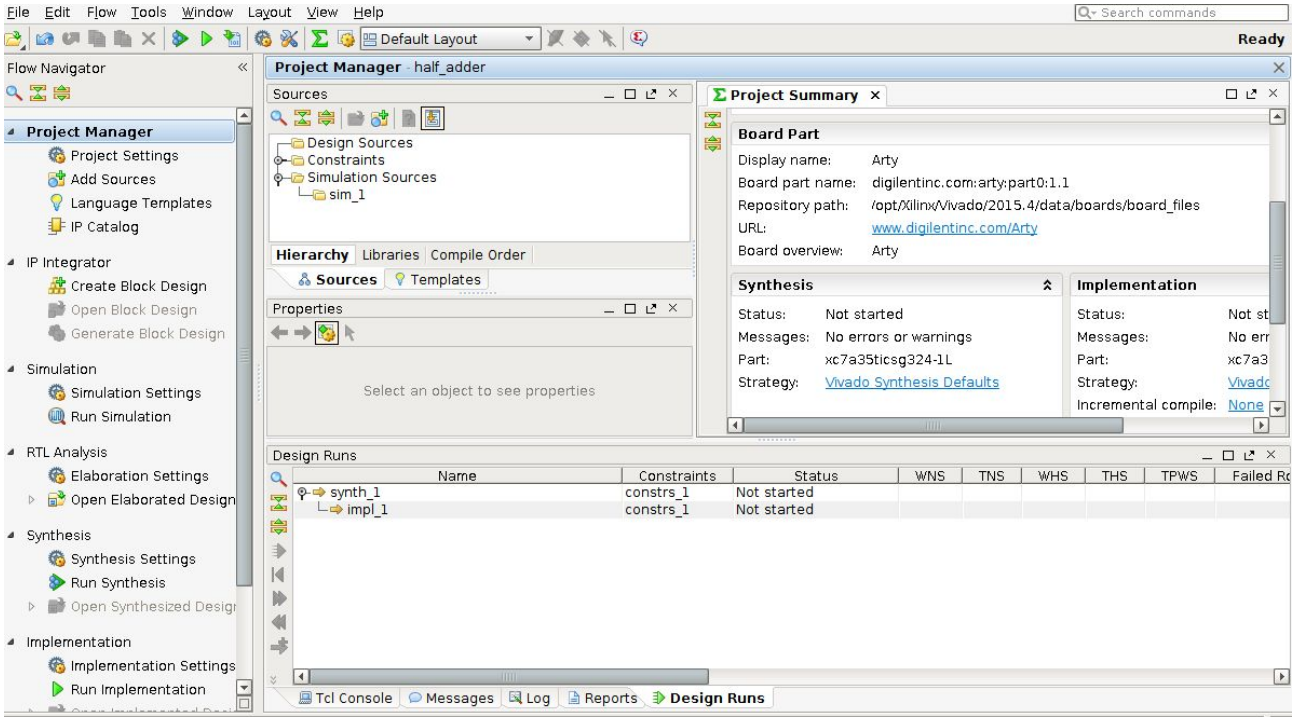
Select ARTY Board from the list and click next:



Display Name	Vendor	Board Rev	Part	I/O Pin Count	File Version	Block RAMs	DSPs	FlipFlops	GTPE2 Transceivers	Gb Transceiv
Arty Z7-20	digilentinc.com	A.0	xc7z020clg400-1	400	1.0	140	220	106400	0	0
Arty	digilentinc.com	C.0	xc7a35ticsg324-1L	324	1.1	50	90	41600	0	0
Basys3	digilentinc.com	C.0	xc7a35tcpg236-1	236	1.1	50	90	41600	2	2
Cmod A7-15t	digilentinc.com	B.0	xc7a15tcpg236-1	236	1.1	25	45	20800	2	2
Cmod A7-35t	digilentinc.com	B.0	xc7a35tcpg236-1	236	1.1	50	90	41600	2	2
Genesys2	digilentinc.com	H	xc7k325tffg900-2	900	1.1	445	840	407600	0	16
Nexys4	digilentinc.com	B.1	xc7a100tcsg324-1	324	1.1	135	240	126800	0	0
Nexys4 DDR	digilentinc.com	C.1	xc7a100tcsg324-1	324	1.1	135	240	126800	0	0
Nexys Video	digilentinc.com	A.0	xc7a200tsbg484-1	484	1.1	365	740	269200	4	4
Zedboard	digilentinc.com	D.3	xc7z020clg484-1	484	1.0	140	220	106400	0	0
Zybo	digilentinc.com	B.3	xc7z010clg400-1	400	1.0	60	80	35200	0	0
ZedBoard Zynq Evaluation and Development Kit	em.avnet.com	d	xc7z020clg484-1	484	1.3	140	220	106400	0	0
Artix-7 AC701 Evaluation Platform	xilinx.com	1.1	xc7a200tfbg676-2	676	1.2	365	740	269200	8	8
Kintex-7 KC705 Evaluation Platform	xilinx.com	1.1	xc7k325tffg900-2	900	1.2	445	840	407600	0	16
Virtex-7 VC707 Evaluation Platform	xilinx.com	1.1	xc7vx485tffg1761-2	1,761	1.2	1030	2800	607200	0	28
Virtex-7 VC709 Evaluation Platform	xilinx.com	1.0	xc7vx690tffg1761-2	1,761	1.7	1470	3600	866400	0	36
ZYNQ-7 ZC702 Evaluation Board	xilinx.com	1.0	xc7z020clg484-1	484	1.2	140	220	106400	0	0
ZYNQ-7 ZC706 Evaluation Board	xilinx.com	1.1	xc7z045ffg900-2	900	1.2	545	900	437200	0	16

Review the settings and click finish.

Now the following screen will appear:

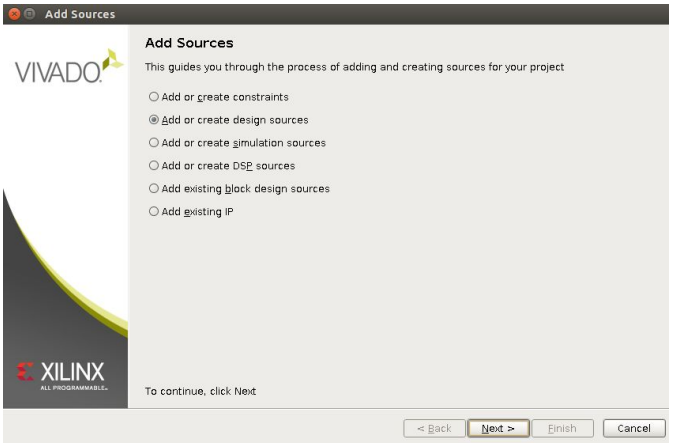


The screen is divided in various different areas:

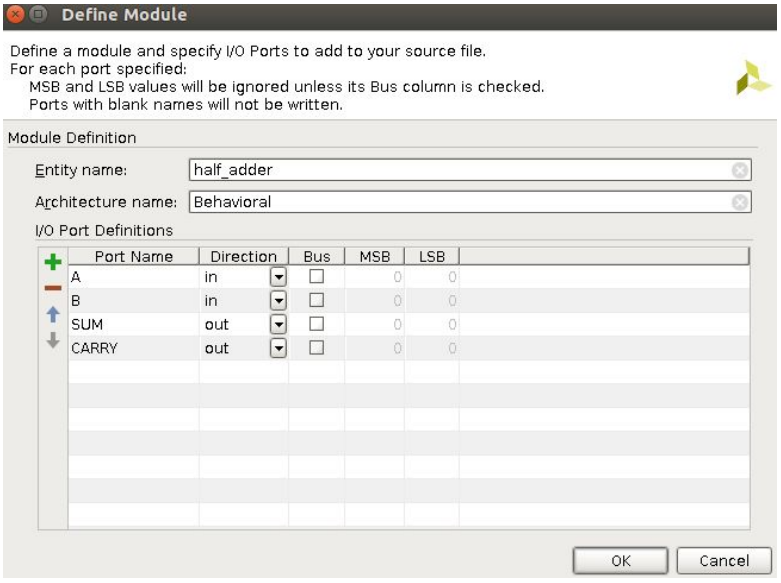
- *Source Window*: it is located at the top of the environment by default, and it contains all the sources associated to the project. For example, *Design Sources* represent HDL files (that can be VHDL or Verilog), while *Constraints Sources* represent constraints files

- *Flow Navigator Pane*: it is located at the left of the environment by default, and it contains the main steps that can be performed in a design with Vivado (such as *Simulation*, *Synthesis*, etc.)

The project just created does not contain source files. Add one by right clicking the mouse into source window and selecting *Add Sources*.



Select *Add or create design sources* and click *Next*. Click on the green cross at the top left and select *Create File*. Specify the name (e.g. *half_adder*, pay attention to not insert spaces in the file name) and click *OK*. Then click *Finish*. A wizard will be opened: it allows to define module parameters (i.e. entity name and architecture name).



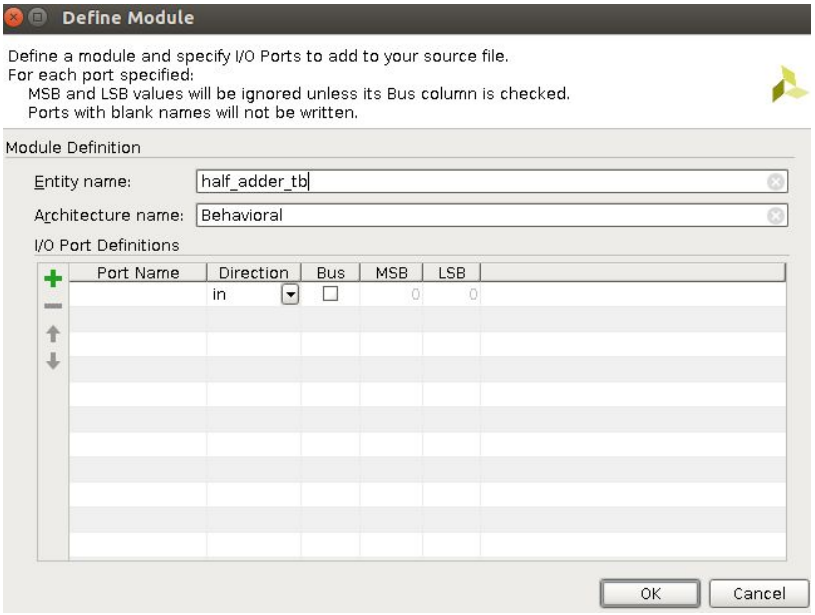
After filling various fields keeping in mind the half adder structure, click *OK* to complete the source file creation. Open the new generated source file by double clicking on it in the source window. Internal to this file you can see that the structure of the VHDL code has been automatically set up. It is possible to note how it is necessary only to write the architecture section (i.e. the functionality of the system). Complete the description as seen during the course lesson:

```
ARCHITECTURE Behavioral OF half_adder IS

BEGIN

 SUM <= A xor B;
 CARRY <= A and B;

END Behavioral;
```

At this point, the description of half adder module is complete. Save the modified source file.

1.2 - Simulation

Verify that the behavior of the block is the one expected: i.e. perform a *Behavioral* simulation.
For this it is necessary to create a *testbench* to stimulate the described circuit. Add to the project another source file, but now select *Add or create simulation sources*. Then create a new file from the top-left green arrow and select *Finish*. Do not insert ports to the entity (it is a *testbench*):



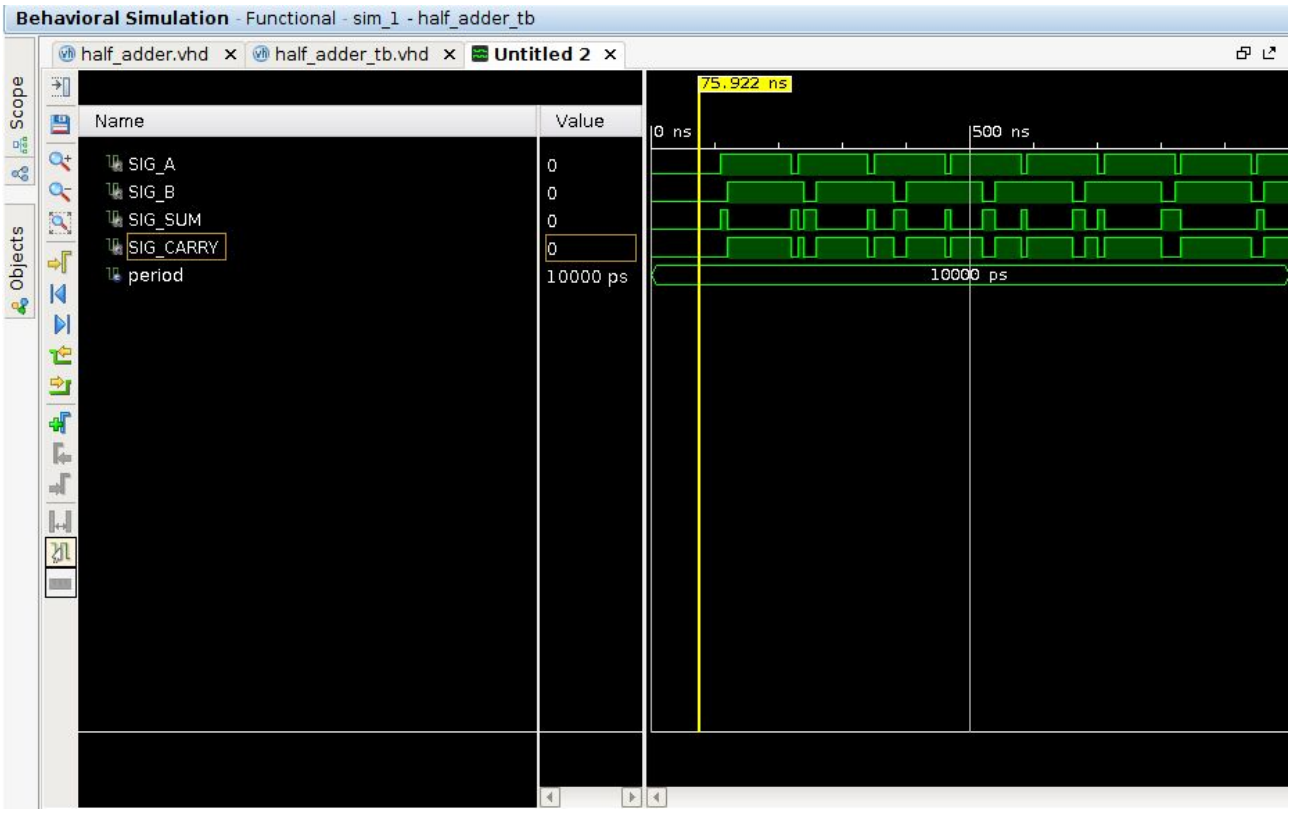
From source window, select the *testbench* from *Simulation Sources*:

Consider the *testbench* contained in the homelab folder. In the code section it can be noted that a constant *period* has been defined: this one can be used to define time interval variations of input signals. By means of the process statement and the wait instruction, it is possible to model the desired signal input evolutions.
Examine, for example, the following piece of code and draw the equivalent timing diagrams. Then, add these stimuli to the right section of the provided *testbench* file:

```
A_stim_proc: process
begin
    wait for 100 ns;
    SIG_A <='0';
    wait for period;
    SIG_A <='1';
    wait for period;

end process;

B_stim_proc: process
  begin
    wait for 100 ns;
```

```
        SIG_B <='0';
        wait for 2*period;
        SIG_B <='1';
        wait for 2*period;

    end process;
```

Once defined input signals evolution, it is possible to run the simulation: in the flow navigator pane, select *Simulation Settings* and check if Vivado Simulator is set and if the top module is the *testbench* that was already written. Then, in the *Simulation Tab*, set the *Simulation Runtime*. At the end, select *OK* and select *Run Simulation* and *Run Behavioral Simulation* in the flow navigator pane.
This last step will launch the *XSim* simulator, and this one will show automatically the timing diagrams of the simulation process (click to various buttons to show waveforms with different levels of detail).



Check simulation results by verifying that half adder outputs behave good with respect to the inputs.

1.3 -Synthesis

Close *XSim*, go back to source window, select the half adder module and click on *Run Synthesis* in the flow navigator pane. At the end, it is possible to check reports, to have a look on synthetized design or continue by running *Implementation*.

1.3 – To Do

Try to develop VHDL description of the following combinatorial circuits:

- Multiplexer

- Demultiplexer
- Decoder
- Coder

Verify that the circuit behavior is correct by means of behavioral simulation. Keep in mind that the description is not uniquely and that if the circuit is simulatable, it does not imply that it is synthesizable. Be aware only to the simulation results, not the implementation.

Section 2: Sequential circuits, I/O and constraints

In a development board, like ARTY, the FPGA is surrounded with hardware components (e.g. UART -> USB converters, VGA, PMODs, LEDs, switch buttons, push buttons, etc.). For the use of this components, it must be possible to connect the implemented blocks into FPGA with external hardware.

In this section, a first sequential circuit into FPGA will be realized, and how to show the outputs by means of LEDs placed on the board will be illustrated [5].

2.1 – Counter

Create a new project in Vivado (e.g. named *homelab_counter*), still targeting the ARTY board, and add a new source file having the entity named *hl_counter*. The possible structure of the counter is reported in the homelab folder. It is possible to note that it is necessary to generate a "slow clock", otherwise the frequency of the counting would be too fast and it will not be visible to human eye through a LED. Try to answer to the following question: if you don't generate slow clock, shall you see the LED forever on or forever off?

Once defined the architecture of the counter, it is possible to go to next phase. At the middle of work, it is possible to perform a behavioral simulation. For this sake, a *testbench* has to be defined, in the same manner of the last exercise (i.e. the half adder), and the counter has to be instantiated into this one and properly excited. Pay attention to the fact that, after simulation launch, by default one is able to look at the behavior of input and output of the top-level section of the system, that is the entity: remember that a slow clock has been used internally to the counter, in order to have the ability to show the LED blinking on the board. From the simulation point of view, this could require a very long simulation time. In order to check the functionality with smaller simulation time, one can have a look to the internal signals of the counter module. This can be done by navigating into *XSim* through *Object Tab* and *Scope Tab*, and by dragging and drop necessary signals. Note that the simulation should be re-launched in order to include these signals.

Going toward the implementation phase, remember that after the downloading of the *bitstream* on the FPGA, the counter outputs drive directly LEDs sited on the board. In order to do this, some constraints have to be defined for the implementation by operating on *Xilinx Design Constraints (XDC)*. An XDC file is a list of functions written in *TCL* language [link] that allows to make some operations targeting constraints in the Vivado Design Suite. Constraints can be of different types: board constraints, timing constraints, etc. In this contest, the interesting is on board constraints. The way (not the unique) to operate with constraint, in this context, is to start from a single XDC file that contains all the board constraints related to the ARTY board, initially commented out. In order to use this XDC file, it must be downloaded from the *Digilent* repository [link]. Have a look on the internal of the file, and note a list of TCL commands. The ARTY board has various inputs and outputs elements, as indicated in RM. Considering that the counter has 1 clock input, 1 reset input, 1 counter output (composed of 4 bits), there is the need of 1 input clock (e.g. provided by an oscillator), 1 input reset (e.g. a push-button) and 4 output counter (e.g. 4 LEDs). After checking that these elements are available on the board, the XDC file can be imported into the project:

- Going in the source pane and adding a source
- Selecting *Add or Create Constraints*
- Adding the XDC file using the top-left green cross (set the *Copy constraints files into the project*)
- Clicking *Finish*

Now the XDC file can be modified, in order to make required board connections. In particular, the following lines can be commented out:

- Connect the clock:

  set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33 } [get_ports { clk }]

- Connect the output:

  set_property -dict { PACKAGE_PIN H5 IOSTANDARD LVCMOS33 } [get_ports { count_out[0] }]

  set_property -dict { PACKAGE_PIN J5 IOSTANDARD LVCMOS33 } [get_ports { count_out[1] }]

  set_property -dict { PACKAGE_PIN T9 IOSTANDARD LVCMOS33 } [get_ports { count_out[2] }]

  set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33 } [get_ports { count_out[3] }]

- Connect the reset:

  set_property -dict { PACKAGE_PIN D9 IOSTANDARD LVCMOS33 } [get_ports { reset }]

The synthesis process and *bitstream* generation can be completed now, by clicking on *Generate Bitstream* in the flow navigator pane. At the end, board can be connected to the PC and to the Virtual Machine. The FPGA configuration can be done by using *Hardware Manager*, in particular:

- selecting *Open Hardware Manager* in the screen at the end of *bitstream* generation or in the flow navigator pane
- clicking on *Open Target* and asking for *auto-connection*
- Clicking on *Program Device* and selecting the *bitstream* generated file (located in the *runs/impl_1* folder of the project tree).

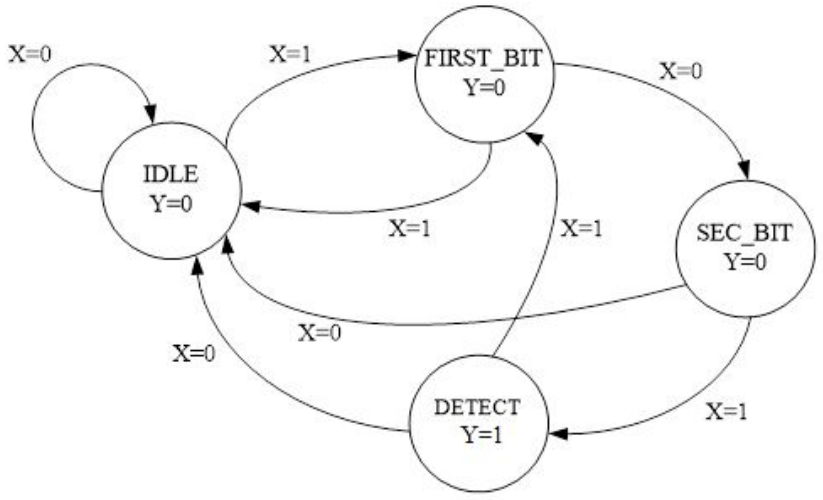
At the end of operation, the LEDs 0-3 on the board should illuminate according to counter evolution. It can be tested also the reset by pushing the BTN0 push-button.

2.2 - Sequence detector (Finite State Machines - FSM)

In this section, it will be illustrated how to realize a circuit able to detect an exact binary sequence on the input line. Suppose that a synchronous detector has to be realized, therefore data input line (X) and clock signal are expected as inputs: data will then be read in the rising edge of the clock. The output (Y) will be equal to 1 only when the sequence is detected.

Using the HDL for system description, XDC for constraints and Vivado Design Suite for implementation, the task is to implement a system that solves the problem and verify the functionality using two switch to "simulating" respectively the clock signal and input signal.

The following state diagram describes a sequence detector in which the detected word on the serial port (101) is "hardcoded".



A draft of VHDL code for a Finite State Machine (Moore type, i.e. the output is determined only by the current machine state, and the next machine state is determined by the input and the current machine state) that implements the behavior of the previous diagram.is provided into the homelab folder. The definition of an enumerated data type is convenient for the state representation. It is left as exercise the completion of the project (by assigning XDC to specify the right connection between sequence detector, board LEDs and board switches and perform the FPGA configuration in the same manner of the previous exercise).

2.3 – To Do

Simulate the GCD of the VHDL lesson: do it for every kind of hardware description, i.e. behavioral, RTL with FSMD and RTL with FSM and datapath. You already have the files, or you have the necessary knowledge to implement missing parts.

Section 3: PicoBlaze

PicoBlaze is an 8 bit RISC soft processor distributed by Xilinx. The following examples are related to the use of this one. Files related to PicoBlaze can be downloaded [link].

3.1 – Input/Output and base of programming

The PicoBlaze is provided by Xilinx in the form of HDL code, together with documentation [link] (remember that the Artix 7 FPGA is used on ARTY board).

The downloaded file is composed by different elements, for example:

- *kcpsm6.vhd* file that represents the PicoBlaze soft-processor description
- PicoBlaze documentation
- Necessary tools for the use of PicoBlaze, including the assembler

In order to use PicoBlaze, the following steps are required:

- Start a Vivado project
- Add PicoBlaze description into the project (*kcpsm6.vhd*)
- Write a program to be assembled by the assembler and executed by PicoBlaze; create a text file with a list of PicoBlaze instructions, for example a program that read inputs and write them in outputs:

  ; read from switches and display to the LEDs

  start: INPUT s0,00

  OUTPUT s0,01

  JUMP start

  Save this file with *.psm* extension, for example *test.psm*.

- Generate the ROM containing instructions to be executed using the *kcpsm6.exe* program on a Windows operating system. In order to do this, a command window has to be started, then reach the executable folder of the assembler and finally launch the program with the following syntax:

  kcpsm6.exe _name_of_psm_file.psm

The utility generates various files: the *test.vhd* file corresponds to the VHDL description of the instruction ROM containing the program previously written in assembly. After importing the *test.vhd* file into project, there are 2 blocks: the soft-processor and its instruction memory. Specifically, the small assembly program previously reported reads input data (from 0x00 address) and copies them to the output (to 0x01 address) indefinitely. Input and output instructions use the s0 register for data buffering. The 2 blocks have to be instantiated and inserted in a higher-level module. Then, the functionality of the design can be verified, for example, by connecting 4 input lines and 4 output lines of PicoBlaze respectively to board switches and LEDs. In this mode, the PicoBlaze is used to detect the change of position of the switches (passed to *on*) and turn on LEDs.

In order to instantiate correctly a PicoBlaze into a higher level module, it is necessary to record the input and output data in two temporary registers (synchronous accessible on the base of the address lines and processor

read and write strobe). An example of the top-module is reported in *embedded_pico.vhd* file contained in the homelab folder. Before provide the synthesis, specify the right pin connections of input and output by means of constraints. At the end, test all the system on the FPGA.

3.2 – Further investigations

Modify the previous written program including different instructions with respect input/output (e.g. try to use the flow control instructions). Verify the functionality by implementing the system on FPGA.

Section 4: Serial Communication

Xilinx provides source files for an UART controller together with PicoBlaze: these ones are contained into the *UART_and_PicoTerm* folder of the downloaded archive. It is possible to use this core to realize a serial communication between PC and the soft-processor. Read the documentation contained in the different folders, in particular in the *UART_and_PicoTerm* folder, in order to better understand the UART IP-core.

4.1 – PicoBlaze & UART

To demonstrate the functionality of the UART controller, it can be analyzed how to add to PicoBlaze the transmission section [6]. The exercise completion with addition of receiver section is left to the reader.

Create a new project and unzip the archive containing PicoBlaze files into the main project directory. Import the following files into the project:

- *kcpsm6.vhd*: the PicoBlaze
- *uart_tx6.vhd*: wrapper that contains components of UART transmission section;

The assembly program contained in the homelab folder (*test_uart.psm*) causes the PicoBlaze transmission (indefinitely loop) of a character string on the serial port. As done on the Section 3, start from the provided program file and copy it into the assembler folder. Then repeat the ROM generation procedure. Once obtained the *test.vhd* file, import it into to the project. At this point there are all the necessary components to verify serial communications: a processor, an UART transmitter and a program memory (ROM) containing instructions to send the processor data to the UART. Create a top-level for the project, in which instantiate various components and connect them in right manner. There is an example code in the home folder (*embedded_pico.vhd*): in this example, the baud rate generator has been configured to work at *38400 bps* speed. The top-level has one input line (clock signal) and one output line (UART TX line). There is a USB-UART connector on the board, that can be used to output what is transmitted by PicoBlaze. In the XDC file, it is possible to define the connection of the clock to the oscillator and of the *txd* signal to the USB-UART controller (pay attention to the fact that on XDC file the direction is intended from the USB-UART controller, so the *txd* has to be connected to the receiver pin). After synthesis execution and device configuration, use a terminal emulator (e.g. *puTTY*) to *listen to* the serial port. Select the proper serial interface and specify the right communication settings. After port opening, the sent message from the PicoBlaze should be printed (that is a kind of "Hello").

4.2 – Further Investigations

Implement the UART receiver section.

Section 5: References and useful readings

[1] ARTY Reference Manual [link]

[2] Pong P. Chu – FPGA Prototyping by VHDL Examples - Wiley

[3] VHDL Tutorial – Learn by examples [link]

[4] PicoBlaze User Guide [link]

[5] 4- Bit Counter with Xilinx ISE 9.2 and Spartan 3E [link] – tutorial that has inspired the example 3.1

[6] Ultra-Compact UART Macros for Spartan-6, Virtex-6 and 7-Series, Ken Chapman [link]

Acknowledgements

We would like to thank Fabio Federici, Walter Tiberti, Gianni Rea, Federico Angeloni, Mattia Micozzi, Celestino De Crescente Pinti, Marco Mirabilio, Nazzareno Valente and Donatella Zimei for the support to solve some issues.