# Quartus II Version 14.0 Tutorial

*Created September 10, 2014; Last Updated January 9, 2017*

This tutorial will walk you through the process of developing circuit designs within Quartus II, simulating with Modelsim, and downloading designs to the DE1-SoC board.

The steps we show you here will be used throughout the class, so take notes and refer back to the appropriate sections when you are working on future labs.

## [0] Installing Quartus II Software

Most of the designs in this class will be done through the Altera Quartus II software. This is preloaded on machines in the CSE department, and you are welcome to do all the work on these PCs. However, if you have a PC of your own that you would like to use, you can install the software there as well.

If you do not want to install Quartus on your own machine, skip to the next section.

1) **Download** the Quartus 14.0 software from the CSE369 website. You'll need both the Quartus software tarfile, and the Cyclone V *`.qdz`* file. Save these both to the same directory.
2) **Extract the tarfile** (using 7zip, WinRAR, etc.).
3) **Run the *`setup.bat`* file**. Make sure you select the "ModelSim-Altera Starter Edition" when running *`setup.bat`*. Install the optional components as well.
4) **Run Quartus II**. If it asks about installing devices, say yes (if it doesn't ask, then likely the files for the Altera devices were already installed). The Quartus II device files are found in the directory you downloaded the individual file into previously. Install the **Cyclone V** files.

## [1] File Setup for CSE369

For each lab in this class, we will create multiple files for your designs, for testing, and for downloading to the DE1-SoC board. To keep things sane, we suggest creating subdirectories for each lab within a class directory. So you might have a cse369labs directory, and create a lab1 subdirectory for Lab1.

Do not reuse the same directory for different labs, because you will want to refer back to a working design when you develop each new lab.

When you start each lab after Lab1, copy the previous directory over as the new directory so that you can reuse many of the files and the setup you did in previous labs.

If you are using the lab machines, put your work onto your Z: drive (shared across all machines). It should be the drive with your NetID on it. If you are using your own machine, you can store the files wherever you will remember to find them.

Download *`Lab1_files.zip`* from the course website and unzip the files into the subdirectory you just created. These files will help you get started quickly with Quartus.

## [2] Creating Verilog Files in Quartus

The initial Lab1 files set up a Quartus *project*, but now we need to add some actual "circuitry."

We will create a simple design of a 2:1 MUX – this is a device with two data inputs `i0` and `i1`, and a select input `sel`. **The output is equal to the `i0` input when `sel==0`, and the output is equal to the `i1` input when `sel==1`.**

1) **Start Quartus II** by double-clicking on the *DE1_SoC.qpf* file.

Your PC may hide the file extension, so if you just see "*DE1_SoC*", hover to the file and make sure the pop-up information text says "QPF File."

2) **Create a SystemVerilog file.** Go to `File`→`New`, select "SystemVerilog HDL File", and hit "OK" (Figure 1). You will do this whenever you want to create a new Verilog file.

System Verilog is "modern" Verilog, with a lot of nice features over previous versions of Verilog. We will use System Verilog exclusively in this class.

3) **Name the file.** The new file is opened for you in Quartus' text editor, but doesn't have a name yet. Go to `File`→`Save As` and give it the name "*mux2_1.sv*" (Figure 2). You should notice that the title bar for the editor pane has now changed.

In Verilog, the filename *must* be the same as the module you are designing.

4) **Populate the file.** Type in the code found in Figure 3 (or just cut-and-paste it in) to *mux2_1.sv*. This creates the module we are developing ("`mux2_1`"), as well as a tester module ("`mux2_1_testbench`") that will help us check whether the design is correct.

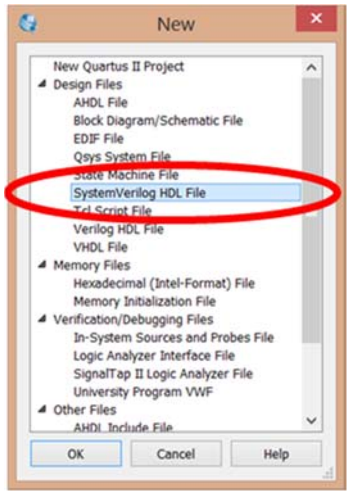
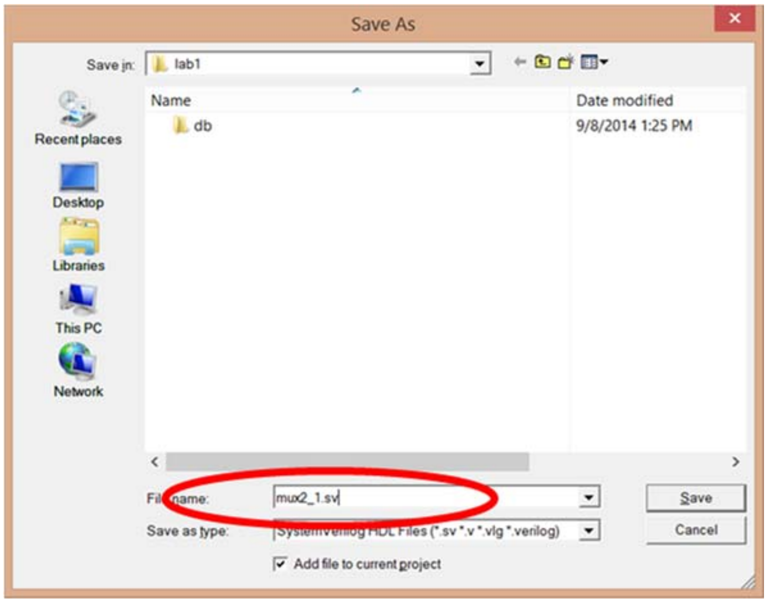


***Figure 1:** Creating a new Verilog file in Quartus*



File name: mux2_1.sv
Save as type: SystemVerilog HDL Files (*.sv *.v *.vlg *.verilog)
☑ Add file to current project

***Figure 2:** Saving and naming Verilog files in Quartus.*

```
module mux2_1(out, i0, i1, sel);
      output out;
      input i0, i1, sel;

      assign out = (i1 & sel) | (i0 & ~sel);
endmodule

module mux2_1_testbench();
      reg i0, i1, sel;
      wire out;

      mux2_1 dut (.out, .i0, .i1, .sel);

      initial begin
            sel=0; i0=0; i1=0; #10;
            sel=0; i0=0; i1=1; #10;
            sel=0; i0=1; i1=0; #10;
            sel=0; i0=1; i1=1; #10;
            sel=1; i0=0; i1=0; #10;
            sel=1; i0=0; i1=1; #10;
            sel=1; i0=1; i1=0; #10;
            sel=1; i0=1; i1=1; #10;
      end
endmodule
```

***Figure 3:*** *Code for mux2_1.sv*

## [3] Synthesizing a Design

Now that we have the design created in Quartus, we need to check that it is valid Verilog:

1) **Set the "top-level" design.** As we go through the class, we will create designs with many different modules all talking to one-another; Quartus needs to know which of the files holds the top-level, complete design. In the upper-left side of Quartus is the "Project Navigator." Make sure the "Files" tab at the bottom of the Project Navigator is selected, and right-click on the file "*mux2_1.sv*," then select "Set as Top-Level Entity" (Figure 4).

2) **Run Quartus' Analysis and Synthesis tool.** Look at the top toolbar for the blue checkmark with the purple triangle and the tiny gate symbol (Figure 5). Press that button to have Quartus check whether the design is at least syntactically correct.

3) **Fix any syntax errors.** The Analysis and Synthesis tool should run for a little while, and then tell you in the message window (near the bottom of Quartus) that "Analysis & Synthesis was successful." If it does not, then check your design and any error messages found in the message window – you can usually double-click on the error message and it will take you to exactly where Quartus thinks the error is. Correct the problems, and re-run Analysis & Synthesis.

Once Quartus declares success, we know that the file is syntactically correct Verilog. However, we don't know whether the design is a proper implementation of the desired functionality. For that, we will simulate the design, which uses the ModelSim simulator to show the actual behavior of our design.

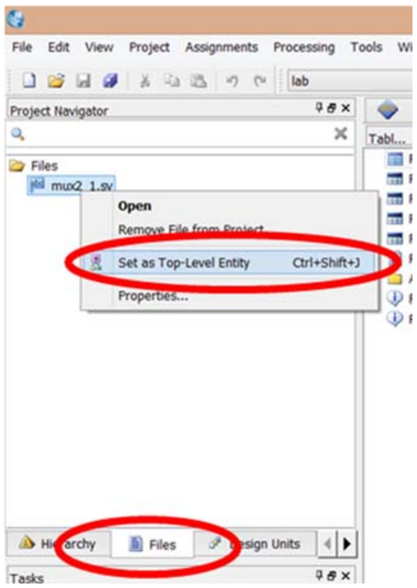
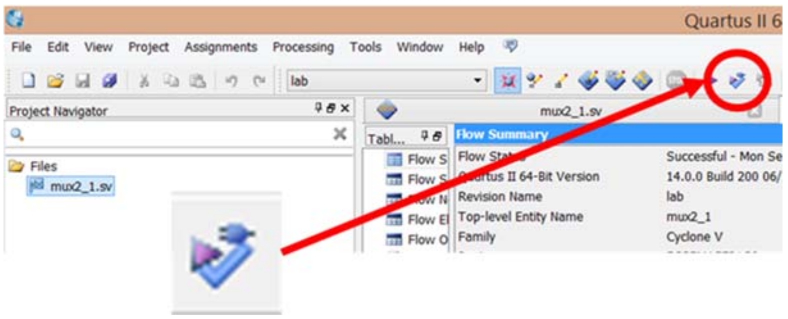


***Figure 4 (left):*** *Setting the top-level file in a Quartus project.*

***Figure 5 (below):*** *The "Start Analysis & Synthesis" button and where to find it.*



## [4] Simulating a Design

In addition to Quartus II, we will be using the ModelSim software, which can simulate Verilog designs before you ever run them on actual hardware. To help make using the tool easier, we have provided the following three files as part of *Lab1_files.zip*:

- *Launch_ModelSim.bat*: A file to start ModelSim with the correct working directory.
- *runlab.do*: A command file for ModelSim that will compile your design, set up the windows for the design, and start simulation.
- *mux2_1_wave.do*: A default file that sets up the simulation window.

1) **Start ModelSim by double-clicking *Launch_ModelSim.bat*.** This should show a blue title screen before the ModelSim opens.

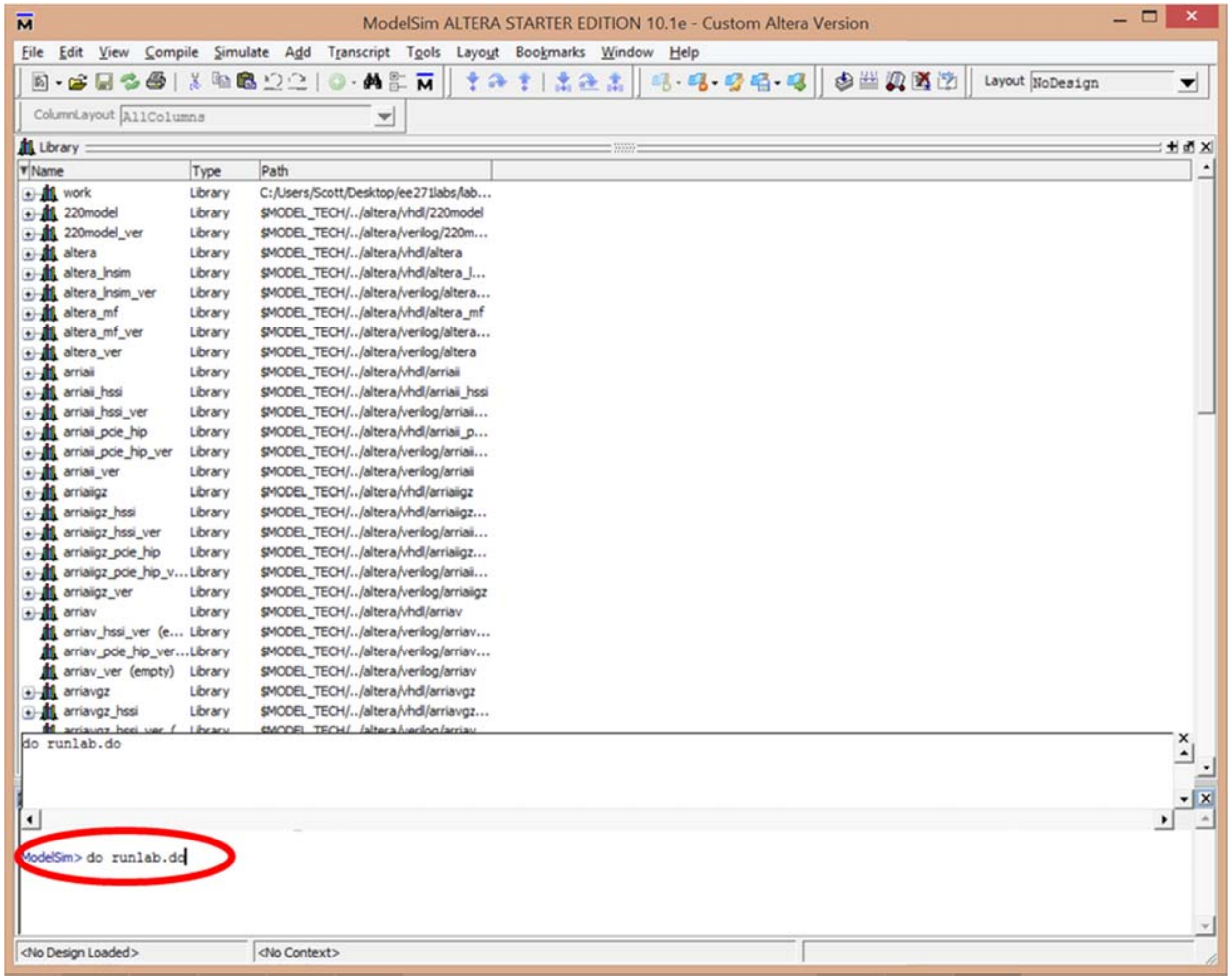
If you instead saw a black window flash by and nothing happened, then your ModelSim is installed at a non-standard location; edit the *Launch_ModelSim.bat* file and put in the correct path to the *Modelsim.exe* executable. Save the file and retry starting ModelSim.

2) **Simulate the circuit by issuing the command "do runlab.do" in the Transcript pane.** The Transcript pane can be found at the bottom of the ModelSim window (Figure 6). The *runlab.do* file will compile and run the simulation for mux2_1.

Hitting <Tab> when you have typed "do r" will auto-complete with the full command, since there are no other files in the Lab1 directory that start with "r."

3) **View the results in the Wave pane** (Figure 7)**.** Time moves from left (start) to right (end), with a green line for each input and output of the design. When the green line is up, it means that signal is true; when the green line is down, it means the signal is false.

Any **red** or **blue** lines indicate that there is a problem in your Verilog files; check that you have done all of the previous steps correctly.


ModelSim ALTERA STARTER EDITION 10.1e - Custom Altera Version
File Edit View Compile Simulate Add Transcript Tools Layout Bookmarks Window Help
Layout NoDesign
ColumnLayout AllColumns
Library

| Name | Type | Path |
|---|---|---|
| work | Library | C:/Users/Scott/Desktop/ee271labs/lab... |
| 220model | Library | $MODEL_TECH/../altera/vhdl/220model |
| 220model_ver | Library | $MODEL_TECH/../altera/verilog/220m... |
| altera | Library | $MODEL_TECH/../altera/vhdl/altera |
| altera_lnsim | Library | $MODEL_TECH/../altera/vhdl/altera_l... |
| altera_lnsim_ver | Library | $MODEL_TECH/../altera/verilog/altera... |
| altera_mf | Library | $MODEL_TECH/../altera/vhdl/altera_mf |
| altera_mf_ver | Library | $MODEL_TECH/../altera/verilog/altera... |
| altera_ver | Library | $MODEL_TECH/../altera/verilog/altera |
| arriaii | Library | $MODEL_TECH/../altera/vhdl/arriaii |
| arriaii_hssi | Library | $MODEL_TECH/../altera/vhdl/arriaii_hssi |
| arriaii_hssi_ver | Library | $MODEL_TECH/../altera/verilog/arriaii... |
| arriaii_pcie_hip | Library | $MODEL_TECH/../altera/vhdl/arriaii_p... |
| arriaii_pcie_hip_ver | Library | $MODEL_TECH/../altera/verilog/arriaii... |
| arriaii_ver | Library | $MODEL_TECH/../altera/verilog/arriaii |
| arriaiigz | Library | $MODEL_TECH/../altera/vhdl/arriaiigz |
| arriaiigz_hssi | Library | $MODEL_TECH/../altera/vhdl/arriaiigz... |
| arriaiigz_hssi_ver | Library | $MODEL_TECH/../altera/verilog/arriaii... |
| arriaiigz_pcie_hip | Library | $MODEL_TECH/../altera/vhdl/arriaiigz... |
| arriaiigz_pcie_hip_v... | Library | $MODEL_TECH/../altera/verilog/arriaii... |
| arriaiigz_ver | Library | $MODEL_TECH/../altera/verilog/arriaiigz |
| arriav | Library | $MODEL_TECH/../altera/vhdl/arriav |
| arriav_hssi_ver (e... | Library | $MODEL_TECH/../altera/verilog/arriav... |
| arriav_pcie_hip_ver... | Library | $MODEL_TECH/../altera/verilog/arriav... |
| arriav_ver (empty) | Library | $MODEL_TECH/../altera/verilog/arriav |
| arriavgz | Library | $MODEL_TECH/../altera/vhdl/arriavgz |
| arriavgz_hssi | Library | $MODEL_TECH/../altera/vhdl/arriavgz... |

```
do runlab.do
```

```
ModelSim> do runlab.do
```

*Figure 6: Entered the command "`do runlab.do`" into the Transcript pane. Press <Enter> to issue the command.*

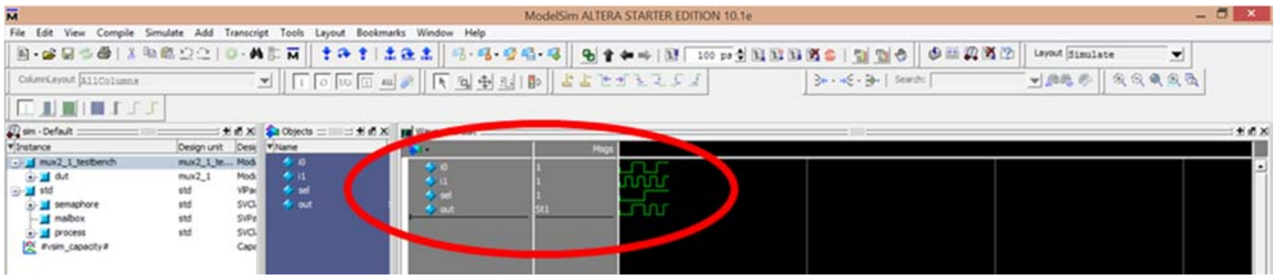


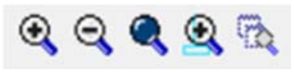
*Figure 7: Simulation results shown in the Wave pane.*

## [5] Navigating the Simulation

The initial waveforms are rather hard to see, so let’s explore the navigation options in ModelSim:

Many of the following commands will only be usable if the Wave pane is *selected*. If you don’t see the Wave pane or ever accidentally close it, go to `View→Wave` to re-open it.

- **Use the Zoom commands:** Found in the toolbars near the top of ModelSim.

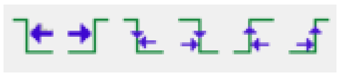


Use the left two commands (+ and – magnifying glass) to zoom so that the green waves fill the Wave pane. Notice that the scrollbar at the bottom now becomes useful, allowing us to move around in the simulation. The time for each horizontal position is shown at the bottom.

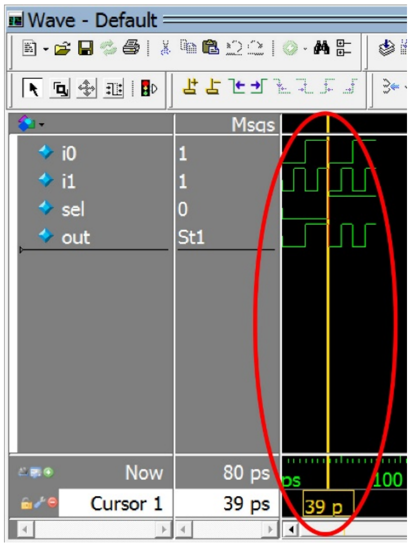
- **View signal values in the Msgs column:** Left-click anywhere within the waveform viewer (the part with the black background) to move the cursor, which is the yellow vertical line with the time in yellow at the bottom (Figure 8). The Msgs column will update with the signal values at the time specified by the cursor (Figure 9).

The signal values you will see are `0`, `1`, `St0` (“strong 0”), and `St1` (“strong 1”). For the purposes of this class, `St0` and `St1` are equivalent to `0` and `1`, respectively.

- **Use the Wave Cursor commands to jump to points of interest:** Also found in the toolbars near the top of ModelSim. To be usable, a *single* signal must be selected/highlighted (either click on a signal name or somewhere on the green waveform for that signal).

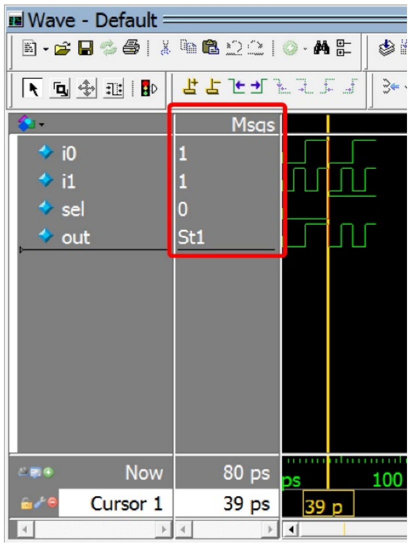


Select the `i1` signal and play with the six cursor movement commands to see what they do.



| | Msgs |
|---|---|
| i0 | 1 |
| i1 | 1 |
| sel | 0 |
| out | St1 |
| Now | 80 ps |
| Cursor 1 | 39 ps |

***Figure 8:*** *You can move the cursor (yellow line) within the waveform viewer (black background) of the Wave pane.*



| | Msgs |
|---|---|
| i0 | 1 |
| i1 | 1 |
| sel | 0 |
| out | St1 |
| Now | 80 ps |
| Cursor 1 | 39 ps |

***Figure 9:*** *The values in the Msgs column will automatically update as you move the cursor.*

## [6] Saving the Simulation View

Once we have adjusted our simulation view to better display our design results, we will often want to save these settings into a file so our next simulation run will return to this Wave pane setup.

1) Make sure that the Wave pane is *active* by clicking anywhere within it (signal list, Msgs column, or waveform viewer).
2) Select `File→Save Format` or press Control-S.
3) Overwrite the file *`mux2_1_wave.do`*. In general, we will name the format file as *`<moduleName>_wave.do`*.
4) [Optional] Verify the new format file by issuing the command "`do runlab.do`" again from the Transcript pane.

Now when you re-run your simulation, even after changing the Verilog files, it will have the Wave pane set up exactly the way we left it!

## [7] More Complex Designs – Create a 4:1 MUX

The 2:1 MUX is a simple, single-file design to get you started. But real designs will have multiple files and won't have all the scripts set up for you. Here we will show you how to build a new, more complex design that will demonstrate how to work with the various ModelSim support files.

0) Exit out of both ModelSim and Quartus, if you have not done so already.
1) Make a copy of the lab1 directory and call it lab1a., so we can use the lab1 directory as a template without overwriting our previous work.
2) Open the Quartus project file *`DE1_SoC.qpf`* in the new lab1a directory. The `mux2_1` file is already there!
3) Create a new SystemVerilog HDL file, type or cut-and-paste the code found in Figure 10, and save it as *`mux4_1.sv`*. Notice that this design uses `mux2_1` as a submodule and has its own testbench.

Every Verilog module should have a testbench, because the quickest way to get a working design is to test each submodule as you write it.

4) Set *`mux4_1.sv`* as the top-level entity and run the Analysis & Synthesis tool. Fix errors as necessary until successful.

```
module mux4_1(out, i00, i01, i10, i11, sel0, sel1);
      output out;
      input i00, i01, i10, i11, sel0, sel1;

      wire v0, v1;

      mux2_1 m0(.out(v0),  .i0(i00), .i1(i01), .sel(sel0));
      mux2_1 m1(.out(v1),  .i0(i10), .i1(i11), .sel(sel0));
      mux2_1 m (.out(out), .i0(v0),  .i1(v1),  .sel(sel1));
endmodule

module mux4_1_testbench();
      reg i00, i01, i10, i11, sel0, sel1;
      wire out;

      mux4_1 dut (.out, .i00, .i01, .i10, .i11, .sel0, .sel1);

      integer i;
      initial begin
            for(i=0; i<64; i++) begin
                  {sel1, sel0, i00, i01, i10, i11} = i; #10;
            end
      end
endmodule
```

***Figure 10:** Code for mux4_1.sv*

## [7a] More Complex Designs – ModelSim Command File

Before we can simulate, we need to modify *runlab.do* for the new design. In the text editor of your choice (e.g. WordPad, Notepad), open *runlab.do* and make the following modifications (Figure 11):

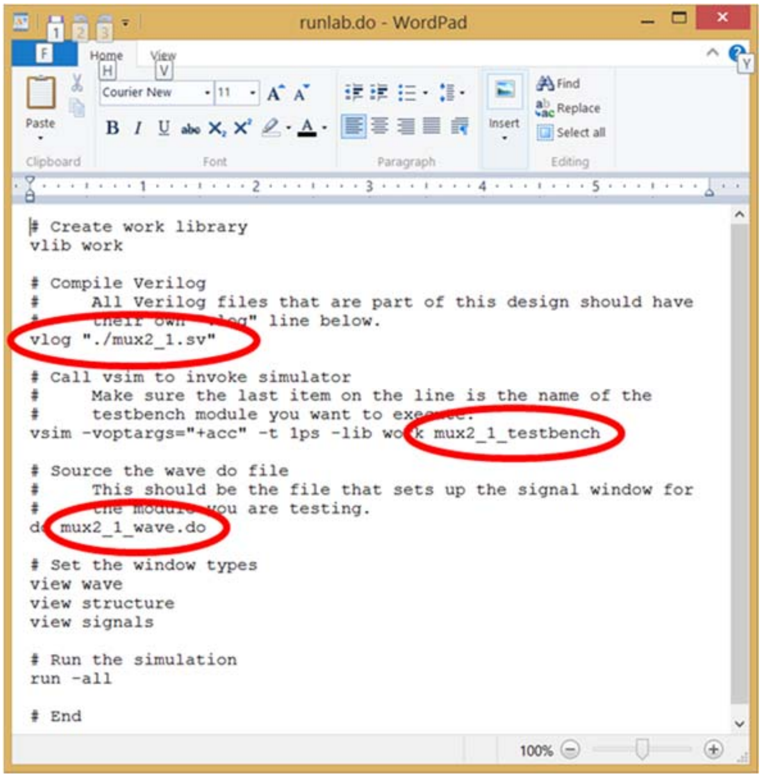
1) Add vlog "./mux4_1.sv" to the compilation section. For all Quartus designs, you will have one "vlog" line for each Verilog file in your design.

2) Edit the "vsim" line to end with mux4_1_testbench instead of mux2_1_testbench to change the module being simulated/tested.

3) Edit the "do" line to end with mux4_1_wave.do instead of mux2_1_wave.do to change the waveform settings. Each module should have its own **_wave.do* file, so that during debugging of a large project you can switch between different modules to test.

Save *runlab.do*, run *Launch_ModelSim.bat* in the lab1a directory, then execute "do runlab.do".

The system should start simulating, show the waveform pane, and then give an error that it cannot open the macro file *mux4_1_wave.do*. That's because we haven't provided the waveform file for you; you need to create it yourself!



```
# Create work library
vlib work

# Compile Verilog
#     All Verilog files that are part of this design should have
#     their own "vlog" line below.
vlog "./mux2_1.sv"

# Call vsim to invoke simulator
#     Make sure the last item on the line is the name of the
#     testbench module you want to execute.
vsim -voptargs="+acc" -t 1ps -lib work mux2_1_testbench

# Source the wave do file
#     This should be the file that sets up the signal window for
#     the module you are testing.
do mux2_1_wave.do

# Set the window types
view wave
view structure
view signals

# Run the simulation
run -all

# End
```

***Figure 11:*** *The three modifications: (1) add file to compile, (2) change testbench to simulate, and (3) change waveform settings.*

## [7b] More Complex Designs – ModelSim Waveform Macro

Our goal is to get the Wave pane properly set up so we can save the waveform settings as a *`*_wave.do`* file.

1) Locate the sim tab (confusingly opened via `View→Structure`), which may be hidden behind the "Library" or "Project" tabs on the left side of ModelSim. This tab shows the various modules in the design.

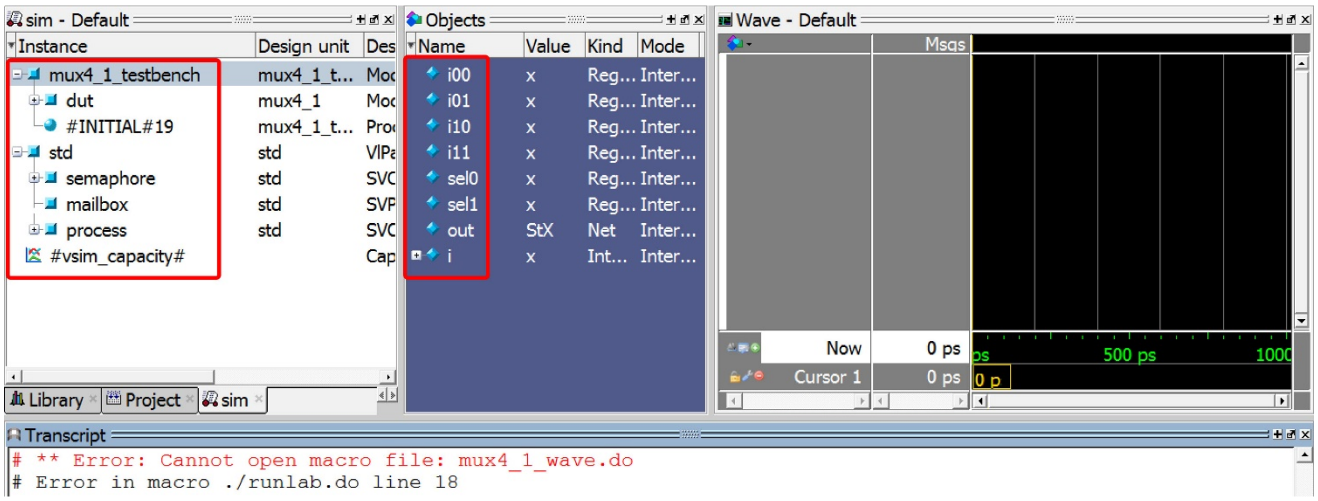
2) `mux4_1_testbench` is the top-level design, which contains `dut` ("device under test"), the name of the `mux4_1` module we are testing. Clicking on the plus next to `dut` shows the three `mux2_1`'s inside of the `mux4_1`: `m0`, `m1`, and `m`. If you click on any of the units in the sim tab, the Objects pane next to it shows the signals inside that module (Figure 12).

3) Click on `mux4_1_testbench` in the sim tab, select all of the signals in the Objects pane except `i`, and drag-and-drop them into the Wave pane.

4) Save the waveform as *`mux4_1_wave.do`* to create the missing file for simulation.

5) Re-run "`do runlab.do`" from the Transcript pane to get a simulation of the entire design.

Examine the waveforms using the navigation techniques. Figure out what the `mux4_1` actually does.



| Instance | Design unit | Des |
|---|---|---|
| mux4_1_testbench | mux4_1_t... | Moc |
| dut | mux4_1 | Moc |
| #INITIAL#19 | mux4_1_t... | Proc |
| std | std | VIPa |
| semaphore | std | SVC |
| mailbox | std | SVP |
| process | std | SVC |
| #vsim_capacity# | | Cap |

Objects

| Name | Value | Kind | Mode |
|---|---|---|---|
| i00 | x | Reg... | Inter... |
| i01 | x | Reg... | Inter... |
| i10 | x | Reg... | Inter... |
| i11 | x | Reg... | Inter... |
| sel0 | x | Reg... | Inter... |
| sel1 | x | Reg... | Inter... |
| out | StX | Net | Inter... |
| i | x | Int... | Inter... |

Transcript

```
# ** Error: Cannot open macro file: mux4_1_wave.do
# Error in macro ./runlab.do line 18
```

***Figure 12:*** *The sim tab is found on the far-left and contains all of the modules in this design. Selecting a module or submodule will show all of the signals contained in that module in the Objects pane just to the right.*

## [8] Process Recap

You now have the commands necessary to develop new designs, commands you will use for all future labs. Just to make sure you've got it, here's a cheat-sheet of the steps for future Verilog designs:

1) Make a copy of a previous lab directory to build off of what you already have (Quartus project file, ModelSim files) while keeping the old design as a reference.
2) For each module you need to write:
    a) Create a new file, write the module definition, and write a testbench for that module.
    b) Set the testbench as the top-level module in Quartus.
    c) Run Analysis and Synthesis and fix any errors it finds.
    d) Edit *`runlab.do`* to include the new module.
    e) Start ModelSim and perform "`do runlab.do`." Fix any errors the compiler finds.
    f) When it complains about a missing *`*_wave.do`* file, set up the Wave pane by drag-and-dropping signals from the Object pane. Save the waveform setup using `File→Save Formatting`, then perform "`do runlab.do`" again.
    g) Check the simulation results, correct errors, and iterate until the module works.

This process has two major features: First, it has you test *every* module before you work on the larger modules that call this unit. This will *significantly* simplify the design process. Second, you have a separate *`*_wave.do`* file for each Verilog file. This keeps a formatted test window for each module, which can help when you discover a fresh bug in a larger design later on. You can always go back and test a submodule by simply editing the *`runlab.do`* file to point to the testbench and *`*_wave.do`* file for the unit you want to test.

## [9] Mapping a Design to the FPGA Hardware

So far we have developed and tested a design completely in software. Once it is working, it is time to use Quartus II to convert that design into a form that can actually be loaded onto the FPGA.

To use the switches, lights, and buttons on the DE1 board, we need to hook up the connections of the circuit design to the proper inputs and outputs of the FPGA. In lab1a, use Quartus to create a new SystemVerilog file called *`DE1_SoC.sv`*, with the following contents:

```
// Top-level module that defines the I/Os for the DE1-SoC board
module DE1_SoC (HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, LEDR, SW);

      output [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
      output [9:0] LEDR;
      input [3:0] KEY;
      input [9:0] SW;

      mux2_1 m(.out(LEDR[0]), .i0(SW[0]), .i1(SW[1]), .sel(SW[9]));

      assign HEX0 = '1;
      assign HEX1 = '1;
      assign HEX2 = '1;
      assign HEX3 = '1;
      assign HEX4 = '1;
      assign HEX5 = '1;

endmodule
```

***Figure 13:*** *Code for DE1_SoC.sv*

You should set this file as the top-level entity. For inputs, the signals `KEY[3]` … `KEY[0]` are the pushbuttons on the front-right of the board, while `SW[9]` … `SW[0]` are the sliders at the front left. They are labelled on the green printed-circuit board. For outputs, the HEX values are the six 7-segment displays (numeric displays like a digital clock) on the left side, and `LEDR[9]` … `LEDR[0]` are the red LEDs just above the sliders.

In the `DE1_SoC` module we hook the inputs of a 2:1 MUX to slider switches, and show the output on the rightmost LED.

We now need to compile the design into a **bitfile**, a file that can be downloaded to the FPGA. To do that, we press the “Start Compilation” button just to the left of the “Analysis & Synthesis” button we have used before:

▶

This will run the multiple steps necessary to compile the design. You can watch the progress of the compilation in the Tasks pane in the lower-left of Quartus.

## [10] Configuring the FPGA with the Bitfile

We now need to send the bitfile to the DE1-SoC.

1) Connect the DE1-SoC to wall power with the power cord. The power cord is black, and it plugs into the black socket “Power DC Jack” next to the red on/off button (Figure 14).

2) Make sure the board is off (i.e. the board should not light up when you plug it in), then plug the provided grey USB cord into the USB-Blaster II port of the DE1-SoC (Figure 14), and to a USB port of the computer you are using to run Quartus II. You can then turn on the DE1-SoC.

3) In Quartus, go to `File→Open`. In the “Files of type” box at bottom, select “Programming Files (*.cdf …” and then double-click on *`ProgramTheDE1_SoC.cdf`* (Figure 15).

4) This will bring up the Programmer dialog box (Figure 16).

   a) If the “Start” button is active, proceed to the next step.

   b) If the “Start” button is greyed out, you need to first run click the “Hardware Setup…” button. This will bring up the “Hardware Setup” dialog box. Set “Currently selected hardware” to “DE-SoC”, and close the dialog box (Figure 17).

5) Click the “Start” button and the DE1 board will be programmed – you’re done!

When you are developing a design, you can keep the Programmer dialog box open so that you can download the design multiple times, including after changing the input files and recompiling the design.

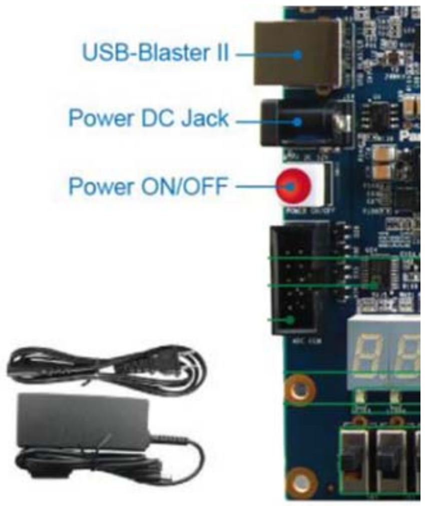
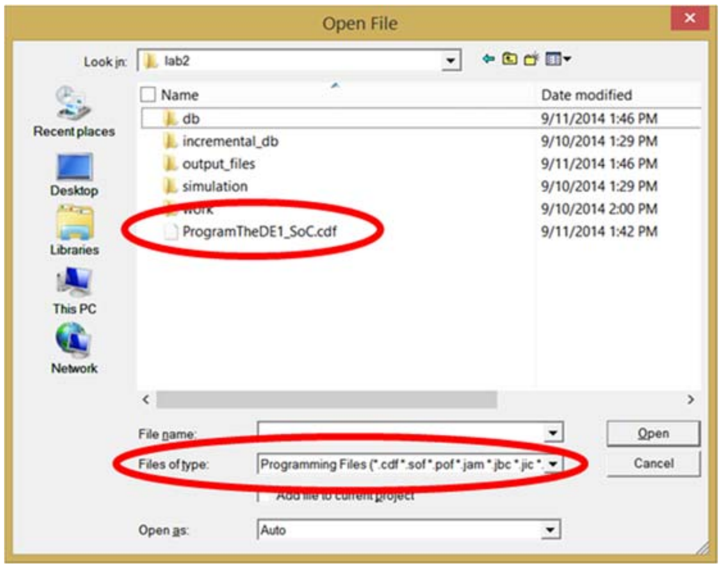


*__Figure 14:__ The power cord (left) plugs into an electrical outlet and the "Power DC Jack" on the DE1-SoC board (right).*

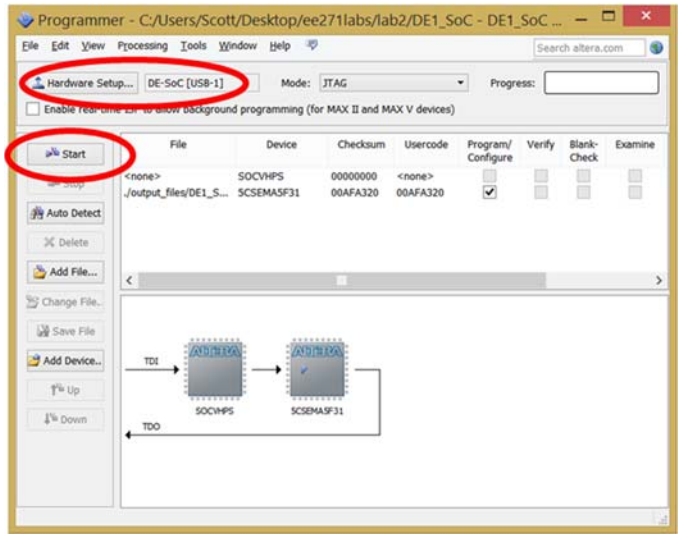


Look in: lab2

| Name | Date modified |
|---|---|
| db | 9/11/2014 1:46 PM |
| incremental_db | 9/10/2014 1:29 PM |
| output_files | 9/11/2014 1:46 PM |
| simulation | 9/10/2014 1:29 PM |
| work | 9/10/2014 2:00 PM |
| ProgramTheDE1_SoC.cdf | 9/11/2014 1:42 PM |

File name:
Files of type: Programming Files (*.cdf *.sof *.pof *.jam *.jbc *.jic *.
Add file to current project
Open as: Auto

*__Figure 15:__ Open the chain description file (cdf) to program the DE1-SoC*



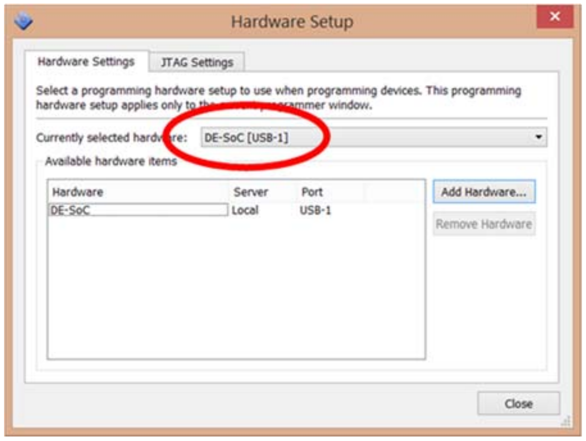
Programmer - C:/Users/Scott/Desktop/ee271labs/lab2/DE1_SoC - DE1_SoC ...

Hardware Setup... DE-SoC [USB-1] Mode: JTAG Progress:

Enable real-time ISP to allow background programming (for MAX II and MAX V devices)

| File | Device | Checksum | Usercode | Program/ Configure | Verify | Blank-Check | Examine |
|---|---|---|---|---|---|---|---|
| <none> | SOCVHPS | 00000000 | <none> | ☐ | ☐ | ☐ | ☐ |
| ./output_files/DE1_S... | 5CSEMA5F31 | 00AFA320 | 00AFA320 | ✓ | ☐ | ☐ | ☐ |

*__Figure 16:__ Programmer dialog box with the "Start" and "Hardware Setup…" buttons highlighted.*



Hardware Setup

Select a programming hardware setup to use when programming devices. This programming hardware setup applies only to the current programmer window.

Currently selected hardware: DE-SoC [USB-1]

Available hardware items

| Hardware | Server | Port |
|---|---|---|
| DE-SoC | Local | USB-1 |

*__Figure 17:__ Hardware Setup dialog box in case the "Start" button in the Programmer dialog box was greyed out.*

## [11] Appendix A: Files in the Default Project

For those who are interested, here are what each of the files contained in `Lab1_files.zip` do:

| Filename | Purpose |
|---|---|
| **DE1_SoC.qpf** | Quartus project file. Top-level that groups all the information together. Preconfigured for the DE1-SoC board. |
| **DE1_SoC.qsf** | Sets up the pin assignments, which connects the signals of the user design to specific pins on the FPGA. |
| **DE1_SoC.sdc** | Tells Quartus about the timing of various signals. |
| **DE1_SoC.srf** | Tells Quartus to not print some useless warning messages. |
| **Launch_Modelsim.bat** | Simple batch file – starts ModelSim in the current directory. |
| **mux2_1_wave.do** | Sets up the waveform viewer for the first design. |
| **ProgramTheDE1_SoC.cdf** | Programmer file, tells Quartus how to download designs to the DE1. |
| **runlab.do** | ModelSim .do file – compiles and simulates the design. |