Electronics for Embedded System

6/11/2023 – 7/11/2023 Laboratory (FPGA programming)

Project #1

Design a simple AND gate in VHDL or Verilog (like the one shown in class). Using this design, practice in compiling, assigning pins, simulation, checking floorplan, timing analysis and timing constraints, and configuring the device. Follow the next steps:

1. Create a new project and the source files (design and testbench)
2. Check syntax and compile the project
3. Simulate the design
4. Assign pins
5. Check floorplan
6. Perform timing analysis
7. Program the device and test the design

In step 6, during timing analysis, try different combinations of pin assignments, timing constraints and floorplan. Examples of possible combinations are:

- No pin assignments, no timing constraints (the tool automatically finds a good floorplan with decent timing, check which pins have been assigned and where is the logic element for the AND gate, then evaluate the timing behavior)
- No pin assignments, specify timing constraints (start with high constraints, around 20ns, and then gradually lower them down to 5ns; check the timing results and the pins that are automatically assigned)
- Assign pins which are close to each other, no timing constraints (the tool places the logic element close to your chosen pins to minimize delays, even if you don't give any timing constraint; try moving the logic element in the floorplan and see how the timing changes)
- Assign pins which are far away, no timing constraints (the tool probably places the logic element close to one pin, or in the middle of the chip; evaluate the timing and try moving the logic element in the floorplan to see how timing changes)
- Assign pins which are far away, specify timing constraints (again start with high constraints and then gradually lower them; you can also try to set asymmetric delays from the two input pins, and this should result in a different placement of the logic element)

Write a table with all your experiments, similar to this one (you can add or remove rows and columns, as you feel necessary)

Input 1 pin	Input 2 pin	Output pin	Constraint t_{pd}	Worst case t_{pd}

Detailed instruction for the various steps are next.

Creating a project and VHDL files

- To create a new project, use the command File → New Project Wizard. **Change the project working directory** to some place where you have write permission, outside the program installation directory. **Don't use spaces** or strange characters in file and directory names (note that "Documents and Settings" or "Program Files" contain spaces), as these will cause wrong behaviors, especially with the NIOS II processor. Give a name to the project, and to the top level entity (usually the same). Choose the Cyclone V 5CSEMA5F31C6 for the DE1-SoC board. In the EDA Tool Settings choose ModelSim-Altera for Simulation, and leave all other options as default. If your project consists of multiple files, you can add them at project creation (if files are already available), or using the command Project → Add/Remove Files in Project after the project has been created.

- To create a VHDL file, use the command File → New and select VHDL File. The editor uses syntax highlighting to show VHDL reserved words. Save the file in the project directory (this should be the default place, but check before saving). Suggestion: *use a separate file for each VHDL entity, and give the same name to the file and the entity contained in it*. This makes debugging much easier.

Checking syntax and project compilation (synthesis)

- To check syntax, use the command Processing → Start → Start Analysis & Elaboration (or double click on the Analysis & Elaboration line in the Tasks window on the left). This is a quick check of your source code, and also generates pin nodes for the Pin Planner (see later). This command will analyze the entire project; if you only want to analyze the current file, then use Processing → Analyze Current File.
- To compile (synthesize) the design, use the command Processing → Start Compilation (or click on the corresponding button in the toolbar, or double click in the line in the Tasks window on the left). You will get a detailed report in the main window, as well as many messages and warnings in the console in the bottom. Check especially for errors and critical warnings.
- Check the compiler reports to determine the number of resources occupied by your design. This information is given in the summary, as well as in more detailed report files that can be viewed by opening the Analysis & Synthesis, Fitter and Assembler folders in the left pane of the compilation report. In particular under Fitter you can check the Pin-Out File or the Input Pins and Output Pins in the Resource Section.
- To change the top level entity of a design, use the command Assignments → Settings, select the General tab, and enter the name of the entity you want to use. As an alternative, in the Project Navigator, select the Files tab, right click on the name of the file that contains the entity and choose Set as Top-Level Entity.

Simulation

- To create the testbench, create a VHDL file with an entity called `testbench` with no input or output ports. Inside the architecture declare your top level entity as a `component` (don't change the top level entity of your design under the Global settings, as the testbench cannot be compiled). Define signals for all input and output ports of the component (and possibly initialize inputs in the signal declaration). Instantiate the component using the label `dut` and map all ports to the defined signals. Create timing waveforms for all inputs using processes and the `wait for` or `after` statements. Save the file as testbench.vhd. Remember to recompile the design and check for errors in the testbench.
- To setup the simulation, use Assignments → Settings and select Simulation under EDA Tool Settings. The tool should be ModelSim-Altera, and the Format for output netlist should be set to VHDL. Under the NativeLink Settings select Compile test bench and click on Test Benches…. In the window that appears click New and in the new window that appears write `testbench` as the Test bench name and as the Top Level module in test bench. Select Use test bench to perform VHDL timing simulation and specify `dut` as the Design instance name in test bench. Specify the length of the simulation using End simulation at. Finally, you have to add the file testbench.vhd to the list of Test bench files (browse for it and click Add). Be sure to click OK on all open windows.
- To perform a gate level simulation you need to generate a functional netlist. This option is not set by default, so you need to change it in the preferences. Use Assignments → Settings and select Simulation under EDA Tool Settings. Click on More EDA Netlist Writer Settings… and turn On the setting for Generate functional simulation netlist.
- To perform an RTL level simulation (i.e., with no timing delays), use Tools → Run Simulation Tool → RTL Simulation. ModelSim should start automatically and runs the simulation for the specified amount of time. In the Wave window you can zoom in and out, place cursors, change the properties of each waveform. The simulation can be restarted if needed. If the VHDL design is changed, it has to be recompiled within Quartus.
- To perform a gate level simulation with extracted delays, use Tools → Run Simulation Tool → Gate Level Simulation, and choose a Timing model (the slow model is the preferred one in most cases) if asked. ModelSim starts automatically like in the previous case, but waveforms will now show delays. Commands and options are the same as the RTL level simulation mode. Delays for the logic implemented on the FPGA are typically not used (the timing models for recent devices are not available), but are included for some of the available IP resources (like for instance the memories).

Assigning and managing pins

- To assign pins, use the command Assignments → Pin Planner. You should see a list of input and output nodes in the bottom. If you don't get the list, then the design has never been analyzed. Run the analysis or a full compilation first. In the Pin Planner, double click the Location entry of the pin you want to assign, and write the full name of the pin (i.e. PIN_T7), or just the location of the pin (i.e. T7), or select it from the drop down list. To delete a pin assignment, right click on the name of the pin (i.e T7, not the name of the port of the VHDL entity which is on the same row) and choose Edit → Delete (or select the pin location and press the Delete key on the keyboard). Check carefully the function of each pin before making an assignment. Note that every time you make an assignment, it is automatically saved (there is no save command). To make the new assignments effective, you have to recompile your design.
- To set the default for unused pins, select Assignments → Device, click on Device and Pin Options…, select the Unused Pins category, and choose As input tri-stated.

Floorplan

- To see the chip floorplan, use the command Tools → Chip Planner. You can zoom in and out on the chip. To select what to see in the chip planner window, use the Layer Settings pane (it is usually visible on the right; if it is not, use the command View → Layer Settings). If you double click on an element in the floorplan, a more detailed view of that element (i.e. a Logic Element, or an I/O Block) will be shown. You can also drag and drop elements to change their placing, preferably using the Editing Mode set to Assignment rather than ECO; you have to recompile the design after any change.
- You can also look at the Netlist at the RTL level or after Technology Mapping under Tools → Netlist Viewers. If you select a node and then right click on it, you can locate it on the Chip Planner, to make it easier to understand where the various blocks have been implemented.

Timing analysis

- To check timing analysis results, look at the compilation report in the Timing Analyzer folder. You can check the Slow Model as well as the Fast Model. For propagation delays, start the Timing Analyzer under the Tools menu, and look into the Report Datasheet under Report. For sequential circuits, look at the Fmax Summary. To change timing analysis settings, open the sdc file or use the Timing Analyzer.
- To specify constraints, use the command Tools → Timing Analyzer or create an SDC file and insert specification using the Insert Constraint command after right clicking on the editor area. You can specify the characteristics of your clocks (give it a name and then set the port, the period and the duty cycle using the rising and falling instants, that have the same unit of measure of the period), the setup and hold times t_{su} and t_h for registers, the clock to output delay t_{co}, and the required propagation delay t_{pd} between inputs and outputs.

Device programming

- To program the device, use the command Tools → Programmer. If the programmer shows No Hardware, click on Hardware Setup, and select the DE-SoC. Use Auto Detect to find the JTAG chain, and change the programmer file for the 5CSEMA5 device. To start configuration, ensure that the box Program/Configure is ticked, and click the Start button. When the progress indicator reaches 100%, the configuration is complete and you can try your design on the real hardware.

Project #2 (optional, if you have time)

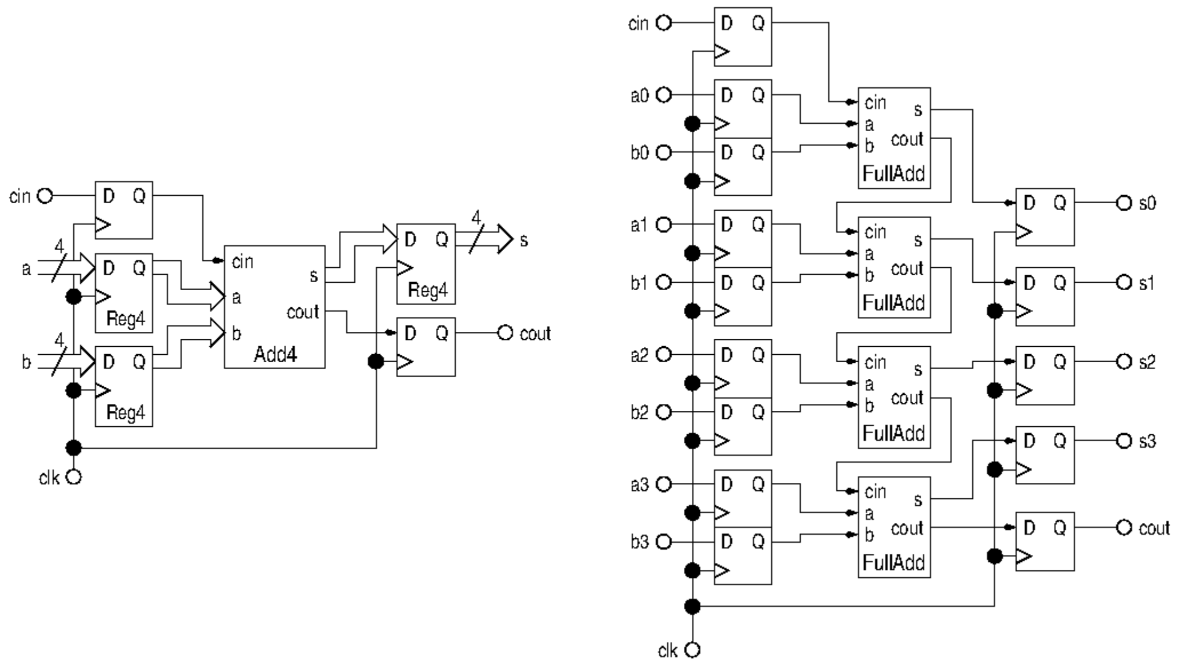
Add a 10 bit counter $c_9, \dots, c_0$ to the previous project to provide inputs to the AND gate; in particular:

- Connect the input clock signal to the 50MHz clock on the board (using the Pin Planner)
- Connect c_8 and c_9 to the inputs of the AND gate (in your VHDL source)
- Add two additional outputs to your design, drive them using c_8 and c_9 (in your VHDL code), and connect them to two of the pins of header connector on the board (using the Pin Planner)
- Connect the output of the AND gate to one of the pins of the header connector on the board (using the Pin Planner)

With the timing analyzer, verify that the maximum clock frequency is more than 50MHz. If you want, simulate the design to check the correctness. Then program the board and show the signals from the header connectors on the oscilloscope. You should see the behavior of the AND gate, and you may be able to measure the delays, as well.

Project #3 (optional, if you have time)

Design a 4-bit ripple carry adder with carry-in and carry-out. Use 8 switches for the two 4-bit inputs, and a push-button for the carry-in signal (remember that push-buttons are normally high). Use LEDs for the output sum and the carry-out signal. If you have time, design a 7-segment display driver to show all input and output values in hexadecimal format. To better measure delays and avoid the high load of pads, put the adder between registers (one 4-bit register to hold one input, another 4-bit register to hold the other input, and a third 4-bit register to hold the output, plus 1-bit registers for carry-in and carry-out) and check the maximum clock frequency of the system. See the next picture for details on the implementation with registers (on the left is the top level view, on the right a more detailed view). Try increasing the number of bits to 8 and check timing analysis to verify performance (do not implement the 8 bit on the board).



Suggestions: design, verify and test on the board a 1-bit full adder by giving the equation for the sum and the carry out. Then build a 4-bit full adder by connecting 4 1-bit adders with a structural design. Check the floorplan and verify that LEs are used in normal mode. Alternative: design a 4-bit full adder using the ieee.numeric_std library and the + operator. Check the floorplan and verify that LEs are used in arithmetic mode. Compare the performance and area of the various designs. You can fill in a table with the obtained results, like the one shown below. Another alternative: compare the performance of an 8-bit ripple carry adder with that of an 8-bit carry look ahead adder.

Number of bits	**LE mode (normal/arith)**	**Area (number of LEs)**	**Worst case t_{pd} (signals and time)**

Project #4 (optional, if you have time)

Design an 8-bit Serial-In Parallel-Out (SIPO) shift register. Implement the shift register on the board using a push-button for the clock, and a switch for the serial input. Use LEDs to show the parallel output. Add an asynchronous active low reset button mapped to a push-button. With the push-button clock you can see bits shifting along the register. Don't use the 50MHz clock, because it is too fast! Look at the floorplan if a register chain is used to implement the design.

Project #5 (optional, if you have time)

Design a 16-bit up counter with the following input signals: clock, reset (asynchronous, active low), enable (synchronous, active high), out and terminal count. Implement the counter on the board. Use a push-button for the reset and a switch for the enable signal. You may want to initially assign the clock to a push-button to verify that everything is working, and then to the 50MHz clock. You can show the count on the 7-segment displays in hex format.

Project #6 (optional, if you have time)

Design a small system that uses one of the embedded memories of the device, configured as a single port memory. Provide the address, datain, clock and write enable signals, and get the dataout signal. Perform both the RTL and the gate level simulations, and see the differences. Try configuring the memory with registered and unregistered outputs, and see the difference in timing in the simulation. To implement the system on the board, use switches and buttons for the inputs, and LEDs for the outputs. If you don't have enough switches and buttons, set higher bits to a fixed value in the VHDL code, and only change the lower values. You are probably going to implement a small memory.
Additional experiments: the internal SRAMs of the FPGA are 9Kbits or 10Kbits. When creating the IP, ask for a bigger memory (e.g. 16Kbytes) and see how it is implemented in the floorplan (see how many SRAMs are used). Try also a dual port memory and simulate reading and writing concurrently on the two independent ports. Implementing a FIFO is also very interesting. You can also try selecting an implementation using the FPGA fabric (i.e. the MLABs on the Cyclone V) and see the difference in area. Try also to configure a memory that is too large for the device, and see the error message that you get from the Fitter.

Project #7 (optional, if you have time)

Design a small system that uses one of the embedded multipliers of the device, configured to have 8 bit inputs and 16 bit output. You don't have enough switches for all inputs, so you may set some of the bits to a known value and only change the others. Show the result on the 7-segment displays if you have them.
Additional experiments: try changing the parameters of the multiplier (size of the inputs and outputs). Try also instantiating several multipliers and perform concurrent multiplications; look at the floorplan. You can also try implementing a multiplier using logic elements and see the difference in area.

Project #8 (optional, if you have time)

Design a small system that uses one of the embedded PLLs of the device. You can generate several output clocks with the following characteristics:

- Higher frequency than the input clock
- Lower frequency than the input clock
- Duty cycle higher or lower than 50%
- Same frequency of the input clock but with a phase shift
- Various combinations of the above

In the implementation on the board you may want to send the output clocks to pins of the header connectors in order to see them using the oscilloscope. Be careful that high frequency signals will not appear as square waves, but more like sinusoids, and there can be a lot of noise. The reason is related to non-ideal behaviors of the board components (parasitic resistances and capacitances, slew rate of the output pins), and to the limited bandwidth of the oscilloscope.