

Quartus Tutorial

August 14, 2017

Quartus Prime Lite Version 17.0

This tutorial will walk you through the process of developing circuit designs within Quartus, simulating with Modelsim, and downloading designs to the DE-1 SoC board.

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

1. Installing the Quartus Software

Most of the designs in this class will be done through the Altera Quartus software. This is preloaded on machines in the EE department, and you are free 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 set up Quartus on your own machine, skip to the next section.

To install the software on your own PC, grab the Quartus 17.0 software from the EE271 website. You'll need the Quartus software, the ModelSim software, and the CycloneV qdz file. Save all of these to the same directory.

Run the QuartusLiteSetup program – after double-clicking it might take a bit longer than you expect to start up. When it asks for components to install, make sure you have selected each of these:

- Quartus Prime
- Devices>Cyclone V
- ModelSim – Starter Edition.

When the software is done installing, make sure to also select “Launch USB Blaster II driver installation”.

Then, start the Quartus software. When it asks about licensing, select “Run the Quartus Prime software”. You may have to start it twice to get it actually to run the first time.

2. Getting Started in Quartus

In this class we will do multiple labs using the Quartus software. As part of this, we will create multiple files for your designs, for testing your designs, and for downloading your design to the DE-1 SoC board. To keep things sane, you should create an overall class directory, and then a subdirectory under that when you start each lab. So, you might have an “ee271labs” directory, and create a “lab1” subdirectory for lab #1. Do not reuse the same directory for

different labs, since you'll want to refer back to a working design when you develop each new lab. However, when you start each lab after #1, copy the previous lab's 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 U: drive (shared across all machines). If you are using your own machine, you can store the files where-ever you'll remember them.

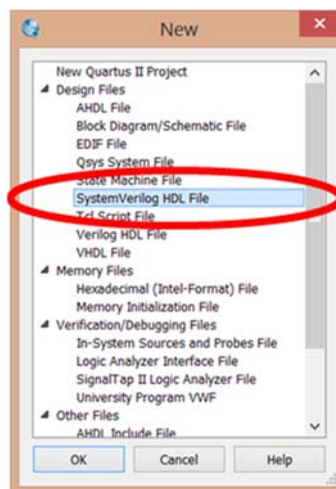
Get the lab #1 files from the class website, and put them into the subdirectory you just created (note: you need to copy them to the new directory – if you leave them in the ZIP file you downloaded from the website you'll have problems). These files will help you get started quickly with Quartus.

3. Creating Verilog Files in Quartus

In the previous steps we created a directory, and moved in files to set up a Quartus project, which told the tool about the DE1 SoC board we are using. We now need to add some actual circuitry to the project. 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. When sel==0 the output is equal to the i0 input, while when sel==1 the output is equal to the i1 input.

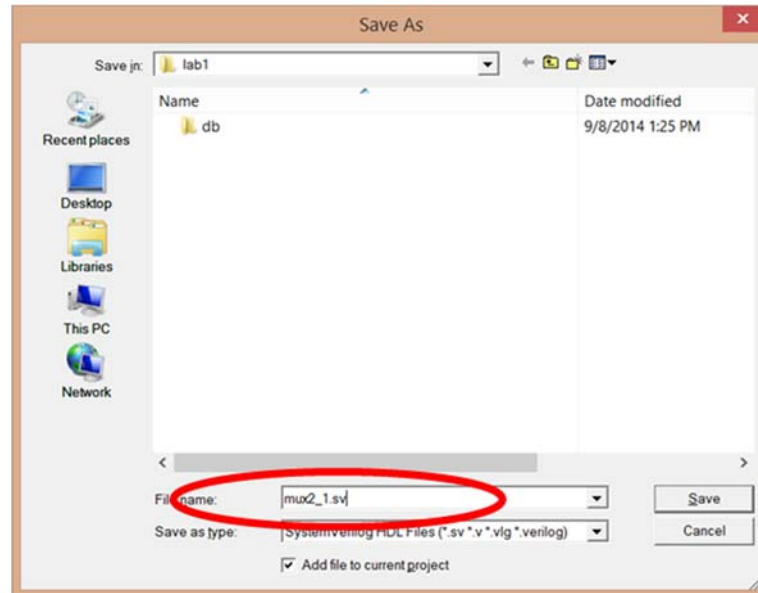
Start Quartus by double-clicking on the DE1_SoC.qpf file, which is the main Quartus file for this project. Your PC may hide the file extension, so if you just see “DE1_SoC”, point to it and make sure the pop-up information text says “QPF File”.

We now need to create a SystemVerilog file (System Verilog is “modern” Verilog, with a lot of nice features over previous basic Verilogs. We will use System Verilog exclusively in this class). Go to File>New (or just hit control-N), select “SystemVerilog HDL File”, and hit “OK”. You will do this whenever you want to create a new Verilog file.



The new file is opened up for you in Quartus's text editor in the middle of the tool. Note that the file doesn't have a specific name yet –fix that by hitting “File>Save As”. Then give it the

name “mux2_1.sv” and save the file. Note that in Verilog the filename MUST be the same as the module you are designing, and in this case we are designing a module called “mux2_1”.



You should notice that the title bar for the editor pane has now changed to “mux2_1.sv”. We now need to put in the circuitry that we are developing. You can type in the following (or just cut-n-paste it in) to the mux2_1.sv window.

```

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

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

module mux2_1_testbench();
    logic i0, i1, sel;
    logic out;

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

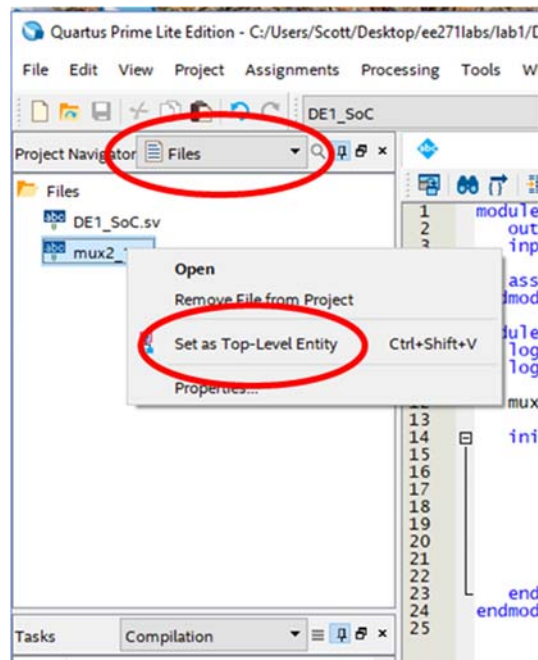
    initial begin
        sel=0; i0=0; i1=0; #10;
        sel=0; i0=0; i1=1; #10;
        sel=0; i0=1; i1=0; #10;
        sel=0; i0=1; i1=1; #10;
        sel=1; i0=0; i1=0; #10;
        sel=1; i0=0; i1=1; #10;
        sel=1; i0=1; i1=0; #10;
        sel=1; i0=1; i1=1; #10;
    end
endmodule

```

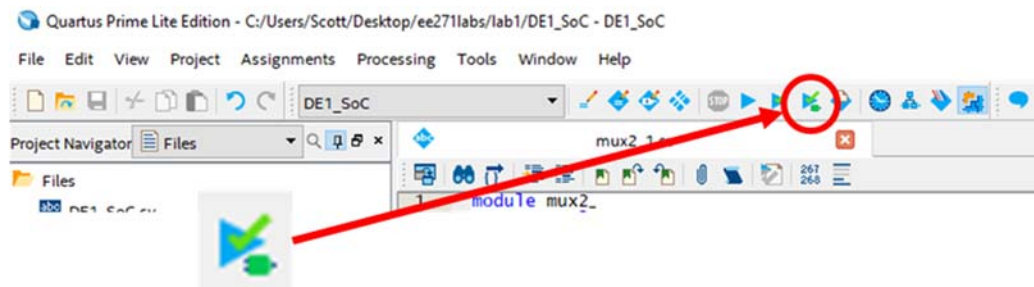
This creates the module we are developing (“mux2_1”), as well as a tester module (“mux2_1_testbench”) that will help us check whether the design is correct.

4. Synthesizing a design

Now that we have the design created in Quartus, we need to check that it is valid Verilog. First, we need to inform Quartus that the mux2_1 file is the “top-level” of the design – as we go through the class we will create designs with many different modules all talking to one-another, and Quartus needs to know which of the files holds the top-level, complete design. Since we have a 1-file circuit this is pretty easy. In the upper-left side of Quartus is the “Project Navigator”. In the pull-down menu next to “Project Navigator” select “Files”. Right-click on the file “mux2_1.sv”. Select “Set as Top-Level Entity”.



You can now have Quartus check whether the design is at least syntactically correct (i.e. you didn't make any spelling mistakes or the like). Look at the top toolbar for the green checkmark with the blue triangle and the tiny gate symbol. Press that button, which will start Quartus's Analysis and Synthesis steps (or hit control-k).



The 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 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.

5. Simulating a design

In addition to Quartus, we will be using the ModelSim software, which can simulate Verilog designs for you. To help make using the tool easier, we provide three files on the website to help:

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 properly.

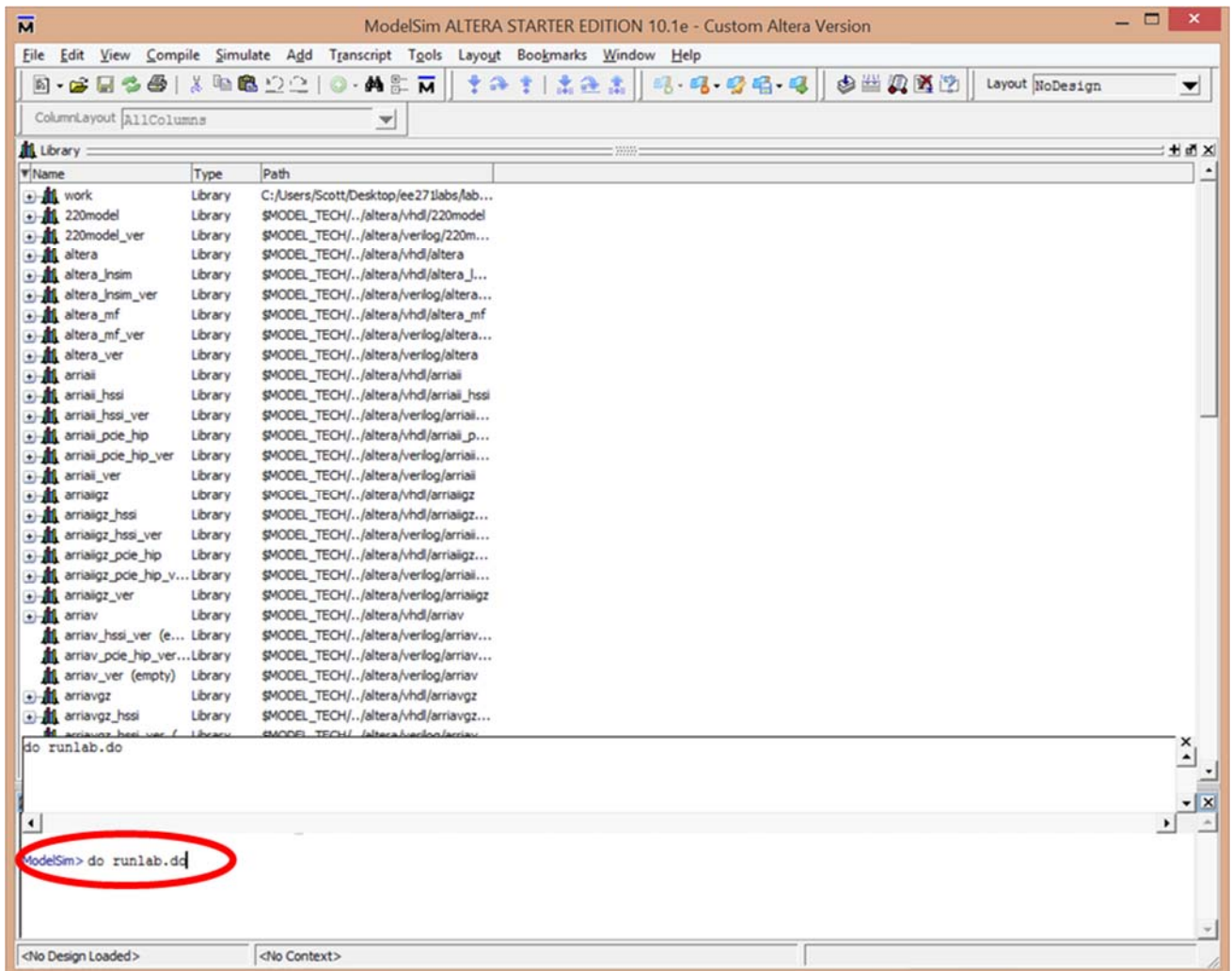
You already added these files into the lab1 directory in a previous step.

To start ModelSim, double-click the "Launch_ModelSim.bat" file. This should show the blue "ModelSim" title screen and start ModelSim. 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 by right-clicking the file, and put in the correct path to the Modelsim.exe executable, save the file, and retry starting ModelSim. The path you enter into "Launch_ModelSim.bat" should resemble the following:

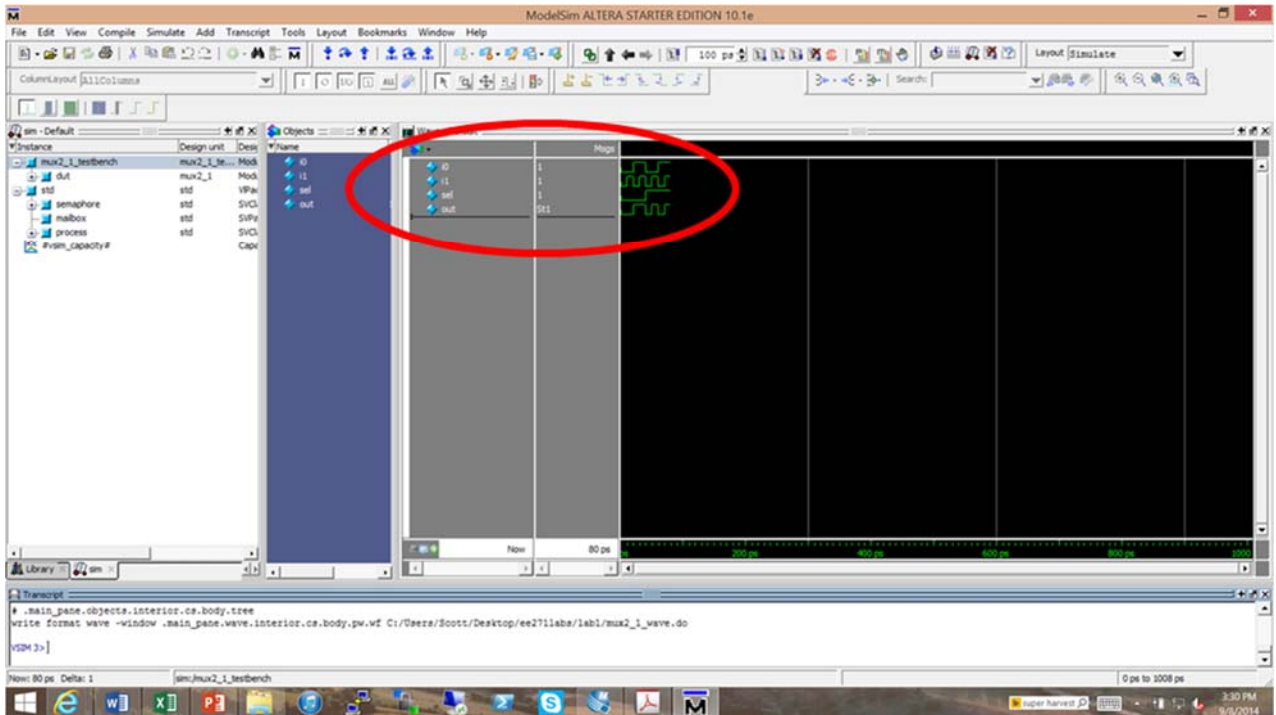
```
D:\intelFPGA_lite\17.0\modelsim_ase\win32aloem\modelsim.exe
```

If your path shows "modelsim_ae", modify it to be "modelsim_ase" instead.

Once ModelSim is started, we can now simulate our circuit. At the bottom of the window is the "Transcript" pane. We can issue commands here, and see ModelSim's responses. For mux2_1, we want to use the "runlab.do" file to compile and run the simulation. To do that, in the transcript pane type "do runlab.do" and hit enter. Note that hitting <tab> when you have typed "do r" already will auto-complete with the full command.



Once you execute the command, ModelSim will simulate the execution of the design, and display the results in the simulation window. Time moves from left (start of simulation) to right (end of simulation), with a green line for each input and output of the design. When the green line is up, it means that signal is true, while if the green line is down it means the signal is false. Note that if you see any red or blue lines it means there is a problem in your Verilog files – check that you have done all of the previous steps correctly.



6. Navigating the simulation

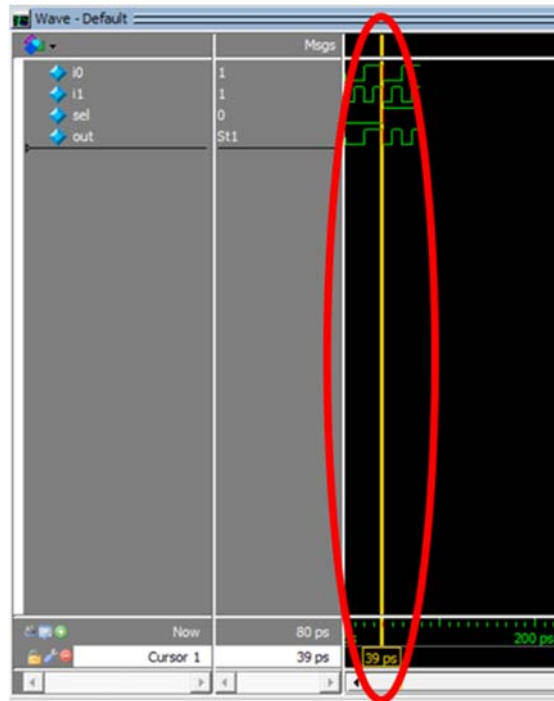
At this point you should have successfully run the simulation, but the waveform window is rather small and hard to see. Let's explore the navigation commands in ModelSim.

Click on the waveform window, and look at the toolbars near the top of the ModelSim window. We first want to use the zoom commands:

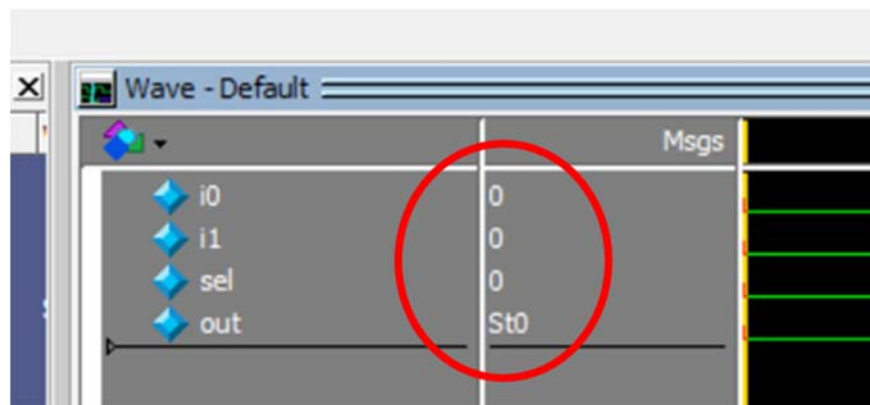


Use the left two commands (+ and – magnifying glass) to zoom so that the green waves fill the waveform window. Notice that the scrollbar at the bottom of the waveform window now becomes useful, allowing us to move around in the simulation. The time for each horizontal position is also shown at the bottom of the window. The third button (black-filled magnifying glass) is also a good way to zoom to get the entire waveform shown in the window.

We can also move around in the simulation and see the value of the signals. Look for the cursor, a yellow vertical line in the waveform viewer, with the time in yellow at the bottom.

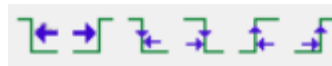


Left-click on one of the green lines in the waveform viewer. The cursor moves to that location, and next to each signal name appears a 0 or 1 value. This means that, at the time specified by the cursor, the signals are at those given values. If the “out” signal says “St1” or “St0” that’s fine – just another way to say 1 or 0.



Left-click in the waveform window at another point on the green waveforms. The cursor will jump to that position, and the Msgs field will update with the values of all signals. This will allow you to move to whatever position is of concern, and look at each signal value.

We can also move to points of interest for a given signal. Click on the green waveform for the “i1” signal. The “i1” label in the leftmost waveform column should become highlighted. Play with the six cursor movement commands to see what they will do:



These commands will help you quickly move through the simulation, finding situations of interest.

Now that we have zoomed in to better display our design, and put a cursor at a point of interest, we will often want to save these settings into a file, so that our next simulation run will return back to this position. To do that, click somewhere in the grey columns of the waveform pane, then select “File > Save Format” from the toolbar. You should overwrite the file “mux2_1_wave.do”. In this way, when you rerun simulation, it will have the waveform window set up exactly the way we just left it, though with new simulation results if you changed the Verilog files (i.e. fixed any bugs there are in your design...). Verify this by clicking on the Transcript window and typing “do runlab.do” now.

7. More complex designs

The 2:1 mux design was set up to be a simple, single-file design to get you started quickly. But, real designs will have multiple files, and won't have all the scripts set up for you. Let's make a more complex design, and show you how to build new designs, especially how to work with the various ModelSim support files.

Make sure you have exited out of both ModelSim and Quartus.

We'll now build a 4:1 mux out of the 2:1 muxes. We could go through all the steps above, but why bother? Instead, simply make a copy of the lab1 directory, and call it lab1a. So you should now have an ee271labs directory with both a lab1 and lab1a subfolder. In this way we can use the lab1 directory as a template, without overwriting all of our old work. Go into directory lab1a and double-click “DE1_SoC.qpf”, the Quartus project file. This starts Quartus in the new directory, with the mux2_1 design already there. We're going to need a new file for our mux4_1, so do File>New and create a SystemVerilog HDL file. Do File>Save As and name it mux4_1.sv. In the file, type or cut-n-paste the following design for the mux4_1:

```

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

    logic  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();
    logic  i00, i01, i10, i11, sel0, sel1;
    logic  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

```

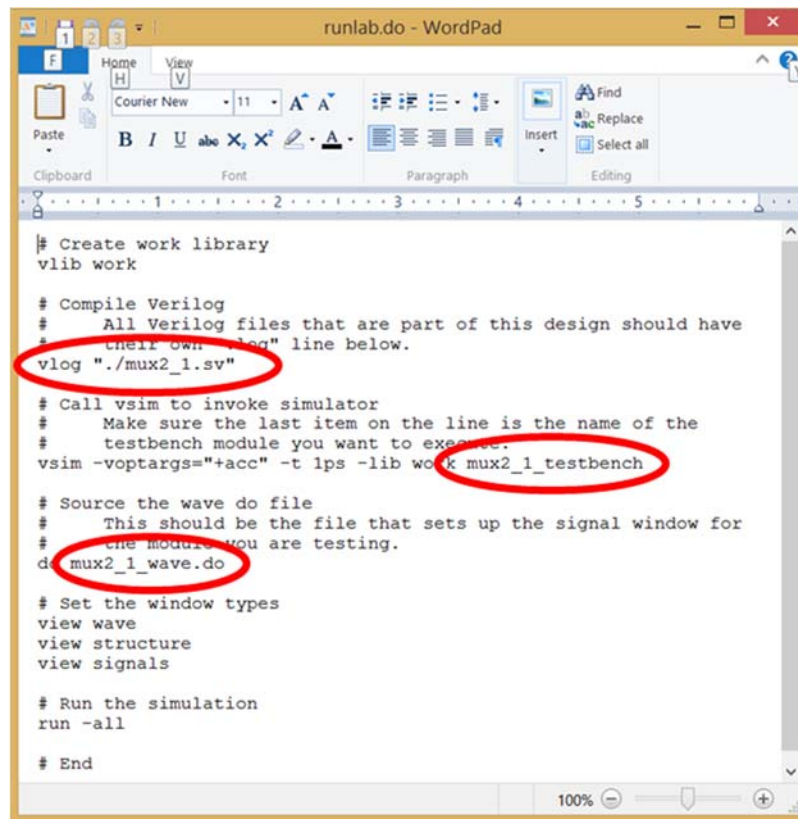
Notice that this design uses the mux2_1 as a subroutine. Notice also that this file has its own testbench – every Verilog module should have a testbench, because the quickest way to get a working design is to test each submodule as you write it.

To check that the design is correct, right-click on “mux4_1.sv” and “Set as Top-level Entity”, then run Analysis & Synthesis from the toolbar. If Quartus doesn’t say “Analysis & Synthesis was successful”, fix whatever errors there are.

Before we perform simulation, we need to fix the runlab.do file to work for the new design. Outside of Quartus right-click on runlab.do in a Windows File Explorer, and edit the file (use WordPad, NotePad, or whatever text editor is on your machine). We need to make the following modifications to the file:

1. Add a line to compile the mux4_1.sv file. Duplicate the current line that starts with “vlog”, and change “mux2_1” to “mux4_1” in the duplicate. For all Quartus designs, you will have one “vlog” line for each Verilog file in your design.
2. Change the module being simulated. Edit the line starting “vsim” to end with mux4_1_testbench, instead of mux2_1_testbench. This tells ModelSim what unit you are testing right now.

3. Change the file that contains the waveform setting file. Edit the line starting “do” to change the “mux2” to “mux4”. Each module will have its own wave.do file, so that during debugging of a large project you can switch between different modules to test.



```
# 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 lps -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

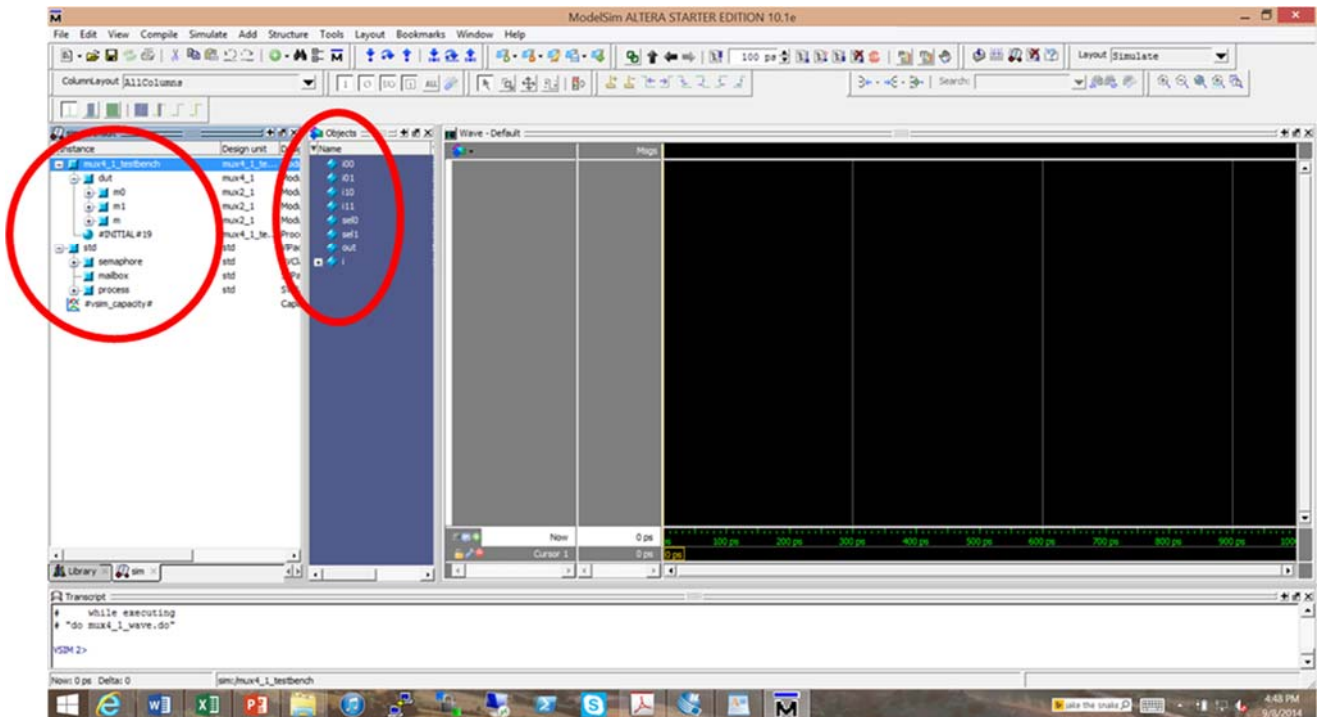
# Run the simulation
run -all

# End
```

Save the file, start ModelSim via the Launch_ModelSim.bat file in the lab1a directory, and execute “do runlab.do”.

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

At the left of ModelSim window is the sim pane, which shows the various modules in the design. “mux4_1_testbench” is the top-level design, and inside that is “dut, 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 pane, the Objects pane next to it shows the signals inside that module.



Click on mux4_1_testbench, select all of the signals in the Objects pane except “I”, and drag them (hold down the left button while moving the mouse) into the waveform pane. This will put all of these signals into the waveform viewer so that you can monitor them. Now save the waveform setup (click on the grey of waveform, select File>Save Format (or cntrl-s), and save it as mux4_1_wave.do). You’ve now created the missing file for simulation. Now, click the transcript window, and “do runlab.do”. You will now get a simulation of the entire design.

Look through the waveform view via the zoom and cursor commands we used earlier. Figure out what the mux4_1 actually does.

8. Recap – starting a new design & simulating the design

In the previous section we created a new mux4_1 design and simulated it. 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 all future Verilog designs you’ll do in this class.

1. Make a copy of a previous lab directory. This keeps the old design as a reference, but allows you to build off of what you already have. This includes the Quartus Project file and the support files for ModelSim.
2. For each module you need to write, do:
 - 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 the runlab.do file to include the new module.
- e. Start ModelSim, perform “do runlab.do”. Fix any errors the compiler finds.
- f. When it complains about a missing *_wave.do file, set up the waveform window by drag-and-dropping signals. Save it by 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 convert that design into a form that can actually be loaded onto the FPGA. Quartus is responsible for doing these steps.

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 DE-1 SoC board

module DE1_SoC (HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, LEDR, SW);
    output logic [6:0]  HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
    output logic [9:0]  LEDR;
    input  logic [3:0]  KEY;
    input  logic [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
```

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 6 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

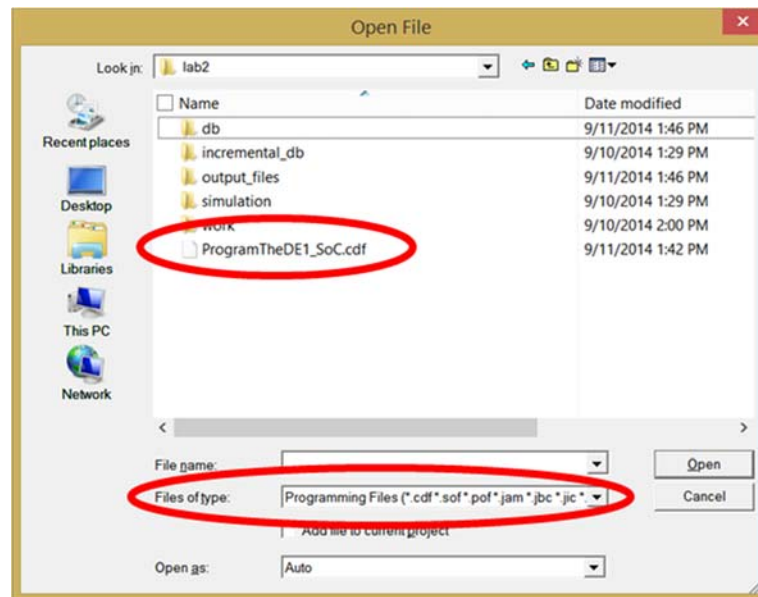
We now need to actually send the bitfile to the DE1 SoC.

Connect the DE-1 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:

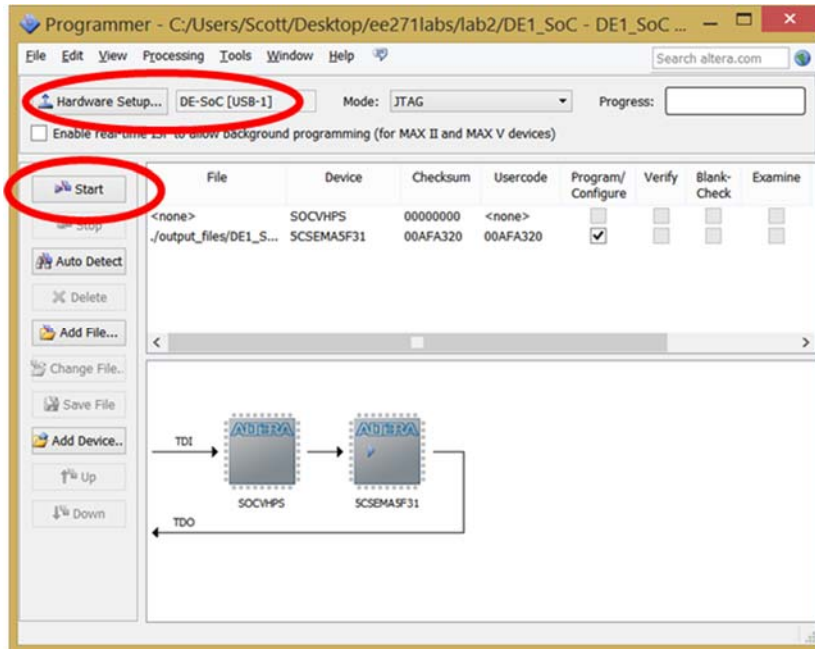


Make sure the board is off (if the board lights up when you plug it in, press the red button). Then plug the provided grey USB cord into the USB-Blaster II port of the DE1-SoC, and to a USB port of the computer you are using to run Quartus. You can then turn on the DE1-SoC.

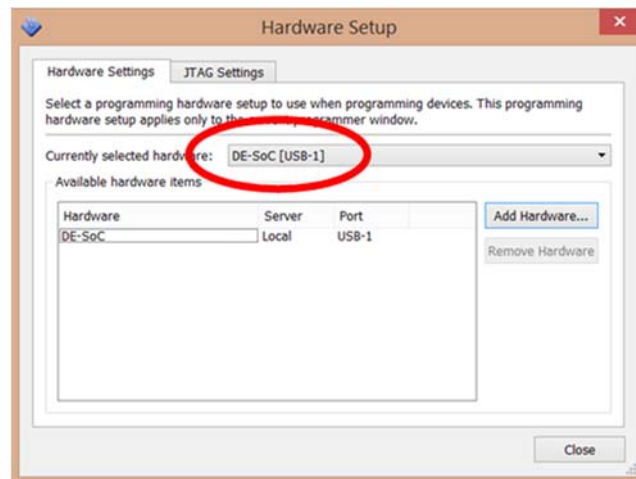
In Quartus, go to “File > Open”. In the “Files of type” box at bottom, select “Programming Files (*.cdf ...)”. Double-click on “ProgramTheDE1_SoC.cdf”.



This will bring up the Programmer dialog box. If the “Start” button is active, click start and the DE1 board will be programmed – you’re done!



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.



Now press the “Start” button on the Programmer dialog box, and this will program the FPGA.

Note that when you are developing a design, you can keep the programmer open so that you can download the design multiple times, including after changing the input files and recompiling the design.

11. Appendix A: Files in the default project

As part of this tutorial we have you copy a set of files into your lab1 directory, and after that they are copied into each subsequent project. For those who are interested, here are what each of those files does:

Filename	Purpose	Source
DE1_SoC.qpf	Quartus project file. Top-level that groups all the information together. Preconfigured for the DE1-SoC board.	17.0
DE1_SoC.qsf	Sets up the pin assignments, which connects the signals of the user design to specific pins on the FPGA.	17.0
DE1_SoC.sdc	Tells Quartus about the timing of various signals.	14.0
DE1_SoC.srf	Tells Quartus to not print some useless warning messages.	14.0
Launch_Modelsim.bat	Simple batch file – starts ModelSim in the current directory.	14.0
mux2_1_wave.do	Sets up the waveform viewer for the first design.	14.0
ProgramTheDE1_SoC.cdf	Programmer file, tells Quartus how to download designs to the DE1.	14.0
runlab.do	ModelSim .do file – compiles and simulates the design.	14.0

Note: when upgrading from 14.0 to 17.0 many files were created by the tools, others were retained from our 14.0 deployment. Information listed in the “Source” column.