ECE275: Sequential Logic Circuits
Lab 3: System Verilog Modules

Zafaryab Haider, zafaryab.haider@maine.edu
Shihab Uddin Ahamad, shihab.ahamad@maine.edu

September 19, 2022

Contents

0 Lab Overview

The goal of this lab will be for you to learn how to utilize System Verilog modules to reuse code in a similar way to C functions.

Verilog modules are similar to C functions in allowing for code re-usability. The difference is that in C, when you call a function, you are re-utilizing the same instructions in memory. In System Verilog and other HDLs, you are actually instantiating new hardware when you utilize a module. Each time you instantiate a module, you will need to give it a unique name.

1 Part 1: Create Something Worth Repeating!

1.1 Binary Coded Decimal and Seven Segment LEDs

In this first section, you will create System Verilog code to represent bits as a digit on a seven-segment LED. You will do this in the format known as BCD (Binary Coded Decimal).

BCD is a method of using groups of 4 bits to represent decimal integers. Where this varies from traditional binary is that 4 bits could theoretically represent 16 values ($2^4 = 16$). With BCD 0000_2-1001_2 represent integers 0-9, but 1010_2-1111_2 are discarded or treated as invalid.

A seven-segment LED is a device with 7 individual LEDs, each representing a piece of a displayed digit. With seven segments, we can display a representation of all 10 digits (0-9). We could also display all letters, but for this lab we will just focus on the numerical digits. Keep in mind that the seven segment sections will turn on when the associated output is set to a 0, and will turn off when the output is set to a 1. This is an example of the opposite logic. This means you can either design in the next step your equations with a 1 representing the segment being on and then negate the statement, or design it with a 0 on the output representing the segment being on and then leave the statement as is. Both options are acceptable. For BCD values above 9, pick turning off all segments, keeping the value at 9, or leaving them as don't care. Any option is acceptable as values above BCD 9 would be considered out of scope.

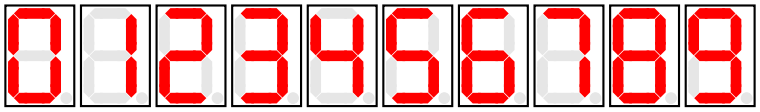

Figure 1: 7 Segment Representations of Each Integer

1.2 Seven Segment LED Truth Table

Your first step in this lab (make sure you read the previous text on BCD and seven segments, do not just skip to this step) will be to create a truth table for the seven-segment LED for a 4-bit BCD value. You will then use the truth table to create equations for each segment in the seven-segment LED for utilizing in the System Verilog code. The format of the truth table has been created for you in Table 1. For each BCD value on the left-hand side, fill in on the right-hand side which LED segments are on. Keep in mind, the segments on the FPGA are on when the ASSIGNED VALUE IS 0! Use Figure 2 to reference for segment names.

BCD Value				LED Segment						
D_3	D_2	D_1	D_0	G	F	E	D	C	B	A
0	0	0	0							
0	0	0	1							
0	0	1	0							
0	0	1	1							
0	1	0	0							
0	1	0	1							
0	1	1	0							
0	1	1	1							
1	0	0	0							
1	0	0	1							
1	0	1	0							
1	0	1	1							
1	1	0	0							
1	1	0	1							
1	1	1	0							
1	1	1	1							

Table 1: Seven Segment BCD Truth Table

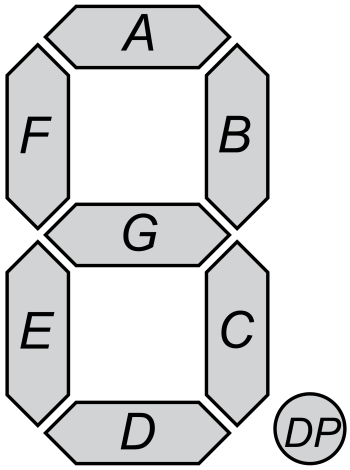

Figure 2: Seven Segment LED Segment Labels

Segment Name	FPGA Pin Name
A	HEX[0]
B	HEX[1]
C	HEX[2]
D	HEX[3]
E	HEX[4]
F	HEX[5]
G	HEX[6]

Table 2: Segment Names vs FPGA Pin Assignments

1.3 Create Equations from the Truth Table

Recall last week, we accomplished the opposite, where the equation was given, and then you used the results to create a truth table. You will have to create the equations from a truth table for this lab. For example the truth table, Table 3, produces the equation m = (! s & x) | (s & y). This can easily be seen to be accurate when viewing the equation and table side by side, but creating the equation is not always an easy task.

x	y	s	m
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Table 3: Multiplexer Truth Table

For this week, you should utilize either the SOP (Sum of Products) or POS (Product of Sums) to create an initial equation for each segment and then simplify it. The simplification step is not required but makes your equations in System Verilog shorter and easier to troubleshoot. You should decide for each segment whether POS or SOP would provide a more simplistic equation.

We will work through creating the multiplexer equation from the truth table, Table 3. First, to decide if we should use SOP or POS, we can count the number of 1's and 0's. If there are more 1's, SOP will create a shorter equation. If there are more 0's, then POS would. In this case, the output m has an equal number of 0's and 1's, shame. We will then just select SOP as it is generally the more intuitive one. This would create the equation:

$$(\bar{x}ys) + (x\bar{y}\bar{s}) + (xy\bar{s}) + (xys)$$

Then doing basic simplification:

$$ys(\bar{x} + x) + x\bar{s}(\bar{y} + y)$$

$$ys + x\bar{s}$$

giving the equivalence

$$ys + x\bar{s} = (!s \ \& \ x) \mid (s \ \& \ y)$$

You should do similar steps for each segment of the LED, giving you 7 equations. These seven equations will be your System Verilog code for the seven segments, HEX_D[6:0].

1.4 Write Your System Verilog Code

For this step, make sure you create a new Quartus Project and then create a System Verilog file in the project. Refer to Lab 1 if you need a refresher on the steps to accomplish this.

For the System Verilog code this week, instead of using the pin assignment names in your code, you will use whatever variable names make sense, and at the beginning of the code, assign the real-world inputs/outputs to these variables. An example of this for the System Verilog code is shown below, with BCDValue being an array of the BCD values and LED_Segment being the 7 segments of the LED display. You can use whatever variable name makes sense to you in place of BCDValue and LED_Segment. Notice that you will be using switches 0-3 to represent the BCD value, with SW[3] representing the most significant bit and SW[0] representing the least significant.

```
module yourprojecttop(
    input [3:0] SW,
    output [6:0] HEX0_D
);
    wire [3:0] BCDValue;
    wire [6:0] LED_Segment;
    assign BCDValue[3:0] = SW[3:0];
    assign HEX0_D[6:0] = LED_Segment[6:0];
    assign LED_Segment[0] = your equation for segment A;
    assign LED_Segment[1] = your equation for segment B;
    assign LED_Segment[2] = your equation for segment C;
    assign LED_Segment[3] = your equation for segment D;
    assign LED_Segment[4] = your equation for segment E;
    assign LED_Segment[5] = your equation for segment F;
    assign LED_Segment[6] = your equation for segment G;
endmodule
```

After completing your System Verilog code, do your pin assignments, and download the results to the board. If the correct segments do not light up for the BCD value set by the switches, double-check your equations, correct them, and re-try. Have your lab TA check you off after completing this section.

2 Part 2: Utilize System Verilog Modules

Verilog modules behave similarly to C functions with the large difference that the hardware is instantiated each time a module is used, so you cannot change the inputs and reuse the hardware.

For this part of the lab, you will make the code you created in the last portion into a module where the hex segments and BCD value can be passed in.

In this way you will be able to use switches 0-3 to represent a first BCD value and switches 4-7 to represent a second. You will then use your created module twice to display two BCD values on two separate seven-segment LED displays.

The format for the System Verilog code for this section is shown below:

```
module yourtoplevel(
    input [7:0] SW,
    output [6:0] HEX0_D,
    output [6:0] HEX1_D
);
    BCD_Display uniquename (SW[7:4],HEX0_D);
    BCD_Display uniquename2 (SW[3:0], HEX1_D);
endmodule

module BCD_Display(
    input [3:0] BCDValue,
    output [6:0] LED_Segment
);
    assign LED_Segment[0] = your equation for segment A;
    assign LED_Segment[1] = your equation for segment B;
    assign LED_Segment[2] = your equation for segment C;
    assign LED_Segment[3] = your equation for segment D;
    assign LED_Segment[4] = your equation for segment E;
    assign LED_Segment[5] = your equation for segment F;
    assign LED_Segment[6] = your equation for segment G;
endmodule
```

The format for instantiating a module is "modulename instanceidentifier (var1, var2, var3...). Similar to a C function the variable var1 would be passed into the first variable name in the module decleration, var2 to the second and so on.

From this, you can see that while using the module BCD_Display multiple times, each time must be given a unique name. Also, remember that each time you use the module, unique hardware will be created. So while this makes your coding easier, theoretically, the created hardware would not differ between using modules or copy/pasting the code many times with unique variable names.

Use your equations from the first part to finish the System Verilog code, then do your pin assignments, compile and download to the FPGA. If you have created your code properly, the BCD value of switch 0-3 should display on seven-segment 0 (HEX0_D), and the BCD value of switch 4-7 should display on seven seven segment 1 (HEX1_D). Have your lab TA check you off after completing this section.

3 Part 3: Add a Multiplexer

You can use the same project you have from part 2 for this section and just extend your System Verilog code.

Create another module that represents the 4-bit multiplexer from Lab 2. This module should take 3 inputs. Two 4-bit values (x and y) and one single-bit input (s). The module should have one 4-bit output (m). The goal of the multiplexer will be to assign m the value of x if the selector is 0 and assign m the value of y if the selector is 1.

After you create the module, modify your top level so that it utilizes the multiplexer to select either x or y to display on the first segment display. Use SW[9] for s, SW[3:0] for x, and SW[7:4] for y. You will need to create a wire variable to hold the multiplexer output to pass to the BCD_Display module. The top level should only contain the wire variable and the instantiations of the multiplexer and display modules. All other programmings should be done inside the modules.

4 Review Questions

4.1 Question 1

If you wanted to add another single-bit variable to the BCD_Display function that could enable/disable the display, what would you need to modify/add? (In the top module where the module is instantiated, and in the BCD_Display module itself)

4.2 Question 2

Why do we need to add a signifier such as "unique name" to a module when we instantiate it?

4.3 Question 3

In terms of hardware created in the FPGA, is there a significant difference between using a module instantiation as we did in the second part or accomplishing the same goal by copying and pasting the first part of the lab twice?

4.4 Question 4

In C programming, you can basically reuse a function an unlimited amount of times without issue. This is not the same for System Verilog modules on an FPGA. What limits the number of times a module could be instantiated on a given FGPA?