Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 – Introductory Digital Systems Laboratory

ModelSim/Verilog Tutorial
Tutorial Comments to David Milliner (dlm@mit.edu)

Introduction

This tutorial is designed to familiarize you with Verilog coding/syntax and simulation in the
ModelSim environment. Verilog HDL is a hardware description language used to design digital
systems. Along with VHDL, Verilog is the primary industry tool for programming digital
systems. ModelSim is the industry standard simulation tool for verifying digital designs.

Directory Structure

When you log into the lab computers you will have access to your own drive (U:/) where you can
store your files for this class. You will probably want to back these files up on your Athena
account from time to time as a backup. You can transfer files back and forth from your Athena
account using WinSCP.

In your U:/ drive create a tutorial folder with the subfolders src and sims. The src folder will
contain the source code for this tutorial and the sims directory is where you will be compiling
your code.

On the lab PCs you will see a shared drive (S:/) from which you should be able to access the files
necessary for this tutorial. Copy the files from the S:/6.111/tutorial2/ folder to your src folder on
the U:/ drive. The files to copy are counter.v, top.v, tb_tutorial.v, full_adder.v, full_adder_4bit.v,
and test_adder.v.

Verilog Source Code and Testbench

The file counter.v is a simple two-bit Verilog counter designed to divide the input clock by four.
This counter is designed to reset back to zero on the positive assertion of the reset signal. Your
counter is instantiated in the top level file top.v. The file tb_tutorial.v is used to simulate the
counter module in counter.v. We recommend you spend a few minutes familiarizing yourself
with this code and the Verilog syntax.

Starting ModelSim

You can access ModelSim either through the PCs in the lab or an Athena Sun Workstation.

On the Lab PC under Windows XP, launch ModelSim from the Desktop icon (or Start -> All
Programs -> ModelSim SE -> ModelSim).

If you encounter problems with licenses, you can run the license wizard: Start -> All Programs ->
ModelSim SE -> Licensing Wizard. The license file should be set to "@mtlcad.mit.edu".

You can also access ModelSim through an Athena Sun workstation. First run 'setup 6.111' to
configure your environment correctly. Then run 'vsim &' to start the application.

Online help and tutorials for ModelSim are available from the Help pulldown menu.
(Help-> SE PDF Documentation-> Tutorial) will bring up the guide for a recommended tutorial.
 The PDF for the user's manual is also available online at:
https://web.mit.edu/6.111/www/s2004/restricted/se_man.pdf (must have MIT certificates).

Library Creation

You will need a location to compile your source and testbench files. This is what your sims
directory is for. If you are not already located in your sims directory then move into this directory
using the ModelSim command line.

On the left-hand side of the ModelSim window you should see a number of libraries (vital2000,
ieee, etc...). Right-click in this window and select to create a new library. You should call the
library sims with a logical mapping to your sims directory.

Compiling in ModelSim

You can compile your Verilog files from inside ModelSim. From the Compile pulldown menu
select Compile. You will see that the default library is the work library. Chose to compile your
files in the newly created sims library rather than the work library.

To compile your files browse to your src directory select the files counter.v, top.v and
tb_tutorial.v and double-click them to compile. We will use the other files later in this tutorial. If
there is an error in your Verilog code it will be reported at this time with an error message. Your
downloaded tutorial files should not contain any errors.

Simulation

You are now ready to simulate the counter module using the testbench tb_tutorial.v. If you are not
currently in the sims directory you should move there at this time. From the ModelSim command
line type

> vsim sims.tb_tutorial

to compile your testbench. To run the testbench type

> run 100us

to restart (not recompile) your simulation type

> restart –f

Viewing Simulations – The Wave Window

To graphically view your testbench pull down the View menu (View -> Wave). From the left-
hand side of the ModelSim window you should see blue dots representing the modules in your
design. This should have occurred when you compiled your testbench. Drag the blue dot
representing the modules in your design into the Wave window. Restart your simulation (type
restart –f) and run for 100us. You should see the clock signal, reset signal, clock divided by four
signal, and two-bit counter value. Zoom out to see the entire wave window. You can select to

have your data represented in a number of different ways although most of you will probably choose to use binary or decimal representation.

Exercise

You should now feel comfortable in the ModelSim environment. As an exercise to demonstrate your familiarity with the environment compile and run the full-adder testbench found at (S:/6.111/tutorial2). The viewgraph from lecture 3 describing this exercise is shown below. You can also try creating your own modules and verifying their functionality.

Testbenches (ModelSim) – Demo this week in Lab by TAs

Full Adder (1-bit)

```
module full_adder (a, b, cin,
            sum, cout);
  input   a, b, cin;
  output  sum, cout;
  reg     sum, cout;

  always @(a or b or cin)
   begin
     sum = a ^ b ^ cin;
     cout = (a & b) | (a & cin) | (b & cin);
   end
Endmodule
```

Full Adder (4-bit)

```
module full_adder_4bit (a, b, cin, sum,
cout);
  input[3:0]  a, b;
  input       cin;
  output [3:0] sum;
  output      cout;
  wire        c1, c2, c3;

  // instantiate 1-bit adders
  full_adder FA0(a[0],b[0], cin, sum[0], c1);
  full_adder FA1(a[1],b[1], c1, sum[1], c2);
  full_adder FA2(a[2],b[2], c2, sum[2], c3);
  full_adder FA3(a[3],b[3], c3, sum[3], cout);
endmodule
```

Testbench

```
module test_adder;
  reg [3:0]  a, b;
  reg        cin;
  wire [3:0] sum;
  wire       cout;

  full_adder_4bit dut(a, b, cin,
                  sum, cout);

initial
   begin
     a = 4'b0000;
     b = 4'b0000;
     cin = 1'b0;
     #50;
     a = 4'b0101;
     b = 4'b1010;
     // sum = 1111, cout = 0
     #50;
     a = 4'b1111;
     b = 4'b0001;
     // sum = 0000, cout = 1
#50;
     a = 4'b0000;
     b = 4'b1111;
     cin = 1'b1;
     // sum = 0000, cout = 1
     #50;
     a = 4'b0110;
     b = 4'b0001;
     // sum = 1000, cout = 0
   end // initial begin
endmodule // test_adder
```
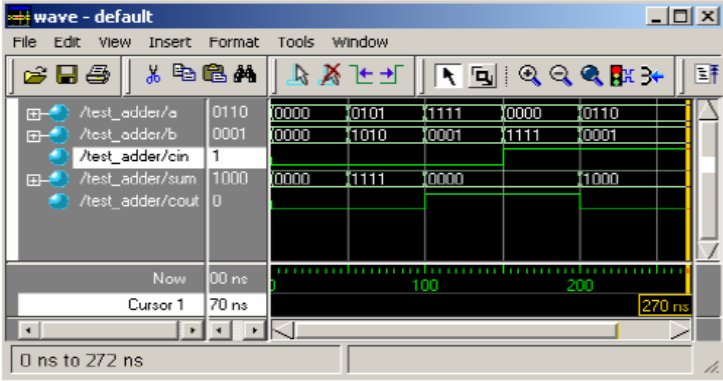
ModelSim Simulation

Courtesy of F. Honore, D. Milliner

Acknowlegements: This tutorial and counter code written by David Milliner. Adder module and testbench written by Frank Honoré.