Lab 3 – EC413 Computer Organization

The purpose of this lab is to

- Review the Xilinx environment, including design entry in Verilog, simulation, and waveform generation and analysis
- Review ***Structural*** Verilog
- Review hierarchical design and other proper HDL style
- Learn to create and use more sophisticated testbenches, including testbenches that facilitate automatic verification
- Learn to perform simple algorithmic performance calculation through gate-count timing
- Practice with construction, test, and analysis of two standard adders.

A note on use of Structural Verilog:
Q: Last semester you learned Behavioral Verilog -- Why not use that here?
A1: The hardware labs in 311 and 413 have multiple missions, including teaching HDLs, good design practice, digital logic, digital systems, and computer organization. In 311, you (may have) learned Behavioral Verilog early so that you could quickly advance in the complexity of digital systems you could build. In 413, our initial emphasis is deep understanding of the underlying behaviors, including logic complexity and pipelined circuits. We've found that students learn these best when they begin with logic where nothing is hidden.
*A2: Another reason to review Structural Verilog is that most HDL systems are a combination of Structural and Behavioral elements. Some of the tradeoffs are obvious – Behavioral is often easier to build and test while Structural is closer to the HW and more likely to give you exactly what you want. But there are also more subtle issues – for a complex component **C** it is often EASIER to use Structural than it is to figure out exactly what all of C's behaviors should be and then reverse engineer those behaviors back into Behavioral Verilog. But don't worry, we'll be back to Behavioral Verilog soon!*

Specification: design, build, simulate, and analyze at least two 64-bit adders:

1. Ripple Carry Adder (RCA)
2. A 2-stage Carry Select Adder
3. (Extra Credit) A more-than-2-stage Carry Select Adder of your own design – see details below

- For all, you may hardwire the overall **carry_in** to 0, but you must generate an overall **carry_out**.
- For the Carry Selects: the overall **carry_out** can use the same MUX as used by the high-order adder block.

Requirements:

- The design ***must*** be hierarchical (i.e., using blocks which you can reuse)
- One of the modules must be a 4-bit ripple carry adder
- You must use Structural Verilog (gate-level specification) for the RCAs. You may use Behavioral Verilog for the MUXes
- You should use Behavioral Verilog for the verification logic. This will be presented in discussion section.
- Base the 64-bit adders on 1-bit full adders made up of ANDs, ORs, and NOTs (and intermediate modules that you construct). In particular, if you use XORs or half adders, then you must create a module first and build them up from ANDs, ORs, and NOTs.
- For timing, set ANDs and Ors to count as one gate delay and NOTs to zero gate delays. The behavioral MUXes (that select between the 0/1 RCA outputs) should also have zero gate delays. (We discussed motivation for these valued in class.)
- Depending on how you design your logic, the following could be useful: in order to avoid changing all the numbers when modifying the delay time (switching between 1 and 0), use the define directive.
- If you do the extra credit, please note that the other MUXes, i.e., the AND and OR in series between RCA stages (beyond the first pair), DO count (have a 1 gate delay each).
- Create two versions of all of the adders:

- → using standard logic gates
- → using the "timed" gates to be provided

You can simplify your design by reusing the logic with ‘define (presented in discussion section).

- Your timing diagram must include representative examples. This is relatively easy in this lab, but becomes more important later in the semester.
    a. Force an overall carry out
    b. Random large A&B input (w/ and w/o carry)
    c. Random small A&B input (w/ and w/o carry)
    d. Random combinations
- Since 64 bits is far too many to analyze wire by wire, your test bench must include verification logic. We'll go over this in the discussion section, but (briefly) this entails creating behavioral level HDL code and comparing its output with your logic. You will also need to test the verification logic explicitly.

Questions to be answered:
What are the times of your adders in gate delays? Are they what you expect? Why or why not?

Extra Credit (up to 10 points). Create another Carry Select but this one with an optimal number of optimally sized adder stages.

What to turn in:
- .v files. Please describe the modules that you designed in building the adders. You can just enumerate the names of the modules and define (in a sentence or two) what its purpose is
- Description of timing results (a few sentences are probably enough)
- Sample waveforms w/ and w/o delay
- Answers to questions

Grading:
An important component of your grade will be style. This means you should use proper comments, make the code neat and readable, and appropriately label modules and signals. For example, a name given to a pin or signal should explain its function: e.g. clocks should be labeled as clk and not C or CL.

Grading guidelines:
- o Pre Lab [10 points]
- o All code involved in the design [20 points]
- o Timing diagram and test benches with verification. Include representative examples [25 points]
- o Proper style [20 points]
- o Demonstrate the adders [25 points]
- o Extra credit [10 points]

To be covered during discussion and the PreLab (separate document) –
- Review Structural Verilog
- Testbenches for combinational logic
- Built-in verification including the behavioral logic you might want to use and a method of testing the verification logic
- Timed gates
- "Define" for making timing visible