Version of Jan/22/05

1 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.

1.1 Source Files for Examples

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

2 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 ugsparc workstations, set up the necessary environment variables by doing:

   ```
   % source /cad2/Xilinx/Modeltech-SOURCEME.csh
   ```

   Then invoke Modelsim:

   ```
   % 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 any Xilinx core macros are instantiated in your Verilog source code, use the following command to simulate your design with the Xilinx Verilog Core Library (GUI: add the library to reference in the "Libraries" tab):

```
# Note that the following is a single command line.  It is broken up
# over multiple lines to format correctly in this document.  The '\'
# character is the csh line continuation character
% vsim -L \
      /cad2/Xilinx/Modeltech_6.0/modeltech/xilinx_libs/xilinx_libs/XilinxCoreLib_ver \
           <working_directory>.<topmost_module_name>
```

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.

2.1 A Simple Example

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

1. Launch the ModelSim GUI.

2. Traverse to the directory that contains full_adder.v

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
```

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
```

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.

3 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.

3.1 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* (http://www.eecg.toronto.edu/~pc/courses/hdl/designexample/) page, which uses symbolic links. This version for Module m07 does not rely on symbolic links in the 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 /rtl directory implements a simple booth-encoded multiplier. Inside the /sim directory are two subdirectories, each of which simulates a different hierarchy of the design.

3. Traverse to the /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 /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.

3.2 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. Go to the fibonacci example.

2. Traverse to the /rtl directory and view its contents. This contains a simple implementation of a fibonacci generator.

3. Traverse to the /sim directory. View the compile.do script. This script is very similar to the compile.do script in the previous example.

4. 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.

5. If you haven't already, launch Modelsim and traverse to the /sim directory. Execute the compile.do script and view the waveform window to see the simulation results.

4 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. Set the simulation options via *Project Options* which can be found in the *Options* 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. The Xilinx libraries usually come installed with ISE and are located in *Xilinx/verilog/mti_se*. The EDK library will need to be generated if they have not been already. Generating the libraries is addressed in the *Platform Studio User Guide*.

5. 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.

6. Once ModelSim is open, type *do system.do* to execute the generated script that compiles all the necessary HDL files.

7. Type *vsim -t ps work.system work.system_conf work.glbl* to start the simulation. *work.system* is the top-level module, *work.system_conf* is the module used to initialize the BRAMs with your software and *work.glbl* contains required global signals. You may need to add options such as "-L XILINXCORE-LIB_VER" if you use CoreGen components.

8. You will need to drive the clock and the reset signal of the system yourself. This can be done with the following commands which can be placed in a script for convenience.

```
force -freeze sim:/system/sys_rst 0 0
force -freeze sim:/system/sys_clk 1 0, 0 {50 ns} -r 100000
run 100 ns
force -freeze sim:/system/sys_rst 1 0
```

9. The simulation can then be run as long as desired.

5 ISE

Check that your ISE preferences have been set to include the path to ModelSim. 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.

3. Go to Project=> New Source.

4. 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>.

5. Click Next.

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

7. Click Next. Click Finish.

8. An Inintialize Timing dialog box will pop up.

9. In the Design Type box specify if you have 1 or more clocks. If 1 clock, make sure the correct signal is chosen.

10. If you have just 1 clock, specify the clock timing.

11. Click OK if 1 clock, otherwise click Next.

12. 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.

13. In HDL bencher, specify your input pattern for all the input signals (blue). (Make sure you are using the correct radix! The radix toolbar buttons are 16=> Hex, 10=> Decimal, 2=> Binary.)

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

15. Save your .tbw file. This will generate a .tfw file for Verilog or a .vhw file for VHDL.

16. In the Sources in Project view, click on the .tbw file you just created. In the Processes for Source view, expand the ModelSim Simulator toolbox. You will find the first 2 items useful: Simulate Behavioral Model and Generate Expected Simulation Results.

17. 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).

6 Using ModelSim at home

You can download MXE II, the free version of ModelSim for Xilinx from

http://support.xilinx.com/xlnx/xil_tt_product.jsp?sProduct=MXE$+$II

Be sure to download it for the language that you intend to use to code your modules (either VHDL or Verilog).

To do mixed language VHDL/Verilog simulations, you will have to use the full version of ModelSim on the UofT PCs or the ugsparcs.

6.1 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.

7 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 9: Using and Modelling OPB Interfaces
Module 12: Using ChipScope