CSE30321 Computer Architecture I

Lab 1: Reviews – Verilog, Xlilinx, Etc. (50 points)
Assigned: Week of September 3rd. Due: Week of September 10th

1 Big Picture Introduction

"Why do I have spend additional time outside of class working in the lab?" or alternatively **"What am I possibly going to get out of the lab that I can't learn in class?"**

Over the course of the semester, you'll be working to complete labs outside of class. Before getting started with Lab 01, I'll try to briefly explain what the lab sessions are designed to do. In short, computer architecture is a course about designing microprocessors. In lab, you'll have the opportunity to apply ideas learned in class to expand the functionality or improve the performance of some real processor design. Moreover, you'll actually use some of the same software and hardware design tools that someone working for a company that designs and builds microprocessors might leverage.

As an example, it certainly is important to weigh both the pros and cons of adding a new instruction to some computer architecture. While the new instruction might make the processor more functional, it might also make it harder to implement too. While the first step in this decision making process might be done "on paper" – and involve thinking about how the new instruction might affect the complexity of your control logic, whether or not you'll need to build a bigger multiplexer, etc. – more detailed simulations are also necessary to make sure the improvement peforms equally well once it is actually fabricated "on chip".

In real life your idea that exists "on paper" would have a life cycle that might look something like this:

1. The new instruction would probably next be simulated at the behavioral level (where you figure out what new control signals might be needed, you check the changes that are made to finite state machine associated with your control logic, etc.)
2. The behavioral model might be synthesized to create a logic gate representation of the hardware that's actually needed to implement the design.
3. You would interact with compiler writers to make sure that the new instruction actually helps improve the performance of programming constructs that are commonly used.
4. The logic gate representation might be processed further to get to a representation of the design that could be sent to a fabrication facility and physically built.

We won't talk much about items 3 and 4 above in this class. (You'd discuss work associated with item 3 in Compilers in the Fall of 2009 and you'd discuss work with item 4 in VLSI Design in the Fall of 2009.) However, in the labs this semester, you *will* get hands on experience with items 1 and 2 – which will help prepare you for future courses that you might take.

There's a "practical" nature to the labs too...

At the end of the semester, you'll be asked to apply the knowledge you've learned in class to make some performance improvements to a simple processor design. You might look at ways to improve performance, lower the implementation costs, etc. The only practical way to manage a design of even this complexity is to use the tools that you'll learn about over the course of the semester in the labs. With this end goal in mind, each lab has been designed to introduce you to different aspects of processor design – so you'll have prior knowledge of all of the design techniques that you'll need to leverage in you're final project.

2 Objectives for this Lab

The goals of this lab are essentially two fold:

1. The lab is designed to introduce and review the tools that you'll use in lab over the course of the entire fall semester. More specifically, we'll review the Verilog concepts learned in CSE 20221 that you'll need to apply in lab assignments in CSE 30321. You'll also become familiar with the ModelSim simulation tool.

2. You'll also learn how to use Verilog to design a finite state machine. (Recall from Lecture 03 that finite state machines can be quite useful in designing and implementing the control signals necessary for a given instruction.)

3 Lab 01 Procedures

1. Recall the assignment from CSE20221 in which the FPGA interfaced with a PS/2 keyboard (link on course web page). Begin by creating a directory and downloading the completed Verilog code and schematic for the CSE20221 assignment (link to zip files on course web page). Open Xilinx ISE and create a new project for this lab ("File → New Project"). Add the sources you just downloaded to the project ("Project → Add Copy of Source...").

2. Design a finite state machine in Verilog to recognize a 4 character stream input from the keyboard. The input stream to recognize is "ABCD". The characters in the input stream can be repeated and should still be recognized, i.e. something like "AAABBCCCDDD" is valid input. Create a Verilog module for the FSM in Xilinx ISE as part of the keyboard project. The input to the module will be the output from "shiftreg11" (i.e. kbd_data[10:0]). The state machine will determine if the input matches the desired string. As input characters are consumed, they are displayed to the seven-segment display by outputting them to "digit_mux2". An LED should be lit when "ABCD" is recognized, and a different LED while "ABCD" has not been recognized.

3. Simulate and debug your design using ModelSim. Create a testbench waveform that will go through 3 inputs that do not match, followed by "ABCD". ModelSim should be selected as the default simulator. If the ISE simulator opens instead, double-click the device (xc3s100e-5TQ144) in the "Sources" tab, and in the window that opens select "ModelSim XE" as the default simulator.

4. When ModelSim opens, you will be presented with a command prompt, a waveform display, and a list of signals from the ISE testbench. The simulation should already have run for the default 1000ns that the ISE testbench specifies. Notethat you may have to right click the waveform and click "Zoom full" to see the results (sometimes it starts zoomed in or out). To run for a longer time, just type "run Xns" at the command prompt, where X is the desired time in nanoseconds. Print out parts of the waveform that demonstrate correct functionality of your design.

4 What to Turn In

1. Demonstrate your design to one of the TAs by the end of your next week's lab time. Sign up for a demonstration time on the course wiki (link on course web page – under Lab Resources). Come to the lab and download your design to an FPGA board and demonstrate it to one of the TAs. FPGA boards are available in 208 Cushing but are NOT to be taken from the room. Any testing you wish to perform before your demonstration time must be done in the lab.

2. Prepare a typed report which should include a description of what is accomplished in this lab, problems encountered, and how they were resolved. Include a printout of the waveform used to verify your design and a description of how the waveform demonstrates correct functionality. Your report will be graded for its technical content as well as its style and grammar. (Note – I would expect a report of about 1 to 1.5 pages that looks something like the first page of this lab handout.)

5 Useful References

- Frank Vahid's Digital Design (Ch. 9)
- Lab website: http://www.cse.nd.edu/courses/cse30321/www/labs.html
- CSE20221 Handouts on Verilog (links on course web page – under Lab Resources)
- ModelSim and Verilog tutorial videos (link on course web page – under Lab Resources)

6 Useful Tips

A useful tip from last year...

”There have been a number of groups having ”strange” problems when testing their lab solutions on the FPGAs. Many groups are counting clock cycles of the PS2 clock to keep track of when a valid letter comes from the shift register. It is expected that the scan code for a key will be the only thing to come through, but in reality the keyboard does the following: it sends the scan code, followed by a ”key-up code” of F0 (when the key is released) and then again by the key’s scan code. Therefore, if you are just waiting for one scan code you will probably not get the results you expect. The Basys reference manual describes this and the data format of the 11-bit scan code with start, stop, and parity bits. If you are having trouble, please read the PS/2 and keyboard sections of the manual carefully to make sure you understand exactly what data your FSM is expected to work with.

Note that this may not apply to you depending on your implementation (I know some groups had success with other methods and didn’t take the key-up code into account).”