EEC0055 - Digital Systems Design

2019/2020

Laboratory 2 - V1.0
4-5 November 2019

In this laboratory the students will implement a custom digital system in a FPGA-based prototyping platform, exercising all the design and verification stages. They will start with a complete design (Verilog RTL modules), execute the FPGA design flow using the XILINX ISE design tools and QuestaSim simulator, and practice with the Digilent Atlys board and the associated software tools. The reference manual of the Atlys board can be found online in https://reference.digilentinc.com/atlys/atlys/refmanual .

Important notice

Prior to the lab class the students should read the laboratory guide, install the design kit archive and browse through the Verilog source files to understand the organization of the whole design. This laboratory project must be completed within the lab class time (2h maximum) and the reference duration indicated for each step should not be exceeded.

1 - Introduction

The design provided for this work implements the block diagram shown in figure 1. The circuit instantiates the rectangular to polar calculator designed in the first laboratory project and includes a set of additional modules to access it from an external computer through a low speed serial link. The block **rec2pol_all** (`rec2pol_all.v`) includes the same datapath designed in the previous laboratory project (`rec2pol`.v) and a finite state machine (`rec2pol_control.v`) that implements the control path to generate the **start** and **enable** signals. The modules **uart** (`uart.v`) and **ioports** (`ioports.v`) implement a set 32-bit ports accessed via a serial port (16 output ports and 8 inputs ports). The whole system is synchronous with a global clock signal running at 100 MHz (10 ns period).

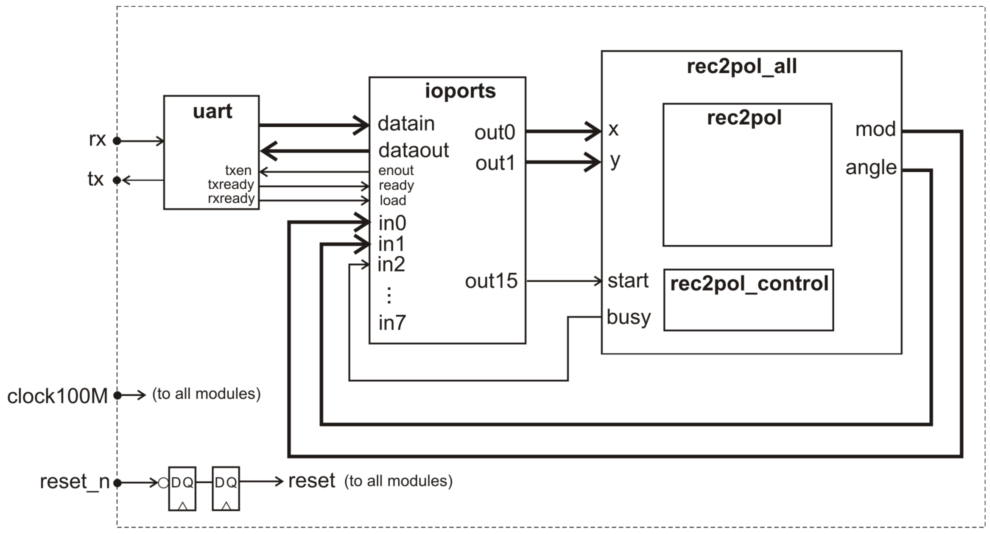


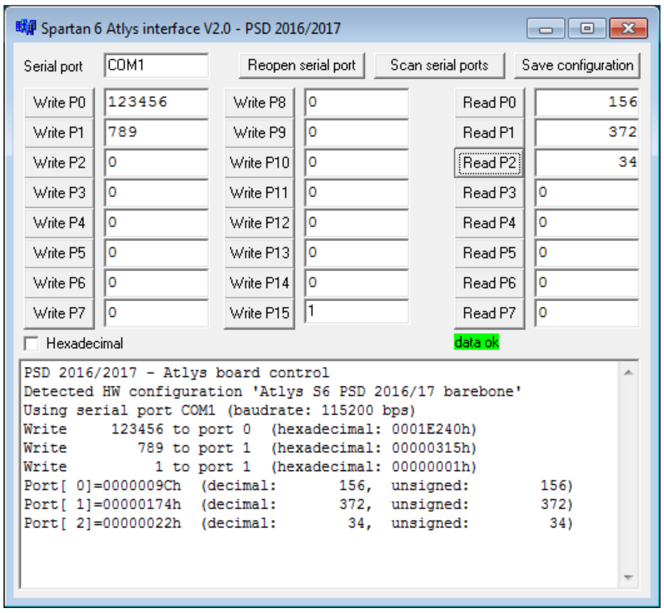
Figure 1 – Simplified block diagram of the reference design.

This circuit can be accessed from an external system (for example a PC) using a reduced set of commands that allow to write and read data to/from the 32-bit ports of module **ioports** (table 1). A basic Windows application (figure 2) implements these commands. Note that this application displays the raw 32-bit unsigned integer values that have to be converted to the appropriate number format to compare the outputs to the expected results. A Matlab function is provided that already does those numeric conversions.

Table 1 – I/O commands implemented by module ioports.

Function	Command	Data (1)	Operation
RESET	8'b0001_xxxx	none	Set all output ports to zero ('xxxx' are don't care)
WRITE	8'b0010_aaaa	4 bytes, write to ioports	Write a 32-bit word to port address 'aaaa' (2)
READ	8'b0011_aaaa	4 bytes, read from ioports	Read a 32-bit word from port 'aaaa' (3)

Notes:
(1) The 32-bit word is written/read in big endian format (MSByte first), following the 1-byte command.
(2) Output port 15 (4'b1111) only keeps the data written during a single clock cycle (useful to generate 1-clock cycle pulses)
(3) Module ioports only implements 8 input ports (addresses 0 to 7). Reading from port 15 (4'b1111) returns a 32-bit identification number set as the parameter ATLYS_HWID (defaults to 32'h2019_2020)



Serial port: COM1

Write P0	123456	Write P8	0	Read P0	156
Write P1	789	Write P9	0	Read P1	372
Write P2	0	Write P10	0	Read P2	34
Write P3	0	Write P11	0	Read P3	0
Write P4	0	Write P12	0	Read P4	0
Write P5	0	Write P13	0	Read P5	0
Write P6	0	Write P14	0	Read P6	0
Write P7	0	Write P15	1	Read P7	0

☐ Hexadecimal

```
PSD 2016/2017 - Atlys board control
Detected HW configuration 'Atlys S6 PSD 2016/17 barebone'
Using serial port COM1 (baudrate: 115200 bps)
Write     123456 to port 0  (hexadecimal: 0001E240h)
Write        789 to port 1  (hexadecimal: 00000315h)
Write          1 to port 1  (hexadecimal: 00000001h)
Port[ 0]=0000009Ch  (decimal:       156,  unsigned:        156)
Port[ 1]=00000174h  (decimal:       372,  unsigned:        372)
Port[ 2]=00000022h  (decimal:        34,  unsigned:         34)
```

Figure 2 – The software application for interfacing with the Atlys board.

In this design module **ioports** connects to the sequential rectangular to polar calculator: the operands **x** and **x** (32 bits) are the data written to ports 0 and 1; the sequential process implemented by module **rec2pol** is initiated by writing 1 to port 15 (asserting input **start**, connected to the LSB) and the final results (the modulus and angle) are retrieved from the input ports 0 and 1. Reading from port 2 will return in the LSB the status of the output busy. However, as the **rec2pol** module will complete one calculation in 33 clock cycles (330 ns) and this is only a small fraction of the time needed to transmit a single byte through the serial port, the results can be read immediately after the write to port 15 to assert the input **start**. The 5 general purpose push-buttons, the 8 slide switches and the 8 LEDs can be later used freely in your design.

2 - Installation of the reference project

Download the archive `PSD-1920-LAB2.zip` and extract all files to a new working directory. Do not install this into the desktop or any other directory containing spaces or other special characters (like accents) in the full pathname. The installed directory tree is:

Directory	**Contents**
`./Matlab`	Matlab scripts (includes code for interfacing with the system)
`./doc`	Documentation (this guide and the manual of Atlys FPGA board)
`./impl/xrec2pol`	ISE project directory (the project is already created)
`./simdata`	Data files used for simulation

`./src/data`	Other source files (user constraints file and ROM contents)
`./src/verilog-IPs`	Intellectual-Property (IP) Verilog modules
`./src/verilog-rtl`	RTL synthezisable Verilog modules
`./src/verilog-tb`	Simulation Verilog modules (testbench)
`./sw/bin`	The software application for accessing the FPGA system

As the simulator QuestaSim will now be launched from the XILINX ISE environment, the simulation process will run in the same directory as the implementation (`./impl`).

The project provided contains all Verilog source files (either for synthesis and for simulation) already located in convenient directories. This includes a correct implementation of the module developed in the laboratory project 1, that must be used if the synthesis or the verification of your own module fail.

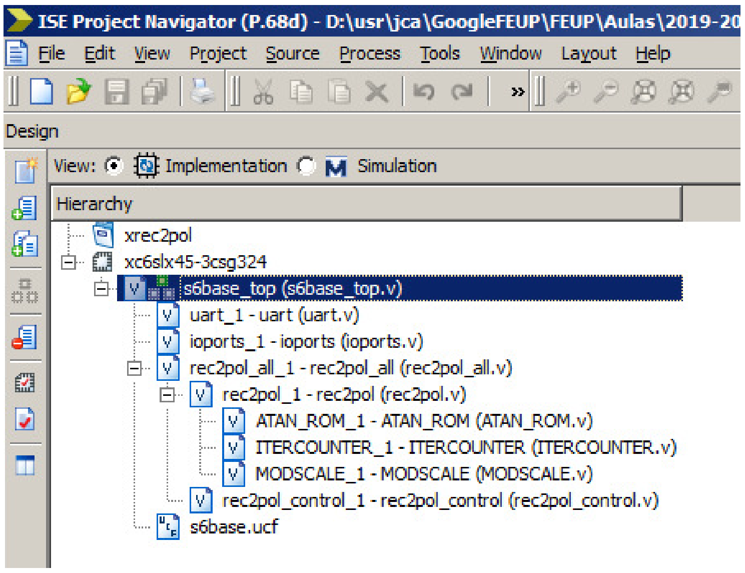
The six main stages to perform the implementation of a digital system to the FPGA platform are:

1. **Functional verification**: this will run a verification procedure by performing logic simulation, using a simple testbench provided in the reference project (`./src/verilog-tb/s6base_tb.v`)
2. **RTL synthesis**: in this stage you will synthesize and compare the synthesis results for different optimization goals.
3. **Post-synthesis verification**: this task will repeat the logic simulation, but using now the logic-level netlist generated by the RTL synthesis.
4. **Physical implementation (map and place&route)**: this stage will build the physical organization of the logic blocks that implement the digital system and create the interconnections among them.
5. **Post-route (or timing) simulation**: this last verification stage uses the timing models for the logic blocks and interconnections to simulate the system with the propagation delays estimated for the real circuit implemented on the FPGA. This is the most accurate simulation done in the ISE design flow.
6. **Testing the circuit**: after the physical implementation you will implement your system in the FPGA (or *configure* the FPGA) and experiment it using the Windows application or a Matlab script to send operands and read results. Although there is no physical fabrication involved in this process, the term "*testing in silicon*" is currently used to refer to the physical implementation and experimentation in the real FPGA device.

3 – Implementatiom

Execute the application "ISE Design Suite 14.6" and close any project that may be open at startup (by default this tool will always open the last project). Open your project by selecting the file `./impl/xrec2pol/xrec2pol.xise`. The module **s6base_top** (file `./src/verilog-rtl/s6base_top.v)` is the top level Verilog module containing the whole system represented in figure 1. In the "Hierarchy" pane (top-left) you can browse through the project hierarchy and all the files associated to the design.

In addition to the Verilog sources, the file s6base.ucf (ucf = "*user constraints file*") contains implementation constraints, as the assignment of the design inputs/outputs to the physical FPGA pins (constraints "`NET … LOC=`") and a timing constraint specifying the minimum required clock period ("`TIMESPEC…`"). This file is essential for the implementation, otherwise the inputs and outputs will be connected to random FPGA pins with unpredictable results.


ISE Project Navigator (P.68d) - D:\usr\jca\GoogleFEUP\FEUP\Aulas\2019-20
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy

```
xrec2pol
xc6slx45-3csg324
  s6base_top (s6base_top.v)
    uart_1 - uart (uart.v)
    ioports_1 - ioports (ioports.v)
    rec2pol_all_1 - rec2pol_all (rec2pol_all.v)
      rec2pol_1 - rec2pol (rec2pol.v)
        ATAN_ROM_1 - ATAN_ROM (ATAN_ROM.v)
        ITERCOUNTER_1 - ITERCOUNTER (ITERCOUNTER.v)
        MODSCALE_1 - MODSCALE (MODSCALE.v)
      rec2pol_control_1 - rec2pol_control (rec2pol_control.v)
    s6base.ucf
```

Figure 3 – The project hierarchy as seen in ISE Project Navigator.

3.1 - Behavioural simulation [20 min]

After building the RTL modules, the first design stage is to verify the whole circuit model at the functional level, as you already did in the previous laboratory project. For this simulation you will use a simplified testbench (**s6base_tb.v**, already included in the project) that simulates the complete design described above. In this design the simulation will use only the primary inputs and outputs of the circuit, sending the operands and retrieving the results through the serial port. The simulation will be done using the QuestaSim simulator, but now the program will be launched from the ISE project navigator.

In the *Design* window select the view "*Simulation*" and choose "*Behavioural*" for the simulation type. In the project hierarchy view, select the testbench module (**s6base_tb**) and execute the process "*Simulate Behavioural Model*". This will launch QuestaSim and run automatically the script "s6base_tb_fdo" that contains the commands necessary to compile the project and run the simulation. You can edit the file "s6base_tb.udo" (udo = "*user do file*") to customize the way the simulation is started. You can also customize the file `./impl/xrec2pol/s6base_tb_wave.fdo` with commands to format the waveform window (a basic waveform configuration is available in the script `./impl/xrec2pol/wave.do`). To rerun a simulation you don't need to close and re-invoke QuestaSim from the ISE Project Navigator, unless the hierarchy of the project has been modified (in that case ISE needs to regenerate a new "s6base_tb.fdo" file). If this is the case, just re-execute in the QuestaSim shell the command "do s6base_tb.fdo".

Verify the simulation results in the output text window and in the waveforms. Feel free to improve the testbench and write in <u>table 1 any relevant comments about this verification phase</u>.

3.2 - RTL synthesis [20 min]

In this stage you will synthesize a few versions of one module of your design and analyse the synthesis results. First, in the *design* window, choose the view "*Implementation*", select the module **ioports** and execute "*Set as top module*", accessed with the right mouse button. This will set this module as being the top level module for the implementation process, which is necessary to execute the RTL synthesis task only for this block.

Run the process "*Synthesize - XST*" and verify the warning messages generated during the synthesis (hint 1: the table "Clock Information:" lists the signals identified as clock signals or control signals of memory elements; hint 2: look for the word "*latch*" in the synthesis report). Identify and fix the issues in the source code related to these warnings and write your solution and conclusions in table 2.

The RTL synthesis is driven by an extensive set of parameters that the designer can tune to meet the required design goals. These parameters can be edited by selecting "*Process Properties*" available for each implementation process (use the right-mouse button over "*Synthesize - XST*"). Verify the current configuration for the two first parameters (optimization goal and optimization effort) and analyse the synthesis results in the "Design summary" window. The most relevant results are the design area measured by the number of slice look-up tables (LUTs) and slice registers (flip-flops) and the estimated maximum clock frequency reported at the end of the synthesis log.

By changing the first two synthesis parameters (optimization goal and optimization effort), try to synthesize the best implementation for this module, either in terms of area (minimize the number of LUTs and flip-flops) and in terms of speed (maximize the clock frequency). Write your results in table 3 and include any additional comments you may find relevant.

Add your **rec2pol** module to the project, set it as the toplevel and execute the synthesis process. If the synthesis fails, do not try to correct the errors at this stage. Just remove it from the project and add the original module. Even if the synthesis succeeds, look for the "*latch inference*" warning. If the synthesis report is clear of these warnings (or any other that may suggest to be critical, as the mismatch of number of bits in connections to the ports of modules), you can proceed with the implementation using your module. Write the results of this design stage in table 4.

3.3 - Synthesis of the whole design [10 min]

Select again the module **s6base_top** as the top level design. Set the synthesis optimization parameters to optimization goal "area" and "effort high", and synthesise now the whole design. Write in table 5 the synthesis results (slice LUTs and number of flip-flops) and compare them with the results obtained for the synthesis of the module **ioports**.

3.4 - Post-synthesis verification [15 min]

Select the simulation process "*Post-Translate*" and perform a logic simulation. This simulation is very similar to the described in section 3.1 although now you will simulate a fully structural Verilog model generated by the synthesis process. Use a text editor (Notepad++) to see the contents of the Verilog file created by the synthesis `./impl/xrec2pol/netgen/translate/s6base_translate.v`. Execute the simulation and write in table 6 your conclusions about this process.

3.5 - Physical implementation (Map and Place&Route) [15 min]

Execute now the process "*Implement design*". This will run the Translate, Map and Place&Route processes, creating the most accurate model of the circuit that represents the physical implementation on the FPGA. All the logic blocks and interconnections will be now annotated with propagation delays calculated from the models of the logic cells and the VT (voltage and temperature) operating conditions. If this succeeds, the summary window will present the final utilization of the FPGA resources for the whole design and a list of detailed reports.

Open the report “Post PAR static timing report” (window “Design summary”, in the section “Detailed reports”) and verify the maximum clock frequency estimated for this implementation. Annotate this clock frequency and the resource occupancy results in table 7 (only the number of slice LUTs and flip-flops).

3.6 - Post-route verification [15 min]
Run a “Post-route” simulation, using the same procedure described in section 3.4. Prior to the simulation, a Verilog netlist will be generated and annotated with the logic and net delays defined in a SDF file (*standard delay format*). Both files are created in the directory `./impl/xrec2pol/netgen/par/`.

Modify the testbench to increase the clock frequency to 30% above the maximum frequency reported in the static timing report. Running a timing simulation under this condition will probably fail. Note that this only fail if the set of stimuli applied to the system under simulation activates one of the logic paths that will not support that clock frequency. Comment the results observed in table 8.

3.6 – FPGA configuration and experimentation in the FPGA board [15 min]
Execute the process “Generate programming file” to generate a *bitstream* file (**s6base_top.bit**) in the root project directory. This file is the final product of the FPGA design and contains the programming data to configure the FPGA logic blocks and interconnections with your circuit.

Programming the FPGA in the Atlys board requires an application provided by the board vendor Digilent (Adept). Run this application, choose the bit file and execute “Program”.

To experiment your system you can either use a dedicated Windows application (`./sw/bin/S6-ATLYS.exe`) or a MatLab function (`./Matlab/hwrec2pol.m`) to send the operands, assert the **start** input and retrieve the results. Include any relevant comment in table 9.

Digital Systems Design - 2019/2020

Laboratory 2 - V1.0

4 – 5 November 2019

Names:____Date:____________

Table 1 – Comments on the functional verification.

Table 2 – Explain and fix the warning messages issued by the synthesis of **ioports**.

Table 3 – Synthesis results of module **ioports** (LUTs, FFs, max clock frequency) for different optimization goals (area and speed).

Table 4 – Results of the synthesis of your module **rec2pol**.

Table 5 – Results of the synthesis of module **s6base_top** (top level).

Table 6 – Results of the post-translate simulation.

Table 7 – Final implementation results (LUTs, FFs and maximum clock frequency).

Table 8 – Results of the timing simulation with the nominal clock frequency and with +30% over-clocking above the maximum clock reported by the STA tool (*static timing analysis*).

Table 9 – Additional comments
