ECE275: Sequential Logic Circuits
Lab 2: Introduction to Verilog and Multiplexers

Pascal Francis-Mezger

September 21, 2020

Contents

1 Lab Overview

The goal of this lab will be to learn the basics of the FPGA boards, and of Verilog by creating multiplexers. The first stage will have you utilize the schematic function of Quartus where you can design a digital logic circuit using graphical versions of the digital logic symbols. This will feel very similar to how you create logic in class. You will then move on to creating this same logic in Verilog.

2 Part 1: Graphical Section

2.1 Create a New Project and Schematic

Create a new Quartus project in the same method as the previous lab, with a name specific to this section of the lab. Something along the lines of lab1part1. Make sure to select the correct FPGA version. Refer to the Lab 1 document for the steps if you do not remember. In this project DO NOT create a Verilog file. We will instead be creating a Block Diagram/Schematic File as shown in Figure 1.

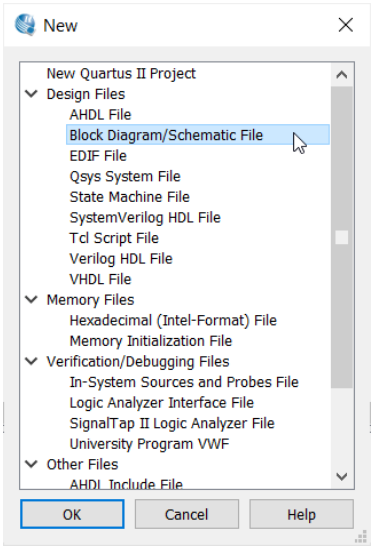

Figure 1: Create a new Block Diagram/Schematic File

2.2 Multiplexer Overview

A multiplexer is in essence a digital selector. One of the inputs allows you to select which of the other inputs would be moved to the output. A simple

multiplexer (as we are building in this section) would have 3 input bits. In our case we will label them as s, x, and y. s would be our selector bit, where if it is a 0, then the value of x is present on the output. If it is a 1, then the value of y is present on the output. If there were more inputs to select from, then s would have to be larger to be able to select from the larger amount of states. For example, if instead of just x and y, we had w, x, y, and z, then s would need to be able to select from four states. To accomplish this s would need to be two bits, where if s is 00, then the value of w is on the output, 01 gives x, 10 gives y, and 11 gives z. The size of the other inputs is also arbitrary. w, x, y, and z could be a single bit or any number of bits, but it does not matter as s just selects which one to move to the output. The symbol for a multiplexer is shown in Figure 2 and represents the multiplexer you will be building today, with an output m.

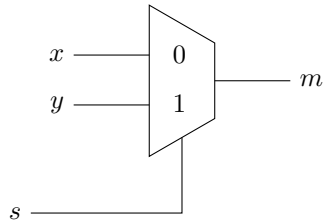

Figure 2: Simple Multiplexer

This can also be represented by the digital logic circuit shown in Figure 3

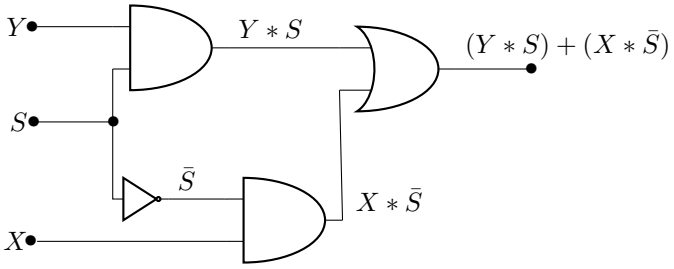

Figure 3: Simple Multiplexer Logic Diagram

2.3 Create the Multiplexer

The schematic file you created in the previous step should be open. Go ahead and save it right away. You will need to save it as name of your top level, so something like lab1part1top.bdf. .bdf files are the Block Diagram/Schematic files, while .v represents Verilog files.

Next, use the pin tool at the top of the schematic (shown in Figure 4) to create 3 input pins and one output pin.. You will need to use the arrow to the right of the box to change the selection to output to create the output pin. These represent pins x, y, s, and m for the multiplexer. Generally it is a good idea to create your inputs on the left and outputs on the right so logic flows intuitively from right to left.

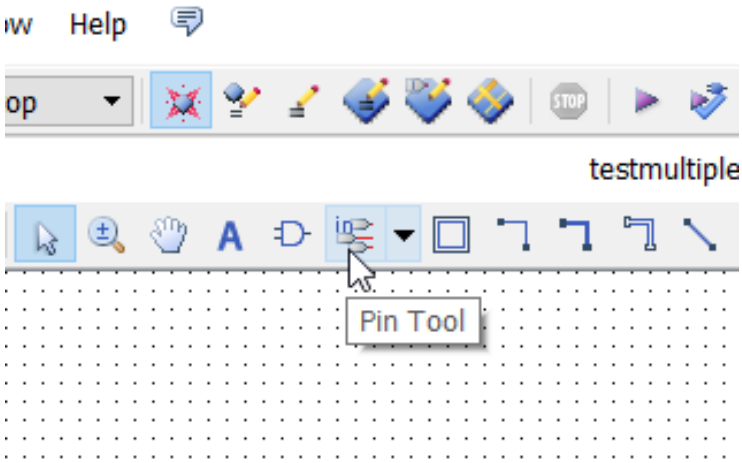

Figure 4: Quartus Schematic Pin Tool

Next, create the logic elements you will need to build the multiplexer from Figure 3, such as the 2 AND gates, 1 OR gate, and 1 NOT. These symbols are under the symbol tool to the left of the pin tool. Make sure to select the logic folder in the symbol tool to find these as shown in Figure 5.

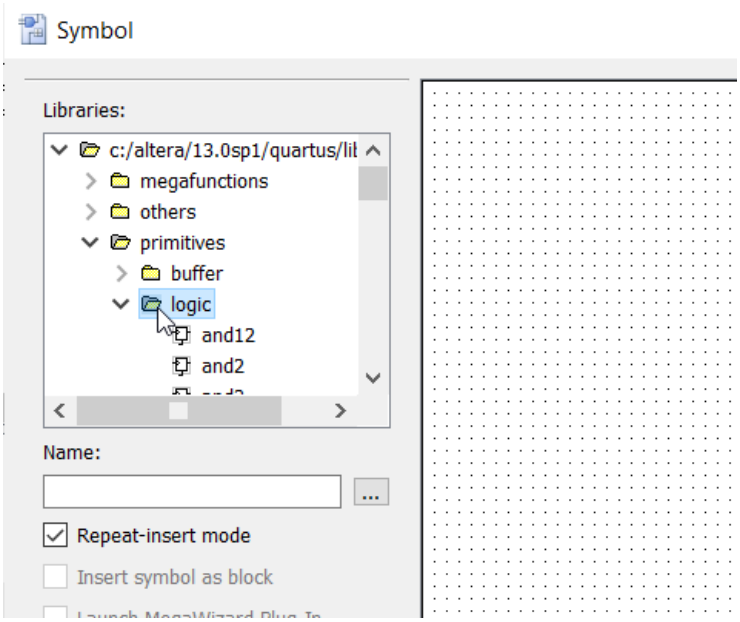

Figure 5: Select the Logic Elements from the Symbol Library

Next attach wires to complete the logic. To start a wire you just need to click and drag from any of the pins on any of the inputs/output/gates.

2.4 Label Pins and Set Pin Assignments

You should now have a graphical logic diagram that represents a multiplexer. The last step is to label the pins and then assign them to real world devices (switches for the inputs, LED for the output). The best idea for the pin labeling is to use the pin names from the .qsf files, so you can copy and paste pin assignments from there. Open the qsf to verify these names, but the switches should be labeled as SW[0], SW[1], etc. and the green LEDs would be LEDG[0], LEDG[1], etc. For this lab use SW[0] as s, SW[1] as x, and SW[2] as y. LEDG[0] will represent the output m. Make sure you understand this portion, as these pins addresses will not be explicitly given to you in the future. If you do not understand these names and how they relate to the .qsf files, ask a TA or the instructor.

After you label the pins select the relevant lines from the .qsf file and paste them into the TCL console in Quartus to assign the pins.

2.5 Run Compilation and Program the FPGA

At this point you should be able to run compilation and then program to the board as you did in Lab 1. If you run into errors during compilation please try to read through the error messages and diagnose the error yourself before asking a TA.

Create a truth table of x, y, s, and m from manipulating the switches and have it checked off by a TA to complete this section of the lab

3 Part 2: Verilog

3.1 Basics of Verilog

The first section was meant to help you see the connection between what you program in Verilog, and the digital logic representation. You will now be creating the same multiplexer you created graphically, but now by utilizing Verilog. Below is the code from last weeks lab. We will take a closer look at it today so you can modify it to create your multiplexer.

```
module lab1top(
    input [9:0] SW,
    output [9:0] LEDG
);
    assign LEDG[9:0] = SW[9:0];
endmodule
```

The first line "module lab1top(" is similar, but not identical to a C function. You will learn more about the differences later, but for now just see it as the module that has your top level name will be the module that runs on your FPGA. My top level was lab1top, so make sure you modify that to match your top level.

The next part inside the the parenthesis, "input [9:0] SW, output [9:0] LEDG); signifies the variables to be used in the module. You can just name the variables in the parenthesis without giving a size or type, as long as you specify those in the module. With this being a simple module, I just designated the switches as inputs, and the LEDs as outputs right in the parenthesis. The [9:0] part of the variable declaration indicates to create the variables SW[0], SW[1],

SW[2],...,SW[9]. These can be utilized similar to C arrays where you can assign to a range SW[5:3] or to a single value SW[3]. You can probably guess that the input keyword signifies those are either inputs from other digital logic, or real world input devices such as switches. Outputs signify being output to other digital logic, or tied to real world outputs such as LEDs. Other options exist, such as wire or register, that would represent a wire holding a value in between logic, or a register to hold a value.

The next part "assign LEDG[9:0] = SW[9:0];" is taking the values of the switches and assigning them to the LEDs. In this case, LEDG[0] is controlled by SW[0], LEDG[1] is tied to SW[1] and so on. The assign statement creates connections between variables similar to how you drew the wires in part 1. The example only utilizes a simple equals, but Verilog supports many math operators such as +,-,*,/, etc. We will be utilizing the more basic boolean logic operators & (AND), | (OR), ∼ (NOT). Use parenthesis to specify order of operations.

3.2 Writing Your Own Verilog

Start the section off by creating a new project, with a blank verilog file. You will be modifying the Verilog code in the previous section to create the multiplexer from Figure 3. The first thing would be to change the inputs/outputs to match our case. The lab would still work if you left them as they are, but it is bad practice to create more variables than you need. We are only using SW[0], SW[1], SW[2], and LEDG[0].

Now change the assign statement to match the logic you need for the multiplexer. You can see this at the far right of Figure 3. The equation in Verilog would be: "assign m = (∼ s & x) — (s & y);" Replace the assign statement in the example Verilog from Lab 1 with this assign statement, and then change the variable names (s,x,y,m) to match the proper input/output addresses. For example, s is SW[0]. After you replace the variables properly, run compilation and then program FPGA. Check that the results for this section match your truth table from the first part.

The last step for this section is to check the created digital logic from your Verilog code. After you do compilation, a compilation list should appear on the left. Navigate to the RTL Viewer Option and double click. This is shown in Figure 6, and the created logic is shown in Figure 7

5.

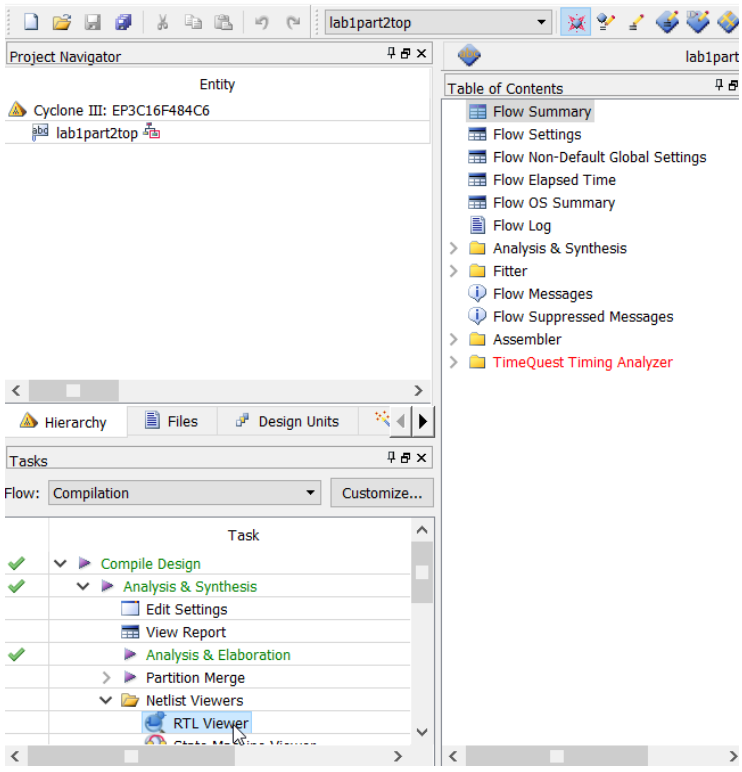

Figure 6: Select the RTL Viewer in the Compilation Window

5.

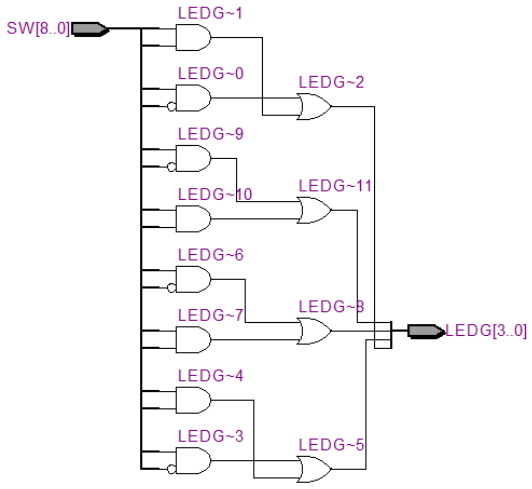

Figure 7: Digital Logic Created by the Verilog Code

4 Part 3: Extend the multiplexer

For this section, create another new project and blank Verilog file. It would be a good idea to copy and paste in your code from the last section, as you can modify it to make this section easier.

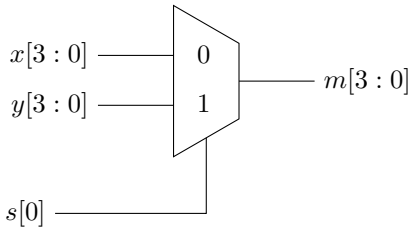

Figure 8: Extended Multiplexer

In the previous part you created a simple multiplexer with a single bit switch,

and 2 single bit inputs. For this portion you will need to create the inputs as 4 bit wide vectors, as well as the output being 4 bits wide. This will show you how to manipulate vectors in Verilog, as it is the most useful basic tool to reduce code complexity. You have already seen how to create input and output vectors from the code from last class (SW[3:0] would be a 4 bit wide vector).

The easiest way to accomplish this section is to just utilize multiple assign statements, and assign each output LED using a very similar equation to the assign line from the previous section. You will just need to update each assign statement to look at the correct location in the vector. So if in the last section the assign line was "assign m = (∼ s & x) — (s & y);", and you replaced s, x, and y with the corresponding SW[X], m with the corresponding LEDG[X] value, in this section if you create multiple lines of the form "assign m = (∼ s & x) — (s & y);" you can update each line with different SW and LEDG values. With this method you would need 4 assign lines.

There are other ways to accomplish the same result to complete this lab, and you are welcome to try a different method if you would like. You just must not use a prebuilt multiplexer module, you must use Verilog code, and in your Verilog you cannot use if statements, or math outside of & (AND), | (OR), ∼ (NOT). Any other method would also likely require you to create a temporary storage variable, you can do this by adding "wire [1:0] VARNAME" to the module variable decleration section. You would replace the [1:0] with the size of the vector you need. You can then use it as a storage variable, but it acts similar to how a wire in between logic gates would.

5 Questions

Answer these questions for a TA at the end of class to complete the lab.

5.1 Question 1

Why is it helpful to use SW[0], SW[1], etc as the variable names instead of using names like s, x, y?

5.2 Question 2

What would the Verilog line "assign LEDG[7:4]=SW[3:0]" accomplish?