Homework 3

ECE3544 CRN:82989

Jacob Abel

October 15, 2018

Note: For problems 2, 3, 4, and 6 you should create a test bench to validate your modules and show correct operation for all input combinations of 0's and 1's (don't worry about combinations with x's and z's). Your PDF solution should include waveforms displaying correct operation of each module.

Problem 1: Explain why the code fragment below will execute forever. Provide a working alternative.

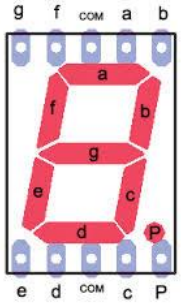
```
1 reg [1:0] loop1;
2 for (loop1 = 0; loop1 <= 4; loop1 = loop1 + 1)
3 begin
4     some statements;
5 end
```

The register is 2-bits wide and therefore has a max unsigned value of 3. This means every possible value is less than 4 and the loop will run even after it overflows back to 0. The below is a functional statement.

```
1 reg [2:0] loop1;
2 for (loop1 = 0; loop1 <= 4; loop1 = loop1 + 1)
3 begin
4     some statements;
5 end
```

Problem 2: A 7-segment display is often used to display decimal numbers. Shown below is one such unit with the segments labelled a-g (we will ignore the decimal point p). Each of the segment outputs is lit on when a '0' is applied to it and is off when a'1' is applied to it. Create a data-flow model using only continuous assignments of a device that accepts a four-bit binary number representing decimal numbers 0-15, and drives the appropriate segments of a 7-segment display to display the number in hex.

Please note: the number 6 should have all segments except "b" on, and the number 9 should have all segments except "e" on. For hexadecimal numbers A, C, E, and F, the display should show upper case, while for B and D, it should show lower case (i.e., the number B should have all segments but "a" and "b" on, while the number D should have all segments but "a" and "f" on).




```
1  // Insert your header here, using the starter files from project
2  // 1a and the documentation lecture as a guide.
3
4  module sevensegdecoder_assign(d3, d2, d1, d0, a_n, b_n, c_n, d_n,
5  e_n, f_n, g_n);
6  input d3, d2, d1, d0; // active high inputs
7  output a_n, b_n, c_n, d_n, e_n, f_n, g_n; // active low outputs
8  //
9  // I N S E R T Y O U R C O D E H E R E
10 // E N D I N S E R T
11 endmodule
```

Incomplete

Problem 3: Repeat problem 2 using a procedural model using only an always block with a case or if statement. The module declaration is given below.

```
module sevensegdecoder_always(d3, d2, d1, d0, a_n, b_n, c_n, d_n,
e_n, f_n, g_n);
input d3, d2, d1, d0; // active high inputs
output a_n, b_n, c_n, d_n, e_n, f_n, g_n; // active low outputs
```

Incomplete

Problem 4: Create a procedural model for a combinational circuit that counts the number of ones in a byte.The input to the circuit should be an 8-bit array named byte_in, with bit 7 as the leftmost bit and bit 0 as the rightmost bit. The output of the circuit should be a four-bit array named ones, with bit 3 as the leftmost bit and bit 0 as the rightmost bit.

Incomplete

Problem 5: Implement the function $F(a,b,c,d) = \Sigma(0,1,2,5,7,9,11,14,15)$ using Shannon's expansion theorem and a 2-to-1 mux. For the expansion, use variable a as the variable to select between $F(0,b,c,d)$ (i.e., the function with $a=0$) and $F(1,b,c,d)$ (i.e., the function with a=1), and then use a as the select line for the 2-to-1 mux. Your solution should be a schematic showing the mux and the functions used as data inputs to the mux.

ab \ cd	00	01	11	10
00	1	1	0	1
01	0	1	1	0
11	0	0	1	1
10	0	1	1	0

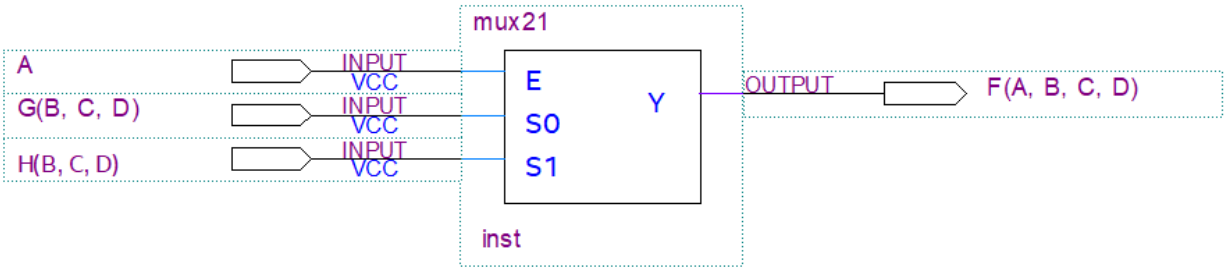
$$F(a,b,c,d) = \Sigma(0,1,2,5,7,9,11,14,15)$$
$$F(a,b,c,d) = \overline{a} \cdot G(b,c,d) + a \cdot H(b,c,d)$$

a \ bc	00	01	11	10
0	1	1	0	1
1	0	1	1	0

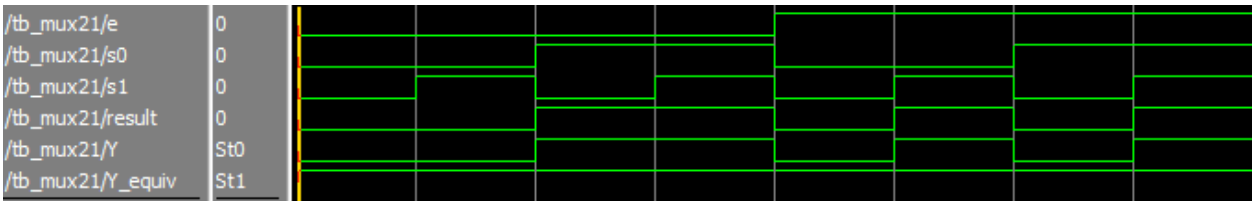
$$G(a,b,c) = \Sigma(0,1,2,5,7)$$

a \ bc	00	01	11	10
0	0	0	1	1
1	0	1	1	0

$$H(a,b,c) = \Sigma(2,3,5,7)$$



Problem 6: Implement your solution for problem 5 in Verilog and verify its operation. Create a behavioural module for a 2-to-1 mux named mux21, then create a module named homework3_problem6 that instantiates your mux21 module. The homework3_problem6 module should have input ports a, b, c and d, and output port F. You should then create a test bench to validate your module and show correct operation for all input combinations.



mux21 Simulation Results (Y_equiv demonstrates equivalence to expected result (result))