Lab 3: Sequential Log, Latches, Flip – Flops, Shift Registers, and Counters

CpE Section 1

Instructor: Telles, Eric

Friday: 4:30 pm – 6:50pm

Thao, Shammah

Part 1: 4- bit adder circuit

Description: Using the first letter and second and letter of our name we get a constant that will be converted to a binary number. Which will be used as our 4-bit adder.

Procedures: Design (create truth table, use K-maps to find equations, and create a circuit schematic diagram) for: A four-bit adder that accepts any 4-bit input and adds a 4-bit binary constant to the input. The output needs five bits

Engineering Data:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
							Predicted						Tested			
	A	B	C	D		sum4	sum3	sum2	sum1	sum0		sum4	sum3	sum2	sum1	sum0
	0	0	0	0		0	0	1	0	1		0	0	1	0	1
	0	0	0	1		0	0	1	1	0		0	0	1	1	0
	0	0	1	0		0	0	1	1	1		0	0	1	1	1
	0	0	1	1		0	1	0	0	0		0	1	0	0	0
	0	1	0	0		0	1	0	0	1		0	1	0	0	1
	0	1	0	1		0	1	0	1	0		0	1	0	1	0
	0	1	1	0		0	1	0	1	1		0	1	0	1	1
0	0	1	1	1		0	1	1	0	0		0	1	1	0	0
1	1	0	0	0		0	1	1	0	1		0	1	1	0	1
2	1	0	0	1		0	1	1	1	0		0	1	1	1	0
3	1	0	1	0		0	1	1	1	1		0	1	1	1	1
4	1	0	1	1		1	0	0	0	0		1	0	0	0	0
5	1	1	0	0		1	0	0	0	1		1	0	0	0	1
6	1	1	0	1		1	0	0	1	0		1	0	0	1	0
7	1	1	1	0		1	0	0	1	1		1	0	0	1	1
8	1	1	1	1		1	0	1	0	0		1	0	1	0	0
9																
0				5 =		0101										
1					Adder is											
2																

Figure 1:TruthTable

In figure 1 feature above, it shows the original a, b, c, and d numbers and the predicted number which are just the original number added to my constant of 0101. The result was given on the far right.

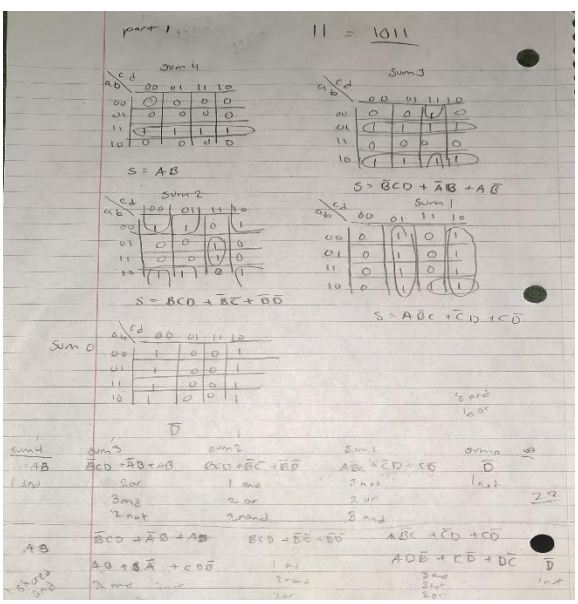


Figure 2:work

In figure 2, using the truth table from figure 1, I created 5 k-map using 4 section each from the truth table. Which at the end gave me multiple equation which will be used to make a multisim, feature in figure 3.

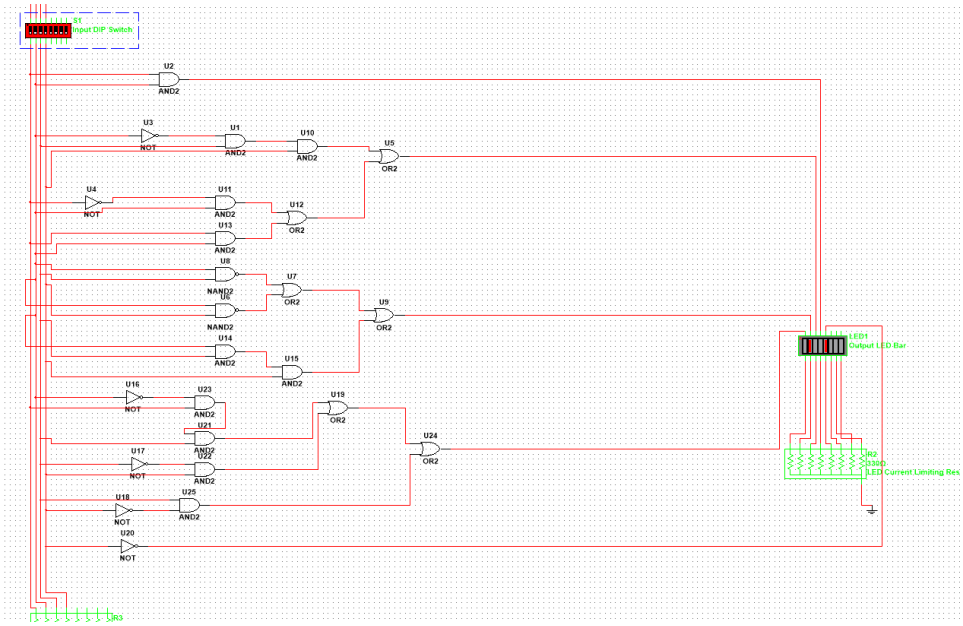


Figure 3:multisim

Part 2: f bit adder Verilog

Description: We basically got to plug our equation that we got from part 1 into the program Quartus and write a Verilog code with it.

Procedure: - Use your five equations for the ADDER circuit from Lab2 part 1 and write a Data Flow model description in Verilog HDL. Be sure you have 5 equations.

Engineering Data:

part2.v part2_tb.v

```
module part2 (a,b,c,d,s0,s1,s2,s3,s4);
   input a,b,c,d;
   output s0,s1,s2,s3,s4;

   assign s0 = (~d);
   assign s1 = ((a & ~b & c ) | (~c & d) | (c & ~d));
   assign s2 = ((b & c & d) | (~b & ~c) | (c & ~d));
   assign s3 = ((~b & c & d) | (~a & b) | (a & ~b));
   assign s4 = (a & b);

endmodule
```

Figure 4: Verilog

We will have to create a Verilog code using our given equation from figure 2. We did a data flow Verilog which allow us to assign the equation to a variable making it much easier to handle. Feature in figure 4. As the Verilog was test, we were able to get a waveform feature in figure 5.

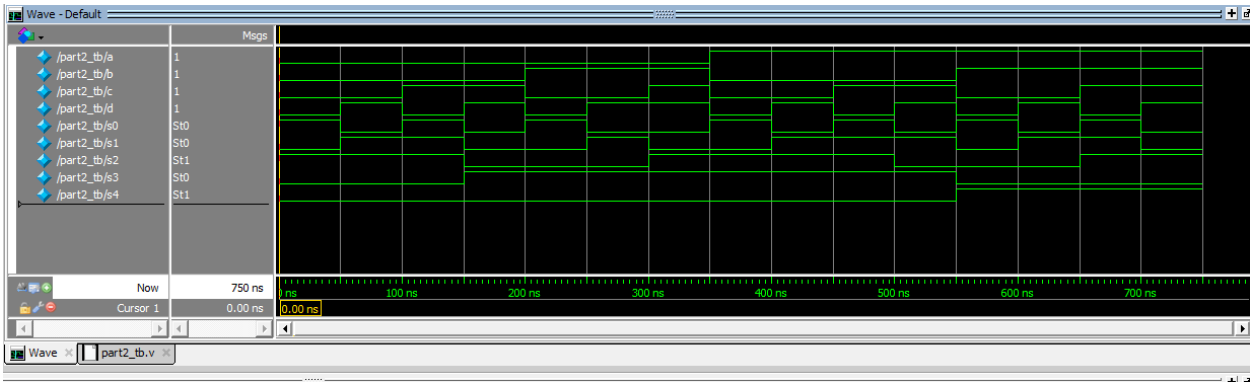


Figure 5:Waveform

Having to test the verilog and waveform using an fpga, we basically used the pin planner from quartus and assign keys for the fpga, using that after using the switch it works, shown in figure 6, we used the switch 1010 which turned on all our lights, similar to the truth table in figure 1.

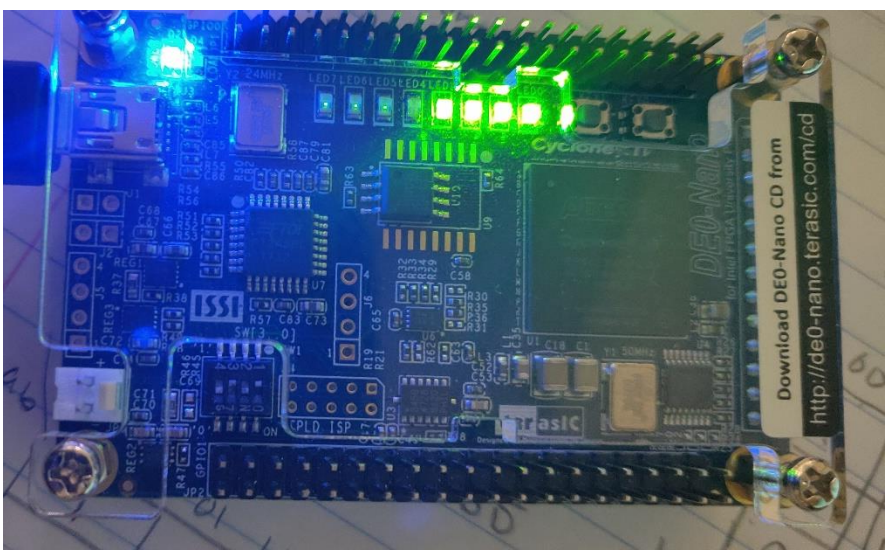


Figure 6:FPGA

Part 3: full 4-bit adder

Description: We had to create a Verilog code that added 4 bits to any 4 bits.

Procedure: Design a full 4-bit adder that adds any 4 bits to any 4 bits. The two 4-bit numbers shall be interpreted as unsigned. Call the 2 numbers A and B. The Function will be A plus B. The number A has

four bits called a3, a2, a1 and a0. The number B has four bits called b3, b2, b1 and b0. The adder shall have five outputs called S4 {or CO for Carry Out}, S3 {Sum 3}, S2 {Sum 2}, S1 {Sum 1}, S0 {Sum 0}.

Engineering Data:

```
module EightBitAdder (a0,a1,a2,a3,b0,b1,b2,b3,s0,s1,s2,s3,s4);
   input a0,a1,a2,a3,b0,b1,b2,b3;
   output s0,s1,s2,s3,s4;

   assign {s4,s3,s2,s1,s0} = {a3,a2,a1,a0} + {b3,b2,b1,b0};

   endmodule
```

Figure 7:Verilog

Using data flow again in this verilog coding, we assigned our 4 variables to be able to add another 4 different variable.feature in figure 7

```
`timescale 1ns / 1ps
module EightBitAdder_tb();
reg a0,a1,a2,a3,b0,b1,b2,b3;
wire s0,s1,s2,s3,s4;
integer i;
   EightBitAdder uut (.a0(a0), .a1(a1), .a2(a2), .a3(a3), .b0(b0), .b1(b1), .b2(b2), .b3(b3), .s0(s0), .s1(s1), .s2(s2), .s3(s3), .s4(s4));
   initial begin
      for(i=0; i<= 256; i = i+1)
         begin
            {a0,a1,a2,a3,b0,b1,b2,b3} <= i;
            #50;
         end
      $stop;
end
endmodule
```

Figure 8:Testbench

In figure 8, creating the testbench, we made a for look which goes all the way to 256 since 2 ^ 8 is 256, using 8 since we got 8 different variables and 2 since we are using 2 compliments. Using figure 8 testbench, we were able to create the waveform in figure 9.

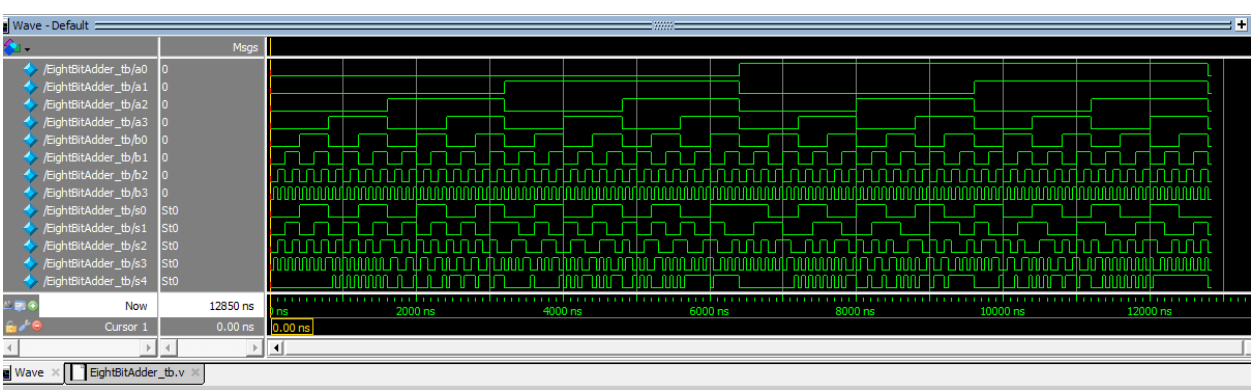


Figure 9:Waveform

Part 4: Design of Comparator using gates

Description: We had to design a 4-input comparator that compares 2-bit numbers.

Procedure: Design a 4-input, 3-circuit that compared two 2-bit unsigned numbers. You can call these numbers a1 a0 and b1 b0. So here a1 is the most significant bit of input A, and a0 is the least significant bit of input A. This circuit should have 3 outputs, which indicate whether A > B, A = B or A < B. You can label these outputs G (which means a1 a0 > b1 b0, where G stands for greater), E (which means a1 a0 + b1 b0), and L (which means a1 a0 < b1 b0). This is called a comparator circuit, because it compares the binary number a1 a0 with the binary number b1 b0

Engineering Data:

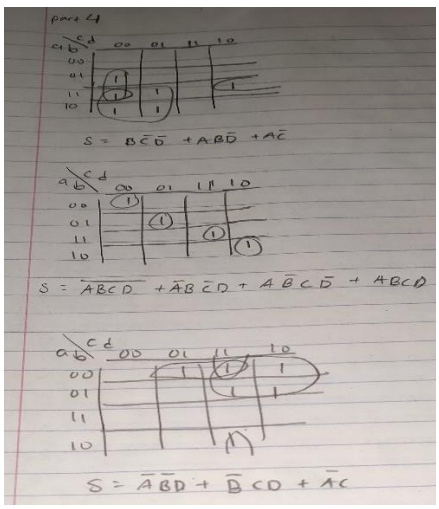


Figure 10:k-map

In figure 10, we got 3 equation using k-map. Which was used to build the Multisim in figure 11.

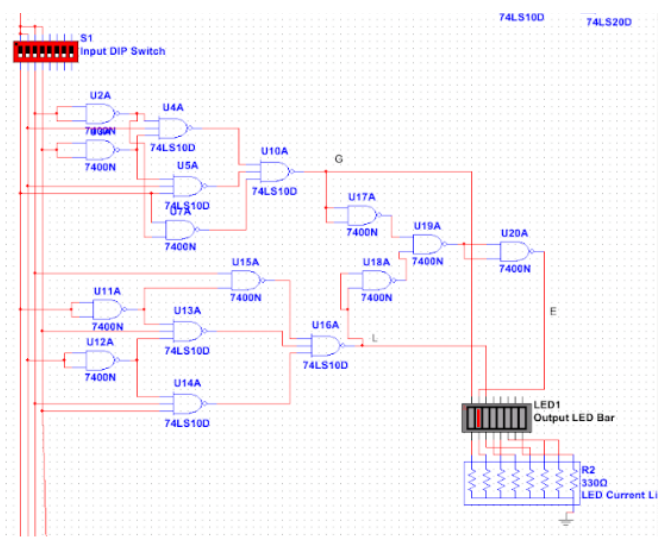


Figure 11:Multsim

The equation that was acquire from figure 10 was then assign to 3 different variable. Using data flow feature in figure 12.

```
module Lab3part4V (a,b,c,d,f,g,h);
   input a,b,c,d;
   output f,g,h;

   assign f = ((b*~c*~d) + (a * b * ~d) + (a * ~c));
   assign g = ((~a * ~b * ~c * ~d) + (~a * b * ~c  * d) + (a * ~b * c * ~d) + (a * b* c* d));
   assign h = ((~a * ~b * d) +(~b * c * d) + (~a *c));

   endmodule
```

Figure 12:Verilog

Using the long way of doing thing, we made a testbench printing out all the truth table variables. Feature in figure 13.

```
`timescale 1ns / 1ps
module Lab3part4V_tb();
reg a,b,c,d; // Inputs
wire f,g,h; // Output
Lab3part4V uut(a,b,c,d,f,g,h);
initial begin
a = 1'b0;
b = 1'b0;
c = 1'b0;
d = 1'b0;
#50; // 50 time units (time unit = 1ns) later change values
a = 1'b0;
b = 1'b0;
c = 1'b0;
d = 1'b1;
#50;
a = 1'b0;
b = 1'b0;
c = 1'b1;
d = 1'b0;
#50;
a = 1'b0;
b = 1'b0;
c = 1'b1;
d = 1'b1;
#50;
a = 1'b0;
b = 1'b1;
```

Figure 13:Truth table

The Verilog and testbench, we made a waveform, feature in figure 14, which shows the 3 different equation at work.

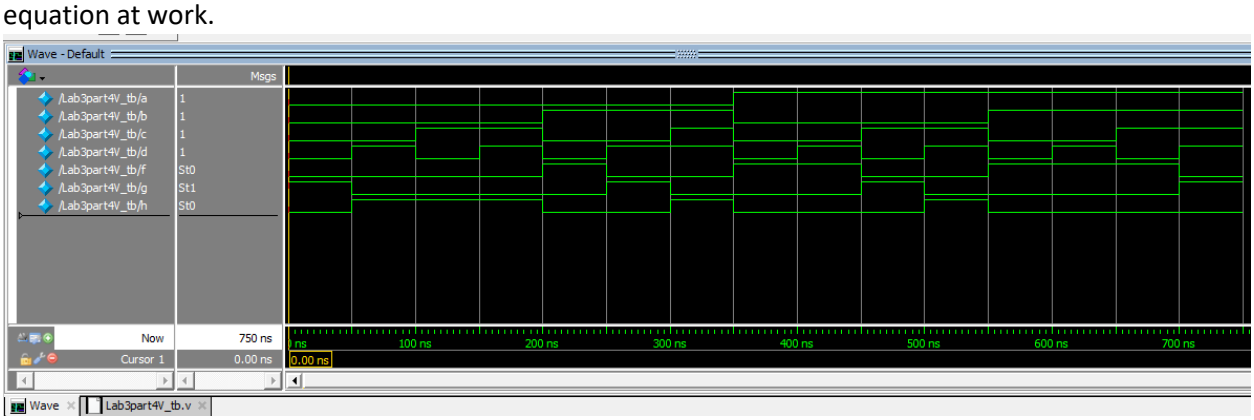


Figure 14:waveform

Part 5: Unsigned 3-bit comparator

Description: we had to make a 3-bit comparator

Procedure: Design an unsigned 3-bit comparator such that “A” is compared to “B”.

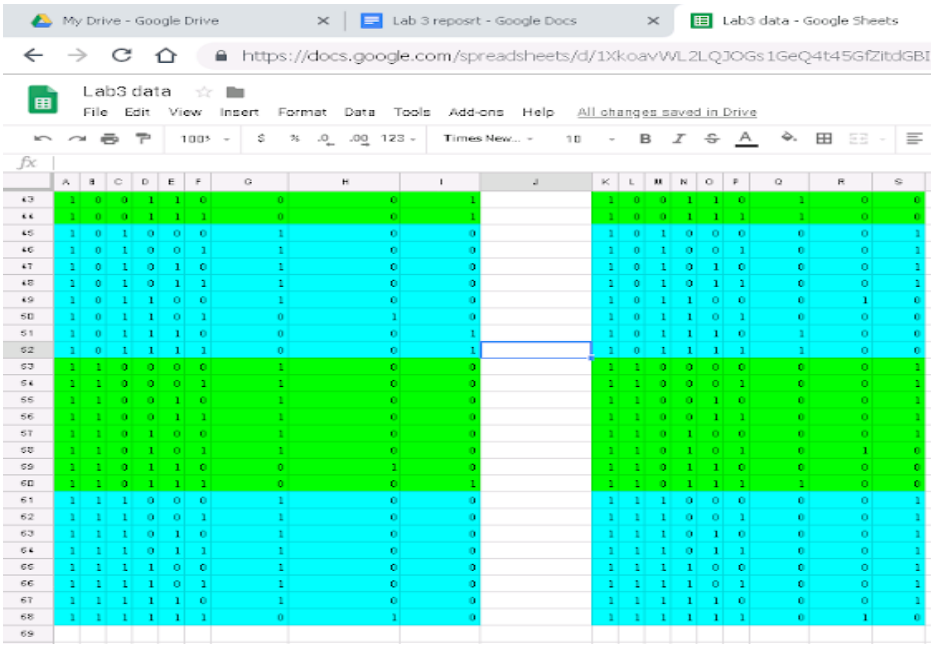
Inputs: A three bits (a2, a1, a0)

Outputs: E A equal to B B three bits (b2, b1, b0) L A less than B G A greater than B E, L, and G are each one bit

Engineering Data:

Unsigned									Signed								
A2	A1	A0	B2	B1	B0	G	E	L	A2	A1	A0	B2	B1	B0	G	E	L
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	1
0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1
0	0	0	0	1	1	0	0	1	0	0	0	0	1	1	0	0	1
0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	1	0	0
0	0	0	1	0	1	0	0	1	0	0	0	1	0	1	1	0	0
0	0	0	1	1	0	0	0	1	0	0	0	1	1	0	1	0	0
0	0	0	1	1	1	0	0	1	0	0	0	1	1	1	1	0	0
0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0
0	0	1	0	0	1	0	1	0	0	0	1	0	0	1	0	1	0
0	0	1	0	1	0	0	0	1	0	0	1	0	1	0	0	0	1
0	0	1	0	1	1	0	0	1	0	0	1	0	1	1	0	0	1
0	0	1	1	0	0	0	0	1	0	0	1	1	0	0	1	0	0
0	0	1	1	0	1	0	0	1	0	0	1	1	0	1	1	0	0
0	0	1	1	1	0	0	0	1	0	0	1	1	1	0	1	0	0
0	0	1	1	1	1	0	0	1	0	0	1	1	1	1	1	0	0
0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0
0	1	0	0	0	1	1	0	0	0	1	0	0	0	1	1	0	0
0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
0	1	0	0	1	1	0	0	1	0	1	0	0	1	1	0	0	1
0	1	0	1	0	0	0	0	1	0	1	0	1	0	0	1	0	0
0	1	0	1	0	1	0	0	1	0	1	0	1	0	1	1	0	0
0	1	0	1	1	0	0	0	1	0	1	0	1	1	0	1	0	0
0	1	0	1	1	1	0		1	0	1	0	1	1	1	1	0	0
0	1	1	0	0	0	1	0	0	0	1	1	0	0	0	1	0	0
0	1	1	0	0	1	1	0	0	0	1	1	0	0	1	1	0	0
0	1	1	0	1	0	1	0	0	0	1	1	0	1	0	1	0	0
0	1	1	0	1	1	0	1	0	0	1	1	0	1	1	0	1	0
0	1	1	1	0	0	0	0	1	0	1	1	1	0	0	0	0	1
0	1	1	1	0	1	0	0	1	0	1	1	1	0	1	1	0	0
0	1	1	1	1	0	0	0	1	0	1	1	1	1	0	1	0	0
0	1	1	1	1	1	0	0	1	0	1	1	1	1	1	1	0	0
1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	1
1	0	0	0	0	1	1	0	0	1	0	0	0	0	1	0	0	1
1	0	0	0	1	0	1	0	0	1	0	0	0	1	0	0	0	1
1	0	0	0	1	1	1	0	0	1	0	0	0	1	1	0	0	1
1	0	0	1	0	0	0	1	0	1	0	0	1	0	0	0	1	0
1	0	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0	0
1	0	0	1	1	0	0	0	1	1	0	0	1	1	0	1	0	0
1	0	0	1	1	1	0	0	1	1	0	0	1	1	1	1	0	0
1	0	1	0	0	0	1	0	0	1	0	1	0	0	0	0	0	1
1	0	1	0	0	1	1	0	0	1	0	1	0	0	1	0	0	1
1	0	1	0	1	0	1	0	0	1	0	1	0	1	0	0	0	1
1	0	1	0	1	1	1	0	0	1	0	1	0	1	1	0	0	1
1	0	1	1	0	0	1	0	0	1	0	1	1	0	0	0	1	0
1	0	1	1	0	1	0	1	0	1	0	1	1	0	1	0	0	0
1	0	1	1	1	0	0	0	1	1	0	1	1	1	0	1	0	0

Figure 15:truth table



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
43	1	0	0	1	1	0	0	0	1		1	0	0	1	1	0	1	0	0
44	1	0	0	1	1	1	0	0	1		1	0	0	1	1	1	1	0	0
45	1	0	1	0	0	0	1	0	0		1	0	1	0	0	0	0	0	1
46	1	0	1	0	0	1	1	0	0		1	0	1	0	0	1	0	0	1
47	1	0	1	0	1	0	1	0	0		1	0	1	0	1	0	0	0	1
48	1	0	1	0	1	1	1	0	0		1	0	1	0	1	1	0	0	1
49	1	0	1	1	0	0	1	0	0		1	0	1	1	0	0	0	1	0
50	1	0	1	1	0	1	0	1	0		1	0	1	1	0	1	0	0	0
51	1	0	1	1	1	0	0	0	1		1	0	1	1	1	0	1	0	0
52	1	0	1	1	1	1	0	0	1		1	0	1	1	1	1	1	0	0
53	1	1	0	0	0	0	1	0	0		1	1	0	0	0	0	0	0	1
54	1	1	0	0	0	1	1	0	0		1	1	0	0	0	1	0	0	1
55	1	1	0	0	1	0	1	0	0		1	1	0	0	1	0	0	0	1
56	1	1	0	0	1	1	1	0	0		1	1	0	0	1	1	0	0	1
57	1	1	0	1	0	0	1	0	0		1	1	0	1	0	0	0	0	1
58	1	1	0	1	0	1	1	0	0		1	1	0	1	0	1	0	1	0
59	1	1	0	1	1	0	0	1	0		1	1	0	1	1	0	0	0	0
60	1	1	0	1	1	1	0	0	1		1	1	0	1	1	1	1	0	0
61	1	1	1	0	0	0	1	0	0		1	1	1	0	0	0	0	0	1
62	1	1	1	0	0	1	1	0	0		1	1	1	0	0	1	0	0	1
63	1	1	1	0	1	0	1	0	0		1	1	1	0	1	0	0	0	1
64	1	1	1	0	1	1	1	0	0		1	1	1	0	1	1	0	0	1
65	1	1	1	1	0	0	1	0	0		1	1	1	1	0	0	0	0	1
66	1	1	1	1	0	1	1	0	0		1	1	1	1	0	1	0	0	1
67	1	1	1	1	1	0	1	0	0		1	1	1	1	1	0	0	0	1
68	1	1	1	1	1	1	0	1	0		1	1	1	1	1	1	0	1	0
69																			

Figure 16: truthtable

The truth table feature in figure 15 and 16 are a combine truth table what was made to compare 3 bits

```
module UnsignedComparator(a2,a1,a0,b2,b1,b0,g,e,l);
    input a2,a1,a0,b2,b1,b0;
    output g,e,l
    reg g,e,l;
    always@(a2 || a1 || a0 || b2 || b1 || b0)
        begin
            case {{a2,a1,a0,b2,b1,b0}}
                0  : begin g<=0; e<=1; l<=0; end
                1  : begin g<=0; e<=0; l<=1; end
                2  : begin g<=0; e<=0; l<=1; end
                3  : begin g<=0; e<=0; l<=1; end
                4  : begin g<=0; e<=0; l<=1; end
                5  : begin g<=0; e<=0; l<=1; end
                6  : begin g<=0; e<=0; l<=1; end
                7  : begin g<=0; e<=0; l<=1; end
                8  : begin g<=1; e<=0; l<=0; end
                9  : begin g<=0; e<=1; l<=0; end
                10 : begin g<=0; e<=0; l<=1; end
                11 : begin g<=0; e<=0; l<=1; end
                12 : begin g<=0; e<=0; l<=1; end
                13 : begin g<=0; e<=0; l<=1; end
                14 : begin g<=0; e<=0; l<=1; end
                15 : begin g<=0; e<=0; l<=1; end
                16 : begin g<=1; e<=0; l<=0; end
                17 : begin g<=1; e<=0; l<=0; end
                18 : begin g<=0; e<=1; l<=0; end
                19 : begin g<=0; e<=0; l<=1; end
                20 : begin g<=0; e<=0; l<=1; end
                21 : begin g<=0; e<=0; l<=1; end
                22 : begin g<=0; e<=0; l<=1; end
                23 : begin g<=0; e<=0; l<=1; end
                24 : begin g<=1; e<=0; l<=0; end
                25 : begin g<=1; e<=0; l<=0; end
                26 : begin g<=1; e<=0; l<=0; end
                27 : begin g<=0; e<=1; l<=0; end
                28 : begin g<=0; e<=0; l<=1; end
                29 : begin g<=0; e<=0; l<=1; end
                30 : begin g<=0; e<=0; l<=1; end
                31 : begin g<=0; e<=0; l<=1; end
                32 : begin g<=1; e<=0; l<=0; end
                33 : begin g<=1; e<=0; l<=0; end
                34 : begin g<=1; e<=0; l<=0; end
                35 : begin g<=1; e<=0; l<=0; end
                36 : begin g<=0; e<=1; l<=0; end
                37 : begin g<=0; e<=0; l<=1; end
                38 : begin g<=0; e<=0; l<=1; end
```

Figure 16:Verilog

```
                21 : begin g<=0; e<=0; l<=1; end
                22 : begin g<=0; e<=0; l<=1; end
                23 : begin g<=0; e<=1; l<=1; end
                24 : begin g<=1; e<=0; l<=0; end
                25 : begin g<=1; e<=0; l<=0; end
                26 : begin g<=1; e<=0; l<=0; end
                27 : begin g<=0; e<=1; l<=0; end
                28 : begin g<=0; e<=0; l<=1; end
                29 : begin g<=0; e<=0; l<=1; end
                30 : begin g<=0; e<=0; l<=1; end
                31 : begin g<=0; e<=0; l<=1; end
                32 : begin g<=1; e<=0; l<=0; end
                33 : begin g<=1; e<=0; l<=0; end
                34 : begin g<=1; e<=0; l<=0; end
                35 : begin g<=1; e<=0; l<=0; end
                36 : begin g<=0; e<=1; l<=0; end
                37 : begin g<=0; e<=0; l<=1; end
                38 : begin g<=0; e<=0; l<=1; end
                39 : begin g<=0; e<=0; l<=1; end
                40 : begin g<=1; e<=0; l<=0; end
                41 : begin g<=1; e<=0; l<=0; end
                42 : begin g<=1; e<=0; l<=0; end
                43 : begin g<=1; e<=0; l<=0; end
                44 : begin g<=1; e<=0; l<=0; end
                45 : begin g<=0; e<=1; l<=0; end
                46 : begin g<=0; e<=0; l<=1; end
                47 : begin g<=0; e<=0; l<=1; end
                48 : begin g<=1; e<=0; l<=0; end
                49 : begin g<=1; e<=0; l<=0; end
                50 : begin g<=1; e<=0; l<=0; end
                51 : begin g<=1; e<=0; l<=0; end
                52 : begin g<=1; e<=0; l<=0; end
                53 : begin g<=1; e<=0; l<=0; end
                54 : begin g<=0; e<=1; l<=0; end
                55 : begin g<=0; e<=0; l<=1; end
                56 : begin g<=1; e<=0; l<=0; end
                57 : begin g<=1; e<=0; l<=0; end
                58 : begin g<=1; e<=0; l<=0; end
                59 : begin g<=1; e<=0; l<=0; end
                60 : begin g<=1; e<=0; l<=0; end
                61 : begin g<=1; e<=0; l<=0; end
                62 : begin g<=1; e<=0; l<=0; end
                63 : begin g<=0; e<=1; l<=0; end
            endcase
        end
```

Figure 17:Verilog

The Verilog code that was made, feature in 17 and 18, was used as a switch case.

Part 6: design a 3-bit comparator signed and unsigned switch

Description: We basically had to edit the code from part 5 and make it so it's less wordy

Procedure: - While the solution for Part 5 was better than doing a six variable K-map, it still was a lengthy design solution (lots of lines of Verilog code). You would suspect that Verilog is better than this! Verilog has the following "logical" operators: = = equality, > greater than, < less than, >= greater than or equal, =< less than or equal to, and! = inequality. Be adventuresome and redesign a solution to the comparator problem by using the relational operators. Compile, download, and test your design. Again, include the segment of the report file that included the test bench source code. Compare the equations with part 5 and comment about any differences in the equations using the two techniques done in each part.

Engineering Data:

lab3pt6.v

```
module Lab3pt6 (in,a2,a1,a0,b2,b1,b0,g,e,l);
   input in,a2,a1,a0, b2,b1,b0;
   output g,e,l;
   reg g,e,l;

      always@(in or a2 or a1 or a0 or b2 or b1 or b0)
         if(in == 0)
            begin
               if({a2,a1,a0} > {b2,b1,b0})
                  begin
                     g<=1;  e<=0;  l<=0;
            end
               else if({a2,a1,a0} < {b2,b1,b0})
                  begin
                     g<= 0;  e<=0;  l<=1;
            end
            else
               begin
                  g<=0;  e<=1;  l<=0;
            end
         end
         else
            begin
               if({a2,a1,a0}>{b2,b1,b0})
                  begin
                     g<=1;  e<=0;  l<=0;
                  end
                  else if({a2,a1,a0}<{b2,b1,b0})
                     begin
                        g<=0;  e<=0;  l<=1;
                  end
                  else
                     begin
                        g<=0;  e<=1;  l<=0;
                  end
               end
            endmodule
```

Figure 18:Verilog

Figure 18, was basically the Verilog code from figure 17 and 18, shorten up using if statement and if then statement.

lab3pt6.v | Lab3pt6_tb.v

```
`timescale 1ns / 1ps
module Lab3pt6_tb();
reg in,a2,a1,a0,b2,b1,b0;
wire g,e,l;
integer i;
   Lab3pt6 uut ( .in(in),.a2(a2),.a1(a1),.a0(a0),.b2(b2),.b1(b1),.b0(b0),.g(g),.e(e),.l(l));
      initial begin
         for(i=0;i<=128;  i= i+1)
            begin
                  {in,a0,a1,a2,b0,b1,b2} <= i;
                  #50;
            end

            $stop;
   end
   endmodule
```

Figure 19:testbench

The testbench for figure 18 is feature in figure 19 which also used a for loop to make it less tedious. The Verilog and testbench was then tested to give up a waveform feature in figure 20.

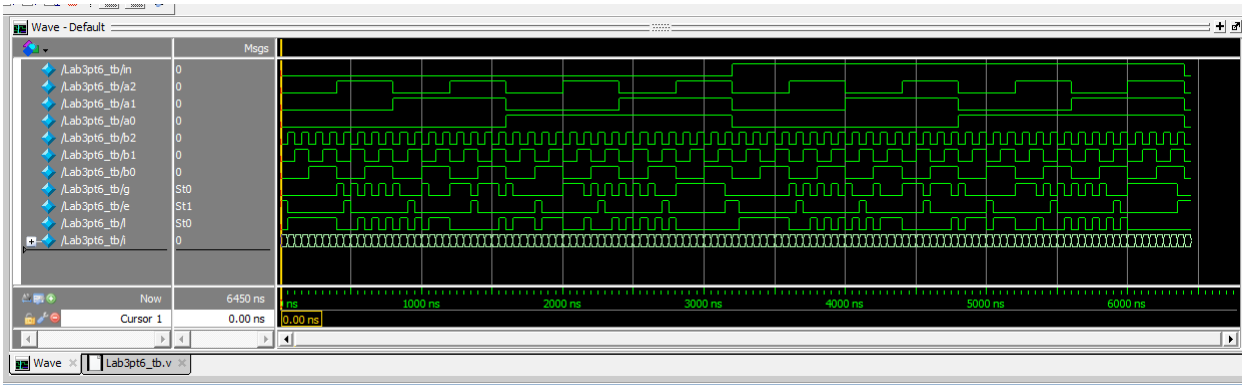


Figure 20:waveform

Part 7 Special design

Description: we make a Verilog design for an 8 input and 2 output.

Procedure: Design a logic system that has eight inputs and two LED outputs. Use your eight pin dip switches as inputs.

Engineering Data:

```
module pt7(a,b,c,d,e,f,g,h,led1,led2);
input a,b,c,d,e,f,g,h;
output led1,led2;
reg led1,led2;
always@(a or b or c or d or e or f or g or h)
begin
if (a==0 && b==0 || b==0 && c==0 || c==0 && d==0
|| d==0 && e==0 || e==0 && f==0 || f==0 &&
g==0
|| g==0 && h==0) |
begin
led1<=1;
end
else
begin
led1<=0;
end

if (h==0 && (a==1 && b==1 && c==1 || b==1 &&
c==1 && d==1
|| c==1 && d==1 && e==1 || d==1 && e==1
&& f==1
|| e==1 && f==1 && g==1 || f==1 && g==1
&& h==1))
begin
led2<=1;
end
else
begin
led2<=0;
end
end
endmodule
```

Figure 21:Verilog

Figure 21, shows the Verilog coding of and 8 input and 2 output coding, using if statement. Trying to figure out the flip-flop switch so it changes every time something is different.

```
`timescale 1ns / 1ps
module pt7_tb();
reg a,b,c,d,e,f,g,h;
wire led1,led2;
integer i;
   pt7 uut ( .a(a),.b(b),.c(c),.d(d),.e(e),.f(f),.g(g),.h(h),.led1(led1),.led2(led2));
      initial begin
         for(i=0;i<=256; i= i+1)
            begin
                  {a,b,c,d,e,f,g,h} <= i;
                  #50;
            end

            $stop;
   end
   endmodule
```

Figure 22:testbench

In figure 22, we have a test bench, using a for loop to create an output of a truth table. Using figure 22 and 21 we made a waveform feature in figure 23.

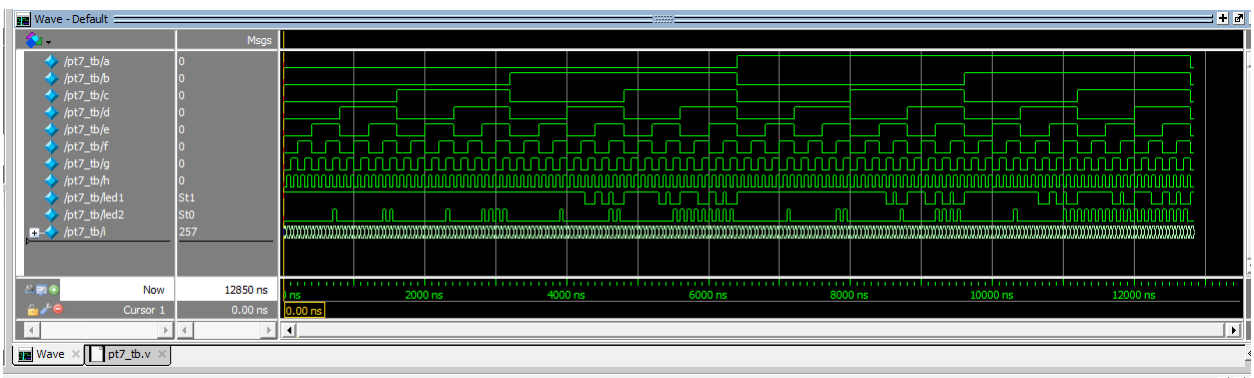


Figure 23:waveform

Conclusion:

In conclusion this lab was hard to do since it was time crunched into one week for such a long lab. The hardest part about this lab is trying to figure out something that haven't been taught in class yet, such as flip flops and comparators. Part 1 of the lab was simple but as we went on it got more different since we had to figure some part out our self. One helpful resource was the sample lab from previous years, it gave us hints to how to get the correct solution