Version for ModelSim SE 6.2e as of January 11, 2007

Introduction

ModelSim is a powerful HDL simulation tool that allows you to stimulate the inputs of your modules and view both outputs and internal signals. It allows you to do both behavioural and timing simulation, however, this document will focus on behavioural simulation.

In behavioural simulation, you can write models that will not necessarily synthesize. An example is a behavioural model for the ZBT RAMs used on the Xilinx Multimedia board. This code cannot be synthesized, but it is intended to give a true reflection of the behaviour of the memory chip so that you can test whether your memory controller is functioning properly.

This module is intended as a quick intro to using ModelSim in the UofT environment. Other resources for ModelSim are linked on the UofT EDK page.

Source Files for Examples

Download and unzip the `m07.zip` file from the UofT EDK Tutorials page.

Using the ModelSim GUI

While there are many things you can do with ModelSim, these are the basic things you will need to get started.

1. If you are using a Windows-based system, launch ModelSim from the Courseware directory. Otherwise, if you are logged in on the ECF Linux workstations, make sure `7326@picton.eecg.utoronto.ca` is in your `LM_LICENSE_FILE` environment variable and then launch `/local/packages/modeltech/bin/vsim`.

2. From the Modelsim GUI, change directory to your project directory, which should include your design files and testbench.

3. Type the following command to create a ModelSim working directory called "work". This is where Modelsim will compile your design to (GUI: menu File → New → Directory):

```
% vlib work
```

4. Before simulating your design, you need to compile the source files and testbench. For hierarchical designs, compile the lower level design blocks before the higher level design blocks. To compile, (GUI: menu Compile → Compile) type the following commands:

```
% vlog <design_file>.v
% vcom <design_file2>.vhd
% vlog <testbench>.v
```

5. To simulate your design, type the following command:

```
% vsim <working_directory>.<topmost_module_name>
```

6. For example, if your working directory is "work" and your design has topmost module named "top" (GUI: menu Simulate → Simulate, select the topmost module name from the "Design" tab):

```
% vsim work.top
```

7. If this is a post-synthesis simulation or if any Xilinx core macros are instantiated in your Verilog source code, you must compile the simulation libraries before simulating. Once the libraries are compiled, use the `-L` flag to reference the compiled libraries (GUI: add the library to reference in the "Libraries" tab).

You can open the Wave window, the Signal window, and the Workspace window from the main GUI by going to the View menu.

In the Workspace window, you can expand the module's hierarchy. The signals in the Signal window will correspond to the level selected in the Workspace window. Expanding and selecting a level in the main ModelSim window Workspace "sim" tab has the same effect. The signals can then be dragged and dropped from the Signals window into the Wave window for viewing.

If you are debugging, you will probably wish to use the same set of signals every time you simulate this module. You can save the format of the signals, radix, dividers, and labels by selecting File → Save Format in the Wave window. This will save the format (not the simulation data) to a `.do` file. Sometimes you may find it useful to modify the `.do` file by hand instead of manipulating signal names from the Wave window GUI.

Once the signals are in the Wave window, you can Restart the simulation by typing "restart -force". You can then run the testbench by clicking on the Run -All button on the Wave window toolbar. Alternately, you could type "run -all" at the ModelSim command prompt in the main ModelSim window to run the testbench.

As you make modifications to your HDL during debugging, you will have to re-compile it within ModelSim before re-simulating. Note that, for simulation purposes, re-synthesizing your HDL in the Xilinx tools will have no impact besides helping to find syntax errors.

New to ModelSim 6.2e

The optimizer, VOPT, is now run by default when you launch VSIM to simulate your designs. This has the benefit of making your simulations run considerably faster, with the side-effect of making it impossible to monitor internal signals by default. You can prevent VOPT from being invoked automatically by including `-novopt` in your VCOM, VLOG, and VSIM invokations. Alternatively, you can enable optimizatino with increased symbol visibility for a smaller performance penalty by using the `+acc` flag to VOPT (similar to the `-g` flag for GCC). For instance, when you launch VSIM, you can use the command `vsim -voptargs="+acc"` to enable full debug access to your design.

A Simple Example

As a basic example, consider the simulation of a full-adder.

1. Launch the ModelSim GUI.
2. Change directory in ModelSim to `lab7/full_adder/`.
3. From the ModelSim command window, type the following command to create a ModelSim working directory called "work".

```
% vlib work
```

4. Compile the `full_adder.v` Verilog file.

```
% vlog full_adder.v
```

5. Start the simulation on the top-level entity (`full_adder`).

```
% vsim work.full_adder
```

Note that ModelSim informs you that VOPT is being run:

```
# ** Note: (vsim-3812) Design is being optimized...
```

6. Add the Wave and Workspace windows to the GUI.

```
% view wave
% view workspace
```

Note: For earlier of versions Modelsim, the workspace window was called the structure window. Hence, the above commands would have to be replaced as follows:

```
% view wave
% view structure
```

7. Position the Wave and Worksapce windows such that they are both visible on the screen. Drag the `full_adder` block from the Workspace window onto the leftmost column of the Wave window. The following five signals should be added to the leftmost column of the Workspace window.

```
/full_adder/a
/full_adder/b
/full_adder/cin
/full_adder/sum
/full_adder/cout
```

Note that the Visibility column in the Workspace window shows that the `full_adder` block was optimized with `+acc=<none>`.

8. Return to the main Modelsim GUI window. Type the following commands to "force" the inputs to set values.

```
% force a 0
% force b 1
% force cin 1
```

9. Run the simulation for 1 microsecond.

```
% run 1 us
```

10. Return to the Wave window. You should see the waveforms resulting from the simulation.
11. You may repeat the above process of setting the inputs, running the simulation and viewing the waveform window. Note that you may have to select the "Zoom Full" button to zoom out completely.
12. When you are done simulating, you can quit the simulator portion of ModelSim with the `quit -sim` command.

Automating the Modelsim Simulation Process

GUI-based commands quickly become very tedious and time-consuming. Instead, a more automated approach is to perform the simulation automatically using scripts. We discuss two methods of automated script-driven simulation:

- Simulation and input generation using a collection of `.do` scripts
- Simulation using a `.do` script and input generation using a behavioral Verilog testbench

Both methods have their advantages. The reader may determine which method is a better fit to their needs.

Simulation and Input Generation using a Collection of `.do` Scripts

This method is best shown by means of an example.

1. The `booth_mult_gizmo` directory is a slight variant of the design found on the Digital Design Example page, which uses symbolic links. This version for Module m07 does not rely on symbolic links in the `lab7/booth_mult_gizmo/sim/` directory, but uses the `compile.do` script to access the correct version of the source HDL file.

2. View the `readme` in the base directory to get a feel for the directory structure. In summary, the Verilog code in the `lab7/booth_mult_gizmo/rtl/` directory implements a simple Booth-encoded multiplier. Inside the `lab7/booth_mult_gizmo/sim/` directory are two subdirectories, each of which simulates a different hierarchy of the design.

3. Traverse to the `lab7/booth_mult_gizmo/sim/my_top/` directory. Again, view the `readme` documentation. Also, review all scripts with a `.do` extension. These are the scripts that are used to automate the simulation process. Pay particular attention to `compile.do`, `init.do`, `setclk.do`, and `mults.do`.

   - `compile.do` compiles the verilog source files to the work directory.
   - `init.do` issues the vsim command to start simulation. It also calls three other scripts — `waves.do`, `setclk.do`, and `busreset.do` — for setting the Wave window and asserting design inputs into a known state.
   - `setclk.do` sets the input clock. Rather than asserting the clock after every 50 nanoseconds, the two "force" commands in this file automatically assert the clock to a 100 nanosecond period with a 50% duty cycle.
   - `mults.do` repeatedly asserts the design inputs, then runs the simulation. The syntax of these commands is identical to those used in the full-adder simulation example. The process is repeated several times to verify full functionality of the multiplier.

4. If you haven't already, launch Modelsim and traverse to the `lab7/booth_mult_gizmo/sim/my_top/` directory. Issue the following three commands.

   ```
   % do compile.do
   % do init.do
   % do mults.do
   ```

5. View the waveform window to see the simulation results.

Simulation and Test Vector Generation Using a `.do` Script and Behavioral Verilog File

Again, an example will be used to show this method of simulation.

1. Change to the `lab7/fibonacci/rtl/` directory and look through its contents. This contains a simple implementation of a Fibonacci sequence generator.

2. Traverse to the `lab7/fibonacci/sim/` directory. View the `compile.do` script. This script is very similar to the `compile.do` script in the previous example.

3. View the `fibonacci_tb.v` Verilog file. This contains all test vectors and other information necessary for simulating the Fibonacci design. Observe the following:

   - The module is a top-level testbench. Therefore, it has no inputs or outputs.
   - Several registers and wires are declared. In fact, registers are used to generate inputs to the design, while wires are used to view the outputs.

- The `fibonacci` block is instantiated in the testbench.
- The following statement automatically asserts the input clock to a 100 nanosecond period with 50% duty cycle.

```
always #(‘top_clk_per/2) top_clk = ~top_clk;
```

- We assert inputs inside the “initial begin” block. After assigning initial values to the inputs, the statement #800 represents a 800 nanosecond delay. (Note: this statement is purely behavioral and definately not synthesizable!). We can use statements such as this to assert the inputs to different values at different times in the simulation.

4. If you haven't already, launch ModelSim and traverse to the `lab7/fibonacci/sim/` directory. Execute the `compile.do` script and view the waveform window to see the simulation results.

EDK

Several simulation tools are also available in EDK. If your design includes a MicroBlaze soft processor, you may use the EDK simulation tools to generate a model that includes the MicroBlaze that will run your software in simulation.

1. In EDK, open the design to be simulated.
2. Before generating the system level simulation, ensure that the software on the generated system will work in simulation. The software to be used in simulation should be in “Executable” mode rather than using the “XmdStub” such that its execution begins immediately. It should be set to initialize the BRAMs as well.
3. The simulation does not include models of the hardware connected to the FPGA. For example, if your software makes calls to `printf()` that writes to the UART, you will only see the TX pin of the UART switching as the characters are transmitted. A simulation model that translates those signals to characters on the terminal is required to see some actual printing.
4. The simulation libraries must be compiled before a simulation can be performed. To do so, first create your own `modelsim.ini` file by running `vmap -c` in a directory you own. Then, set the `MODELSIM` environment variable to the location of that file (make sure your path doesn't contain any spaces). Again, in a directory you own, run:

```
compxlib -s mti_se -arch virtex2 -arch virtex2p -dir . -w -lib unisim \
         -lib simprim -lib xilinxcorelib -lib smartmodel -smartmodel_setup
compedklib -o . -X . -exclude deprecated
```

   This will create the basic Xilinx simulation libraries (UNISIM, SIMPRIM, and XilinxCoreLib), update your `modelsim.ini` to reference these libraries, and build the EDK simulation libraries, all in the current directory (*i.e.*, `.`).

   Note: there is a GUI in XPS for compiling all the required simulation libraries; however, I was unable to get it to work.

   Details on generating simulations libraries are provided in Section 6 — Simulating Your Design of the Synthesis and Simulation Design Guide, accessible in the from the Software Manuals section of the ISE documentation; and in Section 3 — Simulation Model Generator (Simgen) of the Embedded System Tools Reference Manual, accessible in the from the Software Manuals section of the EDK documentation.

5. Set the simulation options via Project Options which can be found in the Project menu. In the HDL and Simulation tab, select Verilog as the HDL and ModelSim as the Simulator Compile Script. Set up the correct paths to the Xilinx and EDK libraries that you just compiled.

6. Generate the simulation model wrappers by running the menu item Tools → Sim Model Generation. Running the menu item Hardware Simulation will open up ModelSim in the correct environment to begin the simulation.

   Note: I get an error dialog saying that EDK can't find the compiled librarires even though I have the path set correctly. I just click OK and leave the path as-is and everything seems to work fine.

7. Once ModelSim is open, type `c` to execute the generated macro that compiles all the necessary HDL files.

8. Type `vopt +acc=v system system_conf glbl -o system_debug` to optimize the design with debugging visibility and then type `vsim -t ps system_debug` to start the simulation. `system` is the top-level module, `system_conf` is the module used to initialize the BRAMs with your software and `glbl` contains required global signals.

9. Type `w` to setup the Wave window with the system's primary signals.

10. Setup the clock stimulus by typing `clk` and then reset the system by typing `rst`.

11. The simulation can then be run as long as desired. For instance, to run for 400 ns, type `run 400 ns`.

ISE

Check that your ISE preferences have been set to include the path to ModelSim. Launch Project Navigator and go to Edit → Preferences. Select the Integrated Tools tab. If the path has not yet been specified, browse to the location of the ModelSim executable (modelsim.exe) for the Model Tech Simulator entry.

1. Code up your design in HDL. You might include Xilinx primitives, CoreGen modules, etc.

2. Create a testbench for the module/sub-module that you want to simulate. The testbench is an HDL file that drives the inputs at specified times and is essentially your "test case". In ISE you can use HDL bencher to generate one easily.

   (a) Go to Project → New Source.

   (b) Select "Testbench Waveform" and enter a name for it in the File Name box. For easy reference, a good name is `test_<name-of-your-module-to-be-tested>`.

   (c) Click Next.

   (d) Select the source file that corresponds to the top-level file for the module you want to test.

   (e) Click Next. Click Finish.

   (f) An Inintialize Timing dialog box will pop up.

   (g) In the Design Type box specify if you have one or more clocks. If one clock, make sure the correct signal is chosen.

   (h) If you have just one clock, specify the clock timing.

   (i) Click OK if one clock, otherwise click Next.

   (j) If you have multiple clocks, select the clocks from the list of signals. Click Next. Then associate every remaining signal with a clock. Click Next. Specify the Clock Timing for each clock. Click Finish.

   (k) In HDL bencher, specify your input pattern for all the input signals (blue).

   (l) Scroll horizontally in time to where you want your test to end. Right click on the column and select Set End of Testbench.

   (m) Save your `.tbw` file. This will generate a `.tfw` file for Verilog or a `.vhw` file for VHDL.

3. Select Behavioral Simulation from the Sources for: drop-down list. Your `.tbw` should appear in the Sources view. Select it, then expand the ModelSim Simulator toolbox in the Processes view. You will find the first 2 items useful: Simulate Behavioral Model and Generate Expected Simulation Results.

4. Right click on Simulate Behavioral Model and select Run. This will generate an `.fdo` file and a `.udo` file and launch ModelSim. In ModelSim, it will run the `.fdo` script. Open the `.fdo` script to see the ModelSim commands that are called (in sequence).

Using ModelSim at home

You can download ModelSim Xilinx Edition-III, the free version of ModelSim for Xilinx. Naturally, as a free product, it has reduced functionality. In particular, simulations run using MXE will be orders of magnitude slower than simulations run on ModelSim SE (all other things being equal).

Alternatively, you can download the full version of ModelSim SE directly from ModelTech. To use the full version, you will need access to a license for the software. You can use the same license server as you use on campus, but if you are connecting from outside the campus network, you will need to tunnel the requisite ports via SSH in order to get around the campus firewall. As a TA if you're unsure how to do this.

Troubleshooting:

In the ISE Sources in Project window, right click on the device (*e.g.*, xc2v2000-4ff896) and select Properties. Make sure that your project properties list the correct HDL for the generated simulation language. If the top-level file of your design is in Verilog, you should specify to generate Verilog. If it is in VHDL, you should specify VHDL. A mismatch will cause weird errors to be generated.

Handy Xilinx Answer Records for ModelSim

2561 How do I compile the Xilinx Simulation Libraries for ModelSim?

10176 "Error: Cannot open library unisim at unisim." (VHDL, Verilog)

12491 How do I save the position of the ModelSim windows?

18016 Does MXE support mixed language VHDL and Verilog simulations? (Note that ModelSim SE/PE do)

18226 Advanced tips for using ModelSim with Project Navigator

Look At Next

Module m09: Using and Modelling OPB Interfaces Module m12: Using ChipScope