# Lab Assignment 2

ECE/CS 3700
Fall 2024

Assigned: Sep 18 Due date: Oct 6

Note: The final check-offs for Lab 2 should be completed during the week of Sep 25. During the same week, we will also start working on Lab 3, which will be a short 1-week lab.

This lab introduces the design of adder circuits. These are covered in Chapter 3 in the textbook, in Sections 3.2 and 3.4. While we catch-up with chapter 3, the TA's will cover the basic concepts in the lab to get you started during the first week.

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 wrote some simple Verilog code to model your circuit and also created a test-bench 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 – just as you did in the previous lab. This implementation will be downloaded onto the FPGA available on the DE10-Lite board. 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 of a 4-bit ripple-carry adder.

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. Then write and simulate a testbench for your 4-bit ripple carry adder to verify that it operates correctly.
- View: (i) an RTL schematic, (ii) Synthesis Report, (iii) Map report and (iv) place & route report.

- 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 LABs and LEs (which includes look-up tables).
- Use the Timing Analyzer tool to find the critical path delay in the design (the longest/slowest path in our circuit). A tutorial for using the Timing Analyzer can be found on Canvas.

- Note: If you wrote structural Verilog code, you will find that the generated RTL schematic closely matches your code (thats because by using structural description, you are asking the compiler to design it just as you described it).

- Note: It is not required to download this design on the board. Verilog simulation and logic synthesis is sufficient to learn about the characteristics of the design. You will download and demonstrate the next design – the lookahead adder – on the FPGA to the TAs.

## II. The Second Assignment - Look-ahead Carry Adder

In this experiment, you are asked to implement a 4-bit look-ahead carry adder, just the way we will design it in the class. [At the time of creating this assignment, we are a couple of lectures away from 'adder design']. Feel free to refer to the textbook, see how you will implement the propagate and generate signals and perform the e look-ahead computation. Use gate instantiations or assign statements for the design, do not use always statements. Also, think of the design structurally.

- In your report, clearly show how you derived the Boolean equations for the (lookahead) carry.
- Simulate the design, synthesize the design and generate an implementation. The implementation of the design is created as a *.sof file in your working project directory.
- Before you have an implementation, (in terms of a *.sof file) you will need to map the inputs and outputs to specific FPGA pins (using the *.qsf 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.
- 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? *Note: It*

*is not required to download the implementation bit-file on the FPGA for this 8-bit adder. Simulation and synthesis will suffice.*

## IV. SUBMISSION REQUIREMENTS AND TIMELINES

- During the first week of lab 2, get your lab 1 checked-off (if pending) and start working on the Ripple Carry adder design. Since we haven't already started studying arithmetic circuits, the TA's will tell you about 1-bit full adders, and ripple-carry circuits. By the end of the first week, you should have written and simulated the testbench for your ripple-carry adder.
- During the second week, 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).
- During the second week, you should also get checked off and simulate the 8-bit adder.

- Report: Write a brief (no more than 3 to 4-pages + schematics and code) 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 and omit the rest.