# COMS40115 Exercise 2: Using the Riviera Simulator (Week 2)

This exercise introduces you to the Riviera simulator. It gives you the opportunity to investigate some of the functionality of Riviera on two example designs - Riviera will be used to debug a calculator design for the first COMS40115 assignment, so it is worth you investing some time into getting to know your debugging tool better.

All files referred to in this exercise are available on the COMS40115 web site. 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:

   ```
   mkdir DV_Ex1
   cd DV_Ex1
   ```

   Create two further directories, one for the library files we create today, and one for wave forms.

   ```
   mkdir lib
   mkdir wav
   ```

   Create one further directory for the source files we use today, and move into it.

   ```
   mkdir src
   cd src
   ```

   Now copy all (Verilog source) files mux421*.v from the COMS40115 web site into the current directory.

2. Starting Riviera:

   Make sure you include `/usr/local/vsimsa/` into your PATH (variable). For instance, if you use the bash shell, it is best to do this in your .bashrc file.

   ```
   export PATH=/usr/local/vsimsa/:$PATH
   ```

   Now start Riviera

   ```
   rungui &
   ```

   You should see one window entitled ”Riviera-...”.

## Simulating Verilog Designs

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.

   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.

   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 systems 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.

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

2. To simulate a design, the Verilog files need to be compiled first. To do this, a design library has to be created and mapped by the user manually.

   In Riviera, choose the Create command from the Library menu. The Create Library window will be displayed. This window can be used to create a library and map a logical library name to a physical library file, create a library only (without mapping), or map a logical library name to an existing library file.

   Select the Library & Mapping radio button. Click the Browse button, select the directory lib, call your library `mux421_example_testbench.lib`, press OK, press OK again in the Create Library window.

   Note that the Library Name field specifies the logical name of the `mux421_example_testbench.lib` library. Observe that the current work library is now `mux421_example_testbench` which is displayed in the library combo box in the main toolbar.

3. After the working library has been prepared, the design files can be compiled. Choose the Compile command from the Compilation menu. The Compile Files window appears showing the current directory and available source files. Select the `src` directory. The top-level file for compilation is the testbench file `mux421_example_testbench.v` - select and compile it.

4. After the compilation of source files has been successfully completed, you need to specify the design top-level unit and initialise simulation. From the Riviera main menu choose Simulation and choose the Initialise Simulation command. You should see four modules (unit names). The one with the red T index `mux421_structural_testbench` is marked as a top-level module. Highlight the `mux421_structural_testbench` top module and press the OK button to initialise simulation. The Initialise Simulation dialog will close and information on the initialisation phase will be printed to the Console window. Review the messages printed to the Console. Pay attention to messages about memory allocation, used libraries, simulation resolution, etc.

5. After the simulation has been initialised, you can display the elaborated structure of the project. To view such a structure, open the Structure Browser window. The Design Structure Browser window and other debugging tools are available from the View menu. The panes in the Structure Browser can be tiled either horizontally or vertically - use the View option to change the panes to tile vertically.

   Select the root of the design in the left pane of the Structure Browser. The set of objects visible in the right pane is automatically adjusted to the selection in the left 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.

6. Select (click on) and then right click on one of the signals, choose Add to Waveform. This should open the Waveform Viewer window. Add all signals to the waveform viewer.

7. After the desired signals have been specified and added to the waveform window, the simulation can be run. To run all test vectors during one simulation step you should use the Run command from the Simulation menu.

Observe the messages printed to the Console. Also, observe the waveform changes - you might need to click the "Zoom to Fit" button from the Waveform Viewer toolbar.

To save the waveform select Save as from the File menu. Select the `wav` directory and give your waveform file a suitable name.

8. To run the simulation again, first restart it by selecting Restart Simulation from the Simulation menu. Observe that all signals are now set to x in the Waveform Viewer window.

## Experimenting with Testbenches

Write your own testbenches to verify the behaviour of the dataflow and behavioural versions of the 4-to-1 multiplexer. For each design, save the waveforms in suitably named files. Use the Compile Waveforms option from the Waveform Viewer window to compare the waveforms you've created - when comparing waveforms, make sure corresponding signals have the same names. If you use the same input stimulus, are the three designs producing the same output? The comparison results are printed to the Console window of Riviera, if differences are detected, the Waveform Viewer window highlights the differences - this will become clearer when you simulate the faulty designs of the next section.

## Debugging a faulty Multiplexer

The three files `mux421_*_faulty.v` contain faulty 4-to-1 multiplexer designs. For each design, build a testbench, simulate it, collect a waveform and compare it to the waveform of the 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.

## A different Multiplexer

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 - compare the waveforms to the code in `mux_int_test.v`. Familiarise yourself with the `always` statement and the `$display` task. In the Waveform Viewer window you can change the format of the signal values, e.g. to decimal by right-clicking on the signal and selecting the Properties option from the menu.

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 Riviera

The Riviera Help Topics option in the Help menu of the Riviera window opens up the on-line help. You might want to work through the Tutorial section, using the files provided in `/usr/local/vsimsa/projects/verilog/`.