J.D.'s Verilog Guide - Part 1

What's covered

Hiya! In this guide, I'll go over how to set up your environment for creating, editing, compiling, executing, and submitting Verilog files on the Timberlake server.

What's not covered (yet)

What I don't go over in this part is the nitty-gritty instructions on how to write the Verilog code itself for either structural or behavioral Verilog-- that's for the next part.

If you're in need of help in this regard, please use this guide to at least get onto timberlake and take a look at the examples Bina provides in the following directory:

/web/faculty/bina/cse241/fall2014/demos

Once you've looked at those, download my Verilog files from Course Documents on UBLearns.

If you want to place those files on timberlake, you can use any SFTP program like FileZilla. My preference is PSFTP as it comes with PuTTY and it's super simple to use. Please email me if you'd like to learn how to do this. I may include it in the next part.

-J.D.

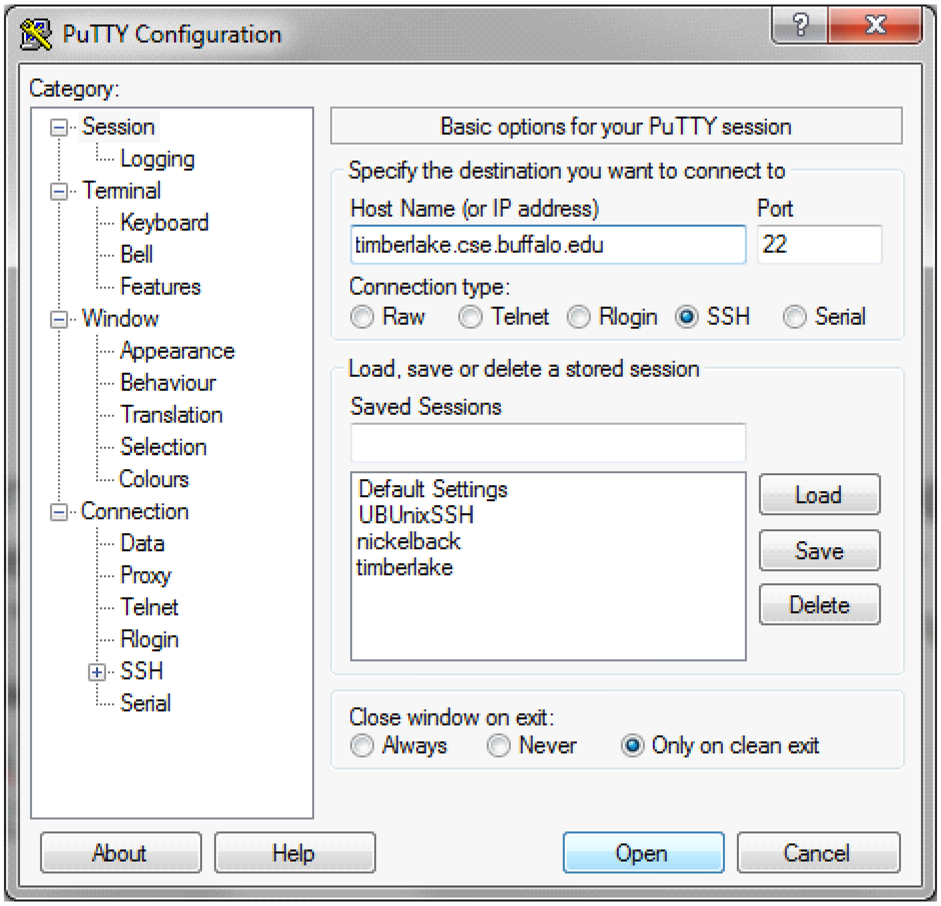
revised 11/21/14

Installation

Connecting to Timberlake

1) Mac or Linux
   a. First, make sure that you have the openSSH package installed on your machine. Mac OS should come with it, as will most of the major distributions of Linux. If you don't, the instructions for this vary from distro to distro, but below are a few. Open your terminal program and then run the following:
      i. Using APT (Debian-based distros):
      ```
      (~)> apt-get install openssh-client
      ```
      ii. Using Pacman (Arch-- my preference):
      ```
      (~)> pacman -S openssh
      ```
      iii. Using yum (RedHat, Fedora, etc):
      ```
      (~)> yum -y install openssh-clients
      ```
   b. Open an SSH connection to Timberlake. The command is pretty much the same across all of them, including Mac:
   ```
   (~)> ssh jdmallen@timberlake.cse.buffalo.edu
   ```
   (but obviously use your UBIT name instead of mine)

2) Windows,
   a. Download PuTTY. If you download it from the UBIT site, it should come packaged with the servers already preset. If not, fill in the fields and make the selections like the screenshot below, then click Open.



Specify the destination you want to connect to

Host Name (or IP address): timberlake.cse.buffalo.edu

Port: 22

Connection type: Raw Telnet Rlogin [✓ SSH] Serial

Load, save or delete a stored session

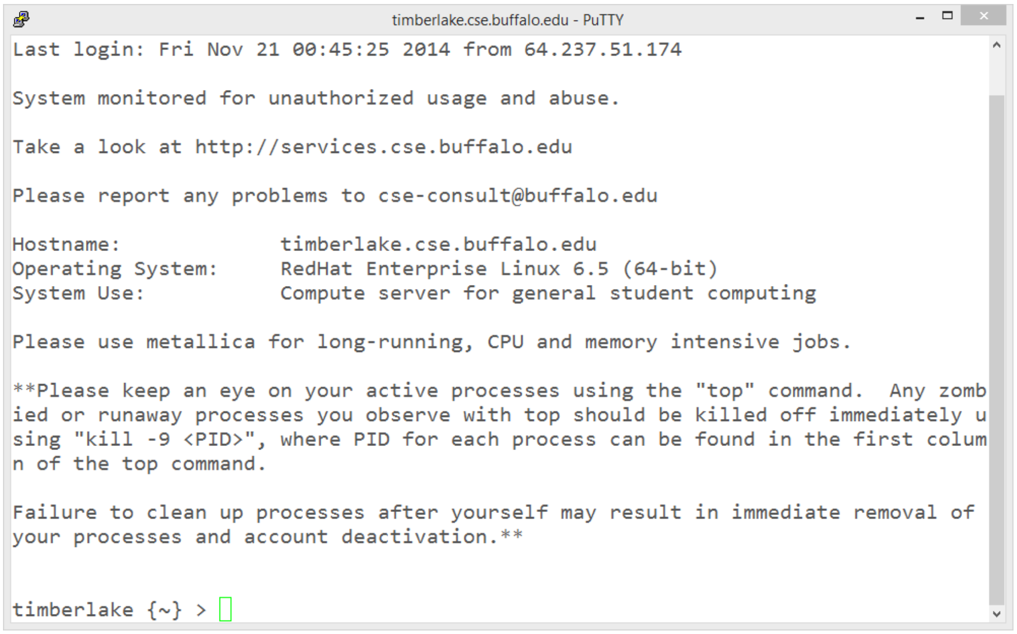
Saved Sessions

Default Settings
UBUnixSSH
nickelback
timberlake

Close window on exit: Always Never [✓ Only on clean exit]

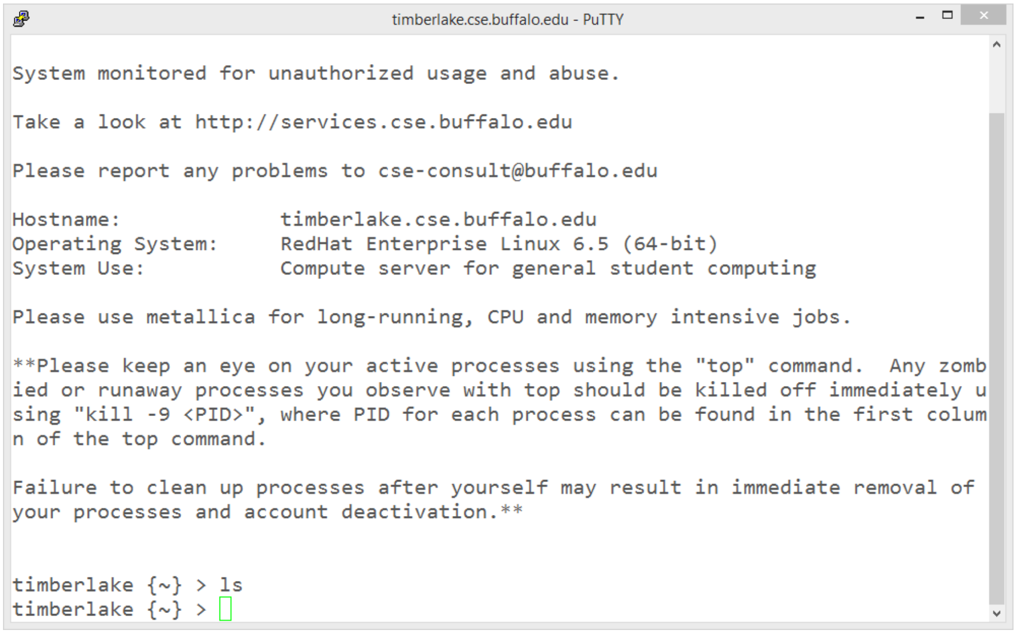
Connected-- now what?

- Congrats! You've successfully SSH'd into your personal file directory on the CSE department's timberlake server. It should look something like the below.



```
Last login: Fri Nov 21 00:45:25 2014 from 64.237.51.174

System monitored for unauthorized usage and abuse.

Take a look at http://services.cse.buffalo.edu

Please report any problems to cse-consult@buffalo.edu

Hostname:              timberlake.cse.buffalo.edu
Operating System:      RedHat Enterprise Linux 6.5 (64-bit)
System Use:            Compute server for general student computing

Please use metallica for long-running, CPU and memory intensive jobs.

**Please keep an eye on your active processes using the "top" command.  Any zomb
ied or runaway processes you observe with top should be killed off immediately u
sing "kill -9 <PID>", where PID for each process can be found in the first colum
n of the top command.

Failure to clean up processes after yourself may result in immediate removal of
your processes and account deactivation.**


timberlake {~} >
```

- If this is your first time in these parts, you should organize a bit. Let's see what you've got in there to start. Type "ls" and hit enter. If this is your first time in there, you'll probably just see this:

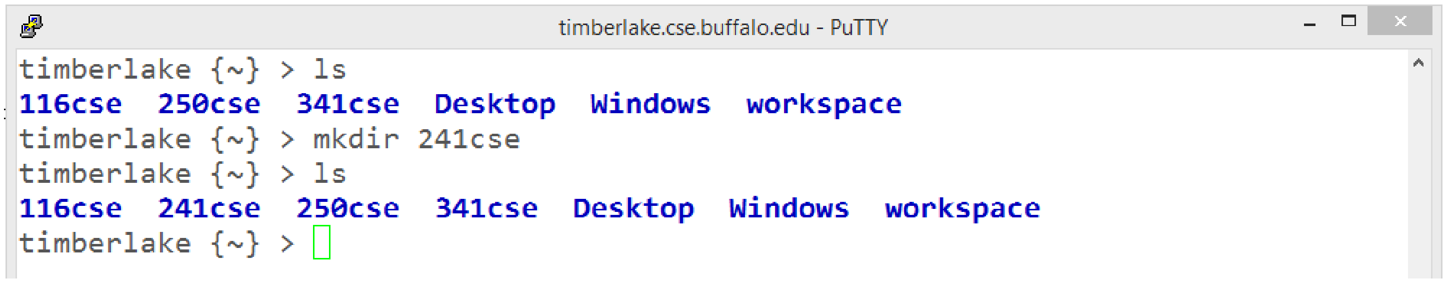


```

System monitored for unauthorized usage and abuse.

Take a look at http://services.cse.buffalo.edu

Please report any problems to cse-consult@buffalo.edu

Hostname:              timberlake.cse.buffalo.edu
Operating System:      RedHat Enterprise Linux 6.5 (64-bit)
System Use:            Compute server for general student computing

Please use metallica for long-running, CPU and memory intensive jobs.

**Please keep an eye on your active processes using the "top" command.  Any zomb
ied or runaway processes you observe with top should be killed off immediately u
sing "kill -9 <PID>", where PID for each process can be found in the first colum
n of the top command.

Failure to clean up processes after yourself may result in immediate removal of
your processes and account deactivation.**


timberlake {~} > ls
timberlake {~} >
```

- But most of you probably have SOMETHING in there.

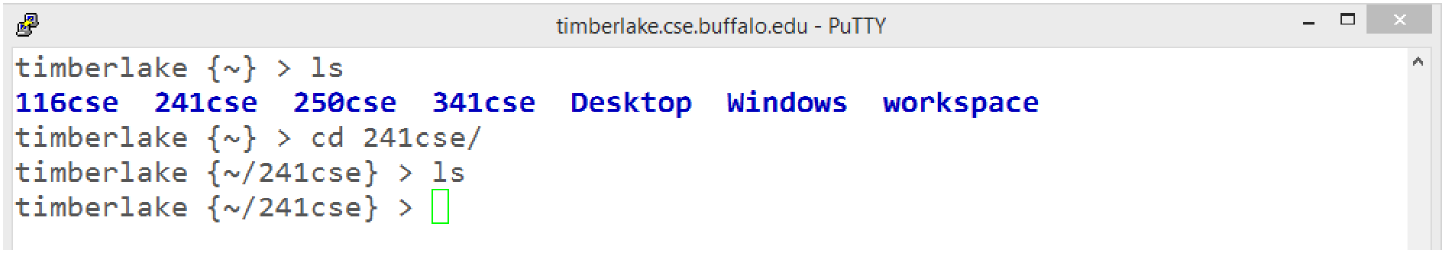
Organizing your Timberlake home directory

- Woo! You just executed a linux command! And probably nothing happened! AWESOME!!11!1!
- Seriously, though, let's make a folder for your 241 stuff by using the **mkdir** or "make directory" command. With Linux, no news is good news. If you didn't get an error, safe to assume it was successful.
- Let's also pretend you had some stuff in there to start. If you took 116, you probably already do.



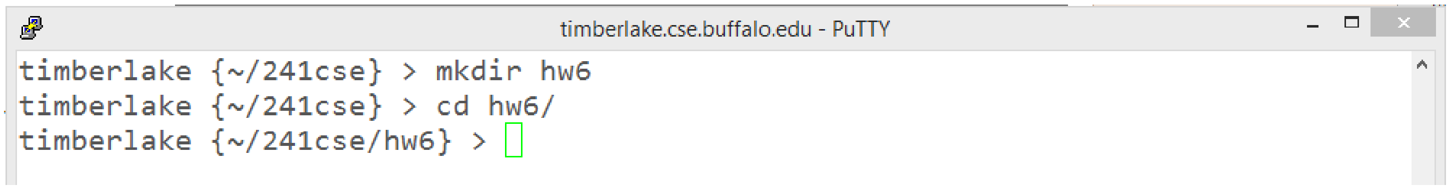
```
timberlake {~} > ls
116cse   250cse   341cse   Desktop   Windows   workspace
timberlake {~} > mkdir 241cse
timberlake {~} > ls
116cse   241cse   250cse   341cse   Desktop   Windows   workspace
timberlake {~} >
```

- Notice the 241cse was created. Let's navigate into it using the **cd** or "change directory" command.



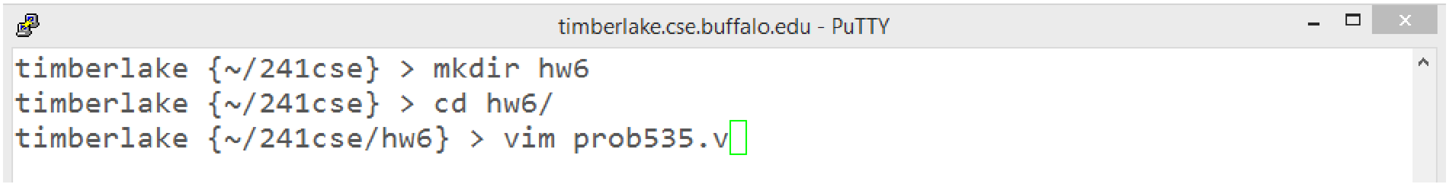
```
timberlake {~} > ls
116cse   241cse   250cse   341cse   Desktop   Windows   workspace
timberlake {~} > cd 241cse/
timberlake {~/241cse} > ls
timberlake {~/241cse} >
```

- Sure enough, it's an empty directory. Let's create a subdirectory for one of our assignments an traverse into it.



```
timberlake {~/241cse} > mkdir hw6
timberlake {~/241cse} > cd hw6/
timberlake {~/241cse/hw6} >
```

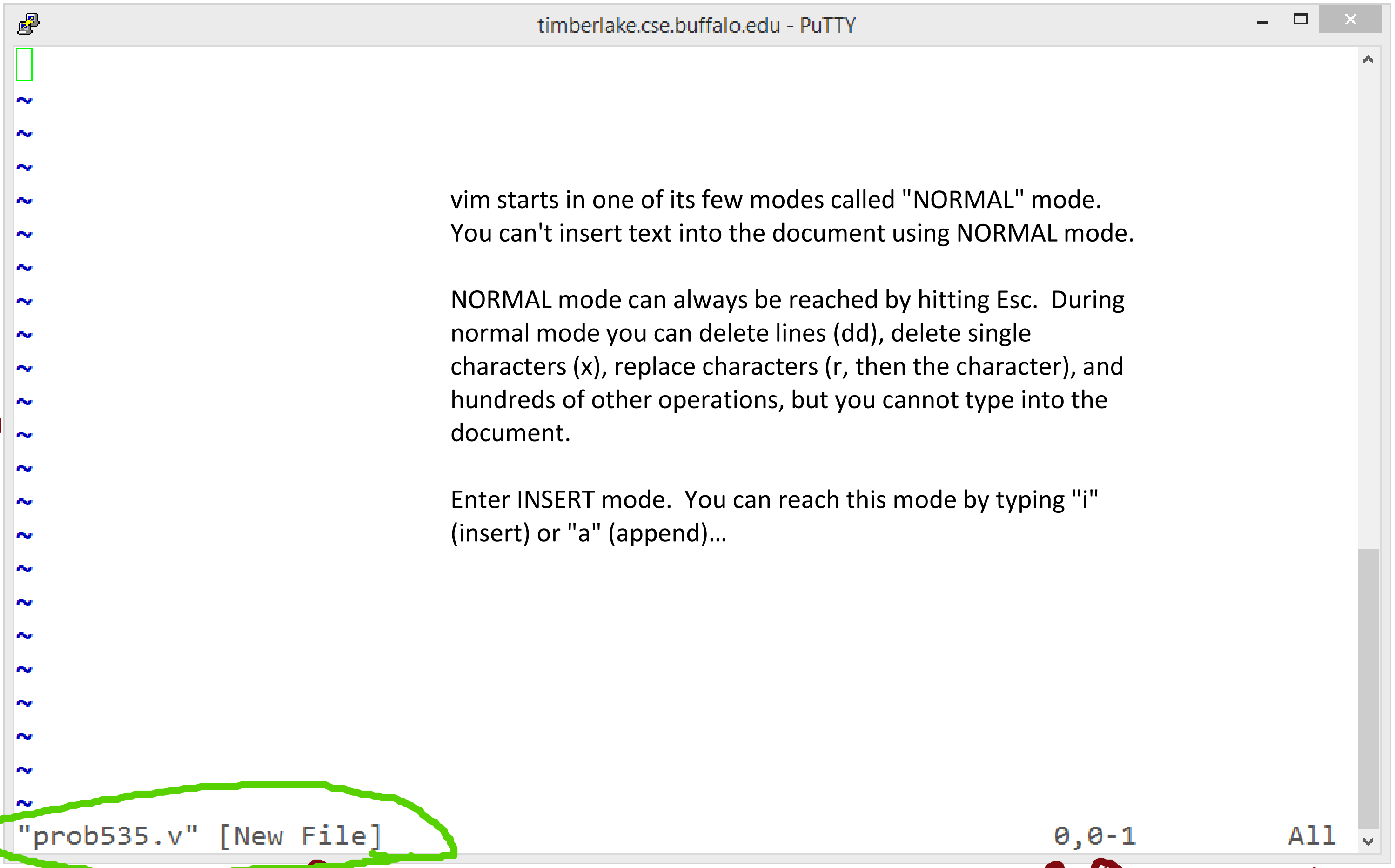
- Now we can start creating some files! Let's use the text editor vim.
  ALSO, if you want to learn more of what vim can do beyond this short guide, you can launch the program called "vimtutor" from the same command line.
- Type in "vim" followed by a space, then the name of the file you want to create/edit. In this case, for hw6, how about "vim prob535.v"? Notice the ".v" at the end. That's the file extension, and it's important for vim to know what syntax to highlight (since it tells vim that it's a Verilog file, as opposed to VHDL, Java, Python, etc).



```
timberlake {~/241cse} > mkdir hw6
timberlake {~/241cse} > cd hw6/
timberlake {~/241cse/hw6} > vim prob535.v
```

- As soon as you hit enter, vim will fill the window… (next page)

The vim (vi Improved) interface



```
"prob535.v" [New File]                                  0,0-1         All
```

vim starts in one of its few modes called "NORMAL" mode. You can't insert text into the document using NORMAL mode.

NORMAL mode can always be reached by hitting Esc. During normal mode you can delete lines (dd), delete single characters (x), replace characters (r, then the character), and hundreds of other operations, but you cannot type into the document.

Enter INSERT mode. You can reach this mode by typing "i" (insert) or "a" (append)...

~

The tildes indicate blank lines. You have to remember that a carriage return or line feed (CR/LF-- hitting the enter key) is a character!

Current filename

File hasn't been saved or "written" yet.

Line

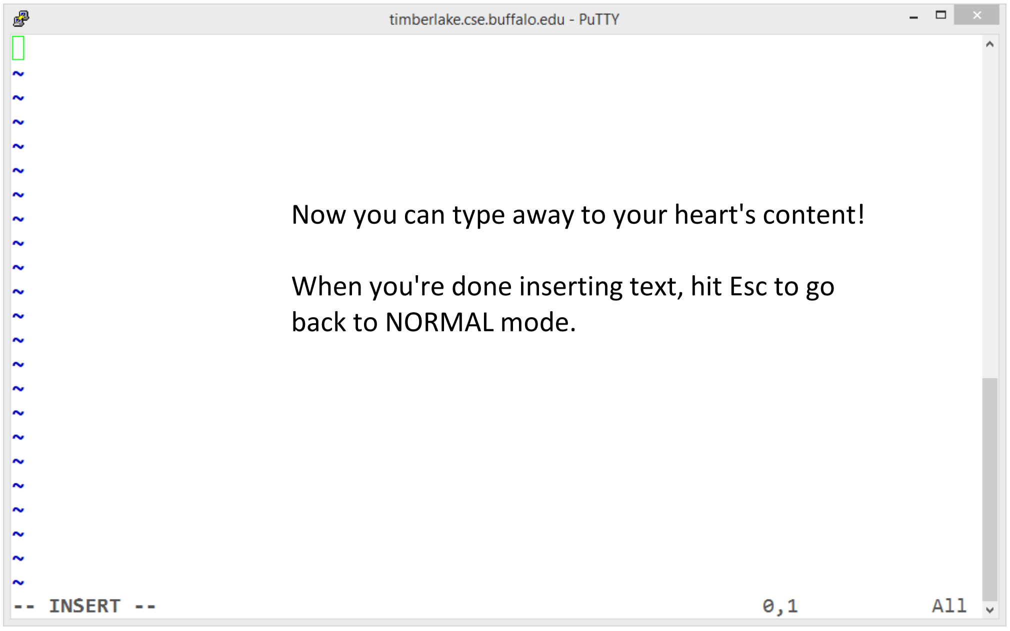
Column

Position in file.

- All-- all contents shown.
- Top-- at top of file.
- Bot-- at bottom of file.
- xx% -- percentage seen of file

This area in the lower-left will also tell you what mode you are in-- more on modes later.

INSERT mode

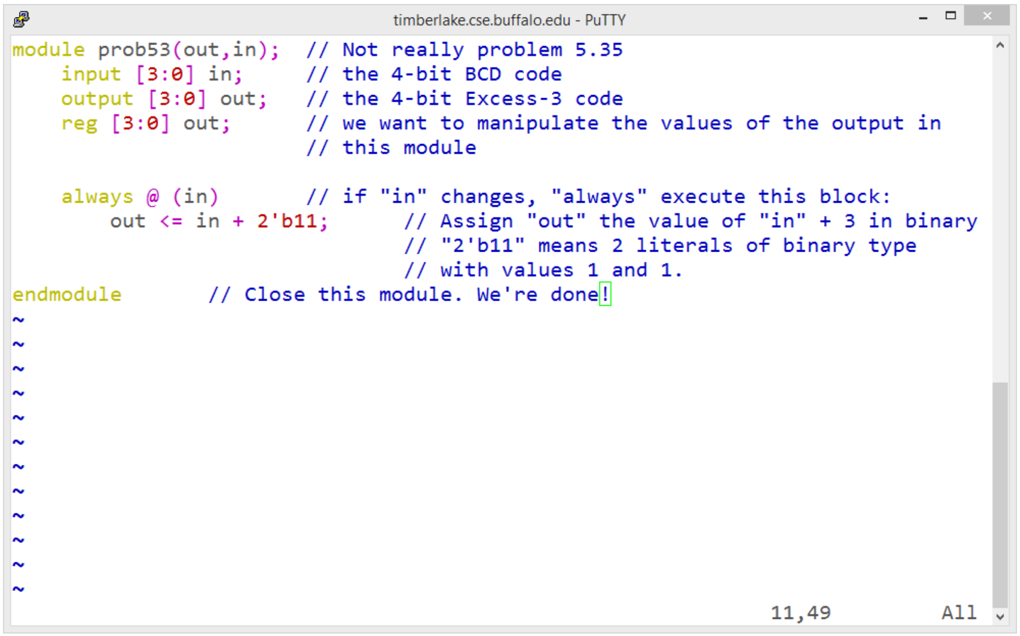


```
timberlake.cse.buffalo.edu - PuTTY
-- INSERT --                                              0,1          All
```

Now you can type away to your heart's content!

When you're done inserting text, hit Esc to go back to NORMAL mode.

- Now that you're inserting, go ahead and type some Verilog code. Problem 5.35 was assigned for HW, so I can't complete that here, but here's a real simple example of behavioral code that increments our input by 3 (a BCD to Excess-3 converter!).



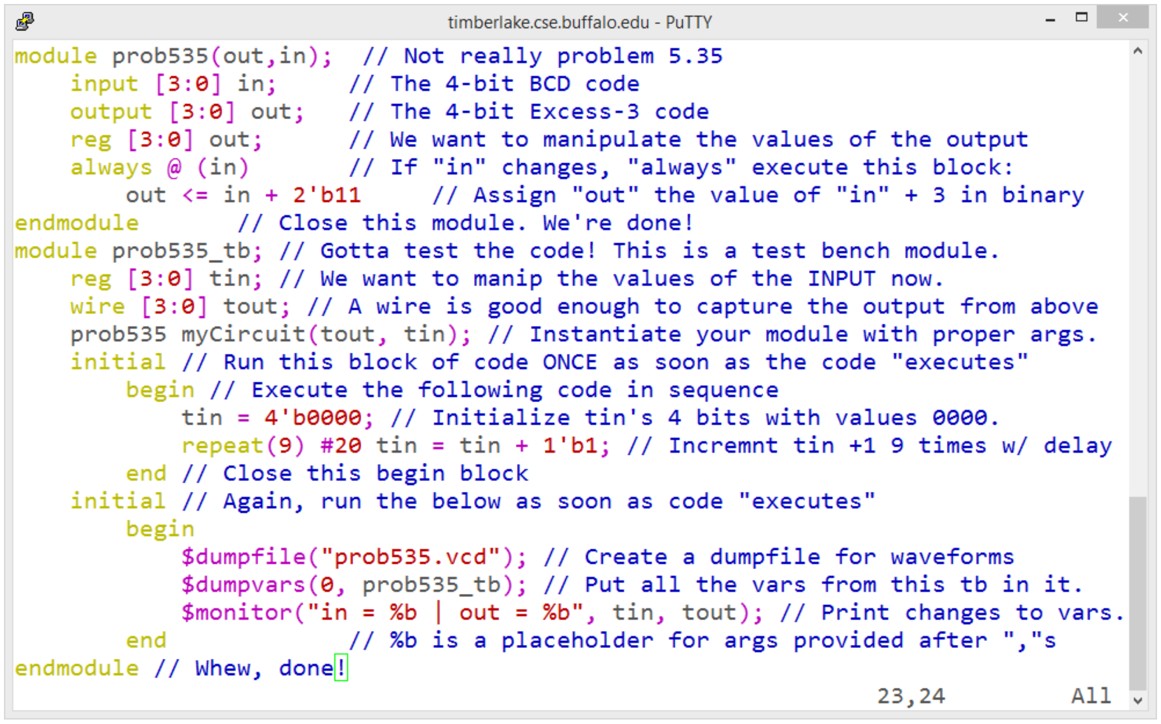
```
timberlake.cse.buffalo.edu - PuTTY
module prob53(out,in);  // Not really problem 5.35
    input [3:0] in;     // the 4-bit BCD code
    output [3:0] out;   // the 4-bit Excess-3 code
    reg [3:0] out;      // we want to manipulate the values of the output in
                        // this module

    always @ (in)       // if "in" changes, "always" execute this block:
        out <= in + 2'b11;      // Assign "out" the value of "in" + 3 in binary
                                // "2'b11" means 2 literals of binary type
                                // with values 1 and 1.
endmodule       // Close this module. We're done!
                                                             11,49        All
```

Here's that code if it's hard to read in the image:

```
module prob535(out,in);  // Not really problem 5.35
    input [3:0] in;     // the 4-bit BCD code
    output [3:0] out;   // the 4-bit Excess-3 code
    reg [3:0] out;      // we want to manipulate the values of the output in
                        // this module

    always @ (in)       // if "in" changes, "always" execute this block:
        out <= in + 2'b11;      // Assign "out" the value of "in" + 3 in binary
                                // "2'b11" means 2 literals of binary type
                                // with values 1 and 1.
endmodule       // Close this module. We're done!
```

Saving / Quitting

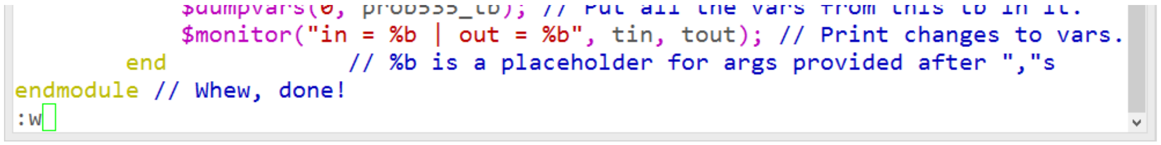
Let's assume we also wrote a test bench.



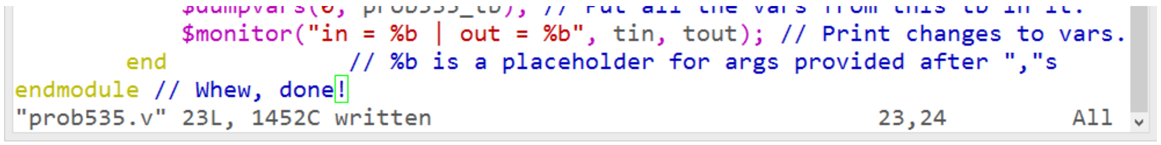
timberlake.cse.buffalo.edu - PuTTY

```
module prob535(out,in);  // Not really problem 5.35
    input [3:0] in;     // The 4-bit BCD code
    output [3:0] out;   // The 4-bit Excess-3 code
    reg [3:0] out;      // We want to manipulate the values of the output
    always @ (in)       // If "in" changes, "always" execute this block:
        out <= in + 2'b11     // Assign "out" the value of "in" + 3 in binary
endmodule       // Close this module. We're done!
module prob535_tb; // Gotta test the code! This is a test bench module.
    reg [3:0] tin; // We want to manip the values of the INPUT now.
    wire [3:0] tout; // A wire is good enough to capture the output from above
    prob535 myCircuit(tout, tin); // Instantiate your module with proper args.
    initial // Run this block of code ONCE as soon as the code "executes"
        begin // Execute the following code in sequence
            tin = 4'b0000; // Initialize tin's 4 bits with values 0000.
            repeat(9) #20 tin = tin + 1'b1; // Incremnt tin +1 9 times w/ delay
        end // Close this begin block
    initial // Again, run the below as soon as code "executes"
        begin
            $dumpfile("prob535.vcd"); // Create a dumpfile for waveforms
            $dumpvars(0, prob535_tb); // Put all the vars from this tb in it.
            $monitor("in = %b | out = %b", tin, tout); // Print changes to vars.
        end             // %b is a placeholder for args provided after ","s
endmodule // Whew, done!
                                                                  23,24         All
```

Now we want to save!
Hit Esc to exit INSERT mode, then type a colon character ":" followed by "w," then hit Enter.

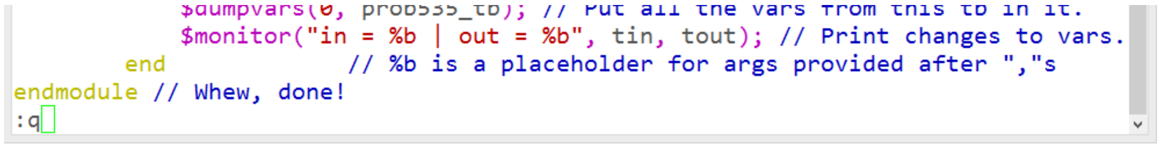


```
            $monitor("in = %b | out = %b", tin, tout); // Print changes to vars.
        end             // %b is a placeholder for args provided after ","s
endmodule // Whew, done!
:w
```

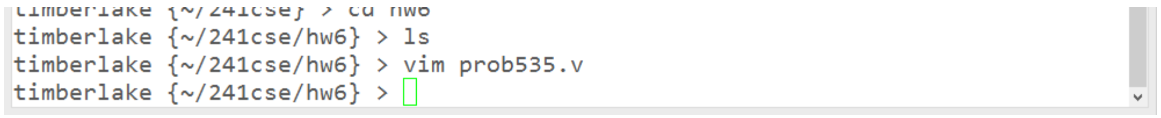


```
            $monitor("in = %b | out = %b", tin, tout); // Print changes to vars.
        end             // %b is a placeholder for args provided after ","s
endmodule // Whew, done!
"prob535.v" 23L, 1452C written                                    23,24         All
```

The file was created in our hw6 directory. Let's go check it out! First, we've gotta quit vim.
While in NORMAL mode, type a colon followed by "q" and hit Enter. If you have any unsaved work, it'll yell at you. If you want to force-quit, add an exclamation point "!" to your command.



```
            $monitor("in = %b | out = %b", tin, tout); // Print changes to vars.
        end             // %b is a placeholder for args provided after ","s
endmodule // Whew, done!
:q
```



```
timberlake {~/241cse/hw6} > ls
timberlake {~/241cse/hw6} > vim prob535.v
timberlake {~/241cse/hw6} >
```

The example code

Here's the code for the previous examples, in case you can't read it or you wanted to try it out.

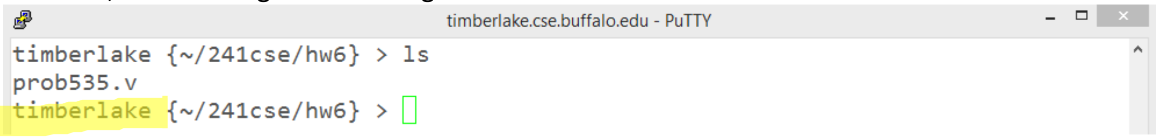
```
module prob535(out,in);  // Not really problem 5.35
    input [3:0] in;     // The 4-bit BCD code
    output [3:0] out;   // The 4-bit Excess-3 code
    reg [3:0] out;      // We want to manipulate the values of the output
    always @ (in)       // If "in" changes, "always" execute this block:
        out <= in + 2'b11      // Assign "out" the value of "in" + 3 in binary
endmodule       // Close this module. We're done!

module prob535_tb; // Gotta test the code! This is a test bench module.
    reg [3:0] tin; // We want to manip the values of the INPUT now.
    wire [3:0] tout; // A wire is good enough to capture the output from above
    prob535 myCircuit(tout, tin); // Instantiate your module with proper args.
    initial // Run this block of code ONCE as soon as the code "executes"
        begin // Execute the following code in sequence
            tin = 4'b0000; // Initialize tin's 4 bits with values 0000.
            repeat(9) #20 tin = tin + 1'b1; // Incremnt tin +1 9 times w/ delay
        end // Close this begin block
    initial // Again, run the below as soon as code "executes"
        begin
            $dumpfile("prob535.vcd"); // Create a dumpfile for waveforms
            $dumpvars(0, prob535_tb); // Put all the vars from this tb in it.
            $monitor("in = %b | out = %b", tin, tout); // Print changes to vars.
        end             // %b is a placeholder for args provided after ","s
endmodule // Whew, done!
```

[And without comments:]

```
module prob535(out,in);
    input [3:0] in;
    output [3:0] out;
    reg [3:0] out;
    always @ (in)
        out <= in + 2'b11
endmodule

module prob535_tb;
    reg [3:0] tin;
    wire [3:0] tout;
    prob535 myCircuit(tout, tin);
    initial
        begin
            tin = 4'b0000;
            repeat(9) #20 tin = tin + 1'b1;
        end
    initial
        begin
            $dumpfile("prob535.vcd");
            $dumpvars(0, prob535_tb);
            $monitor("in = %b | out = %b", tin, tout);
        end
endmodule
```

Icarus Verilog (iVerilog)

- Ok, now we've got our Verilog file:



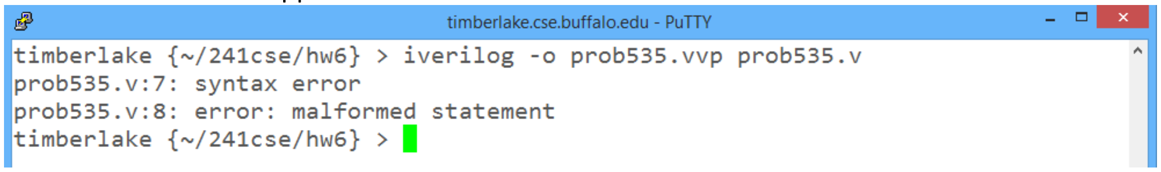
```
timberlake {~/241cse/hw6} > ls
prob535.v
timberlake {~/241cse/hw6} >
```

And we want to make sure it works. This is where iverilog comes in.

- First, we use iverilog to inspect our code. Think of it as a compiler. It will throw syntax errors if you typed anything wrong. It will create a vvp file.
- Then we use the program vvp (which comes with iverilog) to "execute" our code. This is when things like our "$monitor"s in our test benches will get to actually print to the window.
    - We can also capture that output in a text file.
- Finally, we can use GTKWave to analyze the waveform (but not with PuTTY!)

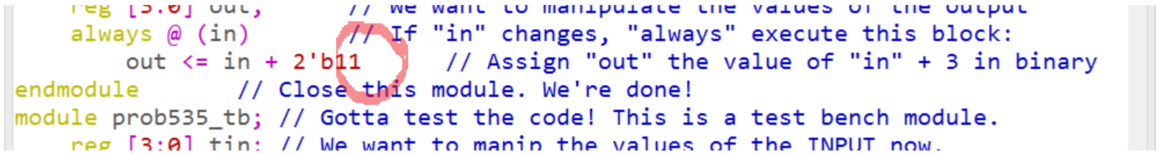
The iVerilog step

- In our hw6 directory, let's execute the following code. I'll break it down in a second.
  `{~/241cse/hw6} > iverilog -o prob535.vvp prob535.v`
    - "iverilog" calls the program.
    - "-o" is an option to provide a name for the output file. In this case, "prob535.vvp"
    - Finally, we include all of our input files. Since we put the test bench module in the same file as the circuit itself, we can just include the one file. If you wrote your test bench in a separate file, you'll need to include that here, too.
- Let's see what happens:



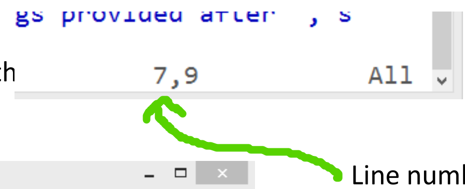
```
timberlake {~/241cse/hw6} > iverilog -o prob535.vvp prob535.v
prob535.v:7: syntax error
prob535.v:8: error: malformed statement
timberlake {~/241cse/hw6} >
```

Uh-oh! Syntax error!
- OK, so it says in in prob535.v on lines 7 and 8.
- Let's go back into vim and take a look!



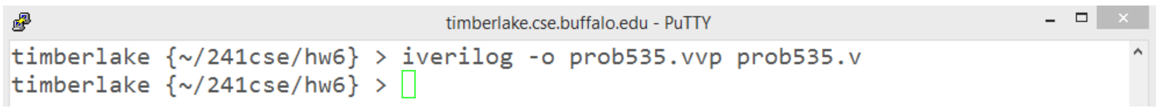
```
    reg [3:0] out;        // We want to manipulate the values of the output
    always @ (in)         // If "in" changes, "always" execute this block:
        out <= in + 2'b11     // Assign "out" the value of "in" + 3 in binary
endmodule         // Close this module. We're done!
module prob535_tb; // Gotta test the code! This is a test bench module.
    reg [3:0] tin;  // We want to manip the values of the INPUT now.
```

- Hmm. Line 7 ("endmodule") and 8 ("module pr...") look fine.
- Let's look just before and after those lines.
- See it? ... I forgot a terminating colon after 2'b11.
- Let's fix it, save our file, and run iverilog again.
- To save you the trouble of typing out that long command again, try hitting th keyboard to see previously typed commands.



```
gs provided after   , s
                7,9            All
```

Line number!

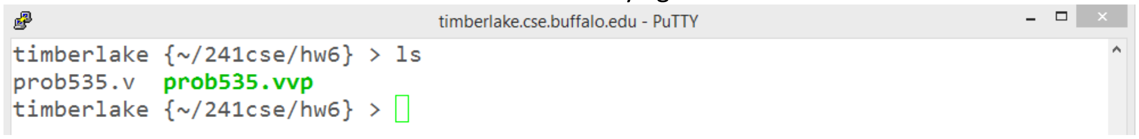


```
timberlake {~/241cse/hw6} > iverilog -o prob535.vvp prob535.v
timberlake {~/241cse/hw6} >
```

- Ta-da! No news is good news!

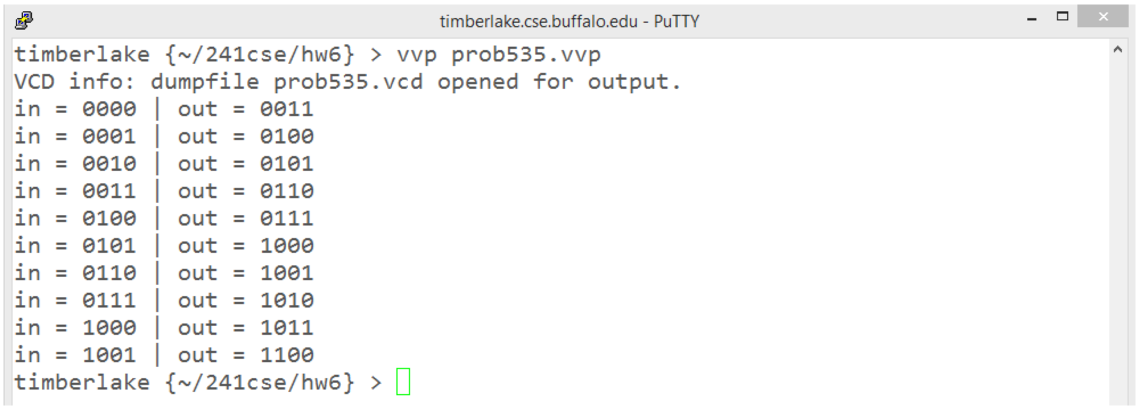
VVP / Capturing output

- Let's take a look at the contents of our directory again.



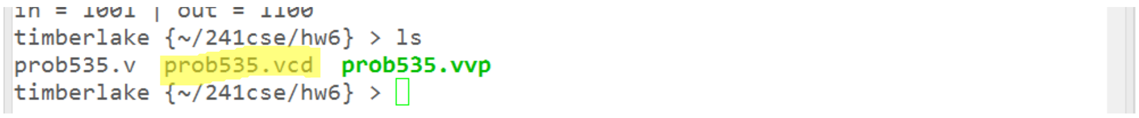
```
timberlake {~/241cse/hw6} > ls
prob535.v  prob535.vvp
timberlake {~/241cse/hw6} >
```

And look, there's the output file we specified! Let's run it! Simple command:
`{~/241cse/hw6} > vvp prob535.vvp`



```
timberlake {~/241cse/hw6} > vvp prob535.vvp
VCD info: dumpfile prob535.vcd opened for output.
in = 0000 | out = 0011
in = 0001 | out = 0100
in = 0010 | out = 0101
in = 0011 | out = 0110
in = 0100 | out = 0111
in = 0101 | out = 1000
in = 0110 | out = 1001
in = 0111 | out = 1010
in = 1000 | out = 1011
in = 1001 | out = 1100
timberlake {~/241cse/hw6} >
```

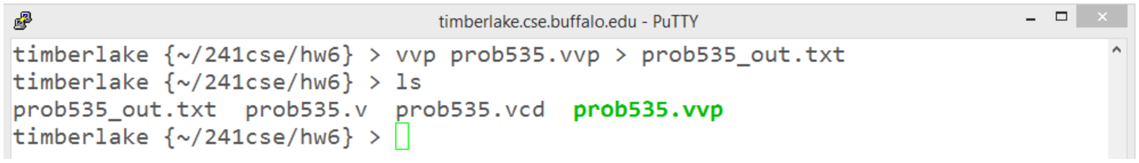
WOOOO!!! We did it! The code did exactly what we wanted it to do, and it even created a dumpfile for us! AWESOME!



```
in = 1001 | out = 1100
timberlake {~/241cse/hw6} > ls
prob535.v  prob535.vcd  prob535.vvp
timberlake {~/241cse/hw6} >
```

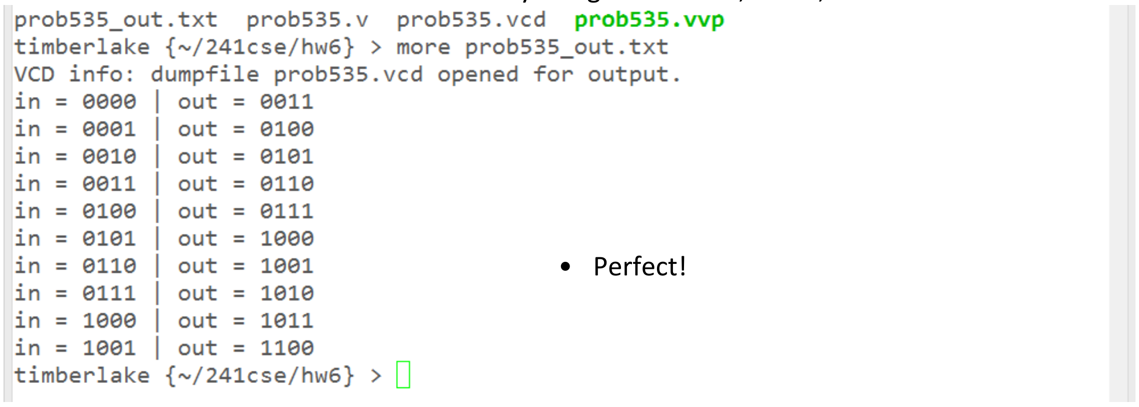
- Let's say we wanted to capture that output in a text file. What can we do?
  We have a few options:
  a. We can use the "script" command to record all terminal activity to a file.
  b. We can pipe or redirect our standard output to a file using the ">" command.
  c. And many others...

  Let's use option (b) for now.
- Execute the following:
  `{~/241cse/hw6} > vvp prob535.vvp > prob535_out.txt`



```
timberlake {~/241cse/hw6} > vvp prob535.vvp > prob535_out.txt
timberlake {~/241cse/hw6} > ls
prob535_out.txt  prob535.v  prob535.vcd  prob535.vvp
timberlake {~/241cse/hw6} >
```

- Let's make sure it has what we want by using the "more", "less", or "cat" commands.



```
prob535_out.txt  prob535.v  prob535.vcd  prob535.vvp
timberlake {~/241cse/hw6} > more prob535_out.txt
VCD info: dumpfile prob535.vcd opened for output.
in = 0000 | out = 0011
in = 0001 | out = 0100
in = 0010 | out = 0101
in = 0011 | out = 0110
in = 0100 | out = 0111
in = 0101 | out = 1000
in = 0110 | out = 1001
in = 0111 | out = 1010
in = 1000 | out = 1011
in = 1001 | out = 1100
timberlake {~/241cse/hw6} >
```

- Perfect!