Laboratory Exercise 3 – ECE241 Fall 2013

Combinational Logic and Displays

This is an exercise in designing combinational circuits that can drive 7-segment displays, and perform a variety of different functions.

Preparation

You are required to complete Parts I to IV of the lab by writing and testing Verilog code and compiling it with Quartus II. Show your Verilog for parts II and IV to the teaching assistants (pasted into your lab book). For Parts II and III, you must simulate your circuit with QSim (using reasonable test vectors) and show the teaching assistant a printout of your timing diagrams annotated with your tests. For Part III, you must also show the TA your K-map, the optimized logic function, and the list of prime implicants and essential prime implicants.

In-lab Work

You are required to implement and test all of Parts I to IV of the lab, and demonstrate parts I and IV to the teaching assistants.

Part I

In this part of the lab, you will extend the work you did in Lab 2. In particular, you will use seven 7-segment displays to show a word whose characters can be rotated. You will need to use seven instances of a 3-bit-wide 7-to-1 multiplexer (Verilog for this is available on the ECE241 course webpage) and also seven instances of the 7-segment decoder circuit shown in Figure 6 of Lab 2. You should have your word displayed on $HEX6$, $HEX5$, $HEX4$, $HEX3$, $HEX2$, $HEX1$, and $HEX0$.

You are to extend the skeleton code below so that it uses seven 7-segment displays. The purpose of your circuit is to display *any* word on the seven displays that is composed of the characters in Table 1 of Lab 2, and be able to rotate this word in a circular fashion across the displays when the keys *KEY*$_{2-0}$ are toggled. That is, your circuit should produce the output patterns illustrated in Table 1 below. Even though there are only 6 different letters to display (P,A,S,2,4,1), can you see why a 7-to-1 multiplexer is used instead of a 6-to-1 multiplexer?

```
module  part1 (SW, KEY, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6);
    input [17:0] SW;          // toggle switches
    input [2:0] KEY;          // keys
    output [6:0] HEX0;        // 7-seg displays
    output [6:0] HEX1;
    output [6:0] HEX2;
    output [6:0] HEX3;
    output [6:0] HEX4;
    output [6:0] HEX5;
    output [6:0] HEX6;


// instantiate seven 3-bit wide 7-to-1 multiplexers and seven 7-segment decoders.

    . . . code not shown

endmodule

// implements a 3-bit wide 7-to-1 multiplexer
module mux_3bit_7to1 (T, U, V, W, X, Y, Z, S, M);
    input [2:0] T, U, V, W, X, Y, Z, S;
    output [2:0] M;

    . . . code not shown: see course webpage

endmodule

// implements a 7-segment decoder for P,A,S,2,4,1
module char_7seg (C, Display);
    input [2:0] C;            // input code
    output [6:0] Display;     // output 7-seg code

    . . . code not shown: use your Lab 2 implementation

endmodule
```

Figure 1. Verilog code skeleton.

KEY_2 KEY_1 KEY_0	Character pattern						
000	P	A	S	S	2	4	1
001	A	S	S	2	4	1	P
010	S	S	2	4	1	P	A
011	S	2	4	1	P	A	S
100	2	4	1	P	A	S	S
101	4	1	P	A	S	S	2
110	1	P	A	S	S	2	4

Table 1. Rotating the word PASS241 on seven displays from $HEX6$ down to $HEX0$.

Perform the following steps.

1. Create a new Quartus II project for your circuit.

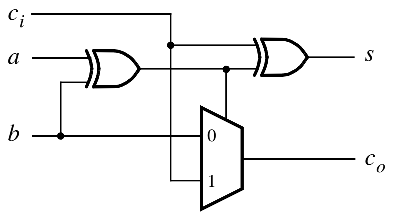
2. Include your Verilog module in the Quartus II project. Connect the keys *KEY*$_{2-0}$ to the select inputs of each of the instances of the three-bit wide 7-to-1 multiplexers. Also connect *SW*$_{17-0}$ to each instance of the multiplexers as required to produce a word such as PASS241. Connect the outputs of the seven multiplexers to the 7-segment displays *HEX6*, *HEX5*, *HEX4*, *HEX3*, *HEX2*, *HEX1*, and *HEX0*.

3. Include the required pin assignments for the DE2 board for all switches, LEDs, and 7-segment displays. Compile the project.

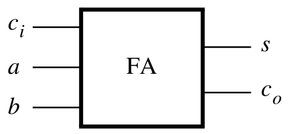
4. Download the compiled circuit into the FPGA chip. Test the functionality of the circuit by setting the proper character codes on the switches SW$_{17-0}$ and then toggling *KEY*$_{2-0}$ to observe the rotation of the characters.

Part II

Figure 2a shows a circuit for a *full adder*, which has the inputs a, b, and c_i, and produces the outputs s and c_o. Parts b and c of the figure show a circuit symbol and truth table for the full adder, which produces the two-bit binary sum $c_o s = a + b + c_i$. Figure 2d shows how four instances of this full adder module can be used to design a circuit that adds two four-bit numbers. This type of circuit is usually called a *ripple-carry* adder, because of the way that the carry signals are passed from one full adder to the next. Write Verilog code that implements this circuit, as described below.



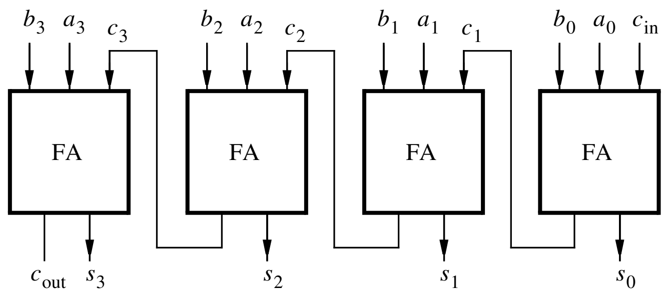
a) Full adder circuit



b) Full adder symbol

b	a	c_i	c_o	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

c) Full adder truth table



d) Four-bit ripple-carry adder circuit

Figure 2. A ripple-carry adder circuit.

1. Create a new Quartus II project for the adder circuit. Write a Verilog module for the full adder subcircuit and write a top-level Verilog module that instantiates four instances of this full adder.

2. Use switches SW_{7-4} and SW_{3-0} to represent the inputs A and B, respectively. Use SW_8 for the carry-in c_{in} of the adder. Connect the *SW* switches to their corresponding red lights LEDR, and connect the outputs of the adder, c_{out} and S, to the green lights LEDG.

3. Simulate your adder with QSim for intelligently chosen values of A and B and c_{in}. Print the simulation waveforms and paste them into your lab book.

4. Include the necessary pin assignments for the DE2 board, compile the circuit, and download it into the FPGA chip.

5. Test your circuit by trying different values for numbers A, B, and c_{in}.

Part III

Given the following 4-variable Boolean function expressed in canonical sum-of-products (SOP) form:

$$f(x_1, x_2, x_3, x_4) = \sum m(5, 7, 8, 9, 10, 11, 13, 15) \qquad (1)$$

For example, minterm 5 is $\overline{x_1}x_2\overline{x_3}x_4$.

Optimize the function using a Karnaugh map (K-map) to find its minimized SOP form. Implement your optimized function as a circuit in hardware using the DE2 board.

1. Draw the K-map for f and use it to optimize the function f.
2. List the *prime implicants* of the function f.
3. List the *essential prime implicants* of the function f.
4. Create a new Quartus II project for the circuit. Write a Verilog module for the optimized circuit.
5. Use switches SW_{1-5} to represent x_1, x_2, x_3, and x_4, respectively. Use $LEDR[0]$ to represent the value of f.
6. Simulate your circuit using QSim with intelligently chosen values of x_1, x_2, x_3, and x_4. Print the simulation waveforms and paste them into your lab book.
7. Include the necessary pin assignments for the DE2 board, compile the circuit, and download it into the FPGA chip.
8. Test your circuit by trying different values for x_1, x_2, x_3, and x_4.

Part IV

Design an *arithmetic logic unit* (ALU) circuit with two 8-bit wide inputs A and B, a 3-bit wide input Q, and an 8-bit wide output Z. Input Q controls the value that is computed by the circuit (based on A and B) and placed on output Z. See Table 2 below for the different computations that can be performed by the circuit. The logical operations (OR, AND, NOT) in the table are *bitwise* operations.

Q_2 Q_1 Q_0	Output Z
000	$\overline{A}$ OR B
001	A OR $\overline{B}$
010	$\overline{A}$
011	A AND B
100	$A + B$ (addition)
101	A NOR B
110	Number of `1`'s in A (in binary)
111	Number of `1`'s in A plus number of `1`'s in B (in binary)

Table 2. Functions that can be performed by the ALU.

Use Verilog to implement your circuit. You do not need to worry about arithmetic overflow for the case when A and B are too large such that $A + B$ (addition) does not fit within 8 bits. (Later on in ECE241, you will learn about how to detect that.) Note: you will re-use this ALU circuit in Lab 4, so do not delete your Verilog implementation!

1. Create a new Quartus II project for the configurable logic unit circuit. Write a Verilog module for the ALU circuit. You are encouraged to use procedural Verilog for this part of the lab (see Appendix A.11 in Brown and Vranesic 3^{rd} edition), though it is not mandatory.
2. Use switches SW_{15-8} and SW_{7-0} to represent the inputs A and B, respectively. Use the red lights, $LEDR_{7-0}$, to represent the value of Z. Use KEY_{2-0} to represent the input Q.
3. Simulate your circuit with QSim for different A, B, and Q values. Ensure the output functionality matches that specified in Table 2.
4. Include the necessary pin assignments for the DE2 board, compile the circuit, and download it into the FPGA chip.
5. Test your circuit by trying different values for numbers A, B, and Q.