

CSCE 230L – Lab 5: Quartus Introduction

February 7, 2014

Purpose

The purpose of this lab is to introduce you to the program that you will use for logic design and for the project: Quartus. You will use Quartus to create logic circuits and eventually put things together to create a simple processor.

In the second part of this lab, you will learn more about Quartus, common errors and debugging methods.

Objectives

- Quartus
 - Creating a project
 - Creating block diagrams
 - Simulation with ModelSim
 - Loading design onto the board
 - Learn more about Quartus II functionality and how to handle errors
 - Learn basic testing and debugging

What to Turn In

Part 1:

.qar file (30 points)

Answer to the questions (20 points)

Part 2:

Modelsim script (the *.do file) (15 points)

Image of simulation results (20 points)

Answer to the questions (15 points)

Part 1

With programs like Quartus, we can design, build, and simulate different **logical systems** from simple components such as a multiplexer to more complicated components such as a processor. Being able to test a design in software saves us the trouble of programming a chip every time we changed something. This allows development to progress faster and cost less as well.

For our labs with Quartus, there are always going to be four main steps: **designing, compiling, simulating/testing, and deployment**. The names themselves are good descriptions of them, but just to clarify. We start off implementing our design via either block diagrams or hardware descriptive language (such as VHDL or Verilog). Once we finished making the design, we then compile our design. In analogy of compilation of programs, the purpose here is to convert the design into a logic model that can be simulated easily. Upon successful compilation, the next step is to setup a simulation of our design to make sure it works correctly. If we find an error, then we want to go back and modify the design. Once simulation shows a working design, we then download it onto the board. The last step isn't always necessary, but it lets us see our design physically working.

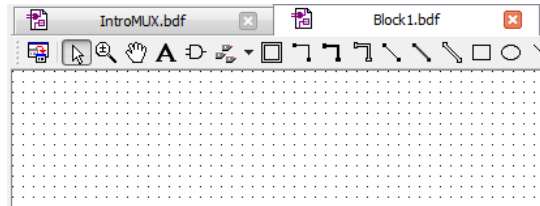
Creating a Project



To start making a design, we need to create a new project. It's a simple process and the following steps will guide you:

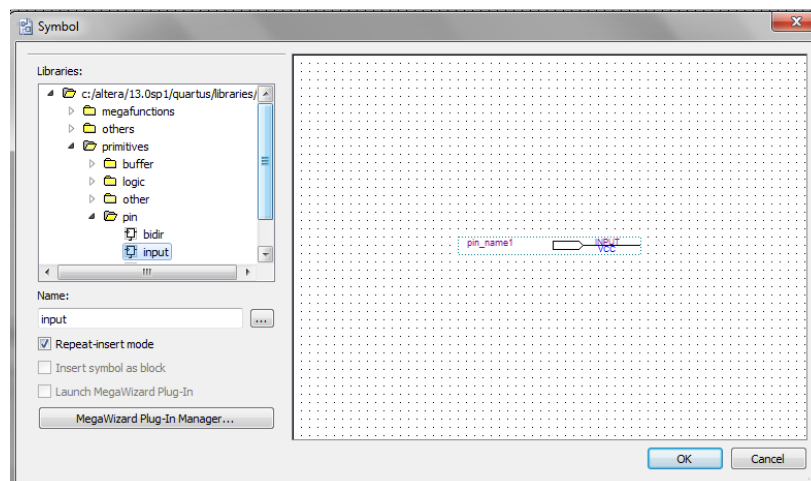
1. Click File, and then New Project Wizard.
2. Click Next. Now you should be able to specify a directory for the project, along with a name.
3. Click Next. Here we can add files we have already written, but for now we have none.
4. Click Next. Now we can select our board. Under device family select **Cyclone II** and then under available devices, find **EP2C20F484C7**.
5. Click Next. Here we can specify any other software tools to use, but we don't need do anything here.
6. Click Next to bring you to a summary and then Click Finish to create the project.

Making a Design


Now that a project is made, it is time to make a design. As mentioned earlier, there are two ways we can make a design. The first way is done graphically with block diagrams. This is what we will be doing in this lab. The second way is to use a hardware description language, such as VHDL or Verilog. To start, go to File and click New... and then select “Block Diagram/Schematic File” under Design Files. We should now have a grid-like section with a menu bar at the top or on the side.

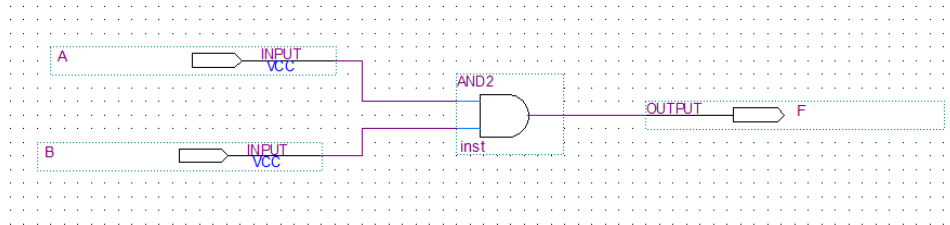


To insert logic gates, inputs, or outputs, we click on the symbol tool  to bring up a window where we can search for a component. For instance, type “input” and observe that it finds input from the pin folder. If the search doesn’t find anything, we can also manually find it. So if we want an AND gate, we can look in the logic folder and find AND gates of various sizes. Select the input symbol, and then click OK. Now anywhere we click in the block diagram will place that symbol there. For our design, we need two input pins. Once they are in place, click the selection tool . Double click on the name of each input pin, and change their names to A, and B. After this, add an output pin and rename it F.




Example: Find the input pin

Now we need to add our logic. Add a two-input AND gate, you can find it in '/primitive/logic'. You don't need to name the logic gates. Once we have all the input and output pins, and the logic gates, we need to connect them together with wires. To do this we will use the orthogonal node tool . Just click on the edge of a gate or pin and drag your mouse. You might want to play around a little bit to get used to adding gates and connecting them.



Example: design

Compiling a Design

When our design is finished, it is time to compile it to allow for simulation. You may need to save it first. Click the compile button  to start this process. Compiling may take a few minutes. When compiling is done, Quartus will either say success or list any error messages.

Simulating and Testing a Design

For our simulation and testing, we will use a program called ModelSim. To set up your Quartus project to use it, follow the given steps.

1. Create a script that will be your simulation (a .do file), download the one given on Piazza. Put the file in your Quartus project. Go to **Project** and then click **Add/Remove Files in Project** and browse for your file.
2. On the **Tools** menu, click **Options**. In the dialog box, click **EDA Tool Options**. Specify the path to ModelSim-Altera executable (for example, C:\altera\91\modelsim_ase\win32aloem). Then click OK. (Notice, it seems that you can only use this version 9.1 in the lab computers)
3. On the **Assignments** menu, click **Settings**. In the dialog box, click **Simulation** under **EDA Tool Settings**. Change Tool name to "ModelSim-Altera". Change Format for output to VHDL. Select '**Script to compile test bench**' option and specify your .do file. Then click OK.

4. Now when you compile your Quartus project, ModelSim should open and run your script file for your design. If you've already compiled, you can go to **Tools->Run Simulation Tool->Gate Level Simulation**. Choose either model then click OK. Notice, if the task window has yellow '?' on each item, you need to compile again until you get green 'v'.

Deployment

We can download our design onto the boards to see the design in action. Go to the block diagram and right-click a pin. Select Locate->"Locate in Pin Planner". Then for each of your nodes, select the following pin locations:

A (input 0): PIN_L21	this will assign slider switch 0
B (input 1): PIN_L22	this will assign slider switch 1
F (output): PIN_Y21	this will assign green LED 7

Once the assignments are finished, close the pin planner and recompile. After a successful compilation, go to Tools->Programmer to open the programming window. Make sure your file is listed, then click "Start" to download the design to the board. Now using SW0 and SW1 as A and B, and LEDG7 as F, you can see the design work!

Archive Quartus Projects

For submitting Quartus projects, we want to submit it as an archive file (.qar). This is done within Quartus. Go to **Project -> Archive Project**. It will ask for a name, which you can leave as is, then click OK. All you need to submit is this .qar file and a text file with answers to any in-lab or post-lab questions. **If you submit your project as a zip, rar, or any other format you will lose points.**

Part 1 Questions

Using the project you created in part1, do each of the following **individually** and explain what happens by using the red error message in the message window.

- a. Open your design file and connect the wires from the inputs together (before they go to the logic gate). Compile and tell what error occurs.

- b. Sometimes the warnings Quartus gives during compilation are actually relevant. Delete the wire from the logic gate to the output pin and recompile. Which of the warnings are relevant to you?
- c. Run a simulation and leave ModelSim open. Now go back to Quartus and run another simulation. What error do you encounter?
- d. Now open your test script (.do file), and change the first line to “vsim files”. Run a simulation. What happens?

Part 2

Using Existing Design Files

Just learning how to use Quartus can be a cumbersome process. Dealing with some of the errors that you’re sure to encounter, and knowing effective ways to test and debug a design can be yet another challenge to Quartus. In this lab you’ll get practice using previously made files in a project, learn some testing and debugging practices, and encounter a few more errors.

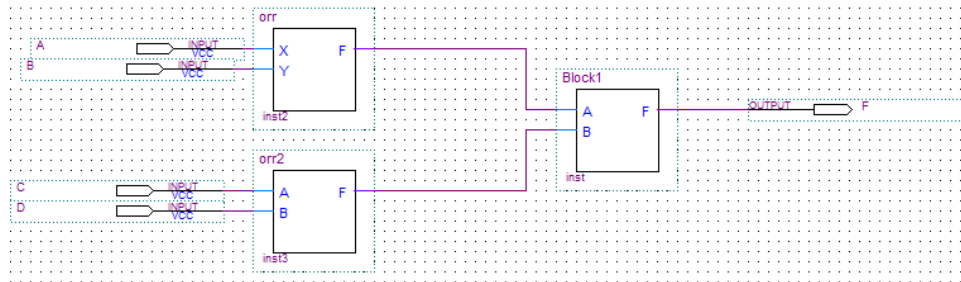
Download the files from Piazza and **create a new project**. In project creation wizard, make sure to include the downloaded files and your files from part1. For the downloaded file, copy them into your new project folder. In general, make sure all the files that you’re using for a project exist in the project folder (i.e. **don’t reference files in other folders**).

Open your design file from part1. Go to Project->“Set as Top-Level Entity”. This makes Quartus know to compile this file. Now compile. Then go to File->“Create/Update”->“Create Symbol Files...”. Save the file “*.bsf” in your project folder. If for some reason Quartus complains that the file is read only, there is a simple solution to trick Quartus:

1. Open Notepad. Don’t type anything.
2. Now save the file, ‘save as type→ All files’, and name it as <design>.bsf. For example, if your design file was lab5_1.bdf, you would name the file “lab5_1.bsf”. Make sure to change the file type to all types.

Now create the following in a **new design file** (e.g. lab5_2.bdf). The ‘Block 1’ is your design in part 1. Set it as the top-level entity and compile. Symbol files are treated just like a

regular gate, except there will be another folder called “Project” when you go to add a gate. For the time being, ignore any warnings.



Testing

Once you have successfully compiled, create a test script (.do file) for your new design. For this circuit, the desired functionality is if the two inputs into Block1 (**your design in part 1**) are both 1, then the output F should be 1, because it is an ‘AND’ logic gate. The two files ‘orr’ and ‘orr2’ are supposed to work as an ‘OR’ logic gate.

So now the question is how can you test this and be convinced that it works? Usually we want to make sure that the output is correct for **all possible inputs**. Copy this test script to a text editor (e.g notepad), and save it as ‘*.do’ and use it as your simulation script (refer to part 1, how to add a simulation test script, ‘**simulating and testing a design**’ part). Observe the output. Is it what you expect? Save your simulation output by going to File->Export->Image.

```
vsim lab5_2
```

```
#the name is the same as your design file
```

```
view wave
```

```
add wave A
```

```
add wave B
```

```
add wave C
```

```
add wave D
```

```
add wave F
```

```
force A 0 0, 1 20 -repeat 40
```

```
force B 0 0, 1 40 -repeat 80
```

```
force C 0 0, 1 80 -repeat 160
```

```
run 320
```

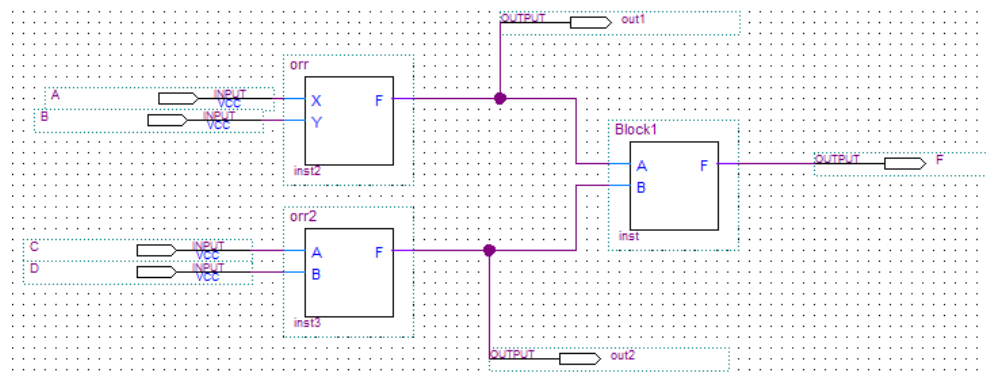
Question 1

Guess what does “U” values mean in ModelSim simulation.

Debugging

Look more closely at the test script and you’ll see that input D was never given a value. Your test script itself can cause these red lines in simulation. Fix the test script to give a value to D: “force D 0 0,1 160”.

Run the simulation again and observe the output. Now you should realize that something else is wrong, because in no case, F can be 1, but where do you start looking? A good method of testing is to add output pins to intermediary wires in the design. Connect output pins as shown below:



Compile this design and then add the output signals to the test script by ‘add wave (the name of your new output pin)’. Run the simulation. Now look over the results and determine in which component the error is coming from. Once you know which one, look at that design file, find the mistake and fix it. When you think you have it fixed, run the simulation again and check for the correct results. When you have it works correctly, **save your simulation output**.

Question 2

Delete the design file (.bdf file) given on Piazza from your project **folder** and compile. What happens? What’s the detailed information you get from compilation?