

Lab Assignment 3

ECE/CS 3700

Spring 2009

Assigned the week of 2/16 onwards, Due date: your respective lab sessions during the week 3/2-3/6.

The objective of this lab assignment is to learn the techniques of Verilog Design, Simulation and Logic Synthesis for the design of unsigned adders. So, in this lab you will experiment with the ripple-carry and lookahead carry adder designs.

In the previous lab, you did write some simple verilog code to model your circuit and also created test-benches to simulate your code. In this lab, we will take the concepts of Verilog design somewhat further and you will learn the concepts of hierarchical design using component instantiation. Once you complete your design and verify its correctness by simulation, you will then *synthesize* your design to create an *implementation*. This implementation can be downloaded onto the FPGA available on the XSA/XST board.

The Synthesis tool (in our case ISE/Webpack) automatically performs logic optimization on your verilog code and generates an optimized Boolean model - this process is what we call logic optimization. This Boolean model is then mapped onto the technology at hand - this is called technology mapping (LUTs in our case). Finally, the mapped model is then placed and routed - the final stage of logic synthesis, called placement & routing (PAR). You will go through this entire process to generate an implementation of your circuit.

Once you have the implementation (in the form of a .bit file), you will download it to the FPGA using the XESS tools. Once the design is on the FPGA, you will then apply external stimulus (using switches) and observe the output (on LEDs).

Really, thats all that you have to do. So, lets get to it. The assignment is in three parts.

I. THE FIRST ASSIGNMENT - VERILOG DESIGN, SIMULATION & SYNTHESIS

(30 points) To design a 4-bit ripple carry adder, do the following:

- Create a module for a 1-bit full adder. You may use either assign statements or gate instantiations, or both. Don't use always blocks for this lab.
- Instantiate 4 1-bit full adders to design a 4-bit ripple carry adder. Simulate your 4-bit ripple carry adder and verify that it operates correctly.
- Now, synthesize the adder and generate the implementation. This process generates: (i) an RTL schematic, (ii) Synthesis Report, (iii) Map report and (iv) place & route report. Analyze these carefully, and see what can you decipher from these reports.
- These reports generate a lot of information - much of which is not very useful for our study. What we are interested in is: (i) Area occupied by the design, in terms of the number of slices and look-up tables; (ii) Delay of the design - the longest/slowest path in our circuit. (**Note:** I synthesized a 4-bit ripple carry adder, synthesized and viewed the

synthesis report. The tool correctly reported that the longest path was from c_{in} to c_o . Wouldn't you want to verify that? The synthesis report can tell you a lot).

- If you view the generated RTL schematic, you will find that the generated design closely matches your structural verilog code (that's because by using structural description, you are asking the compiler to design it just as you described it).
- Do the simulation and synthesis for the 4-bit ripple carry adder design and feel proud that you have passed the first hurdle.

Once you are done with the above, and have gained enough confidence to design ripple-carry adders, you will perform the following experiment.

II. THE SECOND ASSIGNMENT - LOOK-AHEAD CARRY ADDER

(50 points) In this experiment, you are asked to implement a 4-bit look-ahead carry adder, just the way we have designed it in class. Feel free to refer to the textbook, see how you will implement the propagate and generate signals and perform the look-ahead computation. Use gate instantiations or assign statements for the design, avoid always statements. Also, think of the design structurally.

- In your report, clearly show how you derived the Boolean equation for the (lookahead) carry.
- Simulate the design, synthesize the design and generate an implementation. The implementation of the design is created as a *.bit file in your working project directory.
- Once you have an implementation, (in terms of a *.bit file) you will then map the inputs and outputs to specific FPGA pins (using the *.ucf file). The final implementation with pin-info can be downloaded on the FPGA. Connect input switches and output LEDs to the FPGA IO pins and excite your circuit.
- Demonstrate the correct functioning of your design to the TAs.
- For downloading your circuit to the FPGA, you will have to refer to the XESS-TOOLS manuals, available on the class webpage.
- Again, generate the synthesis reports and see for yourself if the delay of the longest path improves or not. Also, observe what happens to the area of the circuit, as compared to that of the ripple carry design.

III. HIERARCHICAL ADDER DESIGN

(20 points) Now that you already have a 4-bit look-ahead carry adder, you can put two of these blocks together (in a ripple configuration) and generate an 8-bit adder. Simulate and verify the correct functioning of the circuit. Aren't you now curious to synthesize the design and view the report? How many LUTs does such a design occupy?

IV. SUBMISSION REQUIREMENTS AND TIMELINES

- During the week of 2/16-2/20, get your lab 2 checked-off and start working on the Ripple Carry adder design. This is your getting started week.

- During the week of 3/23-2/27, you will demonstrate a functioning Verilog design (simulation) for the 4-bit ripple-carry adder design.
- During the week of 3/2-3/6, you will demonstrate the correct functioning of the 4-bit look-ahead carry design. The TAs will look at your code to see how you generated the look-ahead carry. And then they will check off the mapped and downloaded design on the FPGA (with input switches and output LEDs).
- Also, have the simulation for the 8-bit adder checked-off during the week 3/2-3/6. Note: to simulate an 8-bit adder exhaustively, you will have to simulate 2^{16} values. That will be too slow, and you won't be able to view the waveforms. Sure, you can do that to convince yourself that your design works, but please spare the TAs the agony of viewing 65,536 additions. Show 3 interesting cases (e.g. add $255 + 255 + 0$, $255 + 0 + 1$, $127 + 127 + 0/1$).
- **Report:** Write a brief (no more than 2 to 3-pages) summary of the (i) design objectives, (ii) the design process, (iii) observations regarding area/delay trade-offs of designs (tabulate your results, if possible) and (iii) the conclusions that you have derived through these experiments. Attach your Verilog Code, testbenches, simulation results, and a short summary from your synthesis reports. Since we are only interested in area-delay properties of the synthesized netlists, just borrow that information from the reports (feel free to use cut-and-paste) and omit the result of the junk. This should be submitted by 3/6.