CCNY - CSC 343 - Fall 2011

Lab 3 – Introduction to Quartus and Finite State Machines

csc343ta@gmail.com

https://sites.google.com/site/csc343fall2011

TABLE OF CONTENTS

OBJECTIVE: 3

PART I – INTRODUCTION TO QUARTUS 3

Creating a new Quartus project file 4

First block diagram – basic multiplexer 7

Compiling and simulating your design 11

Create VHDL from BDF File 14

Create Symbol Block from VHDL File 14

PART II – FINITE STATE MACHINES 16

Circuit I 16

Circuit II 19

Testing circuit I on the DE2 board 20

Testing circuit II on the DE2 board 23

PART III – FSM DESIGN 24

Objective:

To gain an understanding of the Quartus software, how to simulate designs on the DE2 board and the concept of finite state machines. You will design & implement using these concepts.

Part I – Introduction to Quartus

One purpose of this lab is to learn how to use Quartus and Modelsim software. Most, if not all of you have never used these software packages. This lab is a step-by-step guide on how to create a Quartus project file (qpf) and simulate your design in Modelsim. You will be creating a simple 2-to-1 multiplexer to do this. Below, I briefly explain what a multiplexer is. However, you can google multiplexers if you need further explanation. To those of you who are computer engineering majors, your textbook from EE210 (switching systems) will be very helpful to your labs here. In this part of the lab, we will create two circuits and use them in Quartus – a two to one multiplexer and also a four to one multiplexer.

Quartus is an important tool in developing digital circuits. Before an engineer builds a building, he has an architect design it. The design then may be simulated in special software to see if it can me made structurally sound. This is what Quartus does for us. We use this software to design different kinds of digital circuits BEFORE we implement them. Later on in the semester, you will be testing your designs on a programmable chip. However, it would be pointless to put your design on the chip if your simulation does not work properly on Quartus. The steps in designing digital circuits are as follows:

1) Design circuit using VHDL or visual design in Quartus.
2) Simulate design in Quartus or Modelsim. Modelsim is a more powerful simulating tool.
3) Test digital design in programmable board (in our case, the DE2 board).

The flowchart below gives more detail on the design process of digital circuits.

Creating a new Quartus project file

1) Right click the desktop and click “terminal. Type “Quartus” in the box that appears to open the Quartus software. Click on file=>new project wizard. The following box will appear.

You will see three boxes that you can type in. The top box will ask for the directory that you wish to save your project in. You can click on the small box to the right and save it to a directory you feel comfortable with.

2) After doing this, click on the middle text box. This is where you name your project. I recommend calling it “Basic_Mux”. You will notice while typing, the name of the project will also be typed in the third box as well as the second. Leave it like this for simplicity for now. Click “Next” on the lower right corner.

3) You will now see the following menu.

This menu screen will eventually become important. For now, we will ignore it.

4) Click on “Next” to see the menu shown below.

This is the menu where you select the programmable board you will use to test your designs. In the “Family” drop-down menu, chose “Cyclone II”. Under “Available Devices”, chose EP2C35F672C6. That is the name of the board we will be using for this lab. This will also be known as the DE2 board (it is much easier to say than the EP2C35F672C6 board). After selecting the proper board, click “Finish”. You have just created a new Quartus Project File.

First block diagram – basic multiplexer

A multiplexer is a circuit design that takes 2 or more inputs, and has only one output. Its purpose is to select which one of those inputs will be driven through to the output. There is one more input called a selector. It *selects* which input will be driven to the output. The selector is at least N bits long for 2^N inputs. This means that if there are 2 inputs, the selector is 1-bit long, since there are 2^1 inputs, with N=1. If there are 8 inputs (2^3 inputs), the selector is 3-bits long. For now, we will be dealing with a multiplexer that has 2 inputs, making the selector 1-bit long. As you read on, you will learn how to create a simple one.

1) To open a blank block diagram file, click on file=>new. The following menu will appear.

Click on “Block Diagram/Schematic File” under the “Device Design Files” tab. A large white grid will appear. This will serve as a canvas for your work. Any block diagrams you create will be on this grid.

2) Now that you have a new block diagram, we will create a basic 2-to-1 multiplexer. This multiplexer will consist of two AND gates, an OR gate, and a few inverters. A schematic of a 2-to-1 multiplexer is shown below.

3) To the left of our blank grid, you will see a long vertical toolbar with many symbols to click on. Click on the third one from the top. It looks like an AND gate, and if you leave your cursor on it without clicking, it will say "Symbol Tool". After clicking on this, you will see a box come up like the one shown below.

The folders on the left contain circuits and gates we will use for this design, as well as every other design for all other projects. Although these will not be the *only* design blocks we will be using, they will be used very frequently.

4) From the two folders to the left, click on the plus sign to the left of the bottom folder. It will drop down 3 more folders. Of those three folders, click on “primitives”. This will drop down drop down 5 more folders. Next, click on “logic”. This will drop down many simple circuits and gates for us to choose from. We will start by clicking on “and2”. This is a two-input, one-output and gate. After clicking on this gate, click “ok”. You will see that you can now place this gate wherever you want on the grid. Click anywhere on the grid to place a gate down. You can place as many of this gate you want on the grid until you right click and hit cancel. Try to place the gates in such a way that it will look like the schematic diagram of the multiplexer shown previously.

5) Next, click on the “symbol tool” box again. Under the “primitives” folder, click on “pin”. You will see three choices of pins. Click on “input” and press ok. Place one input pin to the left of each AND gate. To connect the input pins, first click on the box that says “Orthogonal Node Tool”, which is usually two boxes under “Symbol Tool”. Place the cursor over the right side of the pin. You will know the cursor is in the right place when it turns into a + symbol. Click and drag the cursor to either input of the AND gate and release the mouse button. You should now have something that looks like the picture below.

6) Now, in the “logic” folder in the “symbol tool” menu, click on the NOT gate. Place the NOT gate somewhere between the two input pins. Then, place an input pin going into the not gate. Connect two more wires to look like the image below.

7) Next, click on the “logic” folder again in the symbol tool menu. Click on the gate labeled “or2” and click ok. Place this gate to the right of the two AND gates. Connect the output of the AND gates to the inputs of the OR gate. Your design should look similar to the one shown below. Place an “output” pin to the right of your OR gate. Your design is now complete! It should look similar to the image below.

It would be wise to change the name of all the input and output pins. I recommend calling the input pins into the AND gates “C0” and “C1”, the input pin into the inverter “Selector”, and the output pin “Output”.

Compiling and simulating your design

COMPILING

To simulate your design, simply click on processing=>start compilation. If your design has no loose wires or incorrect inputs, all the bars under the status menu on the left side will reach 100%.

SIMULATING

1) Now that you have compiled, you are ready to simulate. First, click on file=>new. In the box that appears, click on the “other files” tab. In the choices that follow, click on “vector waveform file”. A new screen will appear on top of your design. It will look like the image below.

2) Looking at the image above, I placed a red box that says “right click here”. As I am sure you have already guessed, you are going to right click in that area. In the menu that appears, click “insert node or bus”. You will see the following box appear.

From this box, click on “Node Finder”. The menu that will appear is shown below.

Make sure the drop down box called “filter” says “Pins: all”. Click on “List” on the upper right of this screen. A List will appear on the left menu called “nodes found”. Those are the inputs and outputs of our 2-to1 multiplexer. Click on the double arrow >> to send all those inputs and outputs to the “selected nodes” menu. Click “ok”.

3) You are now back to the original vector waveform file screen. You will now notice all of your inputs and outputs. To simulate, we need to give all the inputs initial values. Right click on the first input and select “Value”. From the menu that appears, click on “Forcing Low”. Next, do the same with the second input, but select “Forcing High” instead.

4) Right click the selector. Under the “Value” menu, select “clock. A box will appear. Change the vale of period from 10 to 5, and click ok.

5) Now, we can simulate. Click on Processing=>Start Simulation. If all goes well, you will see a message that says “Simulation was Successful”. By default, Quartus sets its simulator to “timing simulation”. This means that the resulting simulation will have delays. Look carefully to see if your multiplexer functions as it should; taking into account its delay. You can right click and click on “zoom” to zoom in and out. Your resulting waveform will look similar to the image below (this is a zoomed out image).

Create VHDL from BDF File

You can have Quartus create VHDL code automatically from your block diagram files. This is done very easily.

1) Open the 2-1o-1 multiplexer design you just created if it is not open already.

2) Click anywhere on the block diagram file to give it focus.

3) Click File=>Create/Update=>Create HDL design file for current file. When you do, a box will appear.

4) Where it says “file type”, make sure you have “VHDL” selected. Click OK. You have now created a VHDL file!

5) Your VHDL file may not show up automatically. If that is the case, it is located in the project folder you created for your project. Simply look in the directory of your project. It should be named the same as your block diagram file. However, the extension will be .vhd.

Create Symbol Block from VHDL File

1) Click file=>new. In the box that appears, select “VHDL File”. Save this file as “mux_gate”.

2) Write the VHDL code below.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
entity mux_gate is
      port (
            I0 : in STD_LOGIC;
            I1 : in STD_LOGIC;
            I2 : in STD_LOGIC;
            I3 : in STD_LOGIC;
            Sel : in STD_LOGIC_VECTOR(1 downto 0);
            Output: out STD_LOGIC
      );
end mux_gate;

architecture behav of mux_gate is begin
      with Sel select
            Output <=
                  I0 when "00",
                  I1 when "01",
                  I2 when "10",
                  I3 when "11";
end behav;
```

This VHDL code will describe a 4-to-1 multiplexer with a 2-bit selector. The selector must be 2-bits because there are four inputs. Therefore, the selector must have four possibilities. They are 00, 01, 10, and 11.

3) Compile the code.

4) Click File=>Create/Update=>Create Symbol files for current file. This will create a symbol block out of your VHDL code.

5) Open a new block diagram file. Save it as “VHDLmux”.

6) Click on “symbol tool”. In the “libraries” menu on the left side, there should be a folder on the left side called “Project”. This folder contains any symbol blocks that you, the user, have created. If you click on that folder, you should see the “mux_gate” that you have just created. Select it.

7) Place that block on the block diagram file. Add input and output pins to the five inputs and the output. Rename the pins anything you like, EXCEPT the pin that goes into the input labeled “Sel[1..0]”. You can do this by double clicking a pin and type in the name in “pin name(s)” field. Your block diagram should look something like the image below.

8) Double-click the pin that goes into the input named “Sel[1..0]”. Rename this pin “Sel[1..0]”, which is the same as the name of the input in the symbol block. The [1..0] part of the name means that the input is 2-bits long, going from bit 1 down to bit 0.

9) Compile and simulate your design. When simulating, make the selector a “counter” instead of a “clock”. This is so all inputs I0 though I3 have a chance to make it to the output. Does the multiplexer work correctly?

Part II – Finite State Machines

Circuit I

We wish to implement a finite state machine (FSM) that recognizes two specific sequences of applied input symbols, namely four consecutive 1s or four consecutive 0s. There is an input *w* and an output *z*. Whenever *w* = 1 or *w* = 0 for four consecutive clock pulses the value of *z* has to be 1; otherwise, *z* = 0. Overlapping sequences are allowed, so that if *w* = 1 for five consecutive clock pulses the output *z* will be equal to 1 after the fourth and fifth pulses. Figure 1 illustrates the required relationship between *w* and *z*.

Figure 1. Required timing for the output *z*.

You are to write VHDL code for the FSM in Figure 2. In this code describe the state table for the FSM by using a VHDL CASE statement in a PROCESS block, and use another PROCESS block to instantiate the state flip-flops. You can use a third PROCESS block or simple assignment statements to specify the output *z*.

Figure 2. A state diagram for the FSM.

A suggested skeleton of the VHDL code is given below. Observe that the present and next state vectors for the FSM are defined as an enumerated type with possible values given by the symbols *A* to *I*. The VHDL compiler determines how many state flip-flops to use for the circuit and it automatically chooses the state assignment.

(Starter Code)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY Part_I IS
  PORT (
--Enter your code here
);
END Part_I;
ARCHITECTURE Behavior OF Part_I IS
  TYPE State_type IS (A, B, C, D, E, F, G, H, I);
  SIGNAL y_Q, Y_D : State_type; -- y_Q is present state, y_D is next state
BEGIN

  PROCESS (w, y_Q) -- state table
  BEGIN
       case y_Q IS
             WHEN A =>
                  --Enter your code here
             WHEN B =>
                  --Enter your code here
             WHEN C =>
                  --Enter your code here
             WHEN D =>
                  --Enter your code here
             WHEN E =>
                  --Enter your code here
             WHEN F =>
                  --Enter your code here
             WHEN G =>
                  --Enter your code here
             WHEN H =>
                  --Enter your code here
             WHEN I =>
                  --Enter your code here

       END CASE;
  END PROCESS;

  PROCESS (Clock)
  begin
  if (rising_edge(clock)) then
      --Enter your code here
  end if;
      --Enter your code here
  END PROCESS;
END Behavior;
```

Complete the above code and stimulate it in Modelsim. The above code is only one example of doing this part of the lab, you can write your code in any way you wish as long as it uses states and has more than one process.

Circuit II

Design a circuit similar to the one from part I, applying these changes:

- The output z changes after eight occurrence of the same bit and not four.
- The output z changes to 0 after eight consecutive 0s and to 1 after eight consecutive 1s.

Above is the timing diagram for this design.

Testing circuit I on the DE2 board

To test your design on the de2 board you first need to transfer your design to the Quartus II software. Follow these steps to test your design from the first part of this lab on the de2 board.

1- Open Quartus II

2- Click on file=>new project wizard. The following box will appear

3- Choose the directory you wish to save your project. Enter Part_one_Quartus for the name of the project and click next.

4- Click on the browse button on the top right corner and select your VHDL code from part one, and press Add. (You can also add files to your project after creating the project using Project->Add\Remove Files in Project)

5- Click on “Next” to see the menu shown below.

This is the menu where you select the programmable board you will use to test your designs. In the “Family” drop-down menu, chose “Cyclone II”. Under “Available Devices”, chose EP2C35F672C6. That is the name of the board we will be using for this lab. This will also be known as the DE2 board (it is much easier to say than the EP2C35F672C6 board). After selecting the proper board, click “Finish”. You have
just created a new Quartus Project File.

6- Open your VHDL file from part one from file->Open menu.

7- Click on File->Create\Update\Create Symbol Files for Current File

8- Click on File->New, and choose Block Diagram/Schematic File.

9- To the left of our blank grid, you will see a long vertical toolbar with many symbols to click on. Click on the third one from the top. It looks like an AND gate, and if you leave your cursor on it without clicking, it will say “Symbol Tool”.

10- The folders on the left contain circuits and gates we will use for this design, as well as every other design for all other projects. Although these will not be the *only* design blocks we will be using, they will be used very frequently. Choose Part_I under project and press “OK”, and place it on the grid.

11- Click on the Symbol Tool again and this time choose “input” from libraries->primitives->pin. Place three of them on the grid and connect (using the mouse) one to each input.

12- Place an output pin on the grid and connect it to the only output. Here is how the block diagram will look like. (Double click on the pins to change their names)

13- Now we need a pin assignment file. Pin assignment files are files that assign the input and output pins in your design to various parts of the board. For example, you can assign input pins to switches or push buttons that change the binary value of your input from 0 to 1 or visa-versa. You could also assign output pins to LED’s, seven-segment displays. You must decide how you

want to assign the pins in your Quartus design. Pin assignment files are written simply as regular text. Here is the pin assignment file for the above design. Create a new text document and copy and paste the following lines to it.

To, Location
Reset, PIN_V2
Clock, PIN_W26
w, PIN_N25
z, PIN_AE23

Each pin in your design is connected to a part of a de2 board. We get the assignments from the Altera DE2 board user's manual, starting from page 28. The link for it is:
ftp://ftp.altera.com/up/pub/Webdocs/DE2_UserManual.pdf
14- Save your pin assignment as a text file.

15- Click assignments=>import assignments.

16- Click on the box with the three dots to the right of the text box. Select the pin assignment file you just saved. Click OK. You should see your pin assignment appear on your block diagram file.

17- Click on Project->Set as Top-Level Entity, and then compile your design,

18- Click Tools=>Programmer.

19- Check the box under "Programs/Configure".

20- Click "Start". If the blue bar on the upper-right hand corner goes to 100%, you should see your design on the DE2 board.

Testing circuit II on the DE2 board

Test circuit II on the DE2 board like you did for circuit I

Part III – FSM Design

This final part of the lab involves only design and specification. Recall the FSM example given in the lecture – this was the electronic combination door lock. You were given the following behavior requirements:

- Combination is Two-One-Eight
- User has a numeric keypad to enter the combination
- On correct combination, door is to unlock
- Shall remain unlocked until opened and then closed

You were also given the following state diagram:

Also recall that some weaknesses were pointed out in the design. Notably that the combination was "hard-coded" and that the user might enter problematic input such as 2-1-8-5 or 7-2-1-8. Your task is to adjust the circuit to address such weaknesses. You have some creative freedom to do so – you may adjust the behavior requirements slightly, or adjust the device itself. Consider defining what buttons the user might push and how they should behave (consider an ATM pad – does it only have numbers?). Note changing the requirements to something like "Always open" or "User has to bring their keychain" isn't acceptable!

There is no code required for this section. However at the very least you should have a redesigned state diagram and a thorough description of both behavior requirements and any changes you've made for the device. Place this part in a separate section of your lab report as if it were a circuit. This is a very "real world" exercise – typically devices such as this must be thoroughly specified before implementation begins.