DESIGN VERIFICATION Exercise 2:
Introduction to the ModelSim Simulator

This exercise introduces you to the ModelSim simulator. It gives you the opportunity to investigate some of the functionality of ModelSim on two simple example designs. ModelSim is a simulator that can be used to verify the calculator design for the first DESIGN VERIFICATION assignment, so it is worth you investing some time into getting to know this simulator better. The exercises are very simple and are designed to familiarize you with the use of ModelSim and some of its features.

All files referred to in this exercise are available on the unit web pages. This sheet should be sufficient to guide you through the exercise. If there are problems, please let me know. Have fun!

Kerstin

Getting Started

1. Create a directory in your home directory for this exercise, and move into it, e.g.:

   ```
   mkdir DV_Ex1
   cd DV_Ex1
   ```

   Now copy all (Verilog source) files mux421*.v from the unit web pages into the current directory.

2. Make sure you include `/usr/local/modelsim/modeltech/bin` into your PATH (variable), you set the environment variable `MODEL_TECH` and you set the environment variable `LM_LICENSE_FILE` to point to the ModelSim license. For instance, if you use the bash shell, it is best to do this in your .bashrc file.

   ```
   export PATH=/usr/local/modelsim/modeltech/bin/:$PATH
   export MODEL_TECH=/usr/local/modelsim/modeltech/bin
   export LM_LICENSE_FILE=/usr/local/modelsim/modeltech/license.dat:$LM_LICENSE_FILE
   ```

   Remember that changes in the file .bashrc are only effective when you call bash next time.

3. Now start ModelSim.

   ```
   vsim &
   ```

   You should see two windows, one is a "Welcome to ModelSim" dialog and the other one is entitled "ModelSim SE ...". Complete the welcome dialog.

Familiarization with the Design(s) Under Verification and Testbench

1. The files `mux421_structural.v`, `mux421_dataflow.v`, `mux421_behavioural.v` contain the design of a 4-to-1 multiplexer coded in different styles. Familiarise yourself with the coding style used in each design.

2. The file `mux421_gen.v` contains Verilog code that generates 6 binary output signals which will be used as stimulus (input data) for our multiplexer designs.

3. The file `mux421_check.v` contains Verilog code that checks signals. The task `$monitor` continuously monitors the values of the variables or signals specified in the parameter list and displays all parameters in the list whenever the values of any one variable or signal changes. `$monitor` only needs to be invoked once. Only one monitor list can be active at a time. If there is more than one `$monitor` statement in your simulation, the last `$monitor` statement will be the active statement, the earlier ones will be overridden.

Monitoring is turned on by default at the beginning of the simulation and can be controlled during the simulation with the `$monitoron` and `$monitoroff` tasks.

Usage `$monitor` $(p1, p2, p3, ..., pn)$`;`

The parameters $p1, p2, p3, ..., pn$ can be variables, signal names, or quoted strings. NOTE: All Verilog system tasks appear in the form `$<keyword>`. You can find out more about Verilog system tasks such as `$display` and `$time` in the Verilog reference guide in your Evita_Verilog tutorial.

4. The file `mux421_example_testbench.v` contains *one* example testbench for the *structural* coding style description of the 4-to-1 multiplexer design. Familiarise yourself with the testbench code.

5. In addition, there are source files that contain a faulty version of the multiplexer, one for each coding style, and a file `mux421_faulty_testbench.v` which defines testbenches for each faulty version. Familiarize yourself with these testbenches. Notice that the StimulusGenerator and the Checker have been reused for each testbench - just the DUV is different.

Simulating Verilog Designs

1. To simulate a design with ModelSim, the Verilog files need to be compiled into a design library first. To do this, a design library has to be created (to hold the compilation results) and mapped to a physical library.

   In ModelSim, choose "New:Library" from the "File" menu. The "Create a New Library" window pops up. This window can be used to create a new library and a logical mapping to it, or to create a map to an existing library. The default library is `work`.

   Select "a new library and a logical mapping to it". If you don't want to use the default library, you can type a different library name, e.g. "my_work_lib", into the Library Name field, then select OK. This creates a subdirectory named "my_work_lib" - your design library - within the current directory.

   Note that the Library Name field specifies the logical name of the library which is mapped to the physical name. Observe that the new library is now shown in the "Library" pane on the left side of the ModelSim workspace area as an empty library.

   ModelSim can also be operated in a command-line mode from the Console window. Instead of using the ModelSim GUI to create the library and mapping, you could have just typed the following two commands:

   ```
   vlib my_work_lib
   vmap my_work_lib my_work_lib
   ```

   From now on, the command-line alternative will always be given right at the end of an exercise step.

2. After the working library has been created, the design files can be compiled. Choose "Compile" from the "Compile" menu. The "Compile HDL Source Files" window appears showing the current directory, the default current working library (to be compiled into - by default) and available source files. Make sure the working library is set to the one you've previously created. (The default is `work` - other libraries are available from the pull-down menu on the right.) The top-level file for compilation is the testbench file `mux421_example_testbench.v` - select and compile it. Finish by clicking "Done".

   Command-line alternative: `vlog -work my_work_lib mux421_example_testbench.v`

   Review the messages on the ModelSim Console window which show the compiled modules. Notice that `mux421_structural_testbench` is the top-level module. Expand the working library (by double clicking on the + sign in front of it) in the Library pane on the left of the ModelSim workspace area - it now contains four compiled modules. (Enlarge the "Name" column to see their full names.)

3. Once the compilation of source files has been successfully completed, you need to load the design top-level unit for simulation. From the "Simulate" menu choose "Start Simulation. The "Start Simulation"

window appears. Choose the "Design" tab which shows the currently available libraries. Expand your working library. You should see four units available for simulation. (You might need to enlarge the "Name" column again.) Select the `mux421_structural_testbench` unit (the top-level unit), disable optimizations (to gain visibility) and press "OK". The "Simulate" window will close and information on the loading phase will be printed to the Console window. Review the messages printed to the Console.

Command-line alternative: `vsim my_work_lib.mux421_structural_testbench`

Notice also that the ModelSim workspace area has now changed. The "sim" pane displays the design units loaded for simulation (provided you disabled optimizations as described above). It shows the hierarchical structure of the design and the instantiations. By default, only the top level of the hierarchy is expanded.

4. After the design has been loaded, you can see the signals of the design in the "Objects" window.

   Command-line alternative: `view signals`

   Select the root of the design in the "sim" pane. The set of signals visible in the "Objects" window is automatically adjusted to the selection in the "sim" pane.

   Compare the visible objects with the wire declaration in the module `mux421_structural_testbench` contained in the file `mux421_example_testbench.v` - all wires should be observable. Check this for the sub-units of the design.

5. Add all signals observable in the top-level unit `mux421_structural_testbench` to the waveform viewer. In the "Objects window use "Select All" from the "Edit" menu. Then right-click on the selected signals; choose "Add to Wave" and "Selected Signals". This will open the waveform viewer window. You can also select specific signals for waveform viewing by dragging and dropping between the "Objects" and the "Wave" windows.

6. After the desired signals have been added to the "Wave" browser window, the simulation can be run. To run all test vectors during one simulation step you should use "Run" and "Run-All" from the "Simulate" menu.

   Command-line alternative: `run -all`

   Observe the messages printed to the Console. Also, observe the waveform changes - you might need to click the "Zoom Full" button from the "Wave" window toolbar. In addition, a source code window will pop up indicating where the simulation stopped - you can ignore this for now.

7. To run the simulation again, first restart it by selecting "Run" and "Restart" from the "Simulate" menu. Observe that all signals are now set back in the "Wave" window.

   Command-line alternative: `restart -f`

8. Now run the simulation step-wise by repeatedly using "Run" and "Run-Next" from the "Simulate" menu.

   Command-line alternative: `run -next`

9. Experiment with the simulator and waveform viewer in different settings and formats. Run the cursor over a waveform and observe the pop-up comments.

Recording and comparing simulation results

When the waveform viewer is active during simulation, the results of each simulation run are automatically saved into a file called `vsim.wlf` in the current directory. (.wlf stands for wave log format.) This file is overwritten when a new simulation is run in the same directory. You can specify a different name to save simulation results by selecting "Datasets" and "sim" from the "File" menu in the "Wave" window. The currently active simulation is always prefixed by "sim". Note that a simulation session needs to be finished with `quit -sim` in order to produce a valid .wlf file.

1. Start ModelSim.
2. Create a new working library for this part of the exercise.
3. Compile `mux421_example_testbench.v` - this should add 4 modules to your new working library. Check this in the Library pane.
4. Simulate `mux421_structural_testbench`: Double-click on `mux421_structural_testbench` in the Library pane to load this module for simulation. Right-click on `mux421_structural_testbench` in the sim pane, then "Add To Wave". Now run the simulation with `run -all`.
5. Save the simulation result: Select the dataset "sim" from "File" "Datasets". Select "Save As"; call this dataset "structural".

   Command line: `dataset save sim structural.wlf`
6. Quit the simulation `quit -sim`.

   The file `structural.wlf` now contains the golden reference dataset for the structural version of our multiplexer design.
7. Now compile `mux421_faulty_testbench.v` - this should add 6 more modules to your working library, giving a total of 10 modules in the library. Which modules are new?
8. Simulate `mux421_faulty_structural_testbench`.
9. Save the simulation result of the faulty (structural) design. Call the dataset "faulty_structural".
10. Quit this simulation.
11. Compare the simulation results by comparing the two waveforms:
    - Use "File - Datasets - Open" to open the two `.wlf` files just created. Two datasets appear, one called "structural" and one "faulty_structural".
    - In the "Wave" tab, select "Tools - Waveform Compare - Start Comparison". Select "structural" as the Reference Dataset and specify "faulty_structural" as the Test Dataset. Press "OK".
    - Now you need to specify the signals to be compared. Use "Tools - Waveform Compare - Add - Compare by Region" to compare all signals. Select `/mux421_structural_testbench` as the Reference Region. Specify `/mux421_faulty_structural_testbench` as the Test Region. Leave "Compare Signals of Type" unchanged - this will compare signals of all types. Press "OK". The Console should report "Created 7 comparisons." The "Wave" window contains the comparisons created - investigate these.
    - Use "Tools - Waveform Compare - Run Comparison" to run the comparison. The Console should report "Found 1 difference." The difference is marked in the "Wave" window - investigate.
    - You can use "Tools - Waveform Compare - Differences - Show" to get a report of the differences on the Console.
    - When you have experimented enough, use "Tools - Waveform Compare - End Comparison" to finish the comparison.

Experimenting with Testbenches

Write your own testbenches to verify the behaviour of the dataflow and behavioural versions of the 4-to-1 multiplexer. Initially, you should only have to replace the DUV in the testbench file. Later you might want to add extra code to the stimulus generator and the checker modules.

For each design, save the waveforms in suitably named files. Compare the waveforms you've created - when comparing waveforms, specify the regions to be compared and make sure corresponding signals have the same

names. If you use the same input stimulus, are the three (non-faulty) designs producing the same output? The comparison results are printed to the Console window of ModelSim, if differences are detected, the "Wave" window highlights the differences.

Debugging faulty Multiplexers

The three files `mux421_*_faulty.v` contain faulty 4-to-1 multiplexer designs. For each design, build a testbench (or use `mux421_faulty_testbench.v`), simulate it, collect a waveform and compare it to the waveform of the respective non-faulty design. You might want to investigate the source code to see where the bug is - use `diff` to compare the source file to the corresponding non-faulty design source file if the bug is not immediately obvious to you.

A different Multiplexer design: Integer signals

The file `mux_int.v` contains the design of a multiplexer that selects between two numeric inputs and also outputs a response indicating valid data. Which coding style has been used: structural, dataflow or behavioural?

The file `mux_int_test.v` contains a badly designed (i.e. not properly modularised) testbench for the `mux_int` module. Run the testbench, i.e. simulate it. Observe the waveforms - try to understand the code in `mux_int_test.v`. Familiarise yourself with the `always` statement and the `$display` task. In the "Wave" window you can change the Radix of a signal, e.g. to decimal by right-clicking on the signal and selecting the "Radix" option. Notice how the 32-bit wide signals are bundled in the waveform - expand some to see the individual bits.

Redesign the testbench so that it contains one module that generates stimulus and one module that acts as a checker. Experiment with different stimulus data, run the simulation only for/until a set time, or introduce a bug into the multiplexer source code and try to identify it from the waveform or the testbench output.

More on ModelSim

The "Help" menu offers both PDF and HTML documentation on ModelSim under "SE Documentation". It is useful to work through the **ModelSim SE Tutorial**, using the files provided in `/usr/local/modelsim/modeltech/examples/tutorials/verilog/`.