ECE352 – Simulating your processor with ModelSim

1. Open your processor project in Quartus II and go to the *Files* tab. Double-click on *memory.bdf* to open the instruction memory schematic. The simulator does not understand schematic files, so we have to automatically generate a Verilog file from the schematic.

2. Choose the menu entry *File->Create/Update->Create HDL Design File from Current File…* Make sure that the export file name is *memory.v* and that Verilog HDL is selected. Click OK. Leave Quartus.
   (Note: You will later in the project exchange the memory schematic with a new one. Make sure to re-export a new *memory.v* after you did that.)

3. Start ModelSim Altera Starter Edition from the Start Menu

4. *File->Change Directory* to your project directory, probably named *processor_v1_verilog*

5. On the ModelSim command line, you can enter file systems commands like **cd** and **dir** (Unix commands like **ls** and **pwd** also work). Check with **dir** if the expected project files are in the folder.

6. There are three scripts in the project folder, **compile.do**, **simulate.do** and **wave.do**.
   Type **cat compile.do** to see what the first script does:
   a. **vlib work** creates a local folder called *work* in which all the compiled simulation files are written. The folder functions as a working library of compiled components.
   b. The **vlog** command compiles each Verilog file in the project into the working library.

7. Type **do compile.do** to compiles all the components in the project. If you open the tree called "work" in the *library* window in the top left, you can see all the compiled Verilog modules.

8. Type **cat simulate.do** to see what's in the simulation script:
    a. **vsim** starts the simulation. It's **–L** parameters are the built-in Altera component libraries that the simulation uses (e.g. for the instruction RAM block that we are using). The last parameter is the top-level module that we are simulating. Note that this is not the top-level module of the Quartus project (*multicycle*), but instead the testbench module *multicycle_tb*, which instatiates our whole processor as a submodule and generates simulated inputs like the clock and the reset. Run **cat multicycle_tb.v** to see what the testbench looks like.
    b. **mem load** loads data from a *.mem file into the simulated instruction RAM. When you run a test program through our provided assembler, it produces a file *data.mif* which can be used to initialize the RAM in the Quartus project, and a *data.mif.mem* file which the simulator can read. They both hold the same data, just in different formats.

9. Type **do simulate.do** to run the simulation. You can see that the window makeup of the simulator changes. To the left is now a window called *sim* which shows the module hierarchy for the simulated system. If you click on a module, the window *Objects* to the right of the *sim* windows shows all the wires and regs that are in the selected module.

10. Type **do wave.do** to open a new window that shows you simulated signals. You can see signals like *clock* and *reset*. Vectors have a little plus sign beside your name with which you can see every bit separately. However, right now all the signals are indicated as 'x' or 'z' and now waveforms can be seen. This is because we are still at simulation time 0.

11. Type **run 1000 ns**. Now you can see the first microsecond of system simulation. Note how the testbench briefly asserts *reset* and then releases it, as the testbench code described. You can also see the clock oscillate.

12. The best thing about ModelSim is that you can look at internal signals that interest you. In the *sim* window, click on DUT (device-under-test, the name of the processor instance). In the *Objects* window, left-click *MEMwire*, drag it into the bottom of the wave window and release it. You cannot yet see anything because Modelsim did not log the signal data when it simulated. If you type **run 500 ns**, you can see how the signal develops between 1000 and 1500 ns. If you want to see the signal from the start, type **restart** and **run 1500 ns**. On the top of the window, click the magnifier icon with the plus sign a few times to a look at a shorter instant in time, so you can the single signal changes. The *MEMwire* value is still hard to read because it's displayed in binary. Right-click on *MEMwire* and pick *Radix->Hexadecimal*.

13. Type **cat wave.do** to finally see what the waveform window script looks like. We won't go through it in detail, but can see that it is a script that opens the window, adds all the waves we see and sets a few more display parameters. You can save your own window setting by going to the Menu *File->Save Format*. Save the settings as *wave1.do*. If you execute **do simulate.do** and **do wave1.do** again, you see that *MEMwire* is now included and you see the window in time that you had last selected.

14. You can now edit your Verilog files and simulate the modified system. A few <u>important</u> reminders:
    - After each Verilog change, you have to-recompile and re-start the simulation.
    - If you add an extra Verilog file, don't forget to include it in the compile script.
    - When you change the memory type in the first project steps, don't forget to re-generate a *memory.v* file
    - If you generate new memory files with the assembler, do not forget to change the **.mem* file name in the *simulate.do* file
    - If you get different results on the board than from the simulator, make sure that when you simulate with the *XY.mif.mem*, you use *XY.mif* to initialize the memory in Quartus.