M.EEC041 - Digital Systems Design

2022/2023

Laboratory 2 - V1.0
8 November 2022

In this laboratory the students will implement a custom digital system in a FPGA-based prototyping platform, going through 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 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 .

1 - Introduction

The design provided for this lab implements the block diagram shown in figure 1. The modules uart (`uart.v`) and iports (`ioports.v`) implement a set of 16 output ports and 8 inputs ports to/from a digital system (32 bit wide), accessed via a serial port. The module ioports interprets a small set of commands to write data to an output port and read data from an input port. This interface can be accessed using the application command rwport to read a port and to write a port (execute 'rwport' to print a short help text).

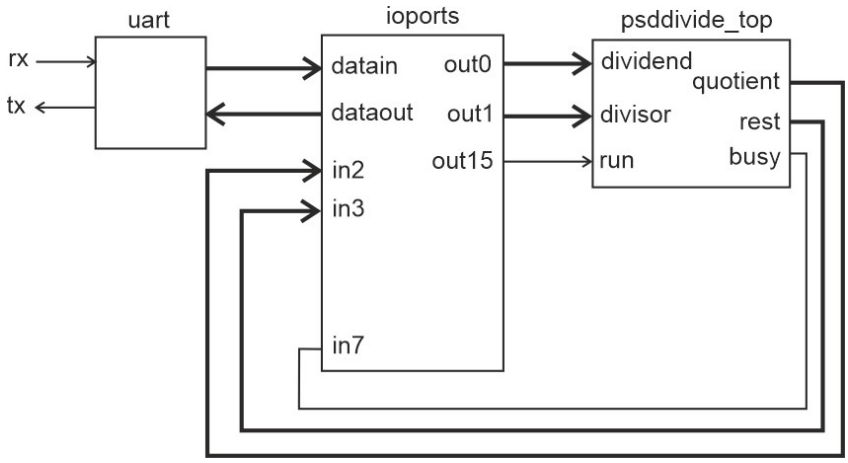

Figure 1 – Simplified block diagram of the reference project.

In the design to be used in this lab, the module **ioports** connects to the sequential unsigned divider developed in the first lab project. The two operands are the data written to ports 0 and 1. The input **run** is activated by writing a 1 to port 15. The final results (quotient and rest) are retrieved by reading from the input ports 2 and 3 (note that these results will be available after 34 clock cycles, a small fraction of the time needed to transmit a single byte through the serial port). The 5 general purpose push-buttons, the 8 slide switches and the 8 LEDs (currently assigned to the 8 LSBits of **dividend**) can be later used freely in your design.

2 - Installation of the reference project

Download the archive PSD-2223-LAB2.zip and extract all files to a new working directory. The installed directory tree is:

Directory	**Contents**
./doc	Documentation (this guide and the Atlys manual)
./impl/psddiv	ISE project directory (the project is already created)

./src/data	Additional source files (User Constraints File - UCF)
./src/verilog-rtl	RTL synthezisable Verilog modules
./src/verilog-tb	Simulation Verilog modules

This project is already complete with all Verilog source files located in convenient directories. In a first stage you will just run the whole verification and implementation process, to finally test the final system in *real silicon* (i.e. in the FPGA). Then you will have the opportunity to include an additional function into the system and redo the verification, implementation and testing process.

After creating the Verilog source files (already given in this design), the six main steps to complete the implementation are:

1. **Functional verification**: this will run a verification procedure by logic simulation, using a simple testbench provided in the reference project (`s6base_tb.v`)
2. **RTL synthesis**: in this stage you will synthesize and compare various versions of your design by setting different optimization goals.
3. **Post-synthesis verification**: this task will redo the logic simulation but using now the logic-level netlist generated by the RTL synthesis process for the whole circuit.
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 simulation step uses the timing models for the logic blocks and interconnections to simulate the system with the propagation delays estimated for the real circuit. 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 test it using the **rwport** command to write and read the input/output ports.

3 – Implementatiom

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

Besides 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.

3.1 – Behavioural simulation [15 min]

The first major step in the design flow is to verify the whole circuit model at the functional level, as you already did in the previous labs. For this simulation you will use a simplified testbench (already included in the project) that simulates the complete design described above, sending the operands and retrieving the results through the serial port. The simulation will be done using the Verilog simulator integrated in the ISE tools, 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 the simulator **ISim**. Although the graphics user interface is different from the QuestaSim, the main commands to launch a simulation and inspect the waveforms are similar.

Verify the simulation results in the output text window and the waveforms. Feel free to improve the testbench and include in table 1 any relevant comments about this verification phase, including improvements you may have done to your testbench.

3.2 - RTL synthesis [15 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 mark this module as being the top level module for implementation, which is necessary for executing the RTL synthesis task only for this block.

Execute 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 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 in "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 in the end and 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.

3.3 - Post-synthesis verification [15 min]

Select again the module **s6base_top** as the top level design. Choose adequate synthesis optimization parameters and synthesise now the whole design. Indicate in table 4 the synthesis results and compare them with the results obtained for the synthesis of the module **ioports**.

Select the simulation process "Post-Translate" and repeat the 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. Open the file `./impl/psddiv/netgen/translate/s6base_translate.v` and analyse its contents.

3.4 - Physical implementation (Map and Place&Route) [10 min]

Execute the process "Implement design". This will run the Translate, Map and Place&Route processes, creating a 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 the clock frequency and the resource occupancy results in table 5 (only the number of LUTs and flip-flops).

3.5 – Post-route verification [15 min]

Run a "Post-route" simulation, using the same procedure described in section 3.3. 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/psddiv/netgen/par/`. You will notice a longer simulation time because the post-route model is now significantly more complex but also the most timing accurate throughout the whole implementation process, based on the timing models provided by the FPGA vendor.

3.6 – FPGA configuration and test in the FPGA [15 min]

Execute the process "Generate programming file" to generate a *bitstream* file (**s6base_top.bit**) in the root project directory. This is the final product of the FPGA design and contains the programming data to configure the FPGA logic blocks and interconnections with your circuit.

Program the FPGA using the command **progfpga <bitstream_file>** and verify that your circuit is working as expected.

Names:_____ Date:_____

Table 1

Table 2

Table 3

Table 4

Table 5

Table 6