Sam Lee

4/2/18

CPE 64

Professor Kiran

Lab Report #3

Part 1:

In this part of the lab, I was assigned to create a four-bit adder that would have four inputs and five outputs in total. In order to start I had to create a truth table that would allow me to use k-maps to deduce the equations. After I had gained the equationsfrom the k-maps I had to create the Verilog code which require me to create the circuit schematic on Multisim. I was only able to use the four types of gates which were AND, OR, NAND, and NOT when creating the schematic. The schematic had all four inputs and all four outputs going to the LED to display the results.

Truth Table and K-Map:

A close up of text on a white background

Description generated with high confidence

Circuit Design:

A close up of text on a white background

Description generated with high confidence

\*Input 0,0,0,0 and Output 0,1,0,1,0

Part 2:

In this part of the lab, I had to take the five equations I had just made in part 1 and create the Verilog code for it. I used my schematic and K-maps as reference when creating the Verilog code for it. The schematic allowed me to identify the total amount of gates I needed for my code. I needed a total of 3 NOT gates, 7 AND gates, and 3 OR gates to create the schematic and my Verilog code. After I had created the Verilog code, I had to make sure it compiled and programmed onto my FPGA board properly. Then, I was able to test my output on the FPGA by comparing it to my truth table results.

Verilog Code:

module Lab3(in1, in2, in3, in4, out1, out2, out3, out4, out5);

input in1, in2, in3, in4;

output out1, out2, out3, out4, out5;

wire in1, in2, in3, in4, in5, out1, out2, out3, out4, out5;

wire and1out, and2out, and3out, and4out, and5out, and6out, and7out;

wire not1out, not2out, not3out;

wire or1out, or2out, or3out, or4out;

and and1(and1out, in2, in3);

and and2(and2out, not2out, not1out);

and and3(and3out, not3out, not1out);

and and4(and4out, in2, in1);

and and5(and5out, and4out, in3);

and and6(and6out, in2, not3out);

and and7(and7out, not2out, in3);

not not1(not1out, in1);

not not2(not2out, in2);

not not3(not3out, in3);

or or1(out1, and1out, in1);

or or2(or2out, and3out, and2out);

or or3(out2, or2out, and5out);

or or4(out3, and6out, and7out);

assign out4 = not3out;

assign out5 = in4;

endmodule

FPGA Board:

A circuit board

Description generated with very high confidence

\*Input 0,0,0,0 and Output 0,1,0,1,0

Part 3:

In this part of the lab, I had to design a four-bit adder that adds any 4 bits to any 4 bits therefore there are 8 inputs. The first four bits for the input are A series or 1st binary number while the last 4 bit the B series are the 2nd binary number to add with the A series. In order to compensate for the lack of switches on the FPGA board, we had to use test benches. Test benches are just Verilog code that simulates the hardware aspect of a FPGA board. Test benches do not have to be synthesizable since were simulating hardware. This allows me to use 8 inputs in total while having only 5 outputs to be displayed in waveforms. When my test bench was complete I had to make sure I was able to compile my code before even running the simulation. After I had made sure that my code compiled I was ready to run the simulation on model sim which displayed the wave form inputs and outputs of my code. These wave forms allowed me to see if my four-bit adder was functioning properly by comparing the results I gained to the results the test bench produced.

Verilog Code:

module adder(a0, a1, a2, a3, b0, b1, b2, b3, out1, out2, out3, out4, c\_out);

input a0, a1, a2, a3, b0, b1, b2, b3;

output out1, out2, out3, out4, c\_out;

assign{c\_out, out4, out3, out2, out1} = {a3, a2, a1, a0} + {b3, b2, b1, b0};

endmodule

Test Bench:

`timescale 1ns / 1ps

module AddTestBench();

reg a0, a1, a2, a3, b0, b1, b2, b3;

wire out1, out2, out3, out4, c\_out;

adder uut (a0, a1, a2, a3, b0, b1, b2, b3, out1, out2, out3, out4, c\_out);

initial begin

a0 = 1'b1;a1 = 1'b1;a2 = 1'b1;a3 = 1'b1;

b0 = 1'b1;b1 = 1'b1;b2 = 1'b1;b3 = 1'b1;

#20;

a0 = 1'b0;a1 = 1'b0;a2 = 1'b0;a3 = 1'b0;

b0 = 1'b0;b1 = 1'b0;b2 = 1'b0;b3 = 1'b0;

#20;

a0 = 1'b1;a1 = 1'b1;a2 = 1'b0; a3 = 1'b0;

b0 = 1'b0;b1 = 1'b0;b2 = 1'b1;b3 = 1'b1;

#20;

a0 = 1'b1;a1 = 1'b0;a2 = 1'b0;a3 = 1'b1;

b0 = 1'b1;b1 = 1'b1;b2 = 1'b1;b3 = 1'b1;

#20;

a0 = 1'b0;a1 = 1'b0;a2 = 1'b0;a3 = 1'b1;

b0 = 1'b1;b1 = 1'b1;b2 = 1'b1;b3 = 1'b1;

#20;

a0 = 1'b1;a1 = 1'b1;a2 = 1'b1;a3 = 1'b0;

b0 = 1'b0;b1 = 1'b1;b2 = 1'b1;b3 = 1'b1;

#20;

a0 = 1'b1;a1 = 1'b1;a2 = 1'b1;a3 = 1'b1;

b0 = 1'b1;b1 = 1'b1;b2 = 1'b1;b3 = 1'b0;

#20;

a0 = 1'b0;a1 = 1'b0;a2 = 1'b1;a3 = 1'b0;

b0 = 1'b1;b1 = 1'b0;b2 = 1'b0;b3 = 1'b1;

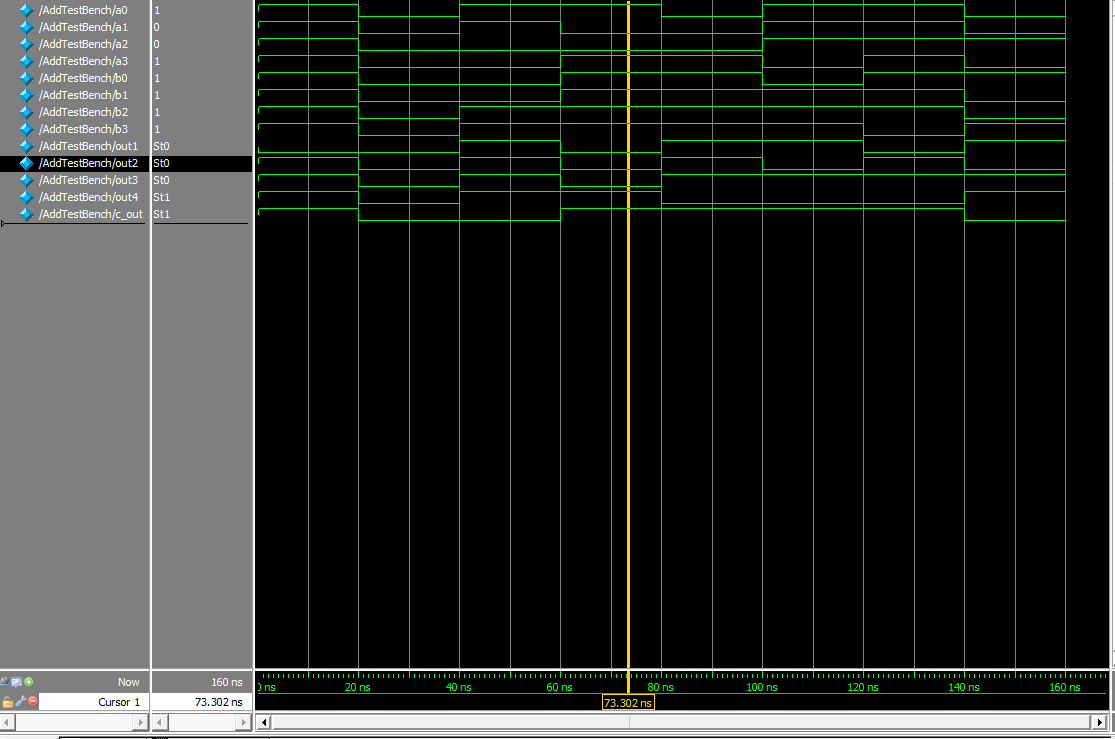
#20;

$stop;

end

endmodule

Wave Form:



\*1001+1111=11000

Part 4:

In this part of the lab, I had to design a 2-bit comparator which compare’s any two bits with any other 2 bits and tells you if the two bits is greater than less than or equal. The truth table has 4 inputs with 2 bit being designated for the A series number and the last two bit being designated for the B series number. The comparator Verilog code was provided in the instructions therefore I only had to program the code onto my FPGA board. From the FPGA board I was able to deduce my truth table. Then from my truth table I was able to use k-maps to create the 3 equations for G (greater than), E (equals to), L (Less than). Then I had to draw the schematic by only using NAND gates. In order to convert to NAND gates I have to take the double Demorgan’s law of each equation I made. However I’m only taking the first Demorgan’s step of each equation as this is the equivalent formula for a NAND gate circuit. After I had converted my equations to NAND equivalents, I was ready to create the schematics. I then created and tested my schematic to make sure I get the same outputs at the truth table states. Once I had completed my schematic test bench for the two-bit comparator. The test bench would output a waveform which simulates the hardware and tells me if my code is correct depending on the result.

Verilog Code:

module comparator(a0,a1,b0,b1,out1,out2,out3);

input a0,a1,b0,b1;

output out1,out2,out3;

reg out1,out2,out3;

always@(a0 or a1 or b0 or b1)

begin

if({a1,a0}<{b1,b0})

out3<=1;

else

out3<=0;

if({a1,a0}>{b1,b0})

out1<=1;

else

out1<=0;

if({a1,a0}=={b1,b0})

out2<=1;

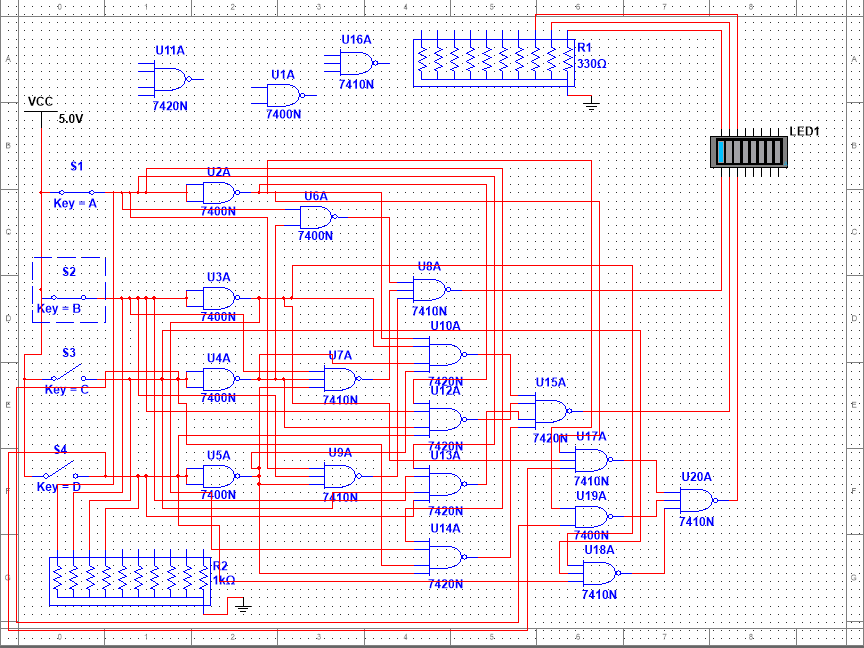
else

out2<=0;

end

endmodule

Circuit Design:



\*Input 1,1,0,0 and Output 1,0,0

Test Bench:

`timescale 1ns / 1ps

module ComparatorTestBench();

reg a0, a1, b0, b1;

wire out1, out2, out3;

comparator uut (.a0(a0), .a1(a1), .b0(b0), .b1(b1), .out1(out1), .out2(out2), .out3(out3));

initial begin

a0 = 1'b0;a1 = 1'b0;b0 = 1'b0;b1 = 1'b0;

#20;a0 = 1'b0;a1 = 1'b0;b0 = 1'b0;b1 = 1'b1;

#20;a0 = 1'b0;a1 = 1'b0;b0 = 1'b1;b1 = 1'b0;

#20;a0 = 1'b0;a1 = 1'b0;b0 = 1'b1;b1 = 1'b1;

#20;a0 = 1'b0;a1 = 1'b1;b0 = 1'b0;b1 = 1'b0;

#20;a0 = 1'b0;a1 = 1'b1;b0 = 1'b0;b1 = 1'b1;

#20;a0 = 1'b0;a1 = 1'b1;b0 = 1'b1;b1 = 1'b0;

#20;a0 = 1'b0;a1 = 1'b1;b0 = 1'b1;b1 = 1'b1;

#20;a0 = 1'b1;a1 = 1'b0;b0 = 1'b0;b1 = 1'b0;

#20;a0 = 1'b1;a1 = 1'b0;b0 = 1'b0;b1 = 1'b1;

#20;a0 = 1'b1;a1 = 1'b0;b0 = 1'b1;b1 = 1'b0;

#20;a0 = 1'b1;a1 = 1'b0;b0 = 1'b1;b1 = 1'b1;

#20;a0 = 1'b1;a1 = 1'b1;b0 = 1'b0;b1 = 1'b0;

#20;a0 = 1'b1;a1 = 1'b1;b0 = 1'b0;b1 = 1'b1;

#20;a0 = 1'b1;a1 = 1'b1;b0 = 1'b1;b1 = 1'b0;

#20;a0 = 1'b1;a1 = 1'b1;b0 = 1'b1;b1 = 1'b1;

#20;$stop;

end

endmodule

Wave Forms:

A picture containing indoor

Description generated with high confidence

\*Input 0,1,0,0,1,0 and Output 0,1,0

Part 5:

For this part of the lab, I had to create a 3-bit comparator with 3 outputs which are E for equal to, G for greater than and L for less than. This comparator should be very similar to the one in part 4 but has more inputs and bits this time. I had to create a test bench for a three-bit comparator and test it out through its waveforms. This test bench and all the previous ones were with unsigned number as we were not counting for negative numbers.

Unsigned Verilog Code:

module Part5 (a0,a1,a2,b0,b1,b2,E,L,G);

input a0,a1,a2,b0,b1,b2;

output E,L,G;

reg E,L,G;

always@( a0 or a1 or a2 or b0 or b1 or b2)

begin

case ({a0,a1,a2,b0,b1,b2})

0 : begin

E<=1;

L<=0;

G<=0;

end

1 : begin

E<=0;

L<=1;

G<=0;

end

2 : begin

E<=0;

L<=1;

G<=0;

end

3 : begin

E<=0;

L<=1;

G<=0;

end

4 : begin

E<=0;

L<=1;

G<=0;

end

5 : begin

E<=0;

L<=1;

G<=0;

end

6 : begin

E<=0;

L<=1;

G<=0;

end

7 : begin

E<=0;

L<=1;

G<=0;

end

8 : begin

E<=0;

L<=0;

G<=1;

end

default :begin

E<=0;

L<=0;

G<=0;

end

endcase

end

endmodule

Unsigned Test Bench Code:

`timescale 1ns / 1ps

module Part5TB();

reg a0, a1, a2, b0, b1, b2;

wire E, L, G;

Part5 uut (.a0(a0), .a1(a1), .a2(a2), .b0(b0), .b1(b1), .b2(b2), .E(E), .L(L), .G(G));

initial begin

a0 = 1'b1;

a1 = 1'b1;

a2 = 1'b1;

b0 = 1'b1;

b1 = 1'b1;

b2 = 1'b1;

#20;

a0 = 1'b0;

a1 = 1'b0;

a2 = 1'b0;

b0 = 1'b0;

b1 = 1'b0;

b2 = 1'b0;

#20;

a0 = 1'b1;

a1 = 1'b1;

a2 = 1'b0;

b0 = 1'b0;

b1 = 1'b0;

b2 = 1'b1;

#20;

a0 = 1'b1;

a1 = 1'b0;

a2 = 1'b0;

b0 = 1'b1;

b1 = 1'b1;

b2 = 1'b1;

#20;

a0 = 1'b0;

a1 = 1'b0;

a2 = 1'b0;

b0 = 1'b1;

b1 = 1'b1;

b2 = 1'b1;

#20;

a0 = 1'b1;

a1 = 1'b1;

a2 = 1'b1;

b0 = 1'b0;

b1 = 1'b1;

b2 = 1'b1;

#20;

a0 = 1'b1;

a1 = 1'b1;

a2 = 1'b1;

b0 = 1'b1;

b1 = 1'b1;

b2 = 1'b1;

#20;

a0 = 1'b0;

a1 = 1'b0;

a2 = 1'b1;

b0 = 1'b1;

b1 = 1'b0;

b2 = 1'b0;

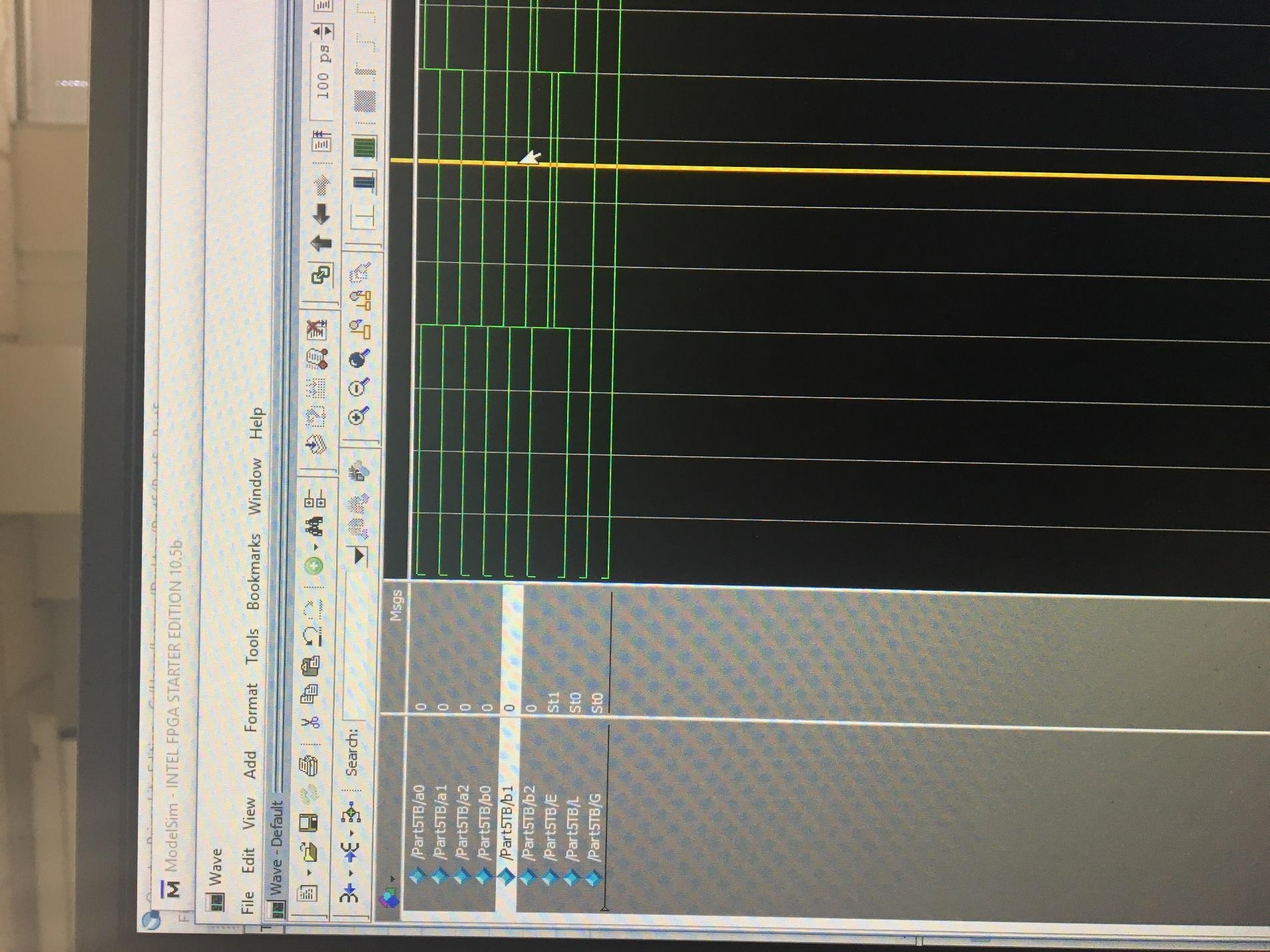
#20;

$stop;

end

endmodule

Unsigned Wave Form:



\*Input 1,1,1,1,1,1 and Output 1,0,0

Signed Verilog Code:

module Part5(a0,a1,a2,b0,b1,b2,E,L,G);

input a0,a1,a2,b0,b1,b2;

output E,L,G;

reg E,L,G;

always@( a0 or a1 or a2 or b0 or b1 or b2)

begin

case ({a0,a1,a2,b0,b1,b2})

0 : begin

E<=1;

L<=0;

G<=0;

end

1 : begin

E<=0;

L<=1;

G<=0;

end

2 : begin

E<=0;

L<=1;

G<=0;

end

3 : begin

E<=0;

L<=1;

G<=0;

end

4 : begin

E<=0;

L<=0;

G<=1;

end

5 : begin

E<=0;

L<=0;

G<=1;

end

6 : begin

E<=0;

L<=0;

G<=1;

end

7 : begin

E<=0;

L<=0;

G<=1;

end

8 : begin

E<=0;

L<=0;

G<=1;

end

default :begin

E<=0;

L<=0;

G<=0;

end

endcase

end

endmodule

Signed Test Bench:

`timescale 1ns / 1ps

module Part5TB();

reg a0, a1, a2, b0, b1, b2;

wire E, L, G;

Part5 uut (.a0(a0), .a1(a1), .a2(a2), .b0(b0), .b1(b1), .b2(b2), .E(E), .L(L), .G(G));

initial begin

a0 = 1'b1;

a1 = 1'b1;

a2 = 1'b1;

b0 = 1'b1;

b1 = 1'b1;

b2 = 1'b1;

#20;

a0 = 1'b0;

a1 = 1'b0;

a2 = 1'b0;

b0 = 1'b0;

b1 = 1'b0;

b2 = 1'b0;

#20;

a0 = 1'b1;

a1 = 1'b1;

a2 = 1'b0;

b0 = 1'b0;

b1 = 1'b0;

b2 = 1'b1;

#20;

a0 = 1'b1;

a1 = 1'b0;

a2 = 1'b0;

b0 = 1'b1;

b1 = 1'b1;

b2 = 1'b1;

#20;

a0 = 1'b0;

a1 = 1'b0;

a2 = 1'b0;

b0 = 1'b1;

b1 = 1'b1;

b2 = 1'b1;

#20;

a0 = 1'b1;

a1 = 1'b1;

a2 = 1'b1;

b0 = 1'b0;

b1 = 1'b1;

b2 = 1'b1;

#20;

a0 = 1'b1;

a1 = 1'b1;

a2 = 1'b1;

b0 = 1'b1;

b1 = 1'b1;

b2 = 1'b1;

#20;

a0 = 1'b0;

a1 = 1'b0;

a2 = 1'b1;

b0 = 1'b1;

b1 = 1'b0;

b2 = 1'b0;

#20;

a0 = 1'b1;

a1 = 1'b1;

a2 = 1'b1;

b0 = 1'b1;

b1 = 1'b1;

b2 = 1'b0;

#20;

$stop;

end

endmodule

Signed Wave Form:

A close up of a door

Description generated with high confidence

\*Input 0,0,0,1,1,1 and Output 0,0,1

Part 6:

For this part of the lab, I had to create a 3-bit comparator which compares both signed and unsigned numbers and displays if variable A was (G) greater than, (L) less than, or (E) equal to variable B. The comparator would have 6 inputs in total and 3 outputs the inputs are split by the variable so 3 bits are designated for variable A while the other 3 bits are for variable B. however there will also be variable called signed bit which dictates whether the variable is signed or unsigned and compares them properly. Signed bits have both positive and negative numbers while unsigned bits only have positive numbers. When writing the Verilog code for this comparator I had to use $signed (“variable A or B”) when comparing variables, A and B. This notation makes sure that the variables are being compared as signed numbers because the signed bit is one. When the signed bit is zero this means that the numbers are unsigned therefore their only positive. To test my code, I had to create a test bench that will assign the bits to variables A and B properly. After I had crated my test bench and made sure it compiled I was ready to generate the waveform. From the waveform results I was able to see if was working properly by comparing its results for E, L, or G to mine that had done by hand.

Verilog Code:

module Part6(a0,a1,a2,b0,b1,b2,us,G,E,L);

input a0,a1,a2,b0,b1,b2;

input us;

output G,E,L;

reg G,E,L;

wire a0,a1,a2,b0,b1,b2;

always@(a0 or a1 or a2 or b0 or b1 or b2)

if (us==1) //unsigned mode

begin

G<={a0,a1,a2}>{b0,b1,b2}; //B is less

E<={a0,a1,a2}=={b0,b1,b2}; //logical equality

L<={a0,a1,a2}<{b0,b1,b2}; //A is less

end

else //signed mode

begin

G<=$signed({a0,a1,a2})>$signed({b0,b1,b2}); //B is less

E<=$signed({a0,a1,a2})==$signed({b0,b1,b2}); //logical equality

L<=$signed({a0,a1,a2})<$signed({b0,b1,b2}); //A is less

end

endmodule

Test Bench For Unsigned and Signed:

`timescale 1ns / 1ps

module Part6TB();

reg a0, a1, a2, b0, b1, b2;

wire G,E,L;

Part6 uut (.a0(a0), .a1(a1), .a2(a2), .b0(b0), .b1(b1), .b2(b2), .G(G), .E(E), .L(L));

initial begin

a0 = 1'b1;

a1 = 1'b0;

a2 = 1'b1;

b0 = 1'b0;

b1 = 1'b0;

b2 = 1'b0;

#20;

a0 = 1'b1;

a1 = 1'b0;

a2 = 1'b1;

b0 = 1'b1;

b1 = 1'b0;

b2 = 1'b0;

#20;

a0 = 1'b1;

a1 = 1'b0;

a2 = 1'b1;

b0 = 1'b0;

b1 = 1'b1;

b2 = 1'b0;

#20;

a0 = 1'b1;

a1 = 1'b0;

a2 = 1'b1;

b0 = 1'b1;

b1 = 1'b1;

b2 = 1'b0;

#20;

a0 = 1'b1;

a1 = 1'b0;

a2 = 1'b1;

b0 = 1'b0;

b1 = 1'b0;

b2 = 1'b1;

#20;

a0 = 1'b1;

a1 = 1'b0;

a2 = 1'b1;

b0 = 1'b1;

b1 = 1'b0;

b2 = 1'b1;

#20;

a0 = 1'b1;

a1 = 1'b0;

a2 = 1'b1;

b0 = 1'b0;

b1 = 1'b1;

b2 = 1'b1;

#20;

a0 = 1'b1;

a1 = 1'b0;

a2 = 1'b1;

b0 = 1'b1;

b1 = 1'b1;

b2 = 1'b1;

#20;

$stop;

end

endmodule

Wave Form Unsigned:

A picture containing indoor

Description generated with high confidence

\*Input 1,0,1,1,1,0 and Output 0,0,1

Wave Form Signed:

A picture containing building

Description generated with high confidence

Input 1,0,1,0,0,0 and Output 0,0,1

Part 7:

In this part of the lab, I had to design a logic system that has eight inputs and two LED outputs based on the conditions I was given. LED #1 is on when two switches are LOW next to each other while LED #2 is ON whenever the least significant switch is not HIGH, AND there are groups of 3 switches in a row that are set HIGH. I was given four examples of each specific outcome and from this I had to create the Verilog code and testbench. From the information given to me I was able to deduce that LED 1 would have all combinations of inputs with NOTs on them for instance (~a&~b)|(~b&~C)|… as this would follow its condition. For LED 2 I had to have combination of 3 inputs ANDED then ORED with other combinations for instance ((a&b&c)|(b&c&d)|…)&~h. however the least significant bit had to be low therefore it received a NOT at the end.

Verilog Code:

module Part7(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) //any 2 switches next to each other are LOW

begin

led1<=1;

end

else //least significant

//switch is HIGH

begin

led1<=0;

end

//least signigicant switch is not HIGH &&

//any 3 switches in a row are low

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

endmodules

Test Bench:

`timescale 1ns / 1ps

module Part7TB();

reg a, b, c, d, e, f, g, h;

wire led1, led2;

Part7 uut (.a(a), .b(b), .c(c), .d(d), .e(e), .f(f), .g(g), .h(h), .led1(led1), .led2(led2));

initial begin

a = 1'b0;

b = 1'b1;

c = 1'b0;

d = 1'b1;

e = 1'b0;

f = 1'b1;

g = 1'b0;

h = 1'b1;

#20;

a = 1'b0;

b = 1'b1;

c = 1'b1;

d = 1'b1;

e = 1'b0;

f = 1'b0;

g = 1'b0;

h = 1'b0;

#20;

a = 1'b0;

b = 1'b1;

c = 1'b1;

d = 1'b0;

e = 1'b1;

f = 1'b1;

g = 1'b1;

h = 1'b0;

#20;

a = 1'b1;

b = 1'b1;

c = 1'b1;

d = 1'b0;

e = 1'b0;

f = 1'b0;

g = 1'b1;

h = 1'b1;

#20;

$stop;

end

endmodule

Wave Form:

A picture containing indoor

Description generated with high confidence

\*Input 0,1,1,1,0,0,0,0 and Output 1,1

Conclusion:

This lab was a great way for me to learn more about Verilog coding and how to generate waveforms as an alternative to using my FPGA or breadboard. For part 1, I gained more practice on combinational logic circuits. This part required me use truth tables and K-maps to create my Verilog code for this four-bit adder. For part 2, I gained even more practice with equations, k-maps, and Boolean algebra. I had to create the Verilog code for the Multisim schematic I created for part one. This required me to count the amount of each type of gate there was and implement it on Verilog. For part 3, I had practiced designing combinational logic circuit with NAND gates. In this part I had to create the equivalent Multisim schematic from the one in part one just by using NAND gates. This required me to take the double Demorgan’s of every equation and just doing the first step to gain my NAND equivalent. For part four I had to design a comparator and test it out. The Verilog code was given so I had to just create the truth table and k-maps for this part. For part 5, I had to create 3bit comparator one for signed number and the other for unsigned. For part 6, I had to create a comparator who compares both signed and unsigned number and create a test bench. Finally, for part 7, I had to do a special design with the conditions given to me on Verilog and test it out by creating a test bench for it.