University of Toronto
ECE532 Digital Hardware
Module m07: Using ModelSim to Simulate Your Designs

Version for ModelSim SE 10.1c as of January 15, 2013

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 external DDR RAMs used on the Atlys board. This code cannot be synthesized, but it is intended to give a true reflection of the behaviour of the memory 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.

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 Start Menu. If you are using a Linux machine, set the necessary paths and environment variables and call `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 have compiled the simulation libraries before simulating. This has already been done on the lab workstations; on private installations, it should have been explained as part of the installation process. The compiled libraries are either automatically linked in if their paths are listed in `modelsim.ini` (in that case you can see them listed in the *Library* tab), or you can reference them with the `-L` flag when using the `vsim` instruction.

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.

ModelSim optimizer

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 optimization 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. 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.
2. 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.
3. 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
```

4. 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

Simulation of a complete MicroBlaze system can be initiated from EDK.

1. These steps assume that after installation of both the ISE Design Suite and Modelsim, you have generated all EDK simulation libraries by either choosing Xilinx Design Tools → ISE Design Suite 14.2 → EDK → Tools → Compile Simulation Libraries in the Windows Start Menu or calling `compedklib` on the command line. Make sure to compile to a path without spaces.

2. In EDK, use the Base System Builder wizard to build a simple system (make sure it does not make use of external DRAM memory, and make the local BRAM at least 32KB large so that all driver code fits in).

3. Export the generated system to SDK (you can uncheck the bitstream/BMM export, which saves you the system implementations steps that are unnecessary for behavioral simulation).

4. In SDK, start a "New Xilinx C Project" and pick the "Peripheral Tests" template. Let SDK also create a Board Support Package for it. SDK will automatically build an ELF file for you.

5. Go back to XPS and select the menu entry Project → Select Elf File...
   Under "Choose Simulation Elf File", click the radio button beside the empty text field. Then press the folder button to the right and find the generated ELF file (if you have not changed any names, you should find it under `.../peripheral_tests_0/Debug/peripheral_tests_0.elf`). If you plan to later implement the system for the board, you can also copy this path to the "Choose Implementation Elf File" setting with the rightmost button. Press "OK".

6. Select the menu entry Project → Project Options and select the "Design Flow" tab. Pick your preferred Hardware Design Language (likely Verilog), in which the testbench and the system top-level file will be generated for simulation. Check "Generate test bench template". Press "OK".

7. Go to Edit → Preferences and select the category "Simulation". Check if the path to the compiled simulation libraries has been correctly set.

8. Click Simulation → Launch HDL Simulator...
   This will generate or update the system simulation files and then start Modelsim. In Modelsim, type `pwd` on the transcript window console. You can see that XPS generated the simulation files in a subdirectory `.../simulation/behavioral`. With the "File" menu, open the files `system.v`, `system_tb.v` and `system_setup.do`. The `system.v` is the top-level file of the MicroBlaze system; browsing through it, you can see the same instantiations of system components as in the XPS "System Assembly View". In `system_tb.v`, the whole system is instantiated and stimulated with a reset and a clock signal. Users can add other input stimuli that will not be overwritten by future system updates. The Tcl script `system_setup.do` defines several one-letter command aliases that make simulation steps more convenient. Feel free to explore the other `*.do` files that it references.

9. Enter `c` to compile the system. Note how the "Library" tab has filled with EDK peripheral components now. Enter `s` to start the simulation. Besides the "Library" tab there is now a "sim" tab which shows a tree hierarchy of all simulated components, again similar to the "System Assembly View", but now with system_tb at the top.

10. If you enter `w`, the Wave window is opened and populated with all signals visible in the system top-level. Start the actual simulation operation with `run 25us`. Click into the Wave windows and press `F` to see the whole simulation time. Clearly the MicroBlaze and its buses get quite busy after a brief initialization phase.

11. Enter `run 500us` on the transcript console and press "F" inside the Wave window again. If you have included an RS232 UART and kept `print` or `printf` instructions in the C code, these outputs are written out slowly over the terminal interface. Calculate how long a single character (10 bit including start and stop bit) takes at a baud rate of 115200 kbit per second. It becomes clear that any simulation that includes such outputs will be quite slow, and that any other hardware interaction that is to be observed in simulation will be held up by the long RS232 gaps. Therefore, it is recommended to comment `printf` instructions out for simulation purposes.

12. Play around with the system simulation at liberty. Note that if you change the code in SDK, you will have to update simulation files with "Generate Simulation HDL files" or a new "Launch HDL Simulator" in XPS, and then re-compile the system in Modelsim.

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?

18226 Advanced tips for using ModelSim with Project Navigator

Look At Next

Module m12: Using ChipScope