EC413 Computer Organization
Lab 4 – Adders and Buses

Overview: The purpose of Lab 4 is:

- Review of the Vivado environment, including design entry in Verilog and waveform generation and analysis
- Review of Structural Verilog
- Review of hierarchical design and other proper HDL style
- Learn to create and use more sophisticated test benches, including test benches which facilitate automatic verification
- Learn to perform simple performance through counting gate timing
- Practice with construction, test, and analysis of two standard adders

A note on use of Structural Verilog:

A 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 complex components it is often EASIER to use Structural than it is to figure out exactly what all of its behaviors should be.
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

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.
- 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, do not use XORs or half adders.
- For timing, assume that ANDs and ORs count as one gate delay, but that NOTs count as zero; we assume that we get NOTs for free in our underlying process. There is one exception: assume that the MUXes that select between the 0/1 RCA outputs have zero gate delays. Again, this is because the underlying process is likely to give you this for free.

- 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.

Deliverables:

- All of your .v files.
- A lab report, including:
  - A description of 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.
  - A description of timing results (a few sentences are probably enough).
  - Sample waveforms w/ and w/o delay, along with a brief explanation.
  - Answers to the following questions:

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

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:

- Pre Lab [10 points]
- All code involved in the design [20 points]
- Timing diagram and test benches with verification. Include representative examples [25 points]

- Proper style [20 points]
- Demonstrate the adders [25points]

To be covered during discussion:

- 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.