Simulating Verilog HDL - Expert Users.

Developed by Karu Sankaralingam for CS/ECE 552. With input from Raghu Raman, Cherin Joseph, Vinay Gangadhar, Chen-Han Ho, and Tony Nowatzki.

Things we won't do any more	Things we will do
• Create project directory etc. using Modelsim • No more mouse clicks, to open files, open project, run simulation etc. • Enter verilog code using Modelsim • Use do files, freeze, tcl scripting etc.	• Use `wsrun.pl -wave` for simulation. Push button solution for many steps. • Use Modelsim waveform viewer for debugging • Use testbenches for verification
We will assume that the items in Getting Started with Mentor have been done. `http://pages.cs.wisc.edu/~karu/courses/cs552/spring2013/wiki/index.php/Main/GettingStartedWithMentor`	

1. Setup tasks	2. Creating modules	3. Creating a testbench
1. Create a subdirectory for each hw problem. Say hw3_1 2. Copy any of the homework modules provided verilog files into this directory. 3. Copy the verilog files for modules from previous homeworks you are using into this directory. If this is homework 1, this step can be ommitted.	We will use the same mux4_1 example as in the novice user tutorial. 1. Use any **text** editor and create the file mux2_1.v. See Appendix/ 2. I recommend gedit, vim, or emacs. 3. Enter verilog code into this file. 4. To create another module, create another file called mux4_1.v. See Appendix. 5. Create all the modules you need. At the end of above steps, if you go to unix prompt and type `ls *.v`, you should see: `clkrst.v dff.v mux4_1_hier.v mux2_1.v mux4_1.v`	1. The testbench is just another verilog module. 2. Use an editor, write and save it. For this example we will call it mux4_1_hier_bench.v

4 Compilation

Up to this step, recall you have not opened Modelsim, and the only tool you have used is your favorite editor.

1. Make sure you are in the directory for this homework and your verilog files are in this directory.
2. From the unix prompt, issue the command: `wsrun.pl` **`-compileonly`** `mux4_1_hier_bench *.v` The -compileonly flag tells the tool to only compile and not simulate also.
3. You will see a bunch of stuff and there will be one of two outcomes.

Outcome 1: Syntax errors	Outcome 2: No Syntax errors - design compiles

You may see this on your screen:
`** Error: mux4_1.v(12): near "[": expecting: IDENT`
• This means in the file mux4_1.v on line 12, you have a syntax error.
• Open mux4_1.v and go fix the syntax error.
• Redo the compilation process until there are no errors.
	Your design compiles successfully and you see no error messages. You will something like this:
	`-- Compiling module clkrst`
	`-- Compiling module mux4_1_hier`
	`-- Compiling module mux4_1_hier_bench`
	`-- Compiling module mux4_1`
	`-- Compiling module mux2_1`
	`Top level modules:`
	`mux4_1_heir`
	You are ready to simulate/verify your design.

5 Simulation

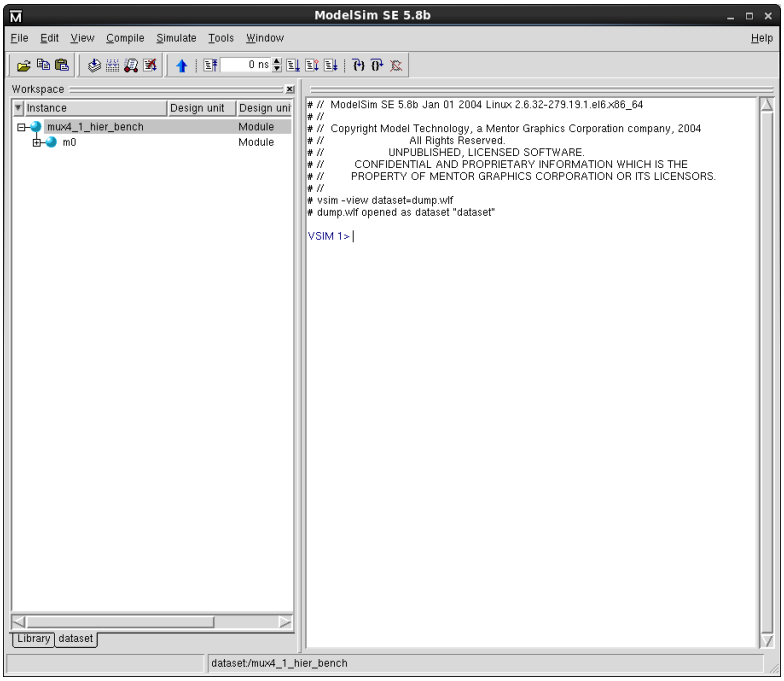
1. Make sure you are in the directory for this homework and your verilog files are in this directory.
2. From the unix prompt, issue the command: `wsrun.pl -wave mux4_1_hier_bench *.v`
   The -wave command, tells the tool to compile, simulate, and open the waveform viewer.

You will see a bunch of stuff and you should see a message that reads:

```
To view waveforms, open with
vsim -view dataset=dump.wlf
Reading /afs/cs.wisc.edu/s/mentor-2004/common/modeltech-5.8b/tcl/vsim/pref.tcl
```

6 Verification

Now your entire design has been simulated and for every cycle and every input transition in your testbench, the output of all signals of all modules has been recorded in a file called dump.wlf. The invocation of `wsrun.pl -wave` should have launched Modelsim, and it will now be open and you should see your module view open up, like the picture below:



1. You can navigate down to any level of the hierarchy by clicking on the + symbols.
2. You can right-click on any module and view all of its signals by choosing the Add → Add to Wave option
3. In general do not use the View source option.
4. Also, we will generally not use the command window to type anything.

If you Add → Add Wave the top module, you will see a window like the one on the left-hand side of the figure in the next page. If you expand the hierarchy and Add → Add Wave the m0 and c0 modules, you will see more signals like the ones on the right-hand side.

The key thing to note is that you did not have to explicitly decide which signals you want to monitor etc. while running simulation.

Verification is the process of looking at the output and debugging: when there is an unexpected (wrong) output, looking at signals that lead to this output to figure out what went wrong. A well designed testbench with also check the outputs and say when the design is wrong, instead of manually checking every output to see if it is correct.

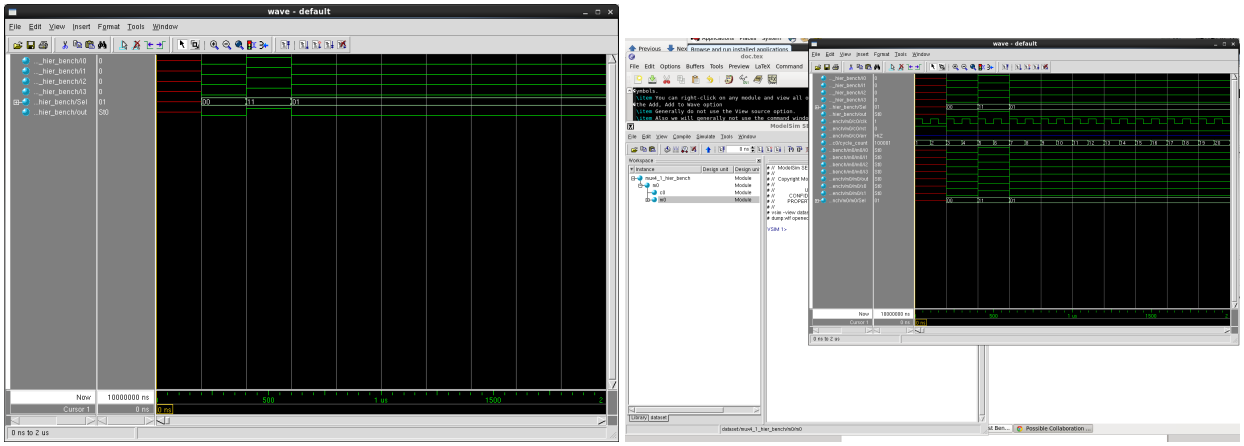


Figure 1: Waveform window

7 Advanced Usage

1. **[Changing radix]** Right click on a signal name then choose Radix → Hexadecimal to change to hexview.
2. **[Saving signal formatting for later simulations]** If you have dragged and dropped a set of signals, found a bug, fixed and want those exact same set of signals, you can retrieve a previous set using the File → Save → Format option. Save to some filename called foo.do. Then, reload using File → Open → Format option and specify foo.do
3. **[Modelsim freezes]** Open another Unix terminal. And type the command: `killmodelsim`
   or also affectionately synonymously referred to as `diemodelsimdie`
4. **[Wierd simulation results]** Go to the directory where you have all the .v files and issue this unix command: `rm -rf __work`. **Be extremely cautions before issuing the command as typos could wipe out all your verilog files etc.**

A Appendix

Example: mux2_1.v

```
module mux2_1(i0, i1, sel, out);
   input i0;
   input i1;
   input sel;
   output out;
   reg outreg;

   always @* case (sel)
   2'b0 : outreg = i0;
   2'b1 : outreg = i1;
   endcase

   assign out = outreg;

endmodule
```

Example: mux4_1.v

```
//a 4-1 multiplexer module
module mux4_1(i0, i1, i2, i3, Sel, out);
   input i0;
   input i1;
   input i2;
   input i3;
   input [1:0] Sel;
   output out;
   wire s0, s1;
   //use mux2_1 as a submodule
   mux2_1 submod1(i0, i1, Sel[0], s0);
   mux2_1 submod2(i2, i3, Sel[0], s1);
   mux2_1 submod3(s0, s1, Sel[1], out);

endmodule
```

Example: mux4_1_hier_bench.v

```
module mux4_1_hier_bench;

   reg i0;
   reg i1;
   reg i2;
   reg i3;
   reg [1:0] Sel;
   wire      out;
   reg       expected_out;

   wire      mismatch;


   mux4_1_hier m0(/*AUTOINST*/
                  // Outputs
                  .out                  (out),
```

```
                // Inputs
                .i0                     (i0),
                .i1                     (i1),
                .i2                     (i2),
                .i3                     (i3),
                .Sel                    (Sel[1:0]));

   assign mismatch = (expected_out == out);

   initial begin
      #200;
      expected_out = 1'b0;
      {i0, i1, i2, i3} = 4'b0000;
      Sel = 2'b00;
      #200;
      expected_out = 1'b1;
      {i0, i1, i2, i3} = 4'b1111;
      Sel = 2'b11;
      #200;
      expected_out = 1'b0;
      {i0, i1, i2, i3} = 4'b0000;
      Sel = 2'b01;
      #200;
      expected_out = 1'b1;
      {i0, i1, i2, i3} = 4'b0111;
      Sel = 2'b01;


   end

endmodule
```