ECE 206/COS 306 Lab 0: Install and Verify Capabilities of Tools

Due Date: Wednesday 09/15/2021 at 11:59 pm
No demonstration required for this lab assignment.

Introduction

In this lab, you will install the tools needed for the course's lab assignments. Additionally, you will become familiar with the tools by compiling and simulating a small amount of Verilog, the hardware description language used throughout the labs.

Directly copy/pasting code from this document may cause unwanted formatting changes and compilation errors. Therefore, you should type out all code contained in this lab.

Part 1: Installing command line tools

Instructions for installing the tools needed for the lab assignments is provided in a separate document titled "ECE 206/COS 306: Software Installation Guide". Please read and follow those instructions. If you run into any difficulties, you can talk to a TA during lab hours, check if the issue has been discussed on Ed Discussion, or post a question on Ed if the issue has not already been addressed.

You must know basic command line usage for this class. You should review the COS 126 command line cheat sheet at http://www.cs.princeton.edu/courses/archive/spr15/cos126/cmdline.html. Unless you are already familiar with the command line in your operating system of choice, it will be

very helpful for you to read in full the 15-minute introduction to the command line in your operating system provided in the COS 126 booksite. The Windows version is available at: http://introcs.cs.princeton.edu/java/15inout/msdos.html. The version for Mac, Linux, and other *nix systems is available at: http://introcs.cs.princeton.edu/java/15inout/unix.html

This course recommends Visual Studio Code as your IDE. We recommend it because it is fast, cross-platform, and contains an integrated terminal which allows Verilog commands to be easily executed. All of the examples/tutorials in this course will use VSCode.

Part 2: Compile and test a small Verilog module

Before continuing into this section, you should become familiar with Chapter 1 and 2 of the Verilog tutorial. Although you will not be tested on your knowledge of Verilog in this lab, you will be required to write your own Verilog from the next lab onward.

We will now compile and test the following very simple circuit in Verilog.

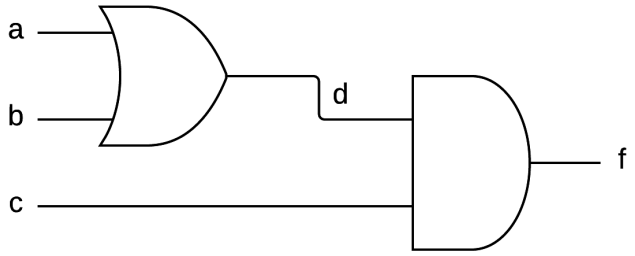

Figure 0.1: A simple combinational circuit

First, you may wish to create a directory to work in. Open up a command line, and navigate to a place where you would like to do your work for ECE 206. Then, from that location, create a directory named *lab0* and change in to that directory:

```
1 mkdir lab0
2 cd lab0
```

Now, inside the *lab0* directory, create a text file containing the following code.

```
module SimpleCircuit(
  input a,
  input b,
  input c,
  output f
);
  wire d;

  assign d = a || b;
  assign f = d && c;
endmodule
```

`SimpleCircuit` is a module that takes three one-bit inputs and produces a single one-bit output. Save this file in the *lab0* directory with the name *SimpleCircuit.v*. Be sure that you are not creating a text file named *SimpleCircuit.v.txt* with an extra, possibly hidden file extension. Next, create a second file and type the following code:

```
module SimpleCircuit_test;
  reg [3:0] counter;
  wire out;

  SimpleCircuit circuit_under_test(
    .a(counter[2]),
    .b(counter[1]),
    .c(counter[0]),
    .f(out)
  );

  always @( * ) begin
    if (counter[3]) begin
      $finish;
    end
  end

  initial begin
    $dumpfile("SimpleCircuit_test.vcd");
    $dumpvars;

    counter = 4'b0000;

    while (1) begin
      #1  // Delay for one second
      counter = counter + 3'b001;
    end
  end

endmodule
```

Save this with the name *SimpleCircuit.t.v* in the same directory. This will be the testbench for `SimpleCircuit`. It contains one instance of `SimpleCircuit`, named `circuit_under_test`, and will generate all possible input values. When simulated, the input and output values will all be recorded to a .vcd file named *SimpleCircuit_test.vcd*. After typing and saving both files, run the following command to compile the two Verilog files:

```
1 iverilog -Wall -o SimpleCircuit_test SimpleCircuit.v SimpleCircuit.t.v
```

`iverilog` is the Icarus Verilog compiler command. The `-Wall` argument tells the compiler to generate all usual warning types, which will help you identify bugs. The `-o` argument tells the compiler that the next argument is the filename to save the output to. The final arguments tell the compiler what source files to compile. If you have typed the files correctly, no warnings should be generated, and the command should succeed without displaying much. If you see any messages, you may have made a typo. Reexamine your code carefully to eliminate it. After successfully compiling the code, if you examine the contents of the directory, you should now see a new file named *SimpleCircuit_test*. This is the compiled version of `SimpleCircuit` and its testbench. To simulate the compiled circuit, run the following command:

```
1 vvp SimpleCircuit_test
```

`vvp` is the command to run the output of the Icarus Verilog compiler. It takes the filename of the compiler output as an argument. After running the simulation, if you check the contents of the current directory, you should see a file named *SimpleCircuit_test.vcd*. This file contains a record of how values changed during the simulation. The recorded data can be viewed by running the following command:

```
1 gtkwave SimpleCircuit_test.vcd
```

On MacOS, if you are getting errors, try this command instead. Keep in mind that you will always need the `open -a` prefix when using gtkwave going forward:

```
1 open -a gtkwave SimpleCircuit_test.vcd
```

This will open a GTKWave window with the `.vcd` file loaded. To view the recorded values, select the `SimpleCircuit_test` module in the SST (Signal Search Tree) pane in the upper left. The `counter` vector of registers and the wire `out` will appear as signals you can select. Click and drag them one at a time to the "Signals" pane in the middle of the window. To the right of each signal in the "Signals" pane, a green waveform will appear in the "Waves"

pane showing how that signal varied over time during the simulation. You can adjust the time scale to show roughly the entire duration of the simulation within the window by clicking the "Time" menu in the top menu bar, selecting "Zoom" and then "Zoom Best Fit". This can also be done by clicking the Zoom Fit button in the toolbar near the top of the window, which looks like a magnifying glass with a square inside of it. You should see that all the signals changed over the course of the simulation. You may further adjust your view by zooming in and out or scrolling. Clicking in the "Waves" pane places a vertical marker at a point in time, and values at that time will be shown in the "Signals" pane.

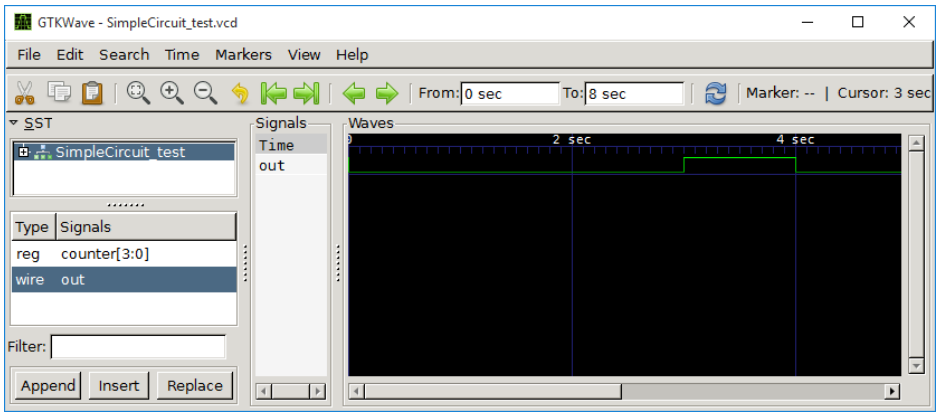

Figure 0.2: Selecting a module in the upper part of the "SST" pane (upper left) will show signals recorded for that module (lower left) which can be dragged to the "Signals" pane (center) to view their values in the "Waves" pane (right).

You may notice that the wires `a`, `b`, `c`, `d`, and `f` did not appear as signals that could be viewed when you selected the `SimpleCircuit_test` module in the SST pane. This is because these wires are internal to the `SimpleCircuit` module, and are not recorded as part of the `SimpleCircuit_test` testbench. If you expand the `SimpleCircuit_test` module by clicking the box with a "+" to its left, you will see the submodule `circuit_under_test`, which, as mentioned previously, is an instance of the `SimpleCircuit` module. If you select `circuit_under_test`, you should be able to see the five wires, `a`, `b`, `c`, `d`, and `f`, which are in a `SimpleCircuit`. You can now click and drag all of these wires over to the "Signals" pane to view its values during the simulation.

Write-Up and Submission

A demonstration is not required for this lab. In future labs, you may be required to demonstrate lab completion to a TA.

Write-Up

For your write-up, answer the following questions briefly but completely. You may create the write-up in any program. You may hand-write and scan your answers to any problems if you would prefer. However, you must submit your write-up as a single PDF document.

Name your PDF *netid_lab0_writeup.pdf*, with *netid* replaced with your Princeton NetID.

1. [**10 points**] Suppose I have a command line open at a directory named *foo*. This directory contains a subdirectory named *bar*. What command or sequence of commands could I issue on your system to find out what files are contained within the subdirectory *bar*, such that after the command or sequence of commands is executed, *foo* is the current directory as it was before the commands were executed?

2. [**10 points**] Include a screenshot of the simulated circuit behavior displayed in GTKWave, showing waveforms for all inputs and outputs of `circuit_under_test` for the entire duration of the simulation. In particular, `counter[3:0]` and `out` should be displayed in the viewer. Additionally, include the waveform for the internal wire `d`. In the screenshot, place a marker at a location such that the "Signals" pane shows the values of all the signals when the inputs were $\texttt{a} = 0, \texttt{b} = 1, \texttt{c} = 1$. Is the output of the circuit what you expect?

3. [**5 points**] How long did the entire simulation run in simulated time?

4. [**5 points**] How much time did you spend on this lab assignment, including time spent installing software?

Submission

Submit the following to *Gradescope*:

- *netid_lab0_writeup.pdf*