Jacob Sickafoose

Lab # 3

Lab Section - 1B

2/2/2020

Description -

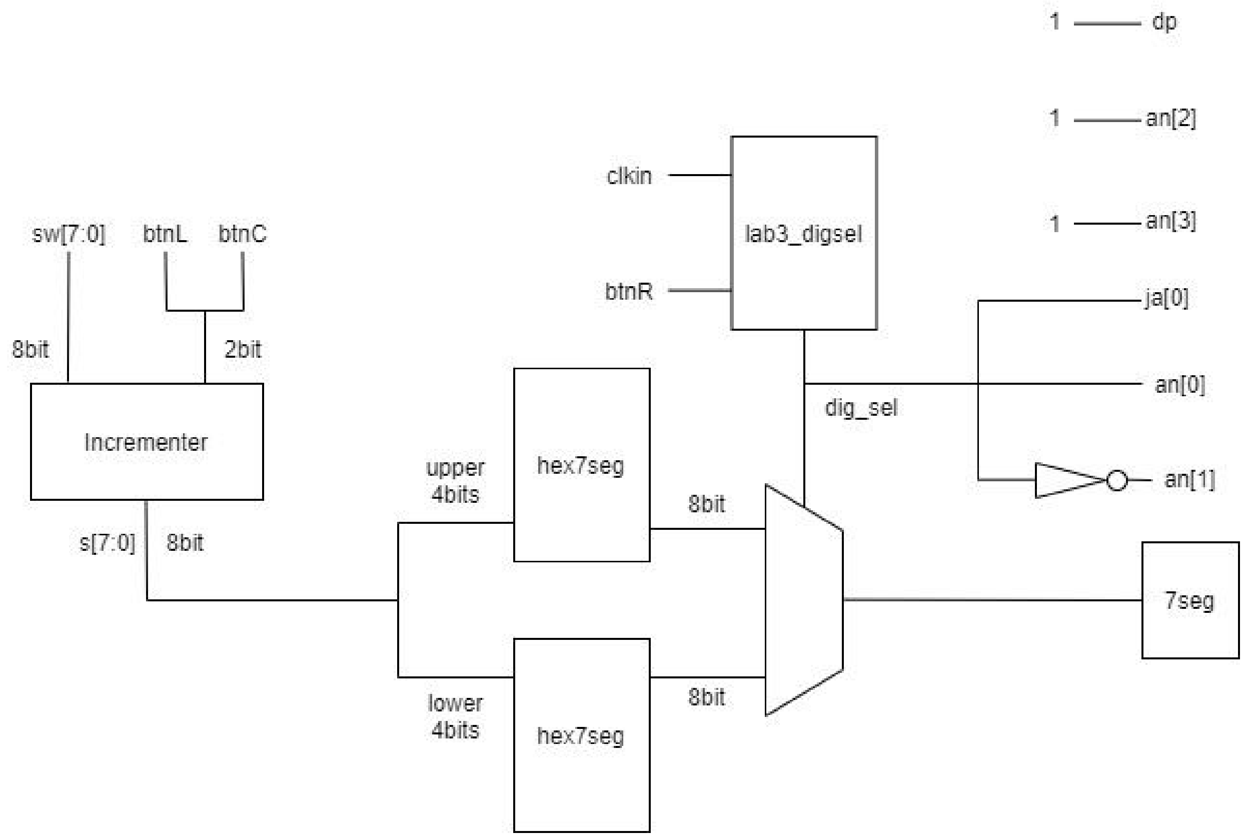
For this lab, we were asked to design another adder. This time making use of 3 different kinds of MUXs. We added the result of an 8-bit number given by the switches, and a 2 bit number given by two buttons. We then were asked to display the result on the two rightmost 7 segment displays. For this, we had to implement the logic again with MUXs and using a clock to refresh the displays.

Method -

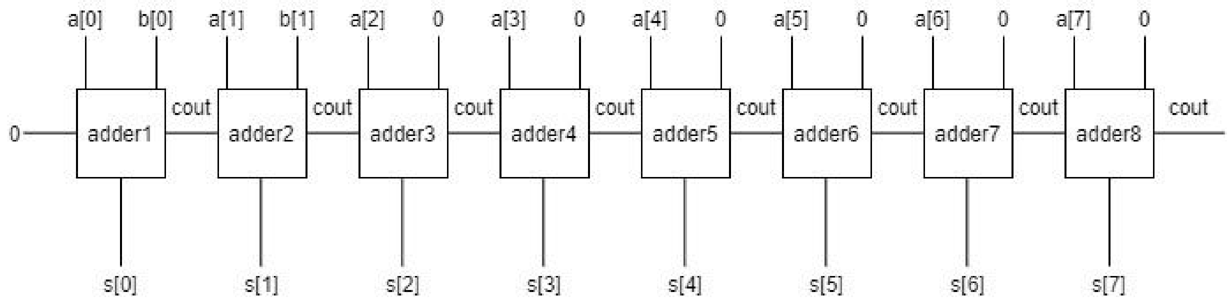
I started this lab by implementing the logic for each of the MUXs. I did this by just making a truth table for the selectors with the desired input line as the output. I then did sum of products to assign the output. Then I implemented the logic for a full adder using MUXs. I started off by making a truth table and using the two most significant bits as selectors on my m4_1 MUX. Using one MUX as the output for sum, another for Cout. By appending 6 zero bits to the 2 bit number, I was then able to use 8 adders to add the 2, 8 bit numbers. Then I was done with the incrementer. For the hex 7 seg converter, we used a separate m8_1 for each of the 7 segments. The 3 most significant bits of the desired 4 bit output, acted as the selector bits for the MUX with the 1 least significant bit deciding the input. I had the lower 4 bits going into one module of hex 7 seg, and the upper 4 bits going into another. The output of the two modules was then fed into our m2_1x8 MUX which selects one of two 8 bit inputs. The selector for this MUX was the resulting dig_sel from the lab3_digsel module given to us. This changes which of the two displays to update, based on our clock.

<u>Design</u> -

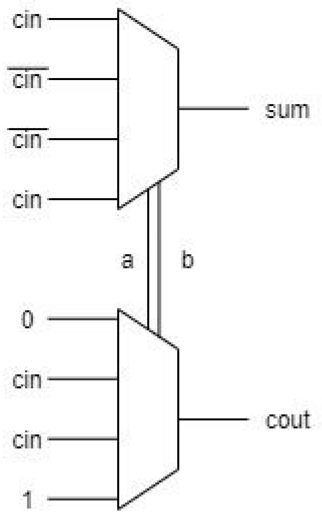
This is my top level. The switches and buttons are input and added in the incrementer. The lower and upper 4 bits of output are then input into the hex7seg modules where they are converted into turning on the appropriate segments. The dig_sel signal alternates which segment display is updated with which hex7seg module output. I also have dig_sel going to the ja[0] pin so I could display it on the oscilloscope.



This is a schematic of the Incrementer. It just strings together all 8 adders. Adding the 2 bits of the buttons to the lower 2 bits of 8bit number the switches produce, and rippling the carry all the way to the end.

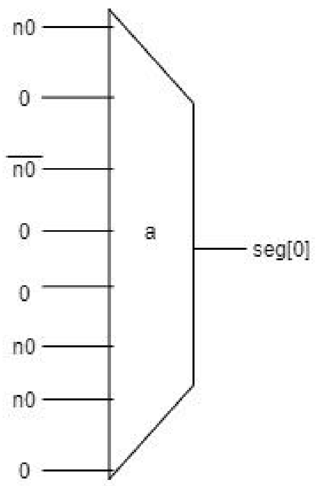


Each of my full adders look like the following, with a and b selecting and cin as the input.



Here is an example of one of my m8_1 MUXs implementing the “a” segment of the 7seg

display.

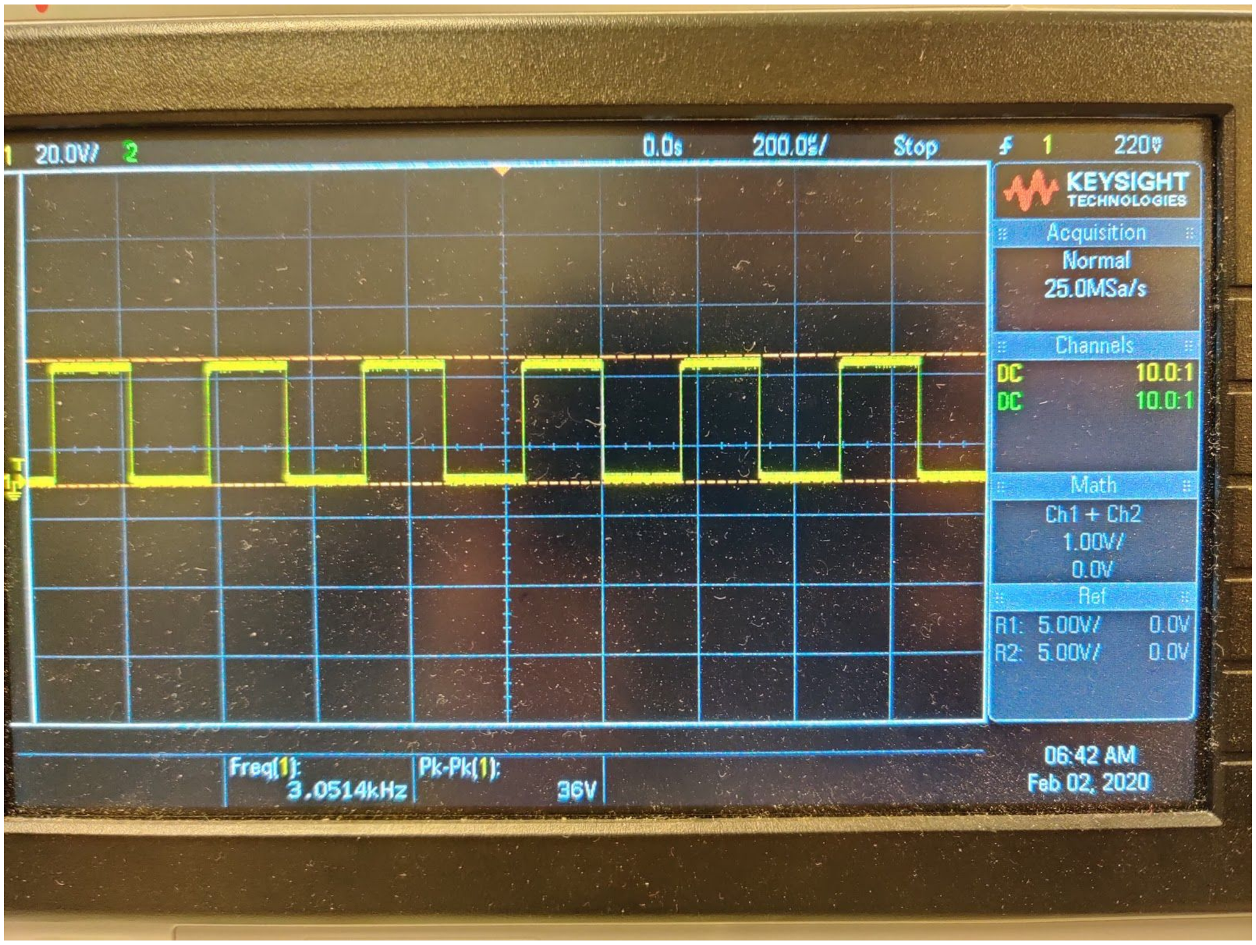


Conclusion -

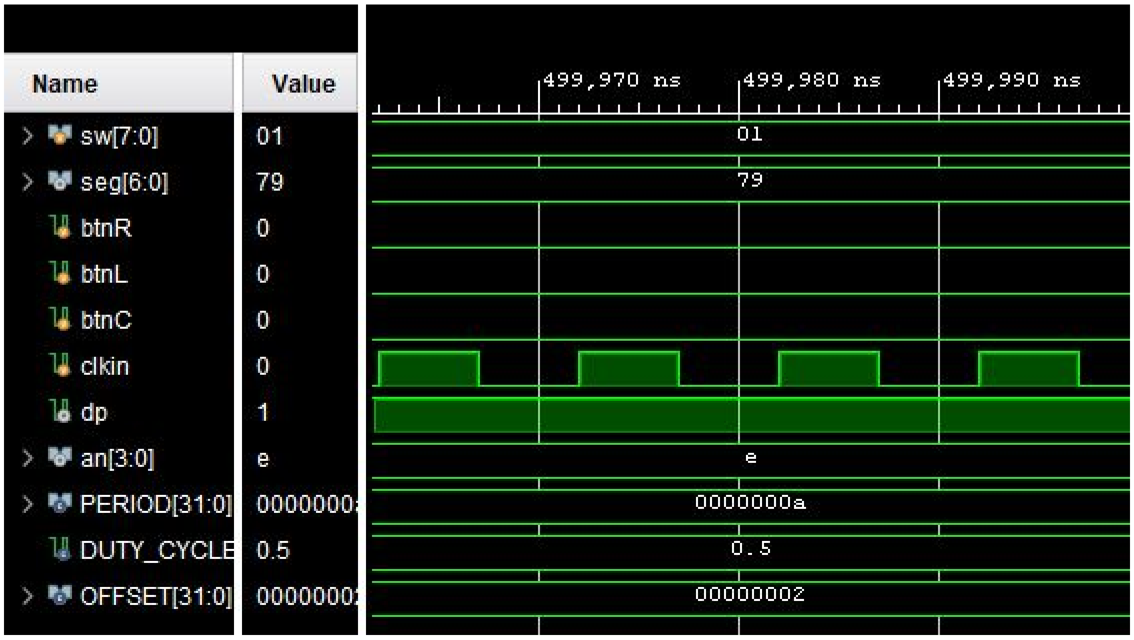
In order to finish this lab, I had to learn how to use MUXs to implement a truth table, as well as how to program MUXs in Verilog. I also had to learn how to use buses in Verilog. I also learned how to use the clock to update the displays. I also learned how to output a signal to a JA pin on the board, as well as how to use the simulation. If I had to do that lab again, it would be alot easier as I am much better at implementing truth tables with MUXs as well as using buses. I also messed up on the full adder and swapped my sum and cout results, costing me a lot of time.

Appendix -

The output of dig_sel on the oscilloscope:

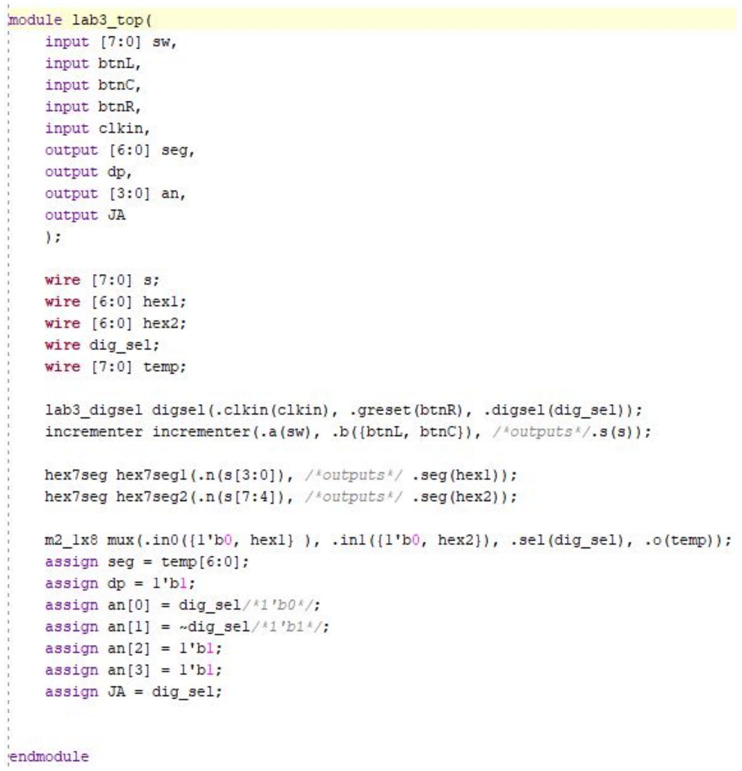


The simulation, which I used to find out that my sum and cout MUXs were swapped:

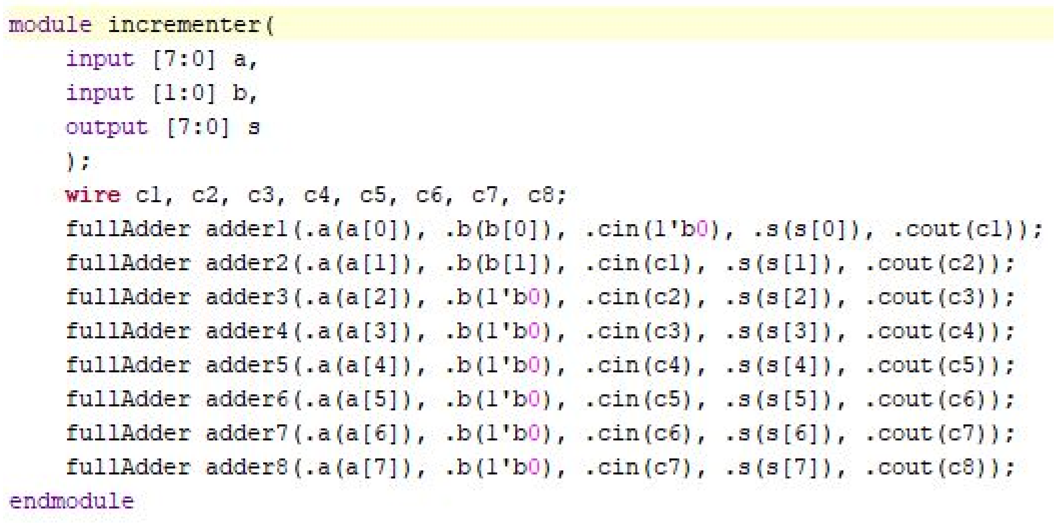


Name	Value
sw[7:0]	01
seg[6:0]	79
btnR	0
btnL	0
btnC	0
clkin	0
dp	1
an[3:0]	e
PERIOD[31:0]	0000000a
DUTY_CYCLE	0.5
OFFSET[31:0]	00000002

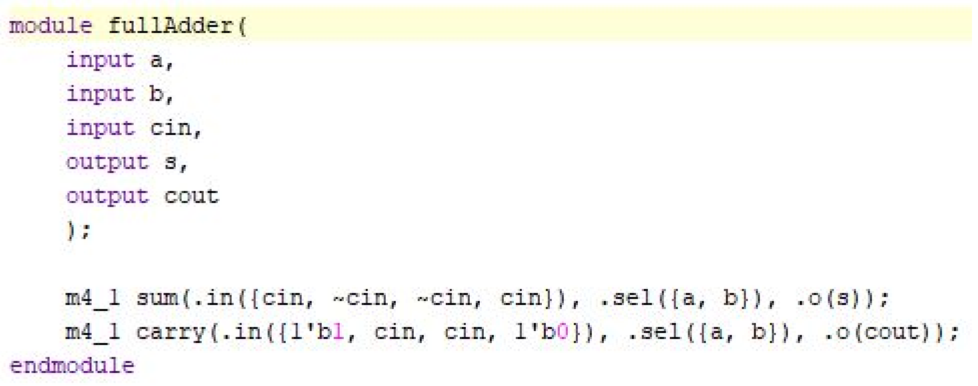
Top level:



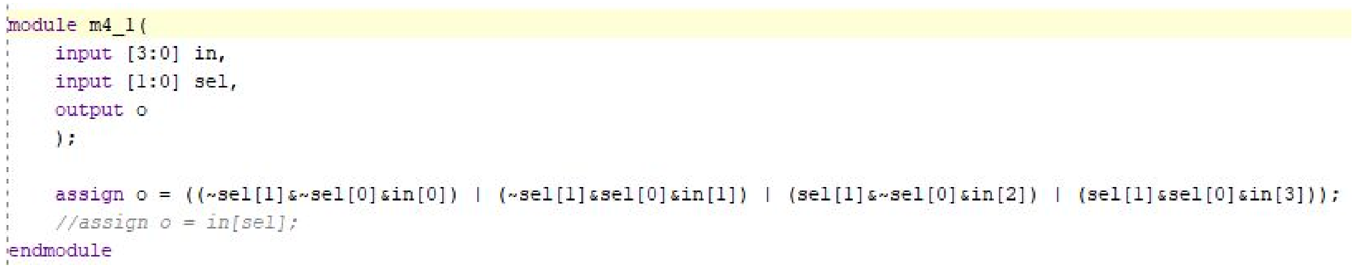
```
module lab3_top(
    input [7:0] sw,
    input btnL,
    input btnC,
    input btnR,
    input clkin,
    output [6:0] seg,
    output dp,
    output [3:0] an,
    output JA
    );

    wire [7:0] s;
    wire [6:0] hex1;
    wire [6:0] hex2;
    wire dig_sel;
    wire [7:0] temp;

    lab3_digsel digsel(.clkin(clkin), .greset(btnR), .digsel(dig_sel));
    incrementer incrementer(.a(sw), .b({btnL, btnC}), /*outputs*/.s(s));

    hex7seg hex7seg1(.n(s[3:0]), /*outputs*/ .seg(hex1));
    hex7seg hex7seg2(.n(s[7:4]), /*outputs*/ .seg(hex2));

    m2_1x8 mux(.in0({1'b0, hex1} ), .in1({1'b0, hex2}), .sel(dig_sel), .o(temp));
    assign seg = temp[6:0];
    assign dp = 1'b1;
    assign an[0] = dig_sel/*1'b0*/;
    assign an[1] = ~dig_sel/*1'b1*/;
    assign an[2] = 1'b1;
    assign an[3] = 1'b1;
    assign JA = dig_sel;


endmodule
```



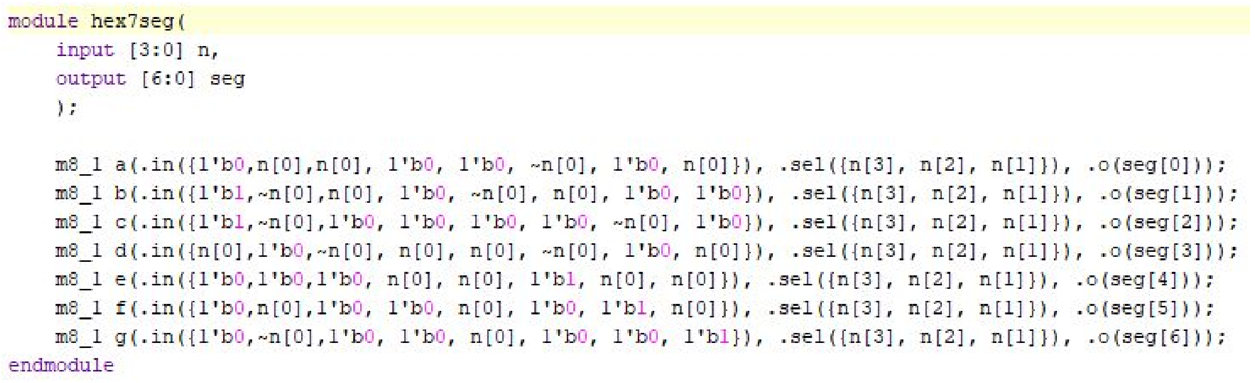
```
module incrementer(
    input [7:0] a,
    input [1:0] b,
    output [7:0] s
    );
    wire c1, c2, c3, c4, c5, c6, c7, c8;
    fullAdder adder1(.a(a[0]), .b(b[0]), .cin(1'b0), .s(s[0]), .cout(c1));
    fullAdder adder2(.a(a[1]), .b(b[1]), .cin(c1), .s(s[1]), .cout(c2));
    fullAdder adder3(.a(a[2]), .b(1'b0), .cin(c2), .s(s[2]), .cout(c3));
    fullAdder adder4(.a(a[3]), .b(1'b0), .cin(c3), .s(s[3]), .cout(c4));
    fullAdder adder5(.a(a[4]), .b(1'b0), .cin(c4), .s(s[4]), .cout(c5));
    fullAdder adder6(.a(a[5]), .b(1'b0), .cin(c5), .s(s[5]), .cout(c6));
    fullAdder adder7(.a(a[6]), .b(1'b0), .cin(c6), .s(s[6]), .cout(c7));
    fullAdder adder8(.a(a[7]), .b(1'b0), .cin(c7), .s(s[7]), .cout(c8));
endmodule
```



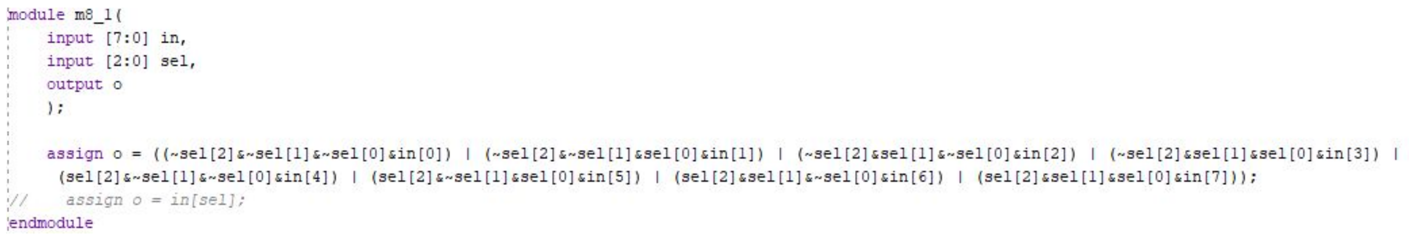
```
module fullAdder(
    input a,
    input b,
    input cin,
    output s,
    output cout
    );

    m4_1 sum(.in({cin, ~cin, ~cin, cin}), .sel({a, b}), .o(s));
    m4_1 carry(.in({1'b1, cin, cin, 1'b0}), .sel({a, b}), .o(cout));
endmodule
```



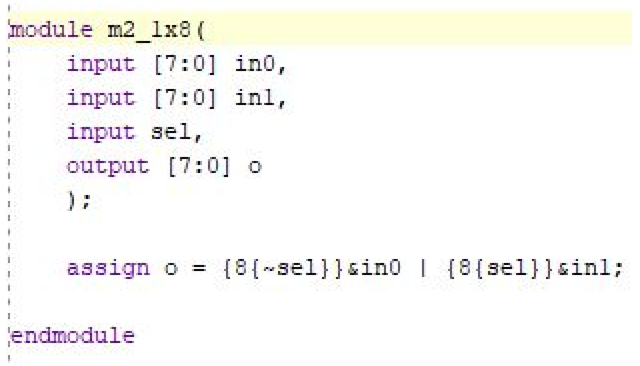
```
module m4_1(
    input [3:0] in,
    input [1:0] sel,
    output o
    );

    assign o = ((~sel[1]&~sel[0]&in[0]) | (~sel[1]&sel[0]&in[1]) | (sel[1]&~sel[0]&in[2]) | (sel[1]&sel[0]&in[3]));
    //assign o = in[sel];
endmodule
```



```verilog
module hex7seg(
    input [3:0] n,
    output [6:0] seg
    );

    m8_1 a(.in({1'b0,n[0],n[0], 1'b0, 1'b0, ~n[0], 1'b0, n[0]}), .sel({n[3], n[2], n[1]}), .o(seg[0]));
    m8_1 b(.in({1'b1,~n[0],n[0], 1'b0, ~n[0], n[0], 1'b0, 1'b0}), .sel({n[3], n[2], n[1]}), .o(seg[1]));
    m8_1 c(.in({1'b1,~n[0],1'b0, 1'b0, 1'b0, 1'b0, ~n[0], 1'b0}), .sel({n[3], n[2], n[1]}), .o(seg[2]));
    m8_1 d(.in({n[0],1'b0,~n[0], n[0], n[0], ~n[0], 1'b0, n[0]}), .sel({n[3], n[2], n[1]}), .o(seg[3]));
    m8_1 e(.in({1'b0,1'b0,1'b0, n[0], n[0], 1'b1, n[0], n[0]}), .sel({n[3], n[2], n[1]}), .o(seg[4]));
    m8_1 f(.in({1'b0,n[0],1'b0, 1'b0, n[0], 1'b0, 1'b1, n[0]}), .sel({n[3], n[2], n[1]}), .o(seg[5]));
    m8_1 g(.in({1'b0,~n[0],1'b0, 1'b0, n[0], 1'b0, 1'b0, 1'b1}), .sel({n[3], n[2], n[1]}), .o(seg[6]));
endmodule
```



```verilog
module m8_1(
    input [7:0] in,
    input [2:0] sel,
    output o
    );

    assign o = ((~sel[2]&~sel[1]&~sel[0]&in[0]) | (~sel[2]&~sel[1]&sel[0]&in[1]) | (~sel[2]&sel[1]&~sel[0]&in[2]) | (~sel[2]&sel[1]&sel[0]&in[3]) |
     (sel[2]&~sel[1]&~sel[0]&in[4]) | (sel[2]&~sel[1]&sel[0]&in[5]) | (sel[2]&sel[1]&~sel[0]&in[6]) | (sel[2]&sel[1]&sel[0]&in[7]));
//    assign o = in[sel];
endmodule
```



```verilog
module m2_1x8(
    input [7:0] in0,
    input [7:0] in1,
    input sel,
    output [7:0] o
    );

    assign o = {8{~sel}}&in0 | {8{sel}}&in1;

endmodule
```

1/b #100 c 12: d 13; e 14:

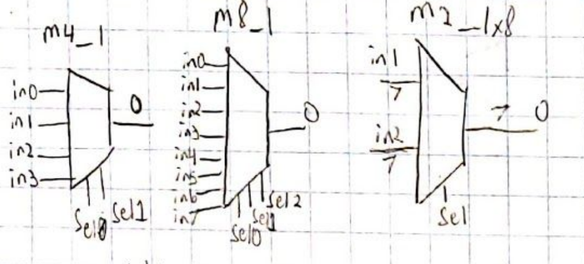
7	3	1	7	4	1	7	5	1	7	6	1
0											
111	011	1	111	100	1	111	101	1	111	110	1

f 15:

7	7	1
111	111	1

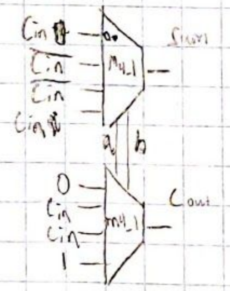
Lab # 3

For this lab, we are making an adder, to add Sw0 - Sw7, an 8bit number, to a 2 bit number btnL and btnC



Full Adder Truth Table

A	B	Cin	Sum	Cout
0	0	0		
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	1	1



A, being our 8-bit number
B, being our 2-bit number

m8_1

Sel0	Sel1	Sel2	in
0	0	0	in0
0	0	1	in1
0	1	0	in2
0	1	1	in3
1	0	0	in4
1	0	1	in5
1	1	0	in6
1	1	1	in7

m4_1

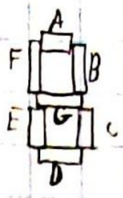
Sel0	Sel1	in
0	0	in0
0	1	in1
1	0	in2
1	1	in3

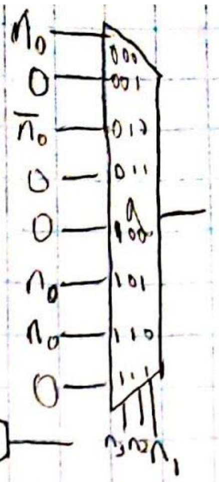
00000000
- - - - - - 00

AS
7139
1:16PM
1/24/20

7 seg Display Truth table

n3	n2	n1	n0	a	b	c	d	e	f	g
0	0	0	0	n_0	0	0	n_0	n_0	n_0	1 1
0	0	0	1	1			1	1	1	1
0	0	1	0	0	0	1	0	n_0	1 1	0
0	0	1	1			$\bar{n}_0$		1	1	
0	1	0	0	$\bar{n}_0$ 1	n_0	0	1	1 1	0	0
0	1	0	1		1		$\bar{n}_0$	1		
0	1	1	0	0	1	0	n_0	n_0	n_0	n_0
0	1	1	1		$\bar{n}_0$		1	1	1	1
1	0	0	0	0	0	0	n_0	n_0	0	0
1	0	0	1				1	1		
1	0	1	0	n_0	n_0	0	1	0	0	0
1	0	1	1	1	1		$\bar{n}_0$			
1	1	0	0	n_0	1	1	0	0	n_0	1
1	1	0	1	1	$\bar{n}_0$	$\bar{n}_0$			1	$\bar{n}_0$
1	1	1	0	0	1	1	n_0	0	0	0
1	1	1	1		1	1	1			





n_0
0
$\bar{n}_0$
0
0
n_0
n_0
0
000
001
010
011
100
101
110
111
g
n_3 n_2 n_1