CDA 4150 Project #0[1]

Due: Do not turn in!

School of EECS, University of Central Florida, Orlando, FL 32816.

1. Objective

The purpose of this lab is to familiarize yourself with the Verilog simulator we will use in CDA 4150. You will be required to write Verilog modules for a byte adder, an 8-bit adder/subtractor, an 8-bit counter, and a digital clock. Please read this entire handout before beginning the lab.

2. Environment Setup

All CDA 4150 projects, including this one, should be done in groups of exactly two students. Please link up with a partner and submit a single solution (though not for this project).

You should use monroe.cs.ucf.edu to run your projects. The steps to connect is:

1. download putty here:
   http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
2. fill monroe.cs.ucf.edu in " Host Name"
3. select " SSH"
4. click open
5. Your login id is your NID and your default password for those who didn't have an account already is Pyymmdd (their birth year, month and day). Please change your password using the passwd command.

To compile and execute all the projects (project 0 through the Final Project) you will need to add heinrich/myusr/local/bin to your path. To do so:

- open ~/.cshrc with your favorite editor (vi or emacs)
- at the top of the file add the line:
  setenv VCS_HOME /mcad/synopsys/vcs6.0
- then add the line:
  setenv SNPSLMD_LICENSE_FILE /mcad/common/license/synopsys/synopsys80bc3bdd.dat
- find the line starting with 'set path = (...'. If the line does not exist ignore the next bullet and instead add the following line to your .cshrc:
  set path = ($path ~heinrich/myusr/local/bin $VCS_HOME/sun_sparc_solaris_5.5.1/bin .)
- add at the end of the line ~heinrich/myusr/local/bin $VCS_HOME/sun_sparc_solaris_5.5.1/bin .
- save the file
- type source ~/.cshrc

Alternatively, it may be that your .cshrc does not contain a PATH statement, but your .login does. In that case, follow the above directions, but substitute .login for .cshrc.

3. Quick Unix Review

3.1. Basic Commands

- **cd** dir - Changes the current directory to dir , which can be either a sub-directory of the current directory or a complete path name starting with a leading slash.
- **ls** - lists files in the current directory.
  -l option gives more information on each file.
  -a option lists all files, including hidden files.
- **cp** source dest - Copies the file source into the file or directory dest.
- **mkdir** dirname - Creates dirname as a sub-directory in the current directory.
- **rmdir** dirname - Removes directory dirname.

[1] The project was designed by Prof. Mark Heinrich for CDA4150 Fall 2004.

- **rm** filename - Removes file filename.
- **clear** - Clears the xterm window of text.
- **pwd** - Prints the current working directory.
- **man** command - Prints the manual page for command.
- **more** filename - Allows you to quickly view the contents of ASCII file filename.

3.2. Paths in Unix

In the Unix environment paths contain forward slashes ('/') instead of the backslashes you may be used to with DOS/Windows. An example of a Unix directory is /usr/bin. The current directory is referred to by a period (.). So you could type cp somefile . to copy somefile into your current directory. Also, a double period (..) refers to the parent directory. So you could type cd .. to move up one level in the directory hierarchy.

3.3. Editors

The two most commonly used editors are vi and xemacs. xemacs -nw will bring up emacs in a terminal window witout X (-nw means 'no window'). Abundant references for how to use these editors are available online.

4. Introduction to Project 0

To get the template files for lab 0, logon to monroe and do the following:

- if you have not previously done so make a directory for CDA 4150 with the mkdir command: mkdir cda4150
- cd cda4150
- cp ~hgao/cda4150/lab0.tgz .
- tar -xzf lab0.tgz

This will make a lab0 sub-directory beneath your cda4150 directory. cd lab0 to begin working.

5. A Sample Verilog Program

We have provided you with sample Verilog code that implements a full adder. The circuit takes 3 bits as input and outputs the sum and a carry bit.

- Go to your lab0 directory (cd ~/cda4150/lab0). All work for this project should be done from this directory.
- Compile the Verilog profect: make
- Run the Verilog simulator: simv

6. Part 1: A Byte Adder

Your first Verilog program will consist of building an 8-bit adder based on the full adder.

- You should write a Verilog module named byteadder (filename byteadder.v) that makes use of the fulladder module to add two 8-bit binary numbers. Your 8-bit adder should receive a carry bit as input and should also output the carry out of the last full adder. The interface for your module should be:

  byteadder(a, b, carryin, sum, carryout)

  where a, b, and sum are 8 bits wide and carryin and carryout are 1 bit wide.
- The demonstration program for your byte adder is demobyteaddder.v. This demonstration program will display on the screen the sum and carry obtained when adding a few pairs of eight-bit numbers.
  You can test your module by typing:
  make demobyteadder
  simv

7. Part 2: Adder/Subtracter

You will now extend the byteadder module from the last section so that it can do subtractions as well. Recall that A - B = A + two's complement(B) (you compute the two's complement of B by inverting all bits and adding one).

- Write a module called addsub (file name: addsub.v) that performs additions or subtractions depending on the value of a control signal. Do not use '+' or '-' anywhere in your code. The format should be

  addsub(a, b, result, control)

  where a, b, and result are eight-bit signals (ignore overflow) and control is a one-bit signal such that control = 1 performs a subtraction and control = 0 performs an addition.
- The demonstration program for your addsub module is demoaddsub.v. This will show how your addsub module works on several pairs of numbers.

  make demoaddsub
  simv

8. Part 3: An 8-bit Counter

You will now design an eight-bit counter that increments its value when a positive edge on a clock signal is detected. The eight-bit counter will also have a synchronous reset control signal that initializes the counter to zero.

- Write a module called counter (file name: counter.v). The format for counter should be

  counter(clk, rst, out)

  where clk is the clock signal, rst the reset signal, and out is the eight-bit count. Your module should increment out whenever a positive edge is detected in clk except if rst is high. If rst is high at the positive edge of the clock, out should be reset to zero (synchronous reset).
- The demonstration program for your counter module is democounter.v. It displays the value of the counter each time the output changes. In this case, you should write code that generates a rst signal so that the counter can be reset to zero whenever it would reach 60. Hence, the sequence should be 0, 1, ..., 58, 59, 0, ...

  make democounter
  simv

9. Part 4: A Digital Clock

You will now use the eight-bit counter module that you designed in the last part to construct a digital clock that displays seconds, minutes, and hours. For this purpose you will need three eight-bit counters S, M, and H. Counter S increments its value whenever a positive edge from the clock signal is detected, and reset to zero whenever it is about to reach 60. Each time counter S resets, counter M is incremented by one. Again, M should reset when it reaches 60 and this event should increment counter H, which should only be allowed to count up to 24 (it's a clock).

- Write a module called watch (file name: watch.v) that implements the digital clock described in the paragraph above. The format for your module should be

  watch(clk, rst, S, M, H)

  where clk is a one-bit clock signal, rst is a one-bit reset signal and S, M, and H are eight-bit outputs.
- The demonstration program for your watch module is demowatch.v. In this case, the provided module only generates clk and rst. You should write code that displays the current time every

minute. Furthermore, you also need to write code that finishes the simulation at 02:05:10 and displays the finish time.

```
make demowatch
simv
```

NOTE: You are allowed to write as many modules as you need in lab 0. It is a good idea to put each module in its own .v file.