

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

ModelSim/Verilog Tutorial

Authors: David Milliner, Frank Honoré, Spring, 2004

Jenny Lee, Spring, 2005

Theodoros Konstantakopoulos, Spring 2006

Javier Castro, Spring 2007

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

The first time you log into the lab computer you should familiarize yourselves with the directory structure and the drives and directories you can store your files for this class.

Double-click on “My Computer”. You will see one local drive, **C:**, and two network drives, **S:** and **U:**

C: is a local drive and should only be used for storing and compiling files temporarily. However, the files on drive C: cannot be retrieved from any other computer in the lab. Using **C:** as your working drive for your source files *will speed up your simulations, compared to using a network drive*. This is particularly true at the end of the term when everyone is in the lab. If you wish to save the files, copy them over to the **U:** drive. Be sure you delete everything you saved on the **C:** drive before each log-off.

S: is a network drive. It is a shared network drive. On this drive there is a ‘6.111’ folder that all 6.111 students have access to it. In this folder (‘S:\6.111’) you will find files useful for tutorials.

U: is a network drive. Each student with a lab account has a private share on this drive. Only the student and administrators have access to the folders on your share. If you double-click on this drive you will see among others, a folder called ‘Desktop’. **This ‘Desktop’ folder is the Desktop you see when you log into a lab computer.** The contents of drive ‘U:’ are accessible from any computer in the lab area.

Please make sure you save all your source files in U:/Desktop.

(Another way to access the contents of your ‘U:’ drive is by double-clicking the ‘My Documents’ folder on your Desktop when you log into a lab computer - drive ‘U:’ points to ‘ecslabs\labroot\users\Username’.)

Although drive 'U:' is accessible from any computer in the lab area, you will probably want to back these files up on your Athena account from time to time. You can transfer files back and forth from your Athena account using WinSCP.

Verilog Source Code and Testbench

In 'S:\6.111\Tutorials' you will find source files for two short tutorials: a counter tutorial and an adder tutorial.

On the Desktop of your local computer, create a 'Tutorial' folder with a subfolder for each tutorial (U:\Desktop\Tutorials\counter and U:\Desktop\Tutorials\adder).

In your 'counter' folder create two subfolders: 'src' and 'sims'. The 'src' folder will contain the source codes for this tutorial and the 'sims' directory is where you will be compiling your code.

Copy the files 'counter.v', 'top.v', 'tb_tutorial.v', from 'S:\6.111\Tutorials\counter' to your 'src' folder in your 'counter' directory.

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.

Opening ModelSim

You can access ModelSim either through the PCs in the lab or an Athena Workstation. On the Lab PC under Windows XP, launch ModelSim from the Desktop icon (or [Start > All Programs > ModelSim SE > ModelSim](#)).

If it's your first time opening ModelSim or 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 '27000@mtlcad.mit.edu'. ModelSim may also ask if it can register '.mpf' files as belonging to it. Clicking OK, in this case, should be fine.

On Athena workstations, 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 pull-down 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 on the course website: '<http://www-mtl.mit.edu/Courses/6.111/labkit/ModelSim.pdf>' (must have MIT certificates).

What are Library and Project?

Before jumping into using ModelSim, there are two important components you should get familiar with: Library and Project. Library provides an environment for you to compile and simulate your design, while Project provides you a place to contain all relevant files and settings for independent design including its working library.

Library:

A directory that contains compiled design units, such as modules. There are two types of libraries: Resource Library and Working Library:

Resource Library:

contains static contents, such as the compiled version of standard modules. Resource libraries, such as `ieee`, can be found in the 'library' pane on the left-hand side of ModelSim.

Working Library:

contains the compiled version of your design. The contents of a working library change every time you compile your design. The default working library in ModelSim is named `work` and is predefined in the ModelSim compiler. The working library when created or linked to your source code can be accessed through the 'library' pane on the left-hand side of ModelSim.

Project:

A collection of various files for designs under test, such as Verilog source files, local working libraries, references to resource libraries, and simulation configuration (*.mpf files).

Creating files in ModelSim

There are two ways to start creating your designs in ModelSim: 1.Creating a project or 2.Opening or creating a Verilog file without a project.

1.Creating a project

Create a new project: [File > New > Project](#)

Specify your project name and specify the Project Location as a directory under `'U:\Desktop\Tutorials\counter'`. Leave the Default Library Name to `work`. Now, you have created a project .mpf file and a working library, which can store useful information, such as your simulation configuration, for future use.

Now, add items to your project by either creating a new Verilog source file or adding an existing file.

2. Creating without a project:

Create a new file: [File > New > Source > Verilog](#) or open an existing Verilog file: [File > Open > File](#). To compile the source files in the ModelSim environment, you must create a working directory or map an existing working directory: [File > New > Library](#) or [File > Import > Library](#) or type `vlib work` in the command line window.

As a good design practice, we recommend you to follow the first option: creating a new project. For Verilog files required for labs, create a new project such as 'lab01' under the default 'work' library as its working library, because you do not explicitly need to specify the work library when compiling. For the final project, which contains many layers of hierarchy, create a new project under a new working library, such as 'final_proj'.

For this tutorial, we are using the second option: opening existing files and creating a new working library called 'sims'. If you are not already located in your 'sims' directory, then move into this directory using the ModelSim command line (`cd 'U:/Desktop/Tutorials/counter/sims'`).

On the left-hand side of the ModelSim window, you should see a number of resource libraries (vital2000, ieee, etc...): Right-click in this window and select to create a new library called 'sims', or type `vlib sims` in the command line.

Compiling in ModelSim

Compile source files: [Compile > Compile](#). You will see that the default library is the 'work' library. Choose to compile your files in the newly created 'sims' library rather than the 'work' library.

To compile your files, browse to your `src` directory and select the files `'counter.v'`, `'top.v'` and `'tb_tutorial.v'`. Selected all files and click on 'Compile'. If there is an error in your Verilog code it will be reported at this time with an error message. Your tutorial files should not contain any errors. Click 'Done' when compilation is over.

You can recompile your files by looking at the 'library' pane. On the left-hand side of ModelSim window, choose the 'library' tab and click on the square containing plus sign next to 'sims' working library. You should see the three files you compiled earlier under 'sims' library. Highlight the files you want to recompile, and when you right-click on those files, you can delete, recompile or simulate the highlighted files.

Simulation

You are now ready to simulate the counter module using the testbench `tb_tutorial.v` in your 'sims' directory. To simulate, you can either use the graphical interface (see *Useful Buttons and Command Line* section) or use the commands described below:

To simulate your testbench:

```
> vsim sims.tb_tutorial
```

To run the testbench:

```
> run 100us
```

To restart your simulation:

```
> restart -f
```

Viewing Simulations – The Wave Window

To graphically view your testbench pull down the View menu: [View > Debug Windows > 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.

In the Wave window, if you do not want to use testbench, you can also specify the behaviors of your input waveforms. Right-click on the input signal 'Create\Modify Waveform'. You can set the input signals to a constant, random, repeater or counter. Once you set the input signals, you can also manually draw the waveform by highlighting the parts of your wave, right-click, and select [Edit Wave > Invert](#) or [Mirror](#) or [Value](#).

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.

By default, when you close the Wave window, all the information is forgotten. Thus, if you want to save your waveforms, you must save a Wave window format file (*.do) under your project: [File > Save > Format](#).

Useful Buttons and Command Lines

Buttons	Description*	Command Line
	Compile this file open the Compile Source File dialog; in a project, compile the file	To compile: vlog -work [working library] [filename.v] ex) vlog -work work tb_tutorial.v To recompile: vlog -work [working library] -refresh
	Compile All compile all files in the open project	Compile > Compile All
	Simulate load the selected design unit or simulation configuration object	vsim -c [working library].[file name] Ex) vsim -c work.tb_tutorial
	Restart reload the design elements and reset the simulation time to zero, with the option of using current formatting, breakpoints, WLF file, virtual definitions, and assertion settings	restart -f
	Run run the current simulation for the specified run length	Run
	Open Wave Viewer	Window > Wave
	Zoom Mode set mouse to Zoom Mode – drag left mouse button to zoom, click middle mouse button to select	N/A
	Insert Cursor add a cursor to the waveform pane	Insert > Cursor

*The button descriptions are copied from User's Manual v5.8b (se_man.pdf) published by ModelSim®.

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\Tutorials\adder). The viewgraph from lecture 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;

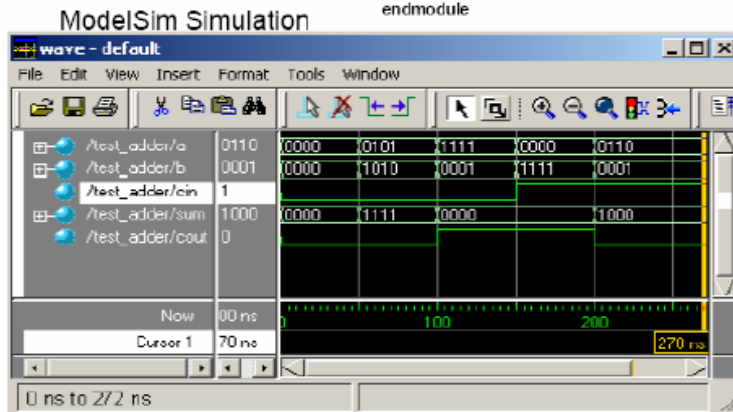
  // instantiate 1-bit adders
  full_adder FA0(a[0],b[0], cin, sum[0], c1);
  full_adder FA1(a[1],b[1], c1, sum[1], c2);
  full_adder FA2(a[2],b[2], c2, sum[2], c3);
  full_adder FA3(a[3],b[3], c3, sum[3], cout);
endmodule
```

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
```



Courtesy of F. Honore, D. Milliner

Acknowledgements: This tutorial and counter code is written by David Milliner and adder module and testbench is written by Frank Honoré. Some sections are added and modified by Jenny Lee.