Lab 3 BCD Adders

July 4, 2017

Cs211 Summer 2017

Xu Mingzhi

Table of Contents

1. Objective

In this lab, we will be taking from what we have learned in the previous labs about full adders, decoders, and such to construct a BCD Adder, or the Binary Coded Decimal Adder which is similar to the 4-bit full adder but instead of outputting it onto the lights to represent the binary result, we will be outputting it onto the seven-segment display to display the result as decimal numbers which are numbers in base 10.

We will be designing the following circuit:

a. Seven Segment Decoder

b. Binary to Decimal

c. Four Bit Full Adder using Bus Tool

d. BCD Adder

2. Seven Segment Decoder

2.1 Functionality and Specification

The seven-segment decoder is a circuit that will read 4-bit binary and output the values onto the seven-segment display in decimal form which usually displays the number 0 to 9.

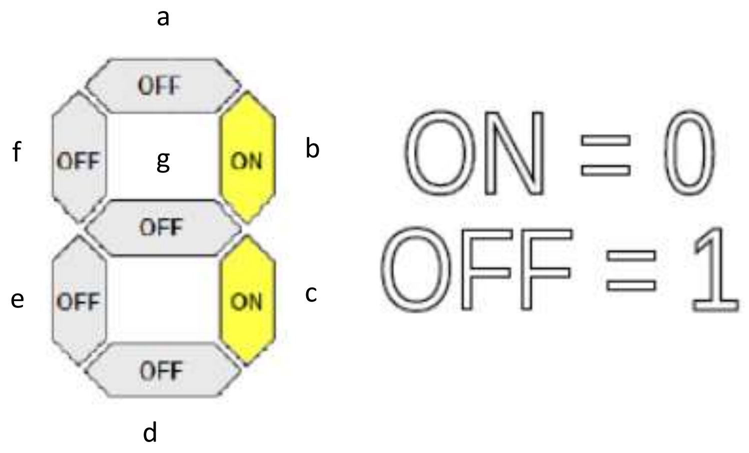
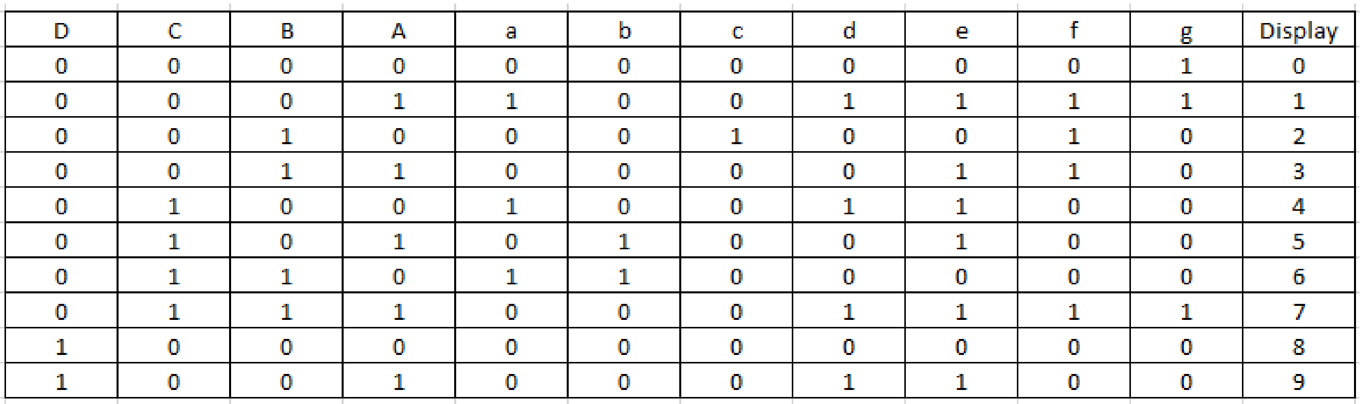


Figure 1: Seven-segment display showing digit 1.

This is the seven-segment display which includes seven elements a to g which are the outputs determine by the four inputs representing the binary value that it will be implemented onto the seven-segment display. When the switch is turned on, the light that is assigned on the elements will be off which mean by default the lights a to g are all on. In order to display the decimal number 1 onto the seven-segment display, when the input is 0001 which is one in binary form, the lights a, d, e, f, and g will be turned off leaving only lights b and c on which is a way to display the number one.



D	C	B	A	a	b	c	d	e	f	g	Display
0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	1	0	0	1	1	1	1	1
0	0	1	0	0	0	1	0	0	1	0	2
0	0	1	1	0	0	0	0	1	1	0	3
0	1	0	0	1	0	0	1	1	0	0	4
0	1	0	1	0	1	0	0	1	0	0	5
0	1	1	0	1	1	0	0	0	0	0	6
0	1	1	1	0	0	0	1	1	1	1	7
1	0	0	0	0	0	0	0	0	0	0	8
1	0	0	1	0	0	0	1	1	0	0	9

Figure 2: Truth table for seven-segment decoder.

This is the truth table for seven-segment display and each output will have its own block diagram and Boolean function which making up a total of seven block diagram and putting them together to make up the seven-segment decoder.

Since a will be 1 which mean the light will be off when the decimal value of 1, 4 or 6 is being displayed which means it will have a Boolean function of

a = D' C A' + D' C' B' A;

and by implementing the Boolean function onto the program call logic friday it will produce a circuit using only NOR gates which looks like this

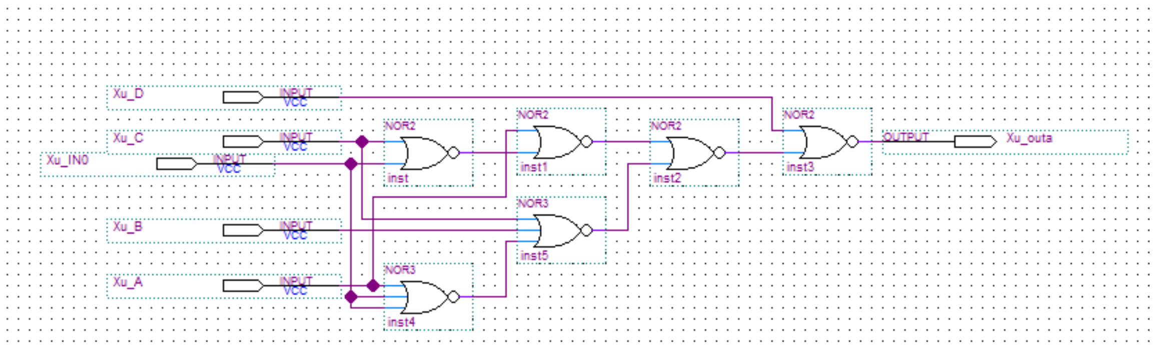


Figure 3: Block diagram for a using only NOR gates.

Since b will be 1 which mean the light will be off when the decimal value of 5 or 6 is being displayed which means it will have a Boolean function of

b = D' C B' A + D' C B A'

and by implementing the Boolean function onto the program call logic friday it will produce a circuit using only NOR gates which looks like this

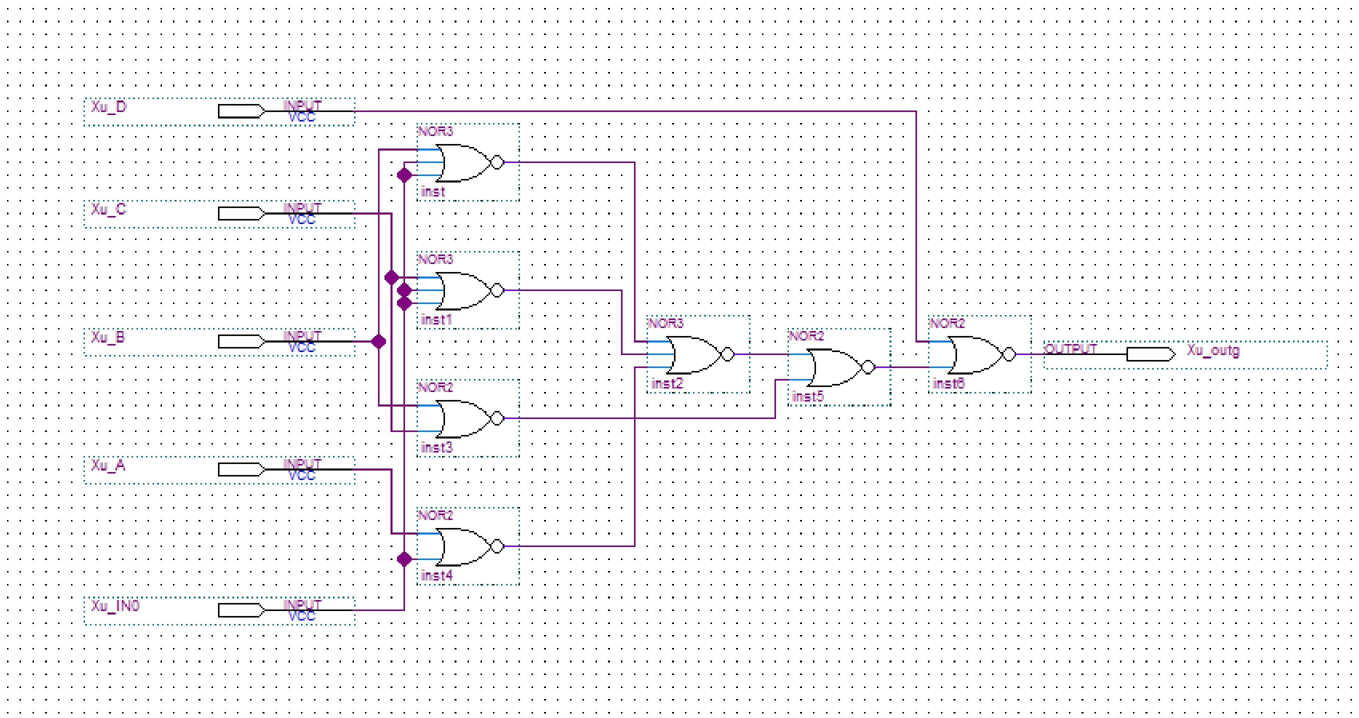


Figure 4: Block diagram for b using only NOR gates.

Since c will be 1 which mean the light will be off when the decimal value of 2 is being displayed which means it will have a Boolean function of

c = D' C' B A'

and by implementing the Boolean function onto the program call logic friday it will produce a circuit using only NOR gates which looks like this

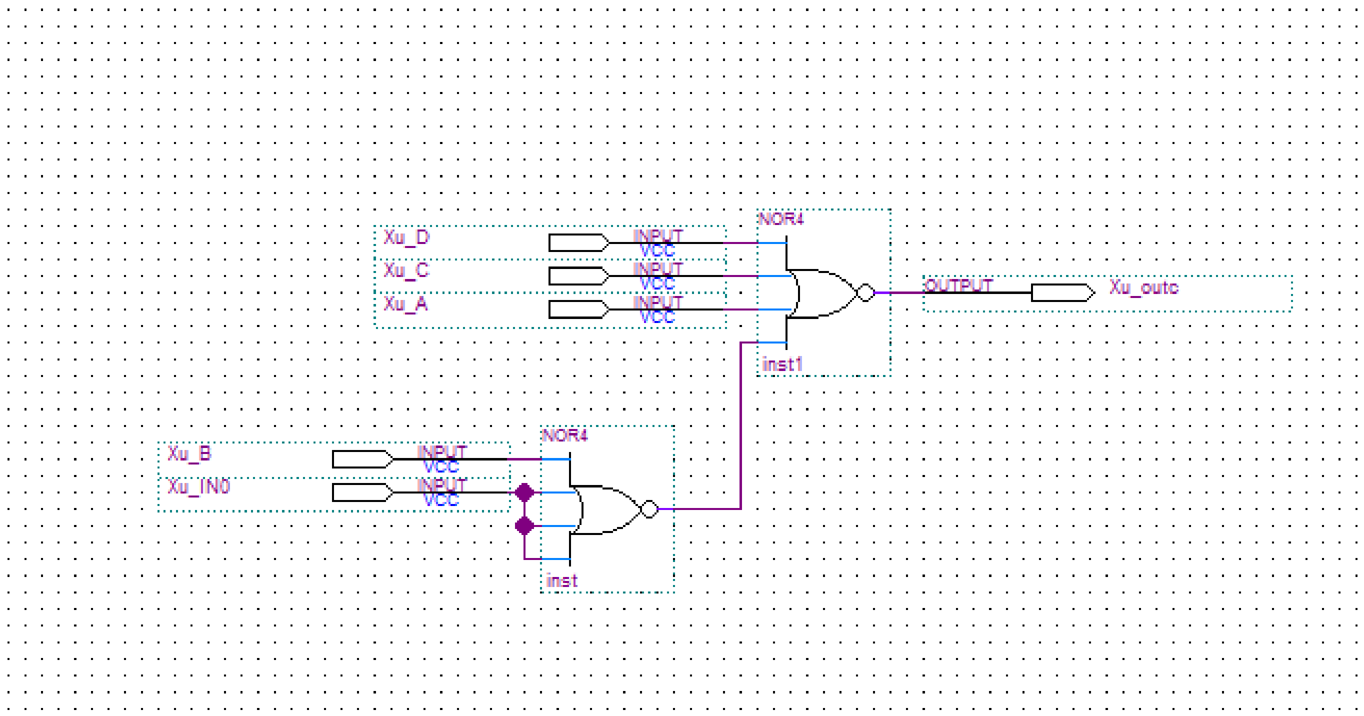


Figure 5: Block diagram for c using only NOR gates.

Since d will be 1 which mean the light will be off when the decimal value of 1, 4, 7 or 9 is being displayed which means it will have a Boolean function of

$$d = C' B' A + D' C B A + D' C B' A'$$

and by implementing the Boolean function onto the program call logic friday it will produce a circuit using only NOR gates which looks like this

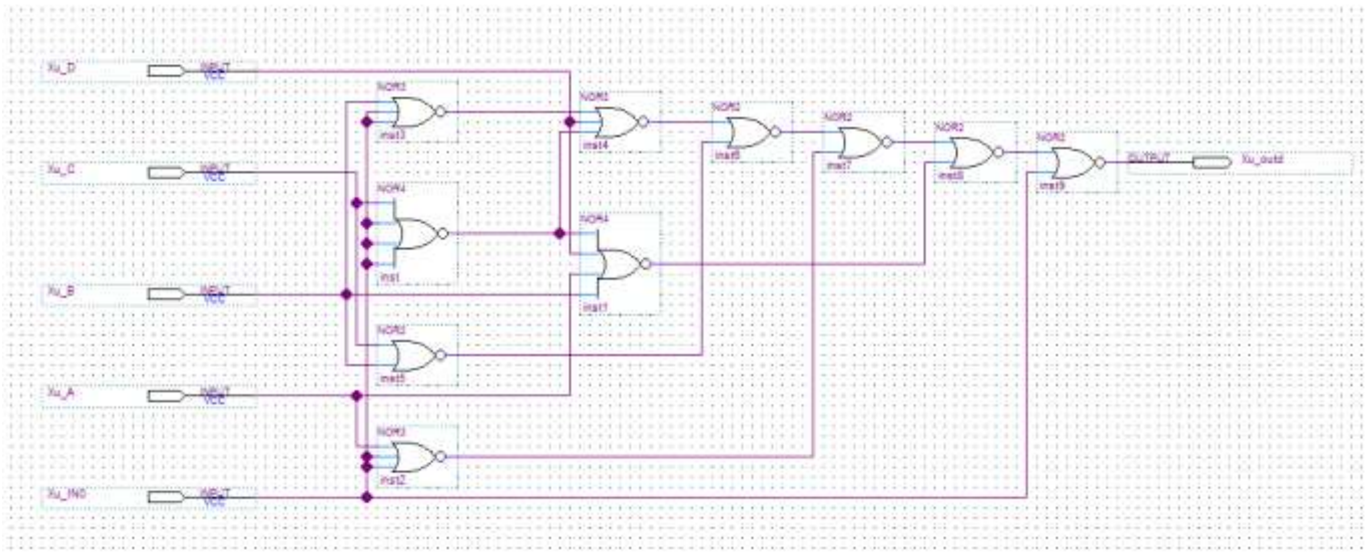


Figure 6: Block diagram for d using only NOR gates.

Since e will be 1 which mean the light will be off when the decimal value of 1, 3, 4, 5, 7 or 9 is being displayed which means it will have a Boolean function of

$$e = D' A + D' C B' + C' B' A$$

and by implementing the Boolean function onto the program call logic friday it will produce a circuit using only NOR gates which looks like this

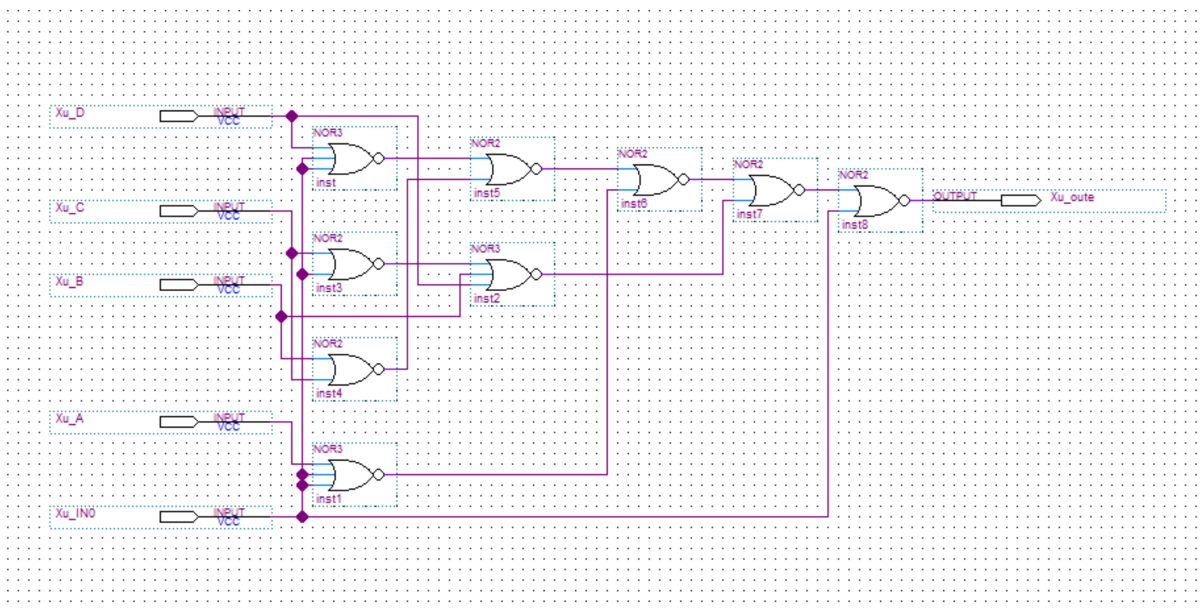


Figure 7: Block diagram for e using only NOR gates.

Since f will be 1 which mean the light will be off when the decimal value of 1, 2, 3, or 7 is being displayed which means it will have a Boolean function of

$$f = D' C' B + D' C' A + D' B A$$

and by implementing the Boolean function onto the program call logic friday it will produce a circuit using only NOR gates which looks like this

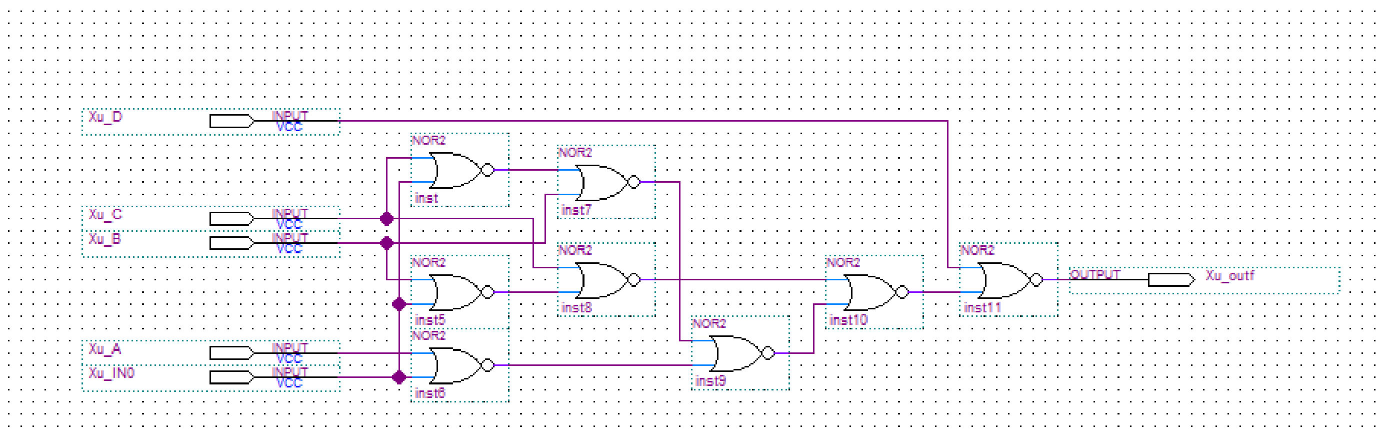


Figure 8: Block diagram for f using only NOR gates.

Since g will be 1 which mean the light will be off when the decimal value of 0, 1 or 7 is being displayed which means it will have a Boolean function of

$$g = D' C' B' + D' C B A$$

and by implementing the Boolean function onto the program call logic friday it will produce a circuit using only NOR gates which looks like this

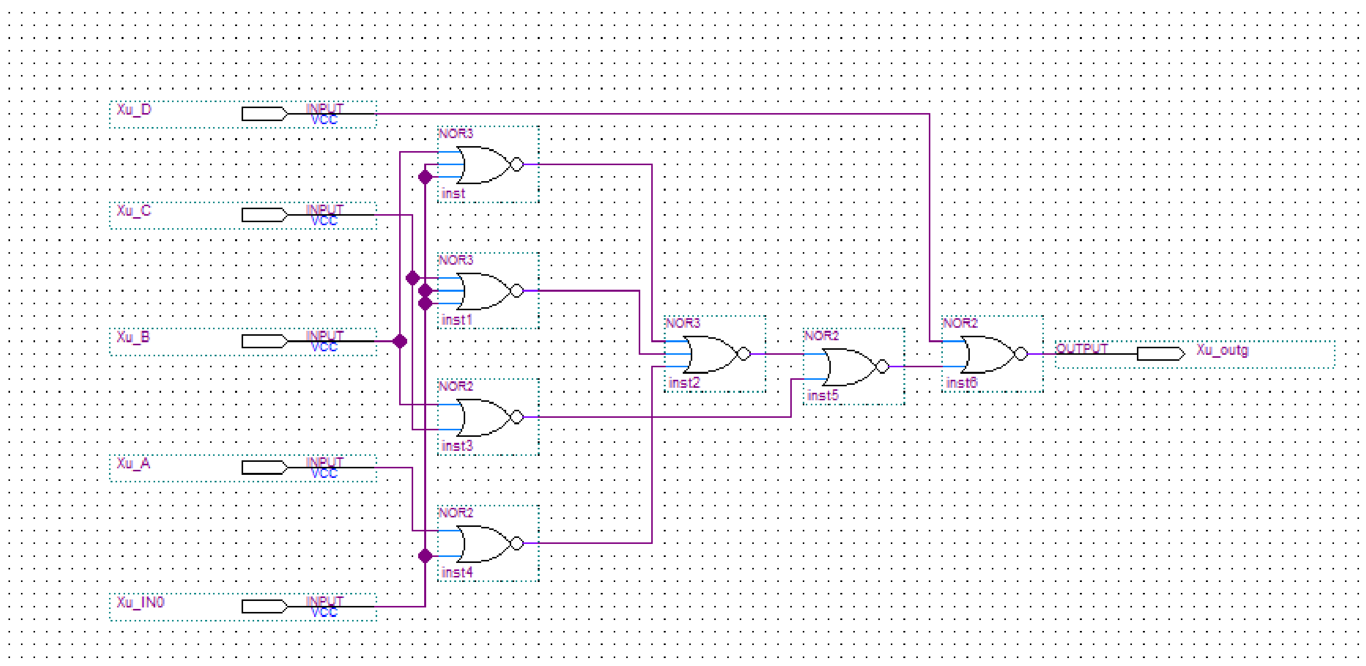


Figure 9: Block diagram for g using only NOR gates.

Now we turn the block diagrams for a to g into a symbol and put it together into one block diagram creating the seven-segment decoder. The circuit mapped by logic Friday has inputs 0 since it is using only NOR gates, it'll be 1 if it was using only NAND gates. Therefore the 0 input will be replaced by GND gate which means it will be defaulted as input 0 so there won't be a need for a switch for that input.

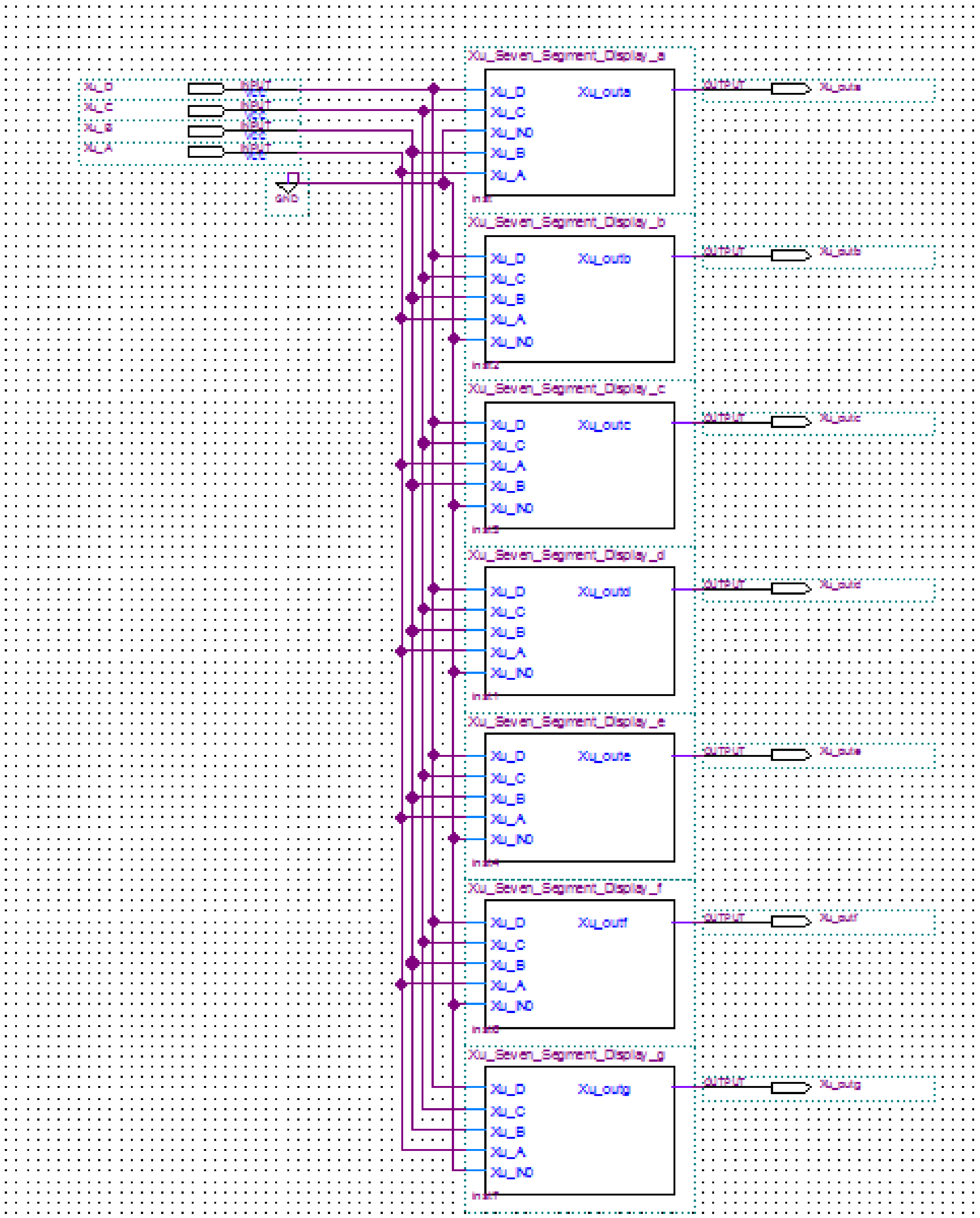
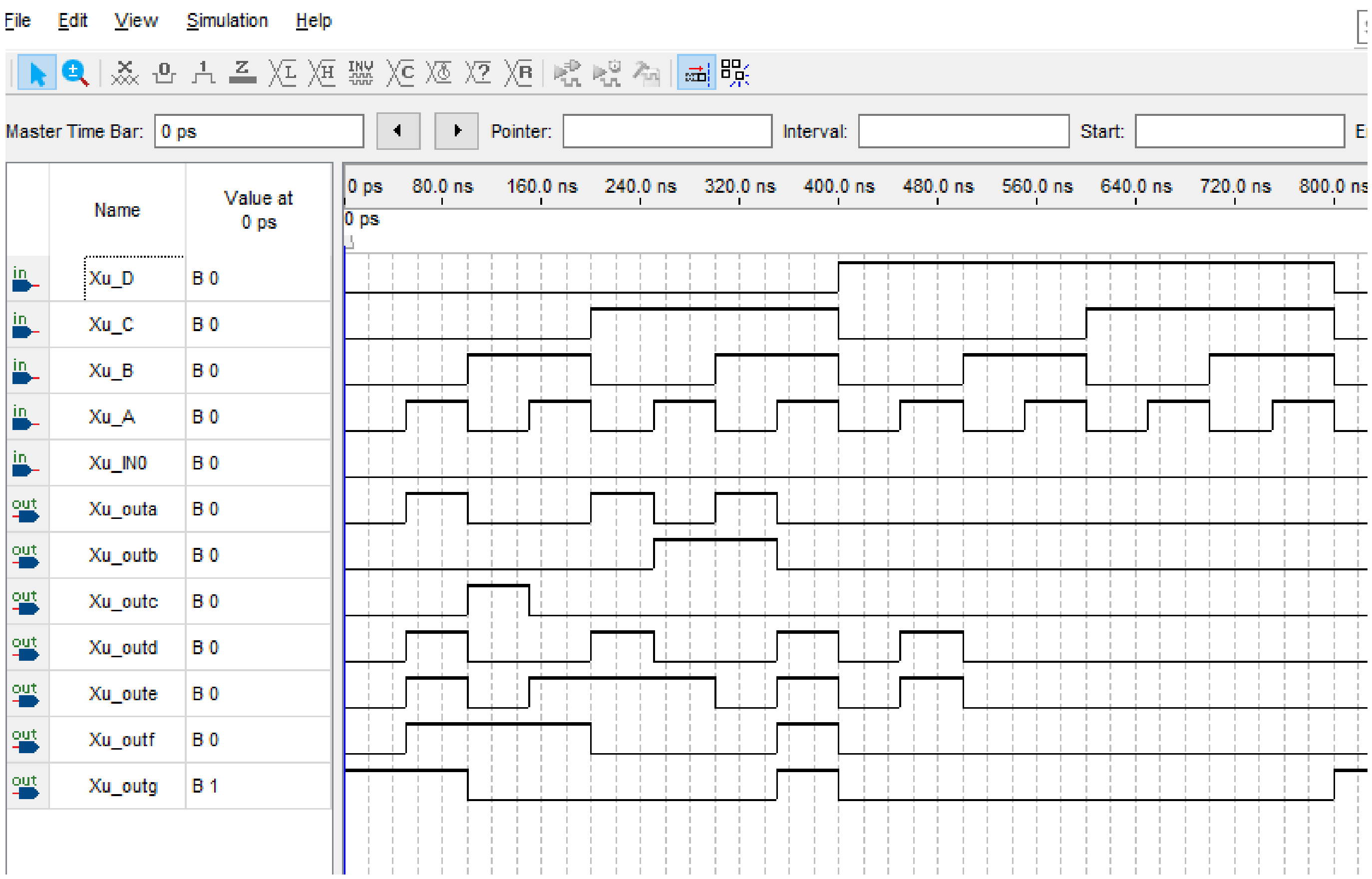

Xu_D
Xu_C
Xu_B
Xu_A
INPUT VCC
GND
Xu_Seven_Segment_Display_a
Xu_Seven_Segment_Display_b
Xu_Seven_Segment_Display_c
Xu_Seven_Segment_Display_d
Xu_Seven_Segment_Display_e
Xu_Seven_Segment_Display_f
Xu_Seven_Segment_Display_g
Xu_NO
Xu_outa
Xu_outb
Xu_outc
Xu_outd
Xu_oute
Xu_outf
Xu_outg
OUTPUT

Figure 10: Seven-segment decoder circuit.

2.2 Simulation

In the simulation, we will give values of 0 and 1 to the inputs at varying intervals of all possible inputs. Input D will have value of 0 and 1 at each 800-ns interval. Input C will have value of 0 and 1 at each 400-ns interval. Input B will have value of 0 and 1 at each 200-ns interval. Input A will have value of 0 and 1 at each 100-ns interval.



Name	Value at 0 ps
Xu_D	B 0
Xu_C	B 0
Xu_B	B 0
Xu_A	B 0
Xu_IN0	B 0
Xu_outa	B 0
Xu_outb	B 0
Xu_outc	B 0
Xu_outd	B 0
Xu_oute	B 0
Xu_outf	B 0
Xu_outg	B 1

Figure 11: Vector waveform simulation of seven-segment decoder.

Since the seven-segment decoder will only display decimal numbers from 0 to 9 we can see that from the inputs 1010 to 1111 which represents 10 to 15 will have no lights turned off which is displaying 8 and it is normal because the seven-segment decoder only functions to display 0 to 9. As we can see that the inputs 0000 to 1001 outputting 0 to 9 corresponds to the truth table.

2.3 Demonstration

The PIN assignment of the inputs and outputs on the DE1-SoC Board.

Xu_A is assigned to SW[0] which is PIN_AB12

Xu_B is assigned to SW[1] which is PIN_AC12

Xu_C is assigned to SW[2] which is PIN_AF9

Xu_D is assigned to SW[3] which is PIN_AF10

Xu_outa is assigned to HEX0[0] which is PIN_AE26

Xu_outb is assigned to HEX0[1] which is PIN_AE27

Xu_outc is assigned to HEX0[2] which is PIN_AE28

Xu_outd is assigned to HEX0[3] which is PIN_AG27

Xu_oute is assigned to HEX0[4] which is PIN_AF28

Xu_outf is assigned to HEX0[5] which is PIN_AG28

Xu_outg is assigned to HEX0[6] which is PIN_AH28

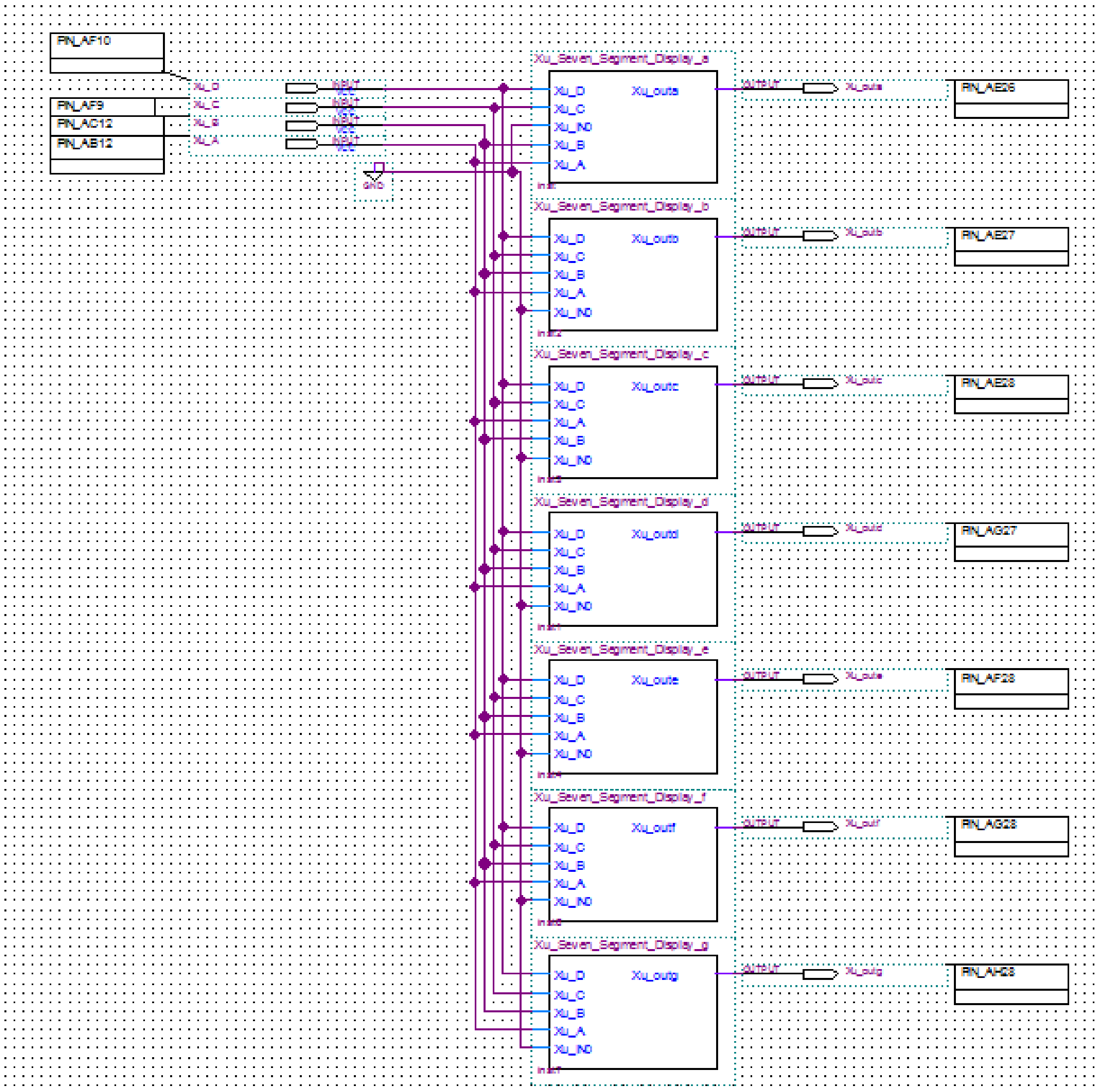

PIN_AF10
PIN_AF9
PIN_AC12
PIN_AB12
Xu_D
Xu_C
Xu_B
Xu_A
INPUT VCC
GND
Xu_Seven_Segment_Display_a
Xu_Seven_Segment_Display_b
Xu_Seven_Segment_Display_c
Xu_Seven_Segment_Display_d
Xu_Seven_Segment_Display_e
Xu_Seven_Segment_Display_f
Xu_Seven_Segment_Display_g
Xu_IND
Xu_outa
Xu_outb
Xu_outc
Xu_outd
Xu_oute
Xu_outf
Xu_outg
OUTPUT
PIN_AE26
PIN_AE27
PIN_AE28
PIN_AG27
PIN_AF28
PIN_AG28
PIN_AH28

Figure 12: PIN assignments of the circuit to DE1-SoC Board.

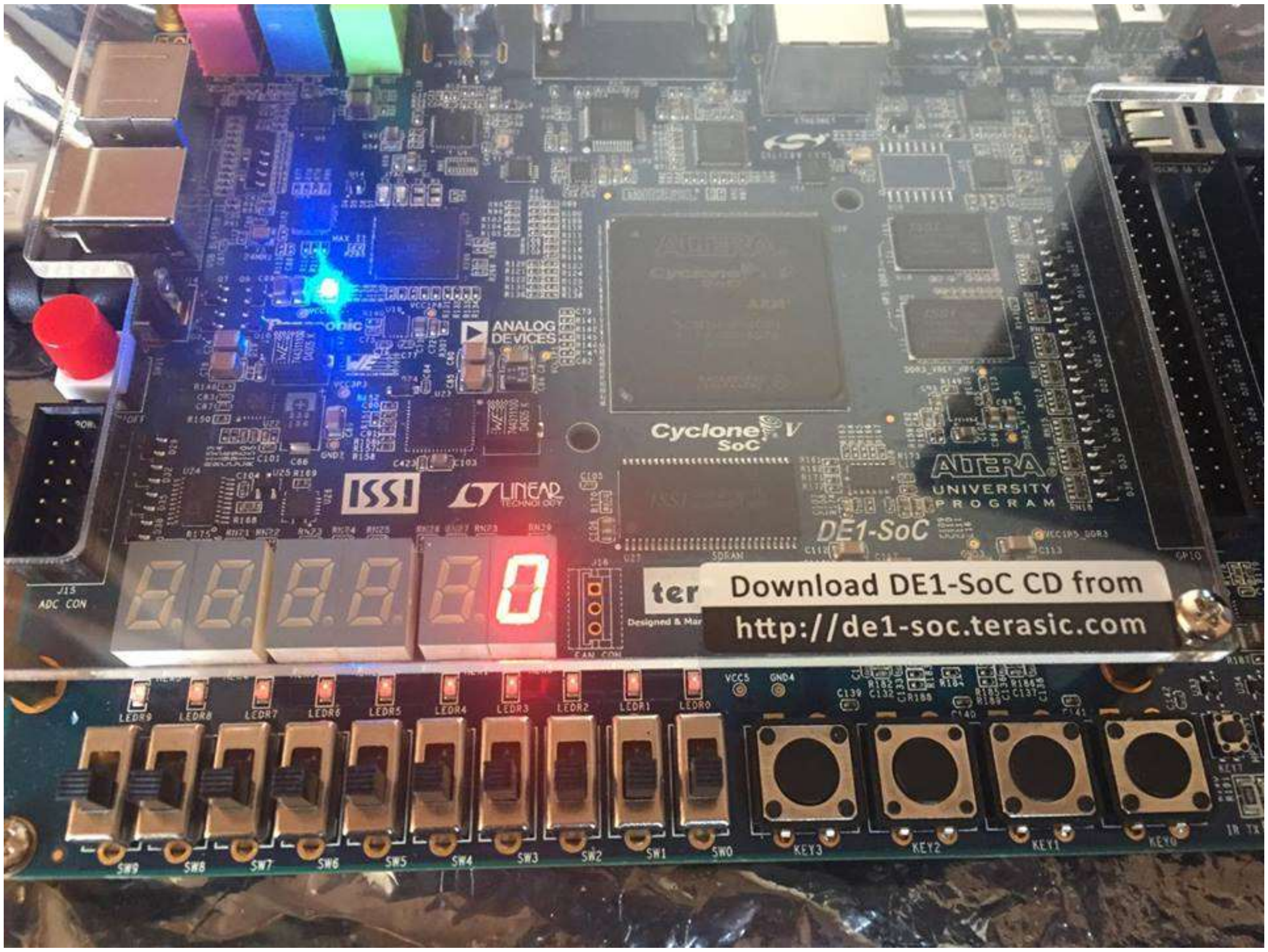


Figure 13: Input 0000 g is off

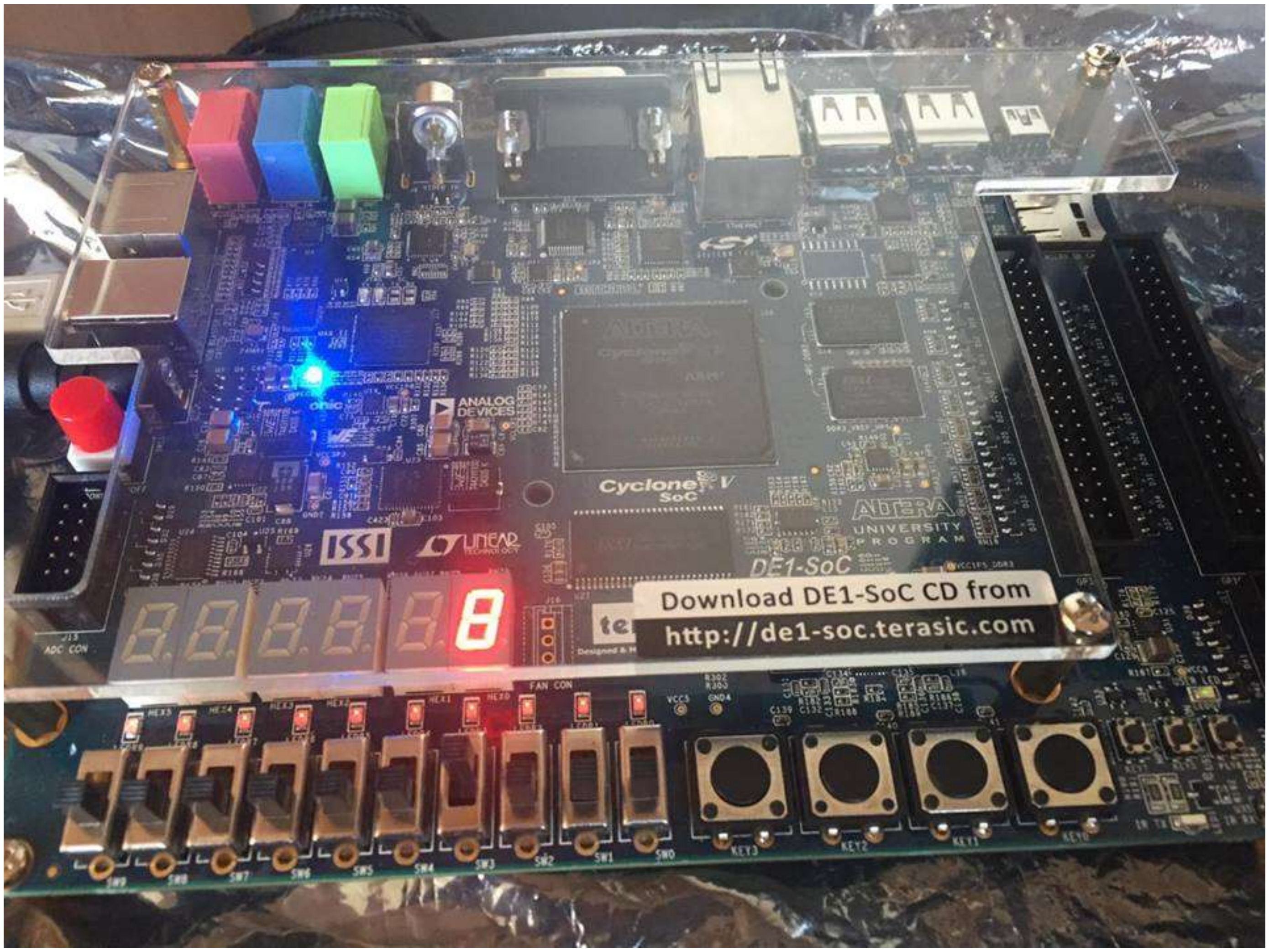


Figure 14: Input 1000 no lights off

3. Binary to Decimal

3.1 Functionality and Specification

The binary to decimal circuit will read 4-bit binary input and the values onto the seven-segment display but this time it will be displaying numbers from 0 to 15 which are the values you can get from a 4-bit binary 0000 to 1111. Since it will be displaying on the seven-segment display, we will be using the seven-segment decoder we have created as one of the component that will make up the circuit for binary to decimal.

HEX1	HEX0	D	C	B	A	a3	a2	a1	a0	Z
0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	1	0
0	2	0	0	1	0	0	0	1	0	0
0	3	0	0	1	1	0	0	1	1	0
0	4	0	1	0	0	0	1	0	0	0
0	5	0	1	0	1	0	1	0	1	0
0	6	0	1	1	0	0	1	1	0	0
0	7	0	1	1	1	0	1	1	1	0
0	8	1	0	0	0	1	0	0	0	0
0	9	1	0	0	1	1	0	0	1	0
1	0	1	0	1	0	1	0	0	0	1
1	1	1	0	1	1	1	0	0	1	1
1	2	1	1	0	0	1	0	1	0	1
1	3	1	1	0	1	1	0	1	1	1
1	4	1	1	1	0	1	1	0	0	1
1	5	1	1	1	1	1	1	0	1	1

Figure 15: Truth table for binary to decimal.

Based on the truth table we can determine the components needed to create the circuit for binary to decimal. First, we will need a comparator outputting Z, the use of the comparator is to tell if the 4-bit binary input is greater than 9 or not. The Boolean function of the comparator will be D AND (C OR B).

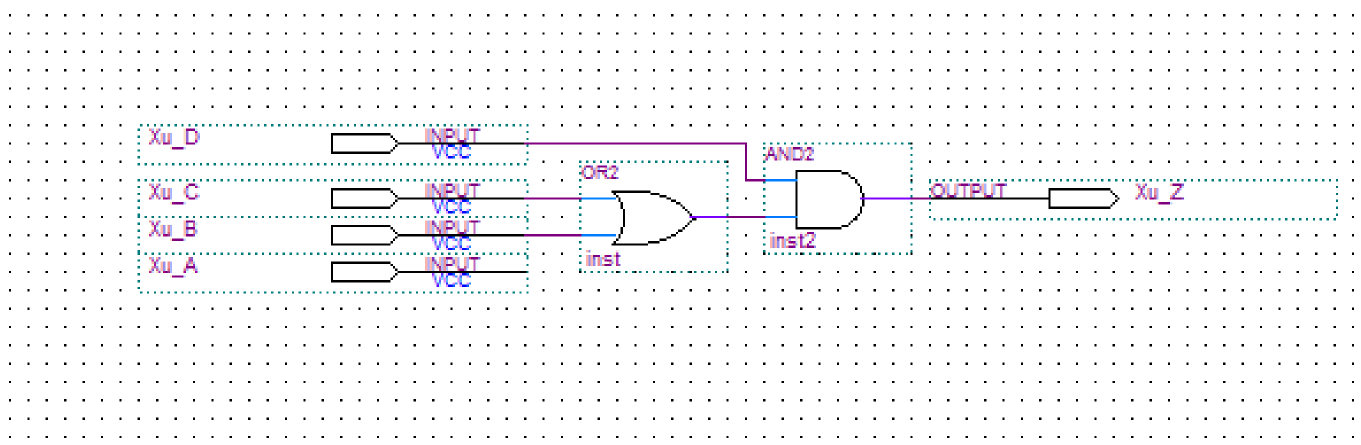


Figure 16: Block diagram of comparator.

Next, we will need a Circuit One which outputs a2, a1, and a0 and these will be the new inputs to go into the seven-segment decoder if the decimal value is greater than 9. For example, 12 in binary is 1100 but it will not go through the seven-segment decoder since it can only read binary from 0 to 9 which is 0000 to 1001, so we need to alternate the 100 into 010 to display 2 on the seven-segment display on d0. The Boolean function for Circuit One are

a2 = C AND B

a1 = C AND NOT B

a0 = (B AND A) OR (C AND A)

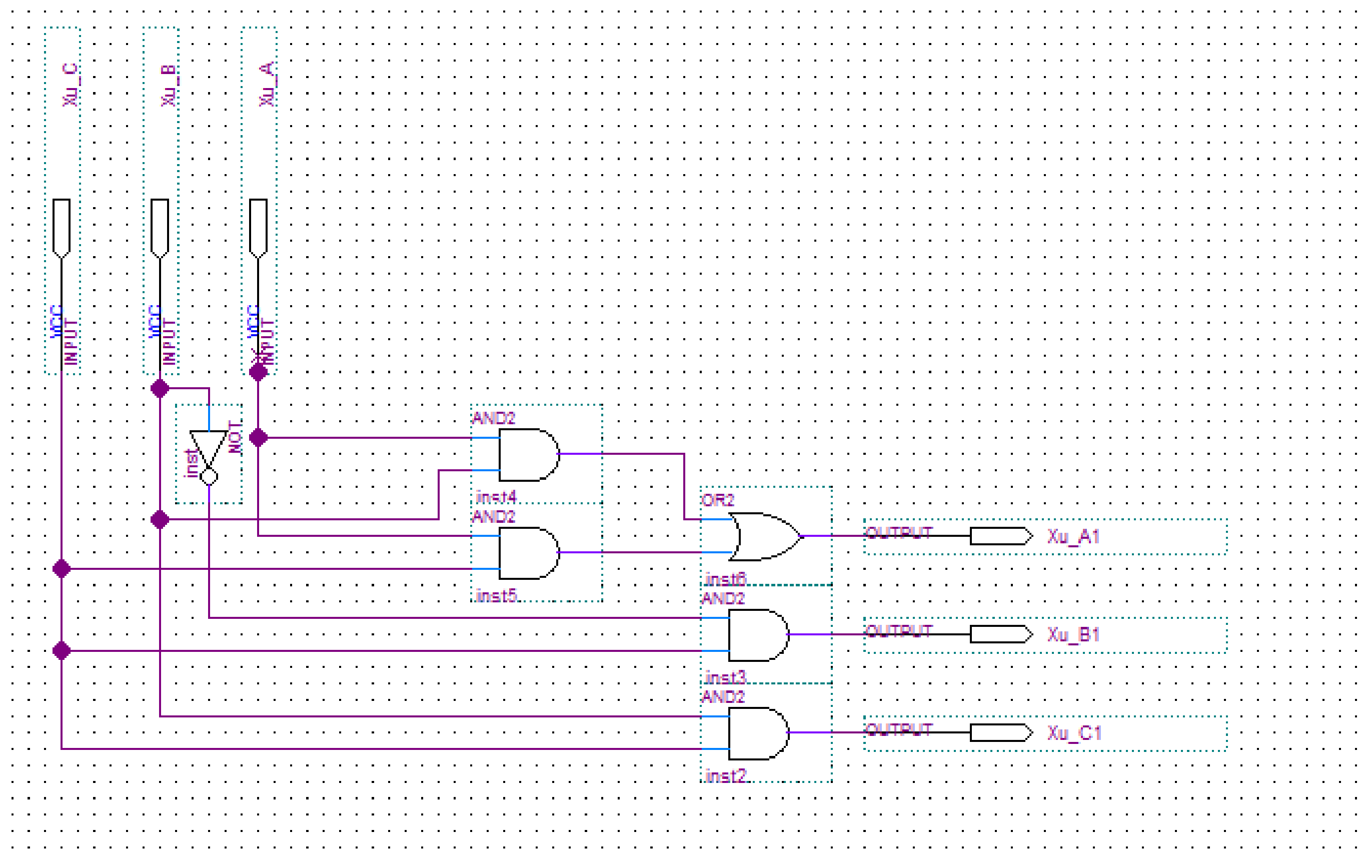


Figure 17: Block diagram for Circuit One.

Then, we will need a Circuit Two which is determined by Z to output d1 to show that if the decimal number is greater than 9 or not. Circuit Two will be a seven-segment display for either 0 or 1.

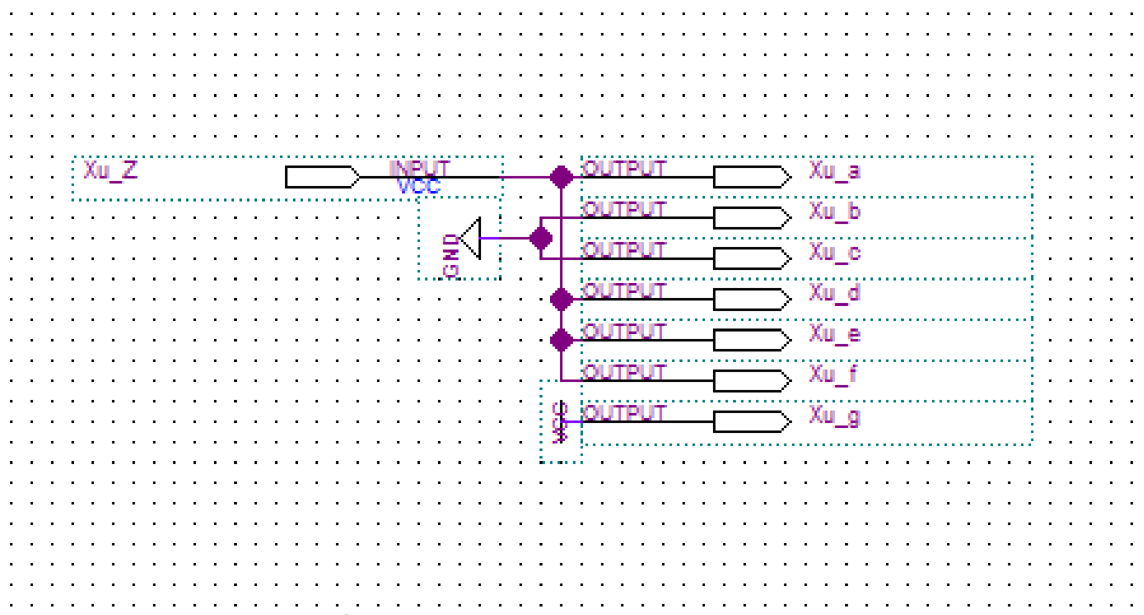


Figure 18: Block diagram for Circuit Two.

Lastly, we will make a symbol for the seven-segment decoder we have created and four 2:1 multiplexers using Z as the selector to determine the inputs for the seven-segment decoder.

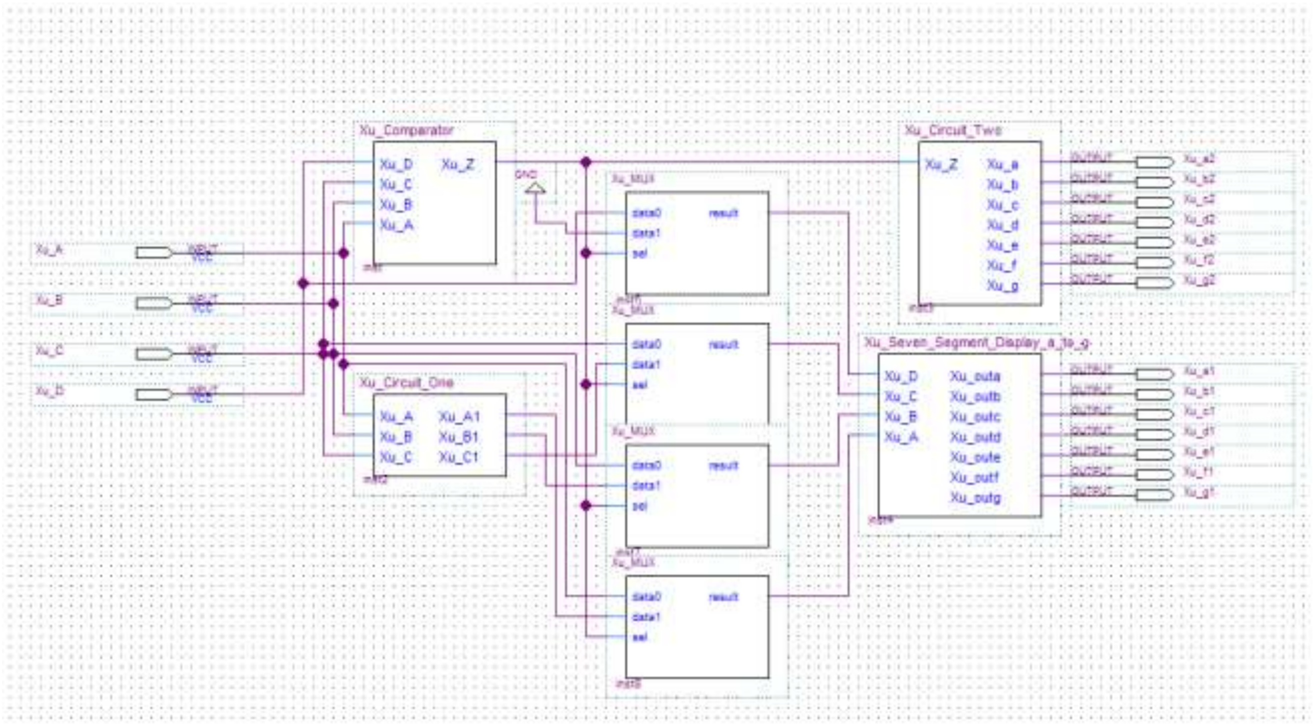
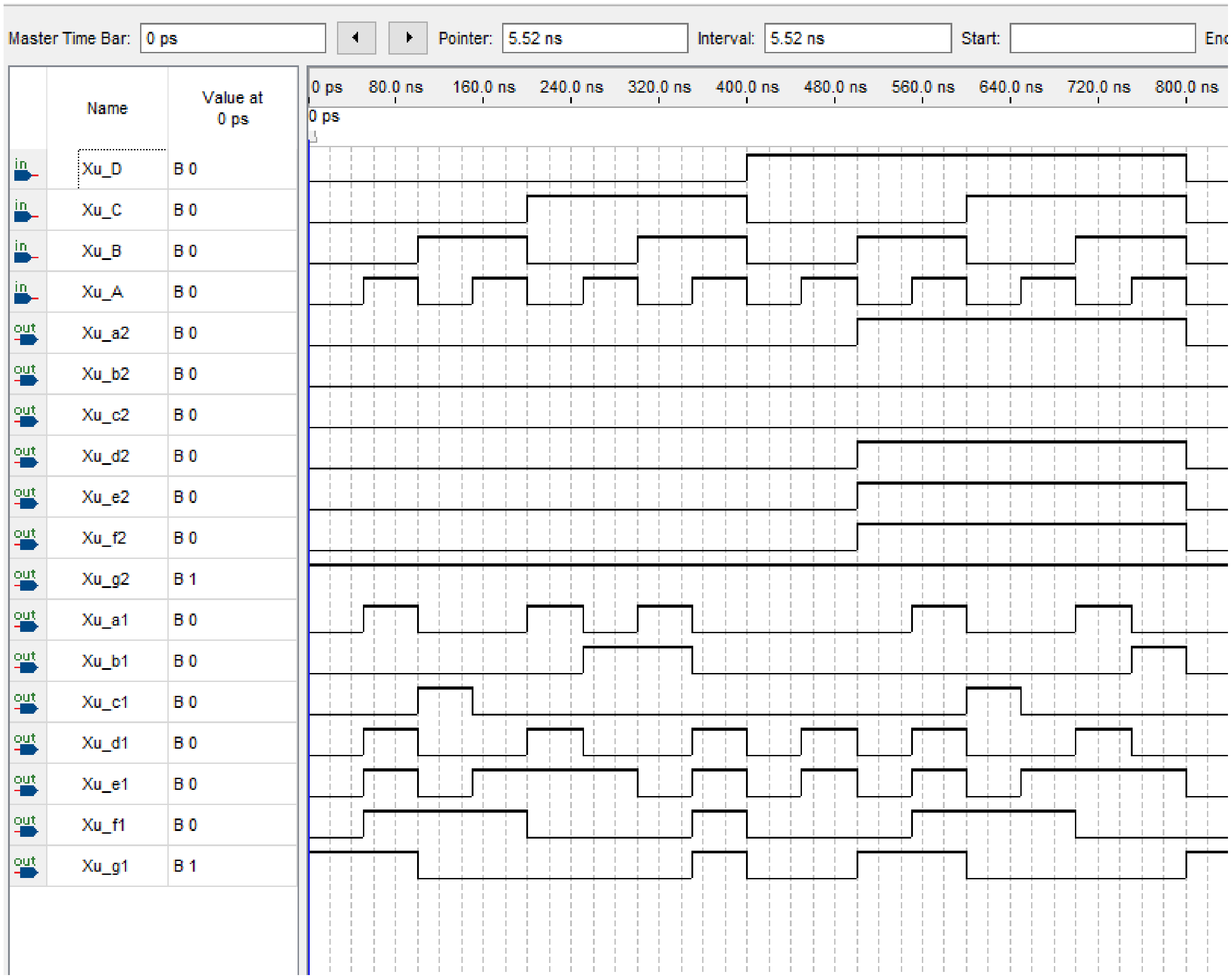


Figure 19: Block diagram of binary to decimal.

3.2 Simulation

In the simulation, we will give values of 0 and 1 to the inputs at varying intervals of all possible inputs. Input D will have value of 0 and 1 at each 800-ns interval. Input C will have value of 0 and 1 at each 400-ns interval. Input B will have value of 0 and 1 at each 200-ns interval. Input A will have value of 0 and 1 at each 100-ns interval.



	Name	Value at 0 ps
in	Xu_D	B 0
in	Xu_C	B 0
in	Xu_B	B 0
in	Xu_A	B 0
out	Xu_a2	B 0
out	Xu_b2	B 0
out	Xu_c2	B 0
out	Xu_d2	B 0
out	Xu_e2	B 0
out	Xu_f2	B 0
out	Xu_g2	B 1
out	Xu_a1	B 0
out	Xu_b1	B 0
out	Xu_c1	B 0
out	Xu_d1	B 0
out	Xu_e1	B 0
out	Xu_f1	B 0
out	Xu_g1	B 1

Figure 20: Vector waveform simulation of Binary to Decimal.

We can observe that the simulation corresponds to the outputs of d0 and d1 on the truth table.

3.3 Demonstration

The PIN assignment of the inputs and outputs on the DE1-SoC Board.

Xu_A is assigned to SW[0] which is PIN_AB12

Xu_B is assigned to SW[1] which is PIN_AC12

Xu_C is assigned to SW[2] which is PIN_AF9

Xu_D is assigned to SW[3] which is PIN_AF10

Xu_a1 is assigned to HEX0[0] which is PIN_AE26

Xu_b1 is assigned to HEX0[1] which is PIN_AE27

Xu_c1 is assigned to HEX0[2] which is PIN_AE28

Xu_d1 is assigned to HEX0[3] which is PIN_AG27

Xu_e1 is assigned to HEX0[4] which is PIN_AF28

Xu_f1 is assigned to HEX0[5] which is PIN_AG28

Xu_g1 is assigned to HEX0[6] which is PIN_AH28

Xu_a2 is assigned to HEX1[0] which is PIN_AJ29

Xu_b2 is assigned to HEX1[1] which is PIN_AH29

Xu_c2 is assigned to HEX1[2] which is PIN_AH30

Xu_d2 is assigned to HEX1[3] which is PIN_AG30

Xu_e2 is assigned to HEX1[4] which is PIN_AF29

Xu_f2 is assigned to HEX1[5] which is PIN_AF30

Xu_g2 is assigned to HEX1[6] which is PIN_AD27

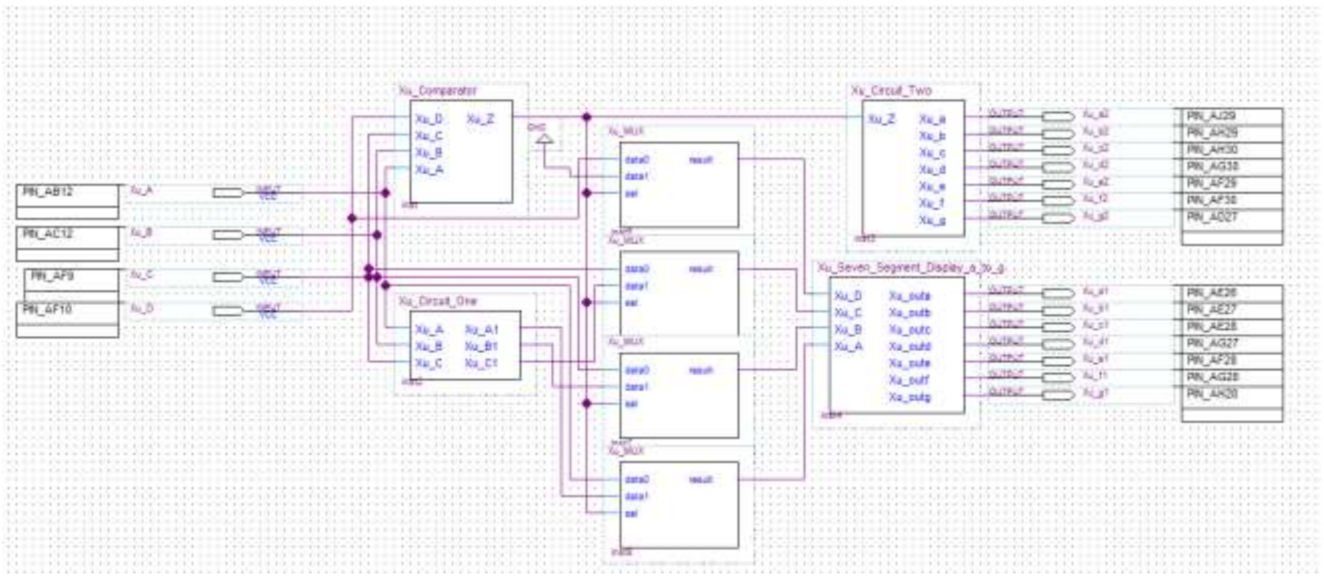


Figure 21: PIN assignment of the circuit to DE1-SoC Board.

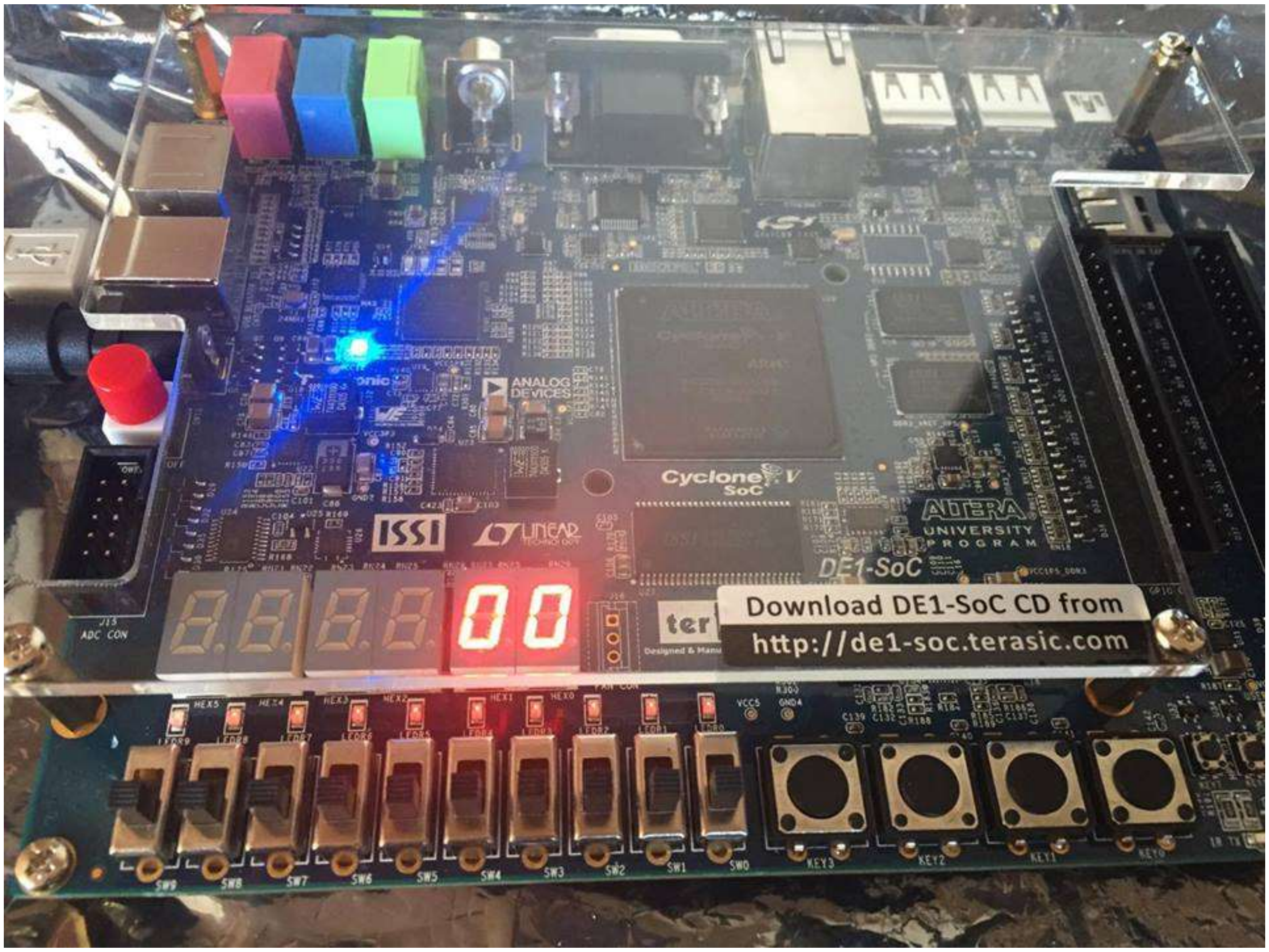


Figure 22: Input 0000 HEX1 display 0 HEX0 display 0

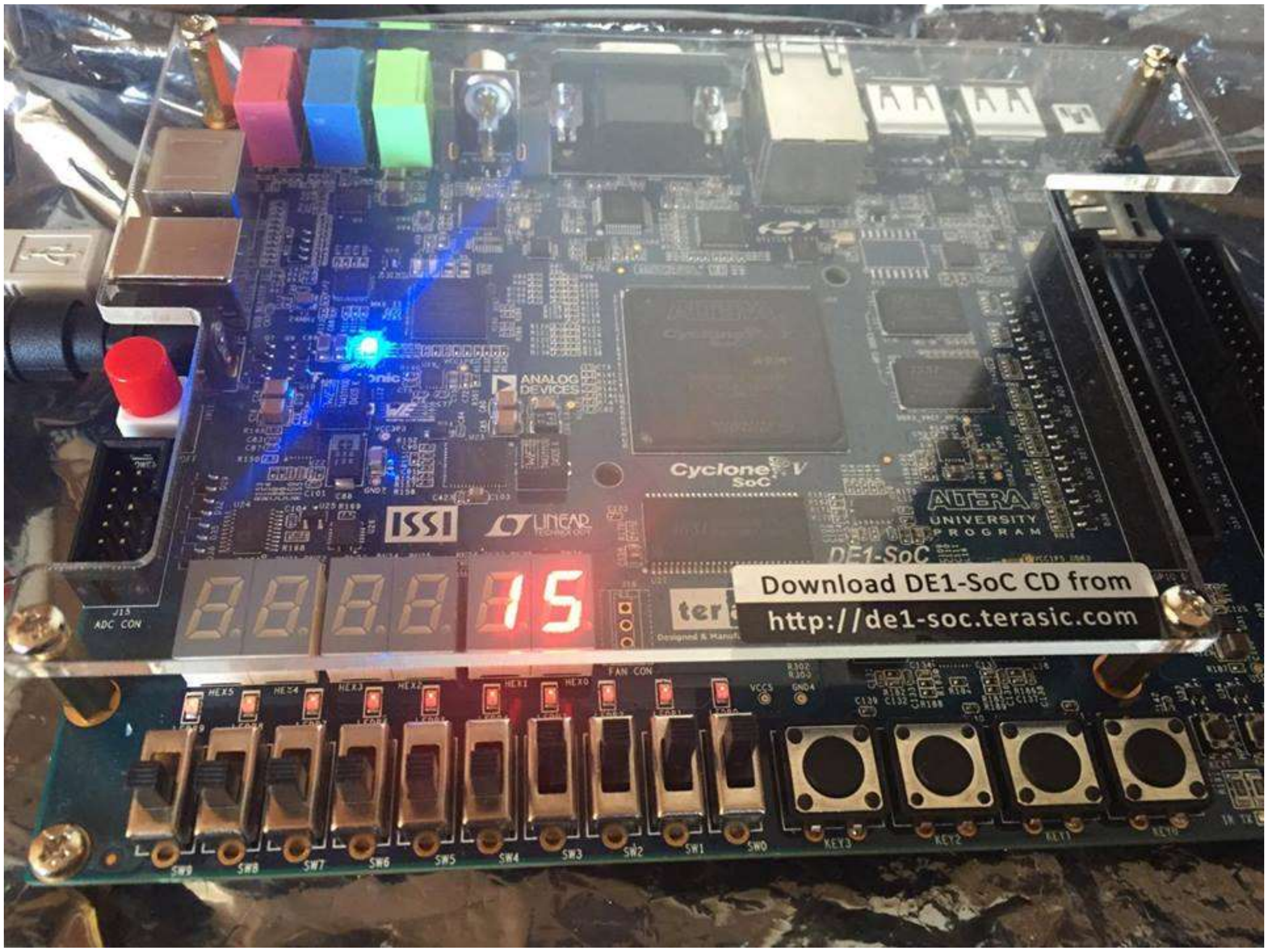
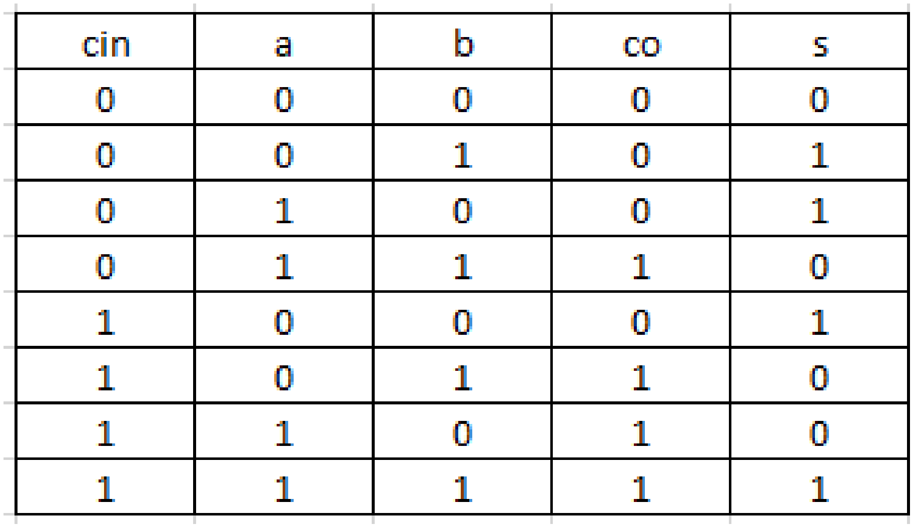


Figure 23: Input 1111 HEX1 display 1 HEX0 display 5

4. 4-Bit Full Adder using Bus Tool

4.1 Functionality and Specification

The 4-bit full adder is something we have already designed in one of the previous labs, but this time we will be doing it by using the orthogonal bus tool. This is the truth table for a full adder.



cin	a	b	co	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

Figure 22: Truth table for 1-bit full adder.

Then we construct the block diagram for a 1-bit full adder using two XOR gates and a 2:1 multiplexer.

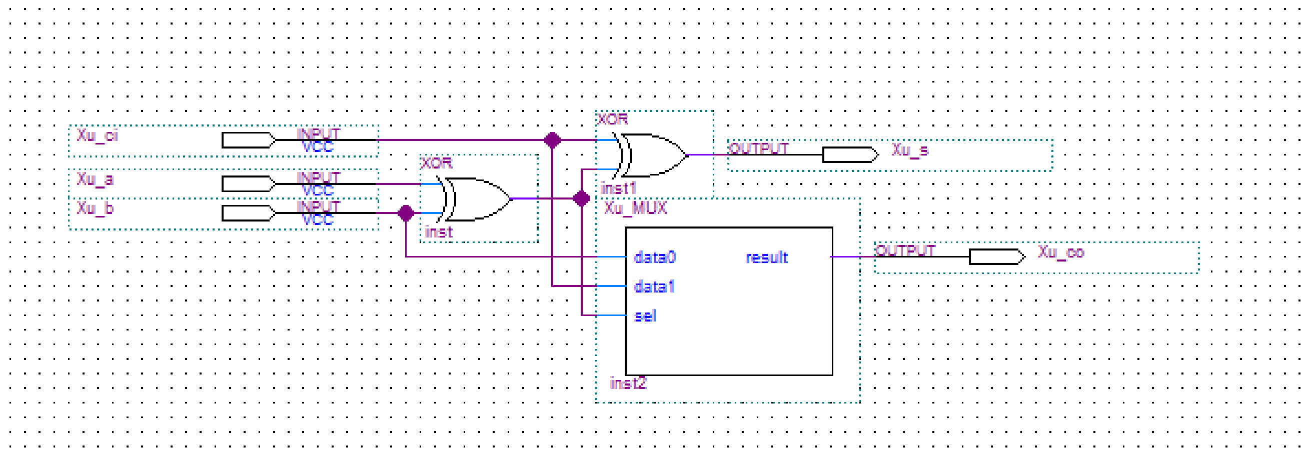


Figure 23: Block diagram for 1-Bit Full Adder.

Then we make a symbol for the 1-bit full adder and by putting four together we can make a 4-bit full adder. Normally it would look like this

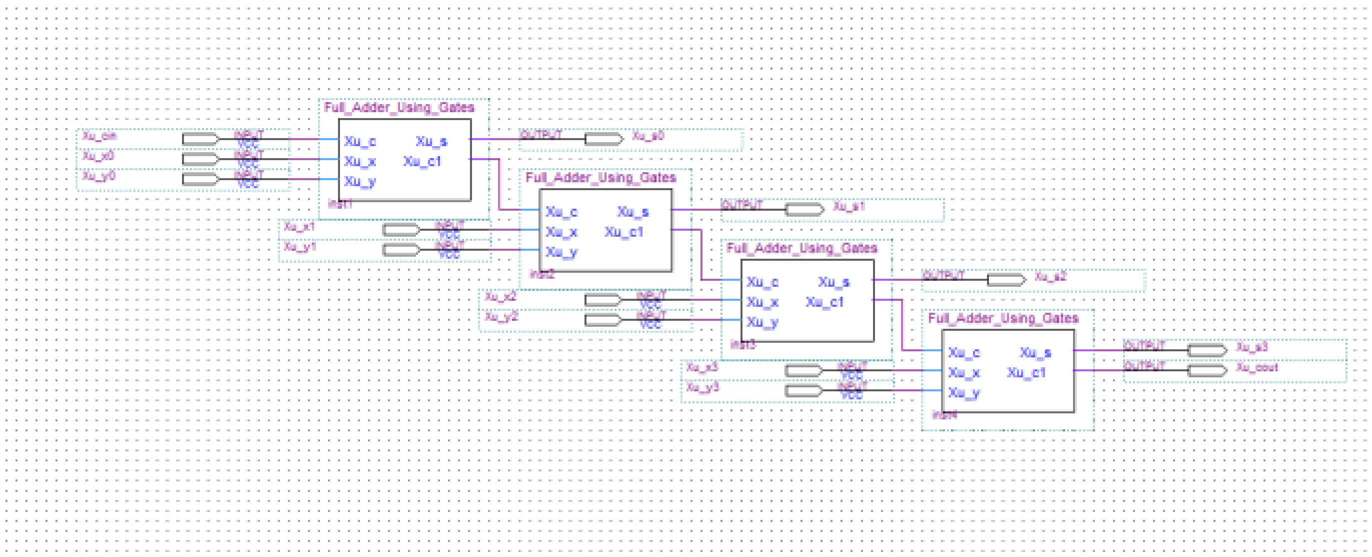


Figure 24: Block diagram for 4-Bit Full Adder.

But this time we are introduced to the orthogonal bus tool. What it does is that it can help you shorten the amount of inputs and outputs by grouping them and circuit lines by connecting it to one bus line and differentiate where the circuit lines go by naming the lines. Therefore the 4-bit full adder using bus tool will look like this

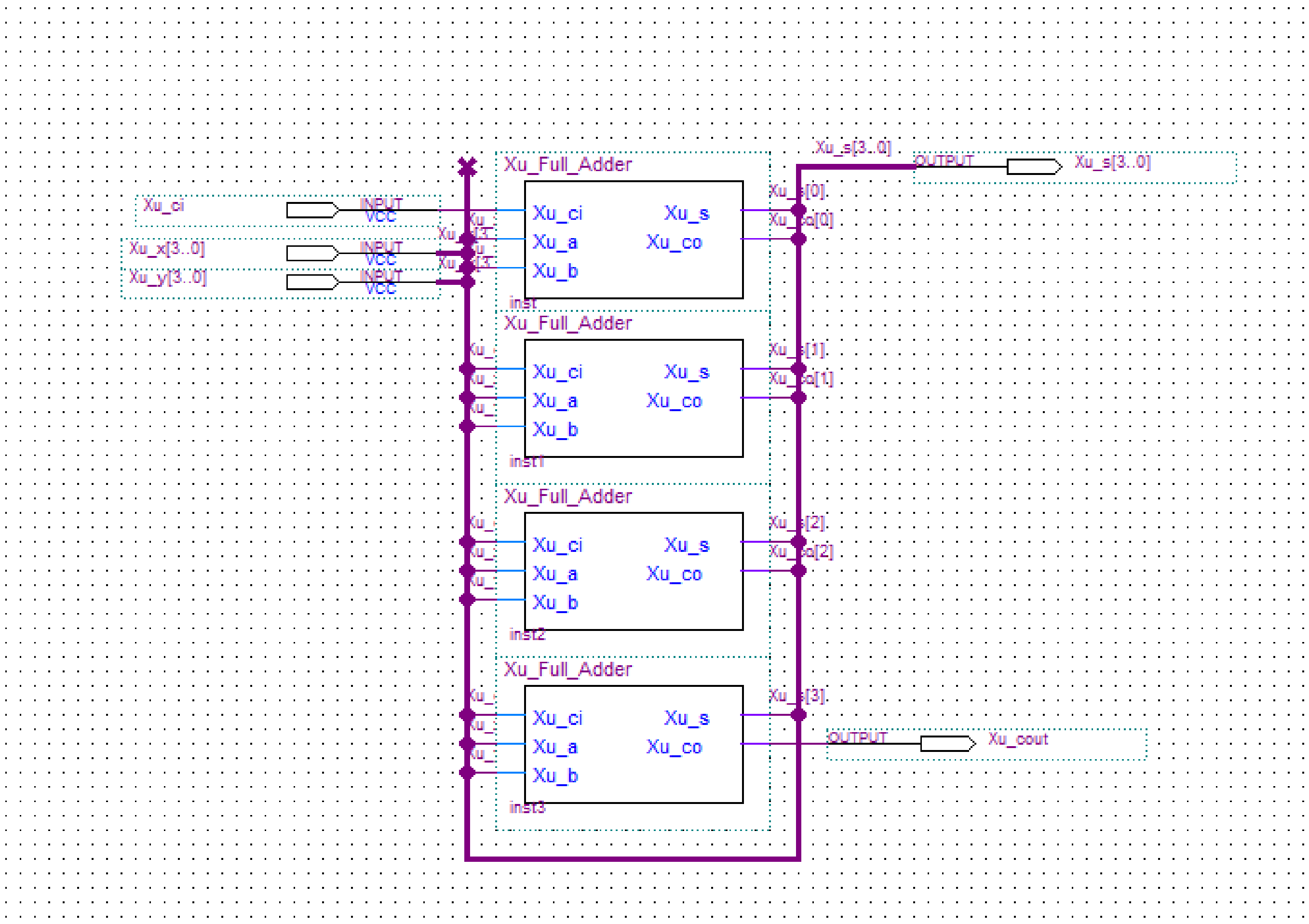


Figure 25: Block diagram for 4-Bit Full Adder using Bus Tool.

4.2 Simulation

In the simulation, we can insert signed decimal value for the grouped input x and y and it will produce a signed decimal value for the grouped output s. Case 1 positive adding positive outputs positive with no carry out. Case 2 positive adding negative outputs negative with not carry out. Case 3 positive adding negative outputs negative with carry out. Case 4 positive adding negative outputs positive with carry out. Case 5 positive adding positive outputs overflow. Case 6 negative adding negative outputs overflow. We can observe that the 4-bit full adder using bus tool is the same as the 4-bit full adder we have created in the previous labs.

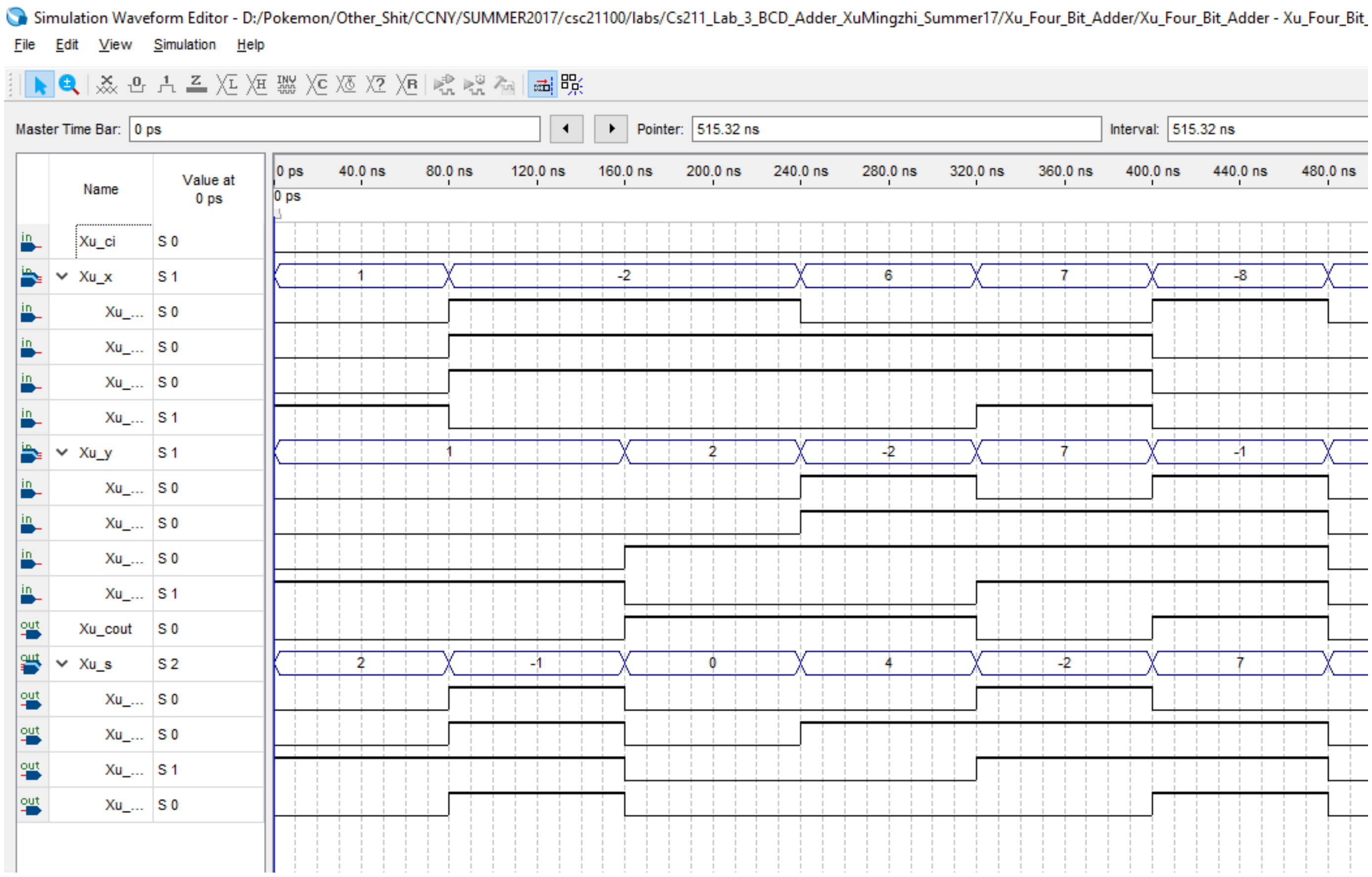

Simulation Waveform Editor - D:/Pokemon/Other_Shit/CCNY/SUMMER2017/csc21100/labs/Cs211_Lab_3_BCD_Adder_XuMingzhi_Summer17/Xu_Four_Bit_Adder/Xu_Four_Bit_Adder - Xu_Four_Bit_
File Edit View Simulation Help
Master Time Bar: 0 ps
Pointer: 515.32 ns
Interval: 515.32 ns
Name
Value at 0 ps
0 ps 40.0 ns 80.0 ns 120.0 ns 160.0 ns 200.0 ns 240.0 ns 280.0 ns 320.0 ns 360.0 ns 400.0 ns 440.0 ns 480.0 ns
Xu_ci S 0
Xu_x S 1
1 -2 6 7 -8
Xu_... S 0
Xu_... S 0
Xu_... S 0
Xu_... S 1
Xu_y S 1
1 2 -2 7 -1
Xu_... S 0
Xu_... S 0
Xu_... S 0
Xu_... S 1
Xu_cout S 0
Xu_s S 2
2 -1 0 4 -2 7
Xu_... S 0
Xu_... S 0
Xu_... S 1
Xu_... S 0

Figure 26: Vector waveform simulation for 4-bit full adder using bus tool.

4.3 Demonstration

The PIN assignment of the inputs and outputs on the DE1-SoC Board.

Xu_x[0] is assigned to SW[0] which is PIN_AB12

Xu_x[1] is assigned to SW[1] which is PIN_AC12

Xu_x[2] is assigned to SW[2] which is PIN_AF9

Xu_x[3] is assigned to SW[3] which is PIN_AF10

Xu_y[0] is assigned to SW[4] which is PIN_AD11

Xu_y[1] is assigned to SW[5] which is PIN_AD12

Xu_y[2] is assigned to SW[6] which is PIN_AE11

Xu_y[3] is assigned to SW[7] which is PIN_AC9

Xu_s[0] is assigned to LEDR[0] which is PIN_V16

Xu_s[1] is assigned to LEDR[1] which is PIN_W16

Xu_s[2] is assigned to LEDR[2] which is PIN_V17

Xu_s[3] is assigned to LEDR[3] which is PIN_V18

Xu_cout is assigned to LEDR[4] which is PIN_W17

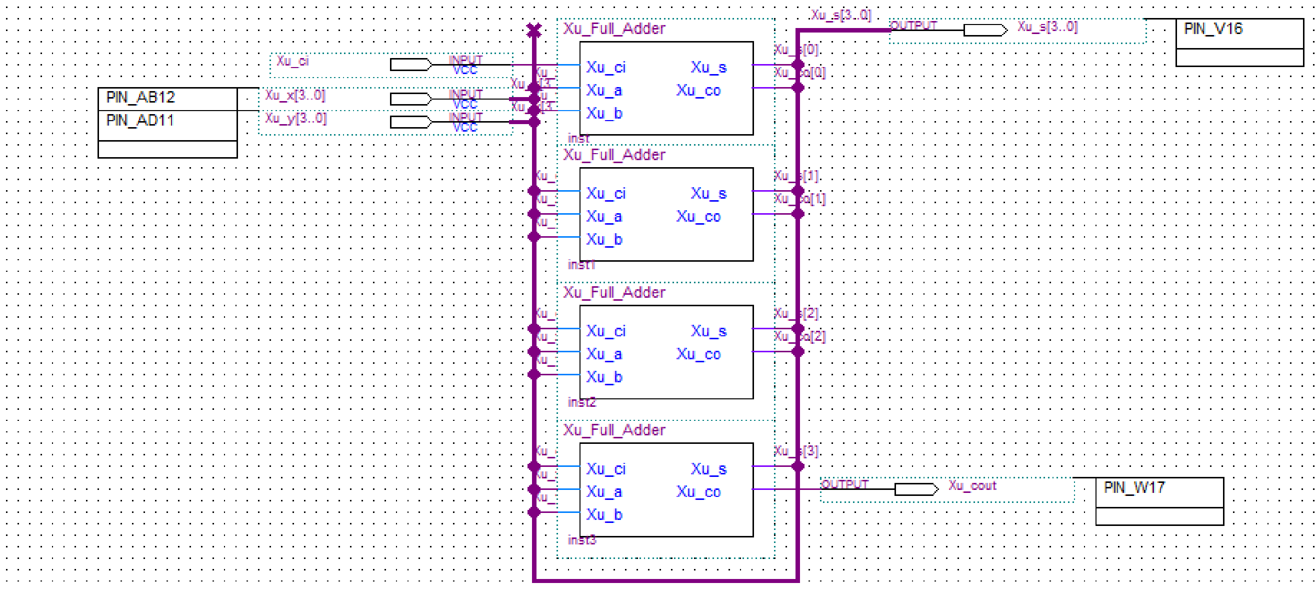


Figure 27: PIN assignment of circuit to DE1-SoC Board.

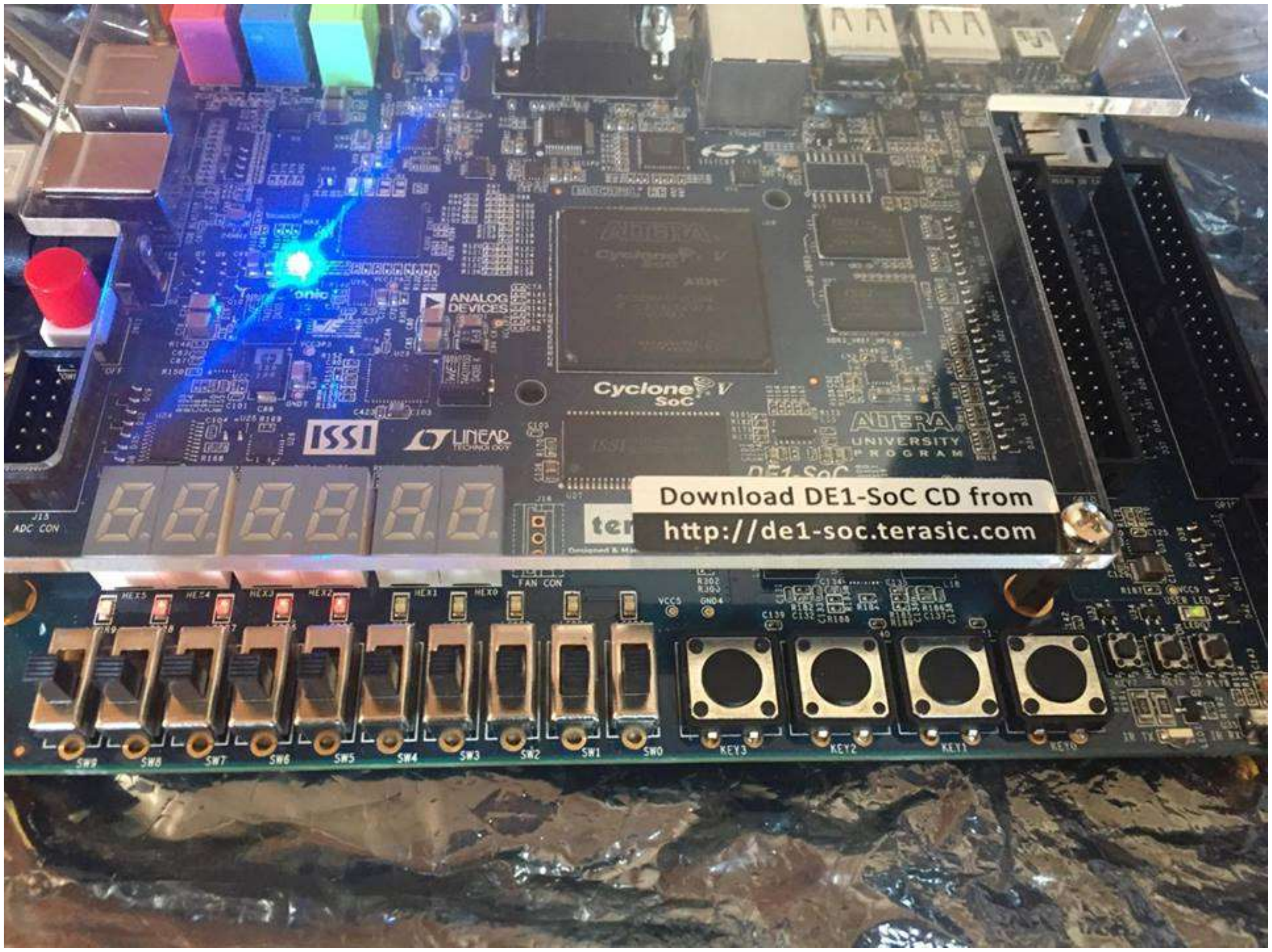


Figure 28: Input x 0000 Input y 0000 Output 0 0000

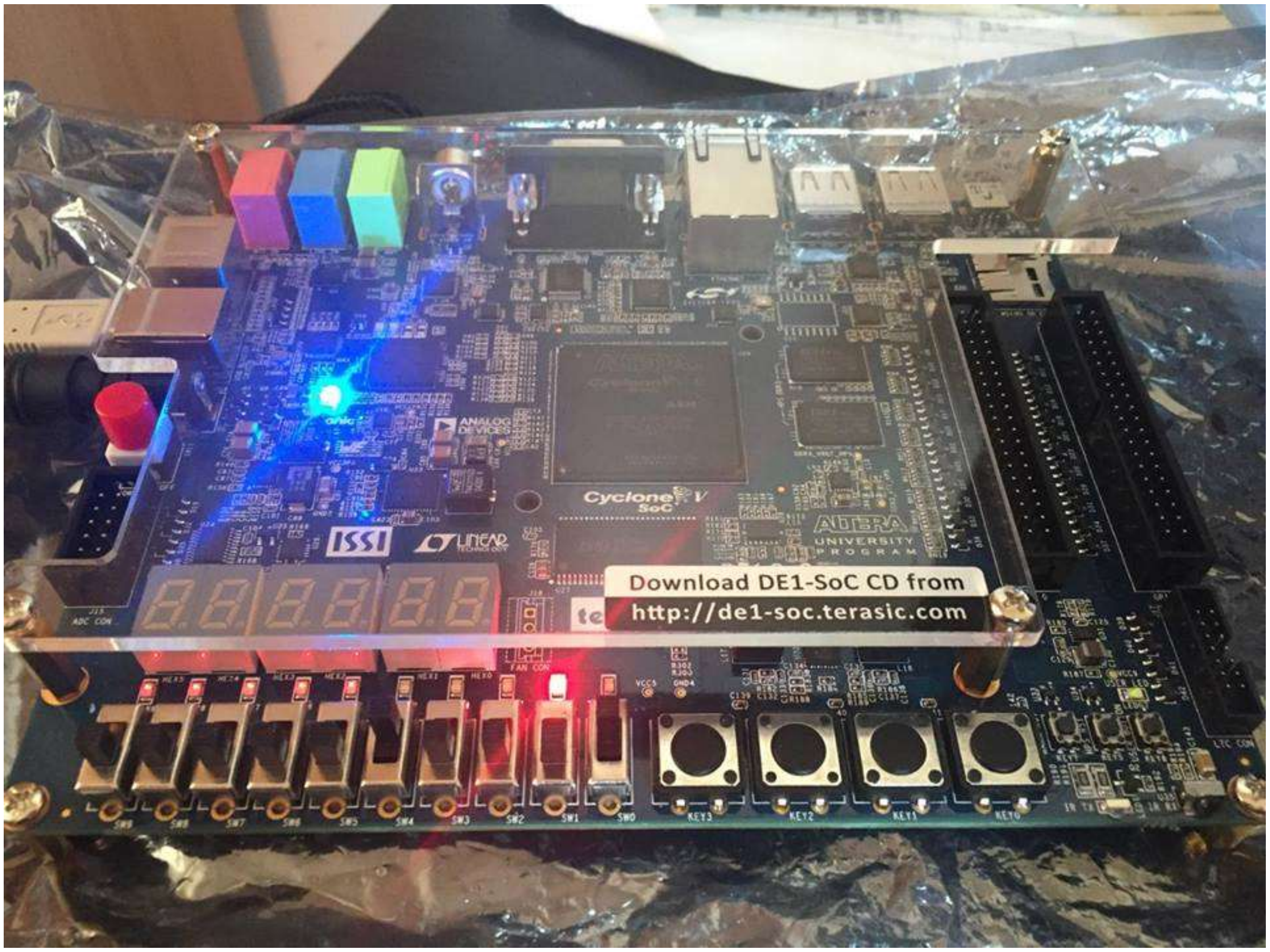
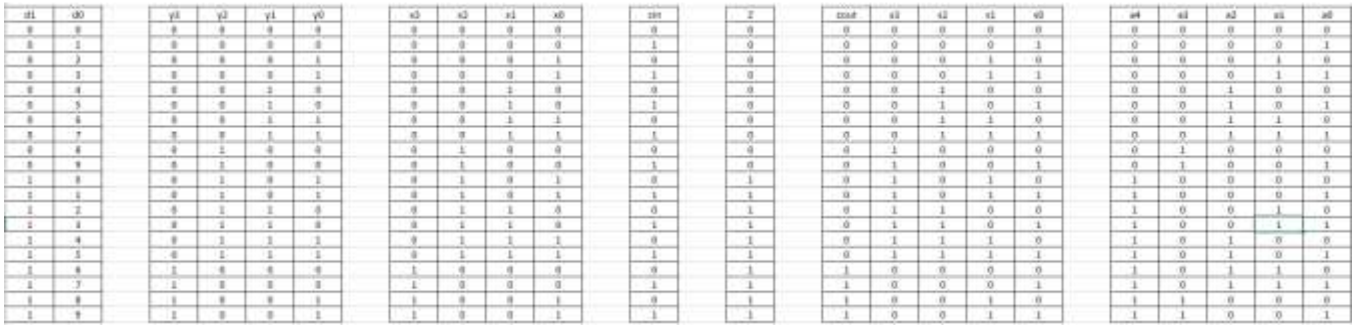


Figure 29: Input x 0001 Input y 0001 Output 0 0010

5. BCD Adder

5.1 Functionality and Specification

The binary coded decimal adder is the main point of this lab, by putting what we have done for seven-segment display, binary to decimal, and the 4-bit full adder using bus tool, we will create the BCD adder that adds two 4-bit binary number that will have a maximum value of 9 and a carry-in which results 9 + 9 + 1 = 19 as the maximum display on the seven-segment display.



d1	d0	y3	y2	y1	y0	x3	x2	x1	x0	cin	Z	cout	s3	s2	s1	s0	a4	a3	a2	a1	a0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1
0	2	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0
0	3	0	0	0	1	0	0	0	1	1	0	0	0	0	1	1	0	0	0	1	1
0	4	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0
0	5	0	0	1	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1	0	1
0	6	0	0	1	1	0	0	1	1	0	0	0	0	1	1	0	0	0	1	1	0
0	7	0	0	1	1	0	0	1	1	1	0	0	0	1	1	1	0	0	1	1	1
0	8	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0
0	9	0	1	0	0	0	1	0	0	1	0	0	1	0	0	1	0	1	0	0	1
1	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	0	0
1	1	0	1	0	1	0	1	0	1	1	1	0	1	0	1	1	1	0	0	0	1
1	2	0	1	1	0	0	1	1	0	0	1	0	1	1	0	0	1	0	0	1	0
1	3	0	1	1	0	0	1	1	0	1	1	0	1	1	0	1	1	0	0	1	1
1	4	0	1	1	1	0	1	1	1	0	1	0	1	1	1	0	1	0	1	0	0
1	5	0	1	1	1	0	1	1	1	1	1	0	1	1	1	1	1	0	1	0	1
1	6	1	0	0	0	1	0	0	0	0	1	1	0	0	0	0	1	0	1	1	0
1	7	1	0	0	0	1	0	0	0	1	1	1	0	0	0	1	1	0	1	1	1
1	8	1	0	0	1	1	0	0	1	0	1	1	0	0	1	0	1	1	0	0	0
1	9	1	0	0	1	1	0	0	1	1	1	1	0	0	1	1	1	1	0	0	1

Figure 28: Truth table for BCD Adder.

In order to create a BCD Adder, we would first need to add the two 4-bit binary inputs together using the 4-bit Full Adder using bus tool we had created earlier and produce a 5-bit output of s0, s1, s2, s3, and cout. Then the next step will be similar to what we have done in the Binary to Decimal section where it contains a comparator, a circuit one, a circuit two, and the seven-segment decoder. Since we have a 5-bit binary result, the BCD Adder will require a different comparator and a different circuit one since these two in the binary to decimal section only complies with a 4-bit binary input. Therefore, the comparator for the BCD adder will have the Boolean function

Z = cout OR (s3 AND (s2 OR s1))

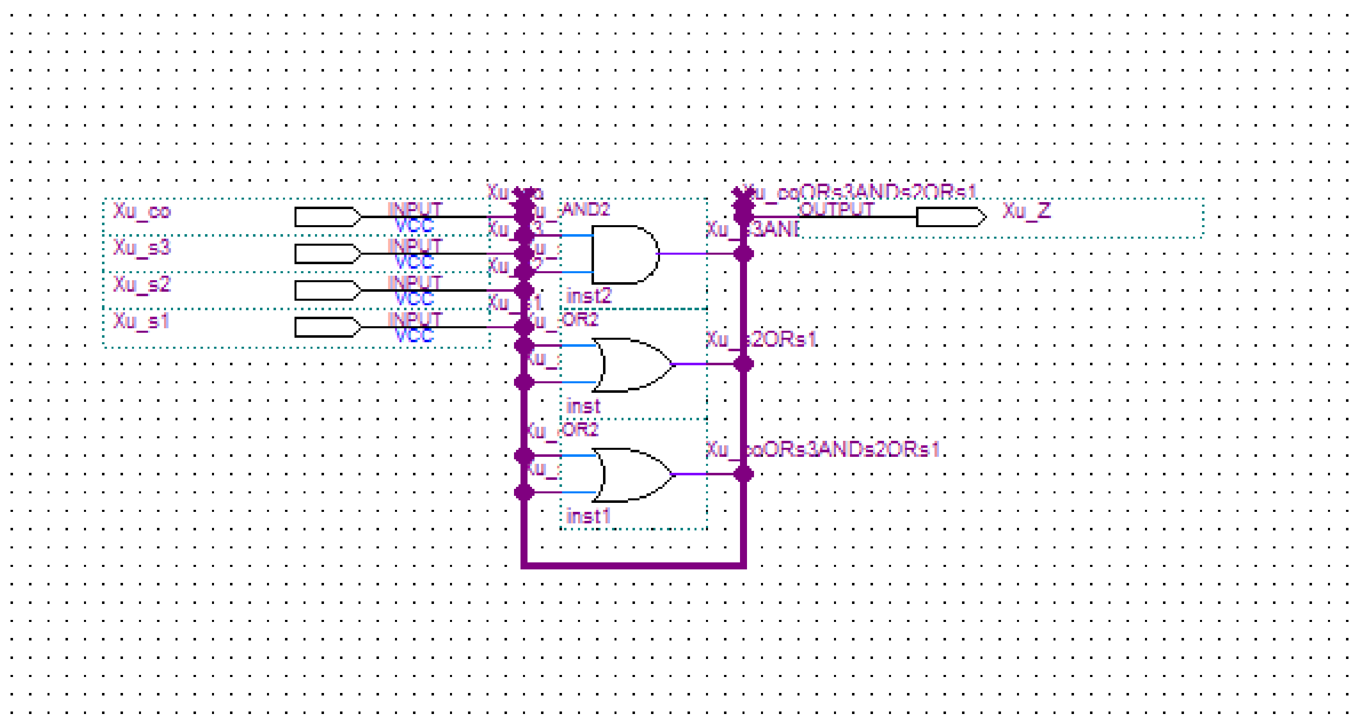


Figure 29: Block diagram for BCD Comparator.

Next we will need to create a Circuit One for the BCD Adder since the one being used in the binary to decimal section is for 4-bit binary input so it only goes up to 15 and we need it to be more than 15 since we have a 5-bit binary result from the 4-bit Full Adder. The Boolean functions for the BCD Circuit One will be

a3 = cout AND s1

a2 = (cout XOR s2) AND (cout XOR s1)

a1 = (s3 XNOR s2) AND NOT s1

a0 = a0

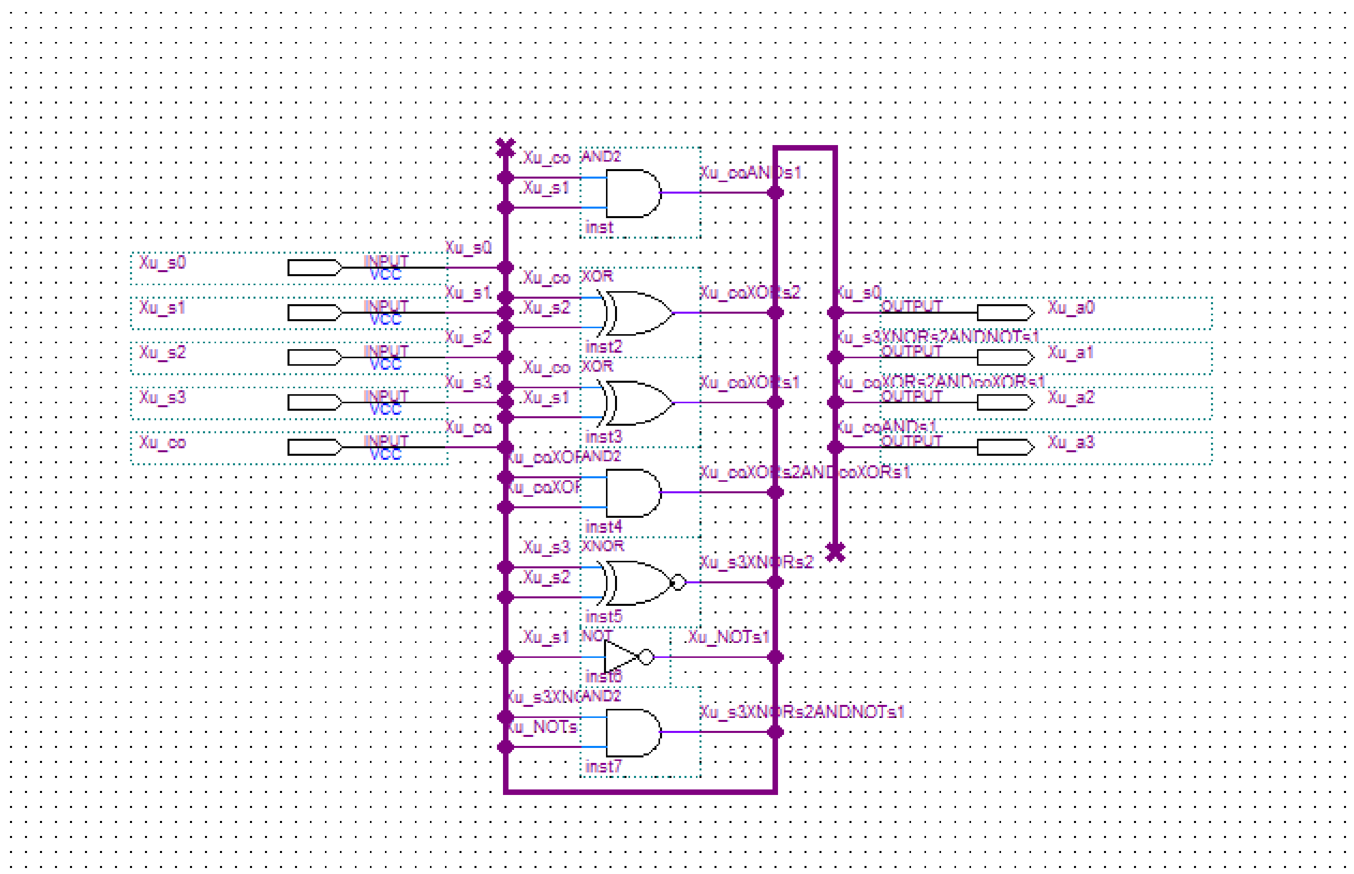


Figure 30: Block diagram for BCD Circuit One.

Then the rest is basically the same as the binary to decimal, we will have four 2:1 multiplexer that alternates the input into the seven-segment decoder which is determined by the selector Z and we can use the same Circuit Two we have already created to display 1 if the result is greater than 9 or 0 if result is less than 9. There are also addition elements to the BCD Adder such as displaying the input value of x and y onto one of the seven-segment display, determining if either input x or y is greater than 9 then LEDR[7] will light up indicating that it's an error input, and LEDR[0] light up to show that there is a cout. Displaying x and y on the seven-segment display is simply just having two more seven-segment decoder that takes x[3..0] and y[3..0] as inputs instead of using the results from the 4-bit full adder. To determine if either x or y input is greater than 9, we can use the comparator we had created for binary to decimal and have two of its which each one takes x and the other one takes y as input and if x OR y is greater than 9 then LEDR[7] will light up.

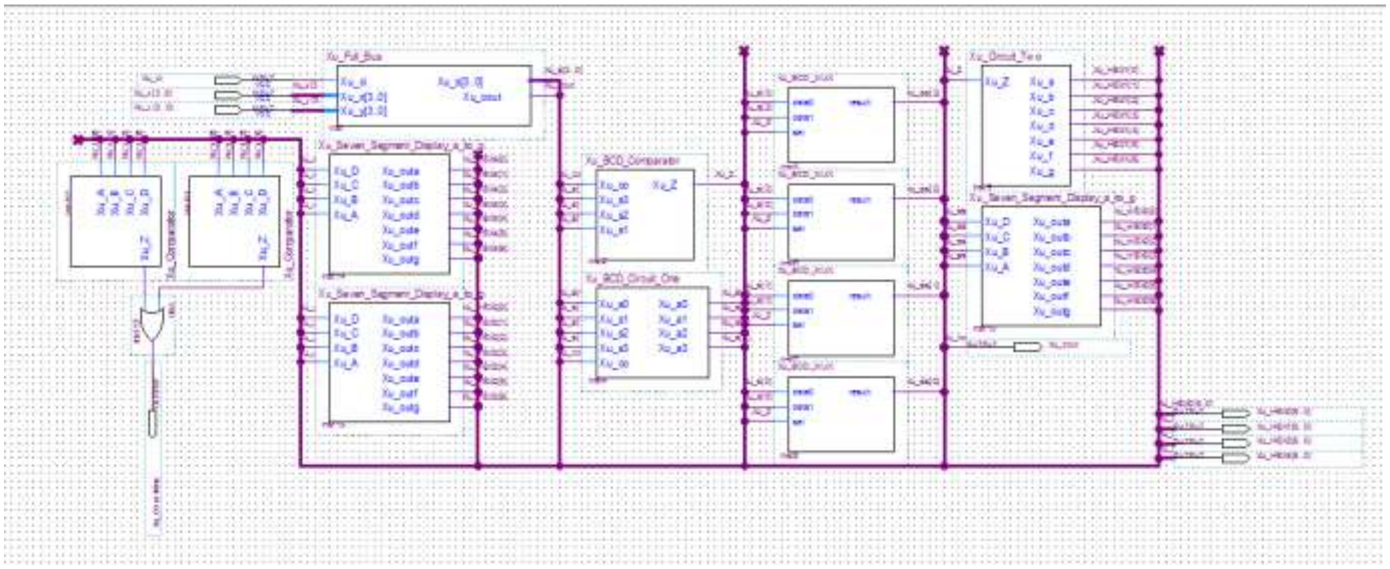


Figure 31: Block diagram of BCD Adder.

5.2 Simulation

In the simulation, we will give values of 0 and 1 to the inputs at varying intervals of all possible inputs. Input x3 and y3 will have values of 0 and 1 at each 960-ns interval. Input x2 and y2 will have values of 0 and 1 at each 480-ns interval. Input x1 and y1 will have values of 0 and 1 at each 240-ns interval. Input x0 and y0 will have values of 0 and 1 at each 120-ns interval. Input ci will have values of 0 and 1 at each 60-ns interval.

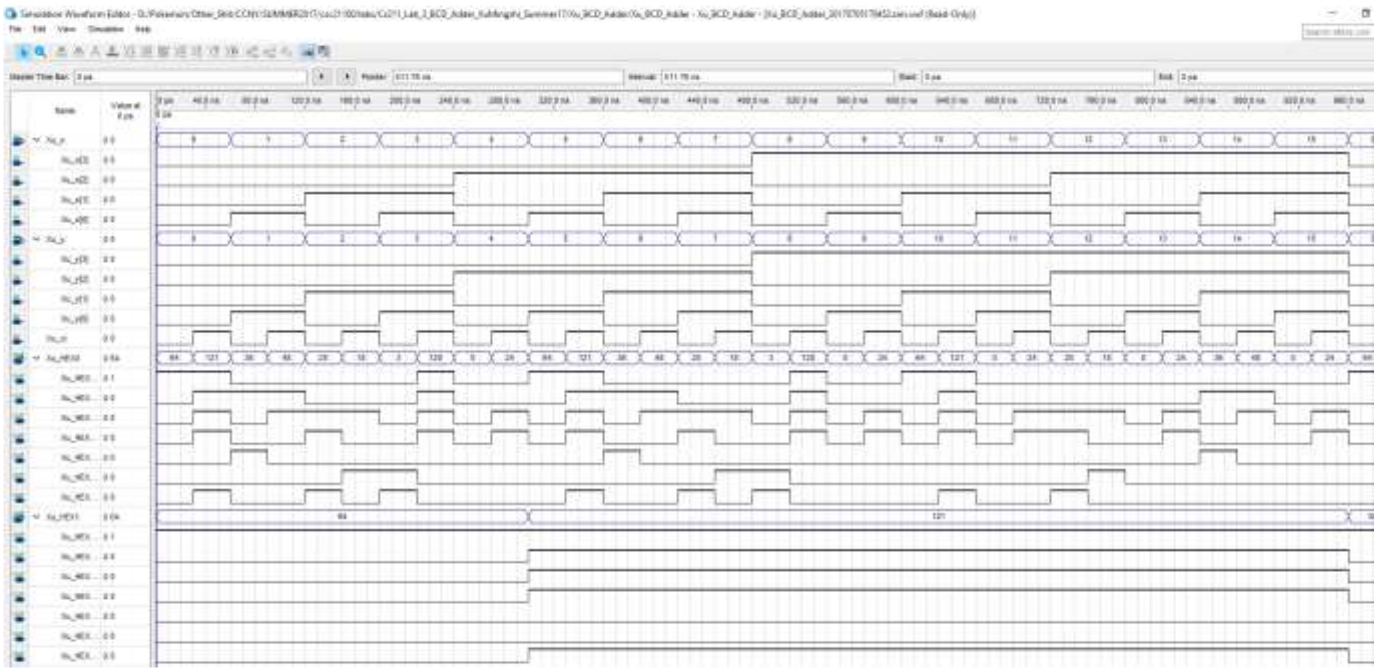


Figure 32: Vector waveform simulation for BCD Adder.

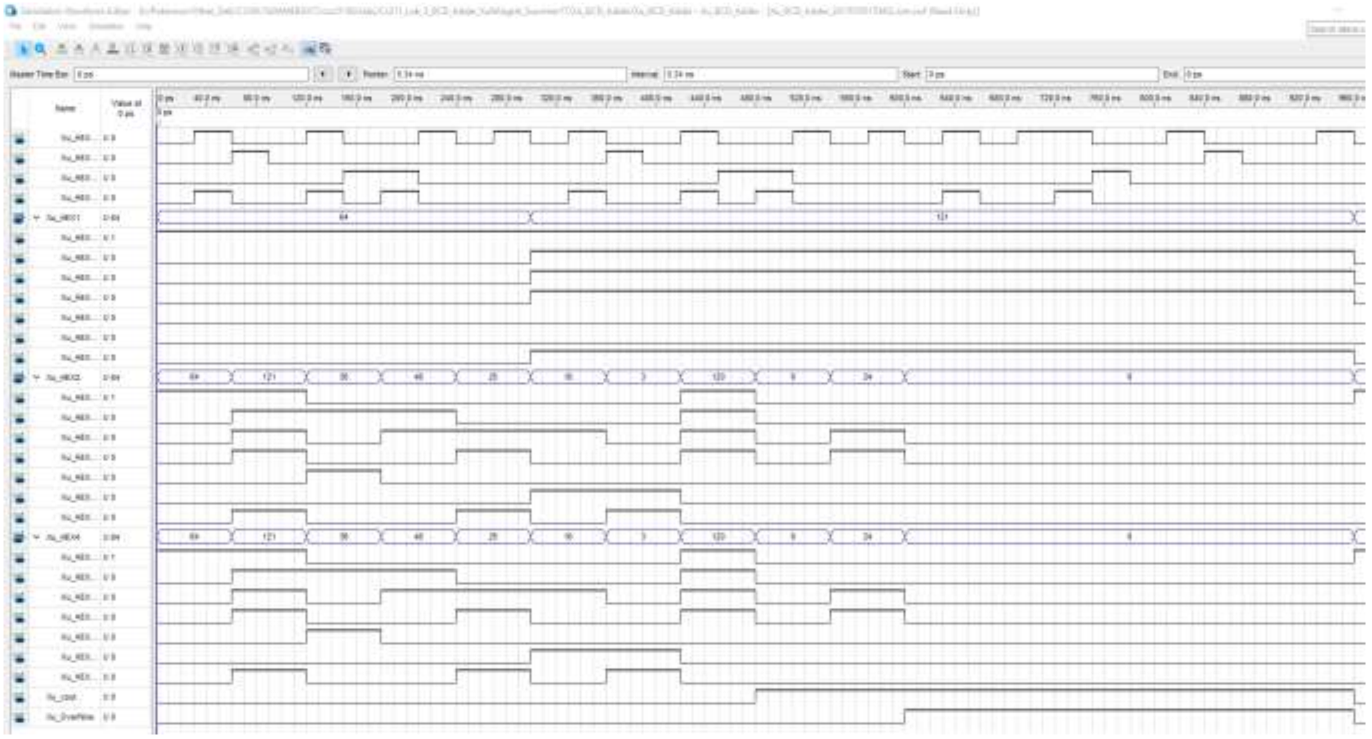


Figure 33: Vector waveform simulation for BCD Adder.

We can observe from the simulation that it corresponds to the truth table.

5.3 Demonstration

The PIN assignment of the inputs and outputs on the DE1-SoC Board.

Xu_x[0] is assigned to SW[0] which is PIN_AB12

Xu_x[1] is assigned to SW[1] which is PIN_AC12

Xu_x[2] is assigned to SW[2] which is PIN_AF9

Xu_x[3] is assigned to SW[3] which is PIN_AF10

Xu_y[0] is assigned to SW[4] which is PIN_AD11

Xu_y[1] is assigned to SW[5] which is PIN_AD12

Xu_y[2] is assigned to SW[6] which is PIN_AE11

Xu_y[3] is assigned to SW[7] which is PIN_AC9

Xu_ci is assigned to SW[8] which is PIN_AD10

Xu_HEX0[0] is assigned to HEX0[0] which is PIN_AE26

Xu_HEX0[1] is assigned to HEX0[1] which is PIN_AE27

Xu_HEX0[2] is assigned to HEX0[2] which is PIN_AE28

Xu_HEX0[3] is assigned to HEX0[3] which is PIN_AG27

Xu_HEX0[4] is assigned to HEX0[4] which is PIN_AF28

Xu_HEX0[5] is assigned to HEX0[5] which is PIN_AG28

Xu_HEX0[6] is assigned to HEX0[6] which is PIN_AH28

Xu_HEX1[0] is assigned to HEX1[0] which is PIN_AJ29

Xu_HEX1[1] is assigned to HEX1[1] which is PIN_AH29

Xu_HEX1[2] is assigned to HEX1[2] which is PIN_AH30

Xu_HEX1[3] is assigned to HEX1[3] which is PIN_AG30

Xu_HEX1[4] is assigned to HEX1[4] which is PIN_AF29

Xu_HEX1[5] is assigned to HEX1[5] which is PIN_AF30

Xu_HEX1[6] is assigned to HEX1[6] which is PIN_AD27

Xu_HEX2[0] is assigned to HEX2[0] which is PIN_AB23

Xu_HEX2[1] is assigned to HEX2[1] which is PIN_AE29

Xu_HEX2[2] is assigned to HEX2[2] which is PIN_AD29

Xu_HEX2[3] is assigned to HEX2[3] which is PIN_AC28

Xu_HEX2[4] is assigned to HEX2[4] which is PIN_AD30

Xu_HEX2[5] is assigned to HEX2[5] which is PIN_AC29

Xu_HEX2[6] is assigned to HEX2[6] which is PIN_AC30

Xu_HEX4[0] is assigned to HEX4[0] which is PIN_AA24

Xu_HEX4[1] is assigned to HEX4[1] which is PIN_Y23

Xu_HEX4[2] is assigned to HEX4[2] which is PIN_Y24

Xu_HEX4[3] is assigned to HEX4[3] which is PIN_W22

Xu_HEX4[4] is assigned to HEX4[4] which is PIN_W24

Xu_HEX4[5] is assigned to HEX4[5] which is PIN_V23

Xu_HEX4[6] is assigned to HEX4[6] which is PIN_W25

Xu_cout is assigned to LEDR[0] which is PIN_V16

Xu_OverNine is assigned to LEDR[7] which is PIN_W20

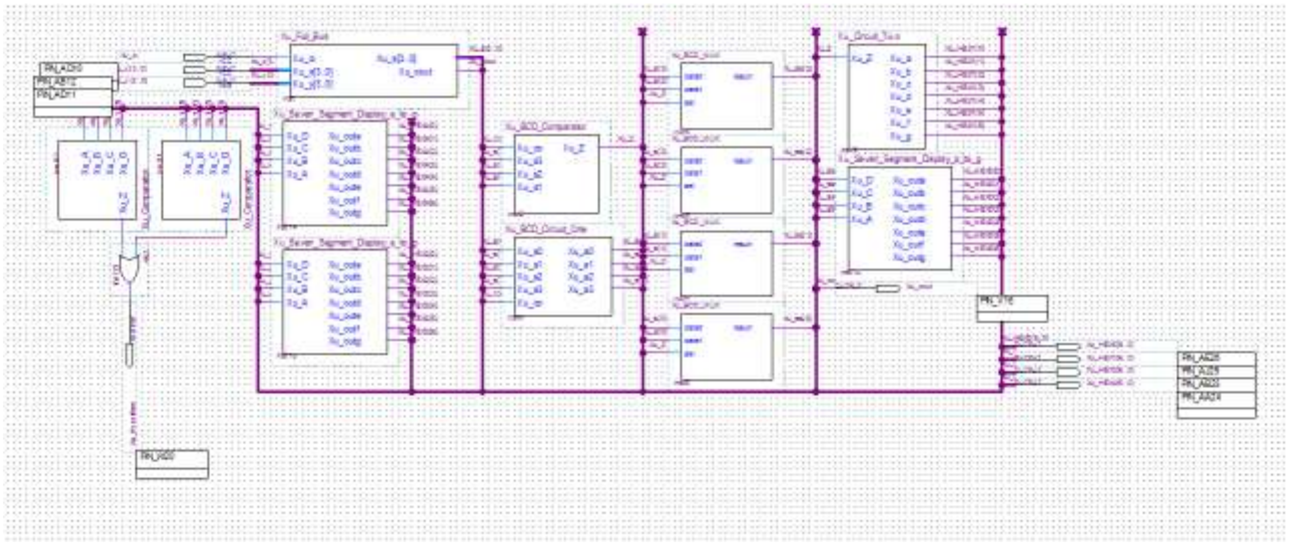


Figure 34: PIN assignment of circuit to DE1-SoC Board.

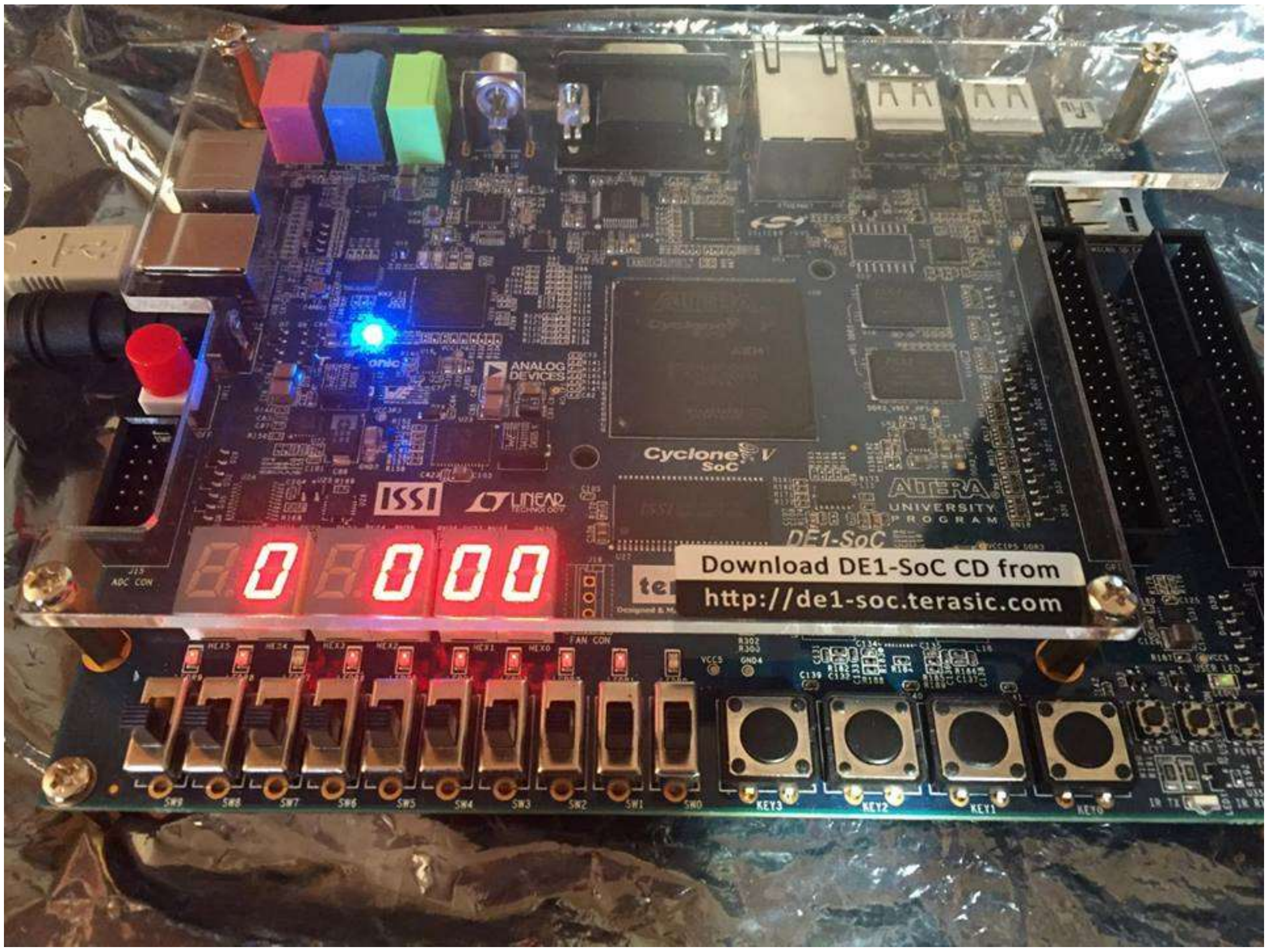


Figure 35: Input x 0000 HEX4 display 0 Input y 0000 HEX2 display 0 Input carry-in 0 Outputs 00

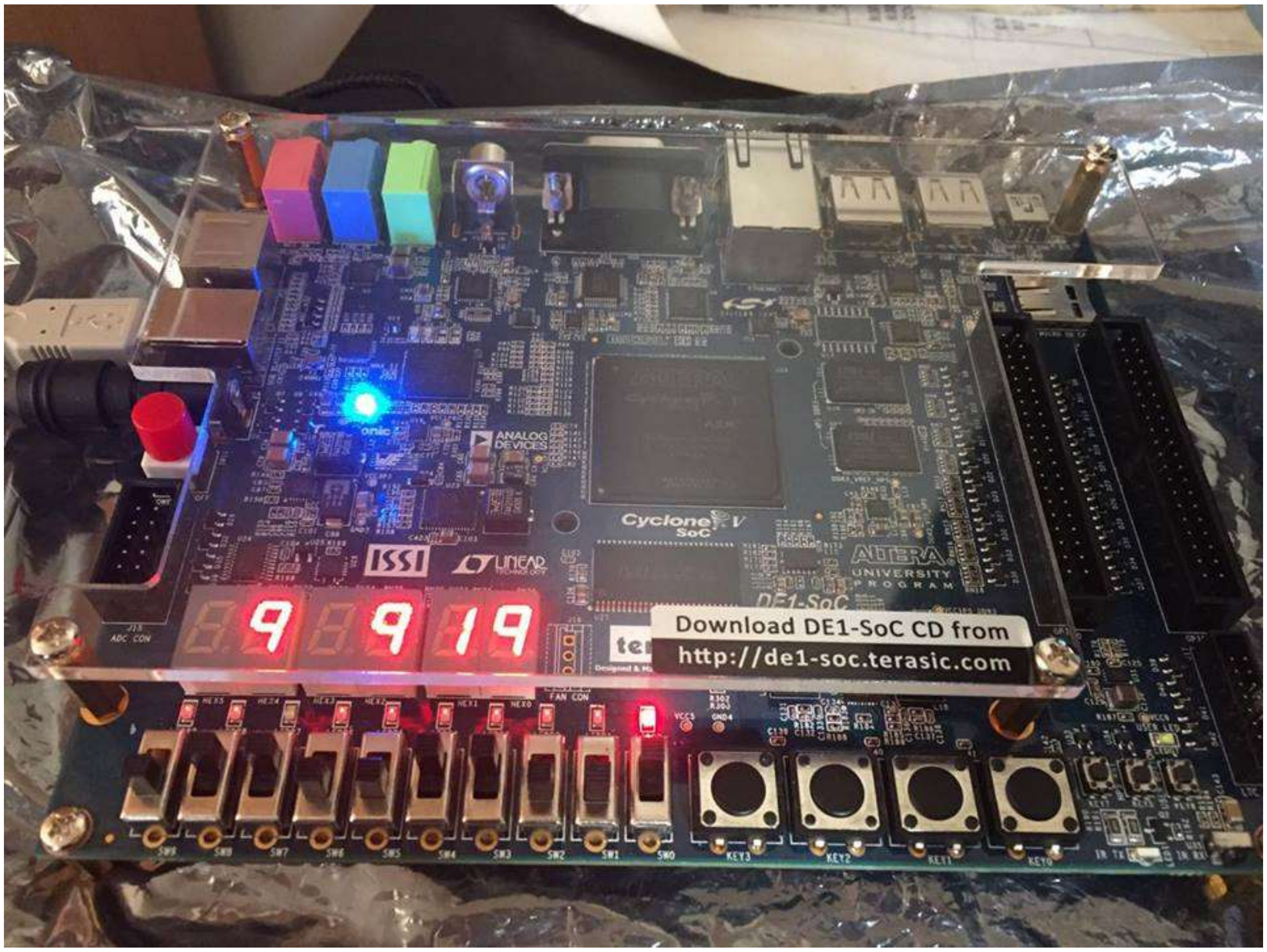
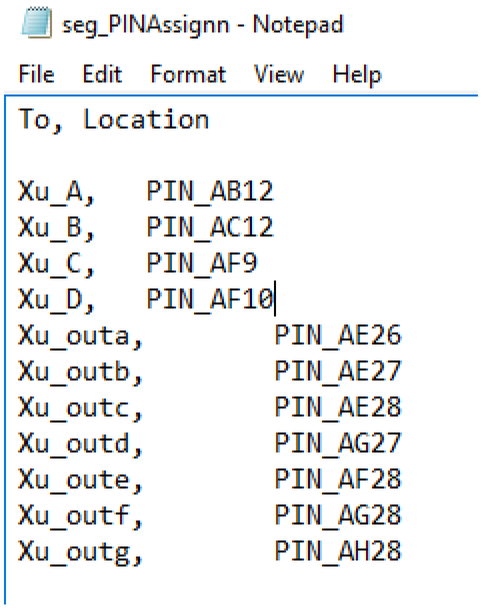


Figure 36: Input x 1001 HEX4 display 9 Input y 1001 HEX2 display 9 Input carry-in 1 Outputs 19 and also Carry-out light on LEDR[0]

6. Conclusion

Overall, in this lab we have learned how the seven-segment display on the board works and the use of orthogonal bus tool. The orthogonal bus tool is very useful, it helps reduce the amount of inputs and outputs by grouping them and the amount of the circuit lines going around which becomes more organized by naming the circuit lines so it'll know where and which inputs or outputs it's going to and from. Since this is the first time that I've use of the orthogonal bus tool, I might be still be inexperienced as you can see from the block diagram of the BCD Comparator and the BCD Circuit One. I believe that I'll be able to improve my ways of using the bus tool in future labs.

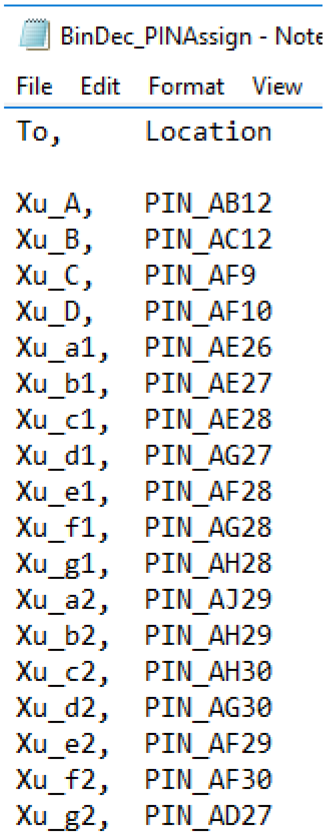
7. Appendix



```
To, Location

Xu_A,    PIN_AB12
Xu_B,    PIN_AC12
Xu_C,    PIN_AF9
Xu_D,    PIN_AF10
Xu_outa,        PIN_AE26
Xu_outb,        PIN_AE27
Xu_outc,        PIN_AE28
Xu_outd,        PIN_AG27
Xu_oute,        PIN_AF28
Xu_outf,        PIN_AG28
Xu_outg,        PIN_AH28
```

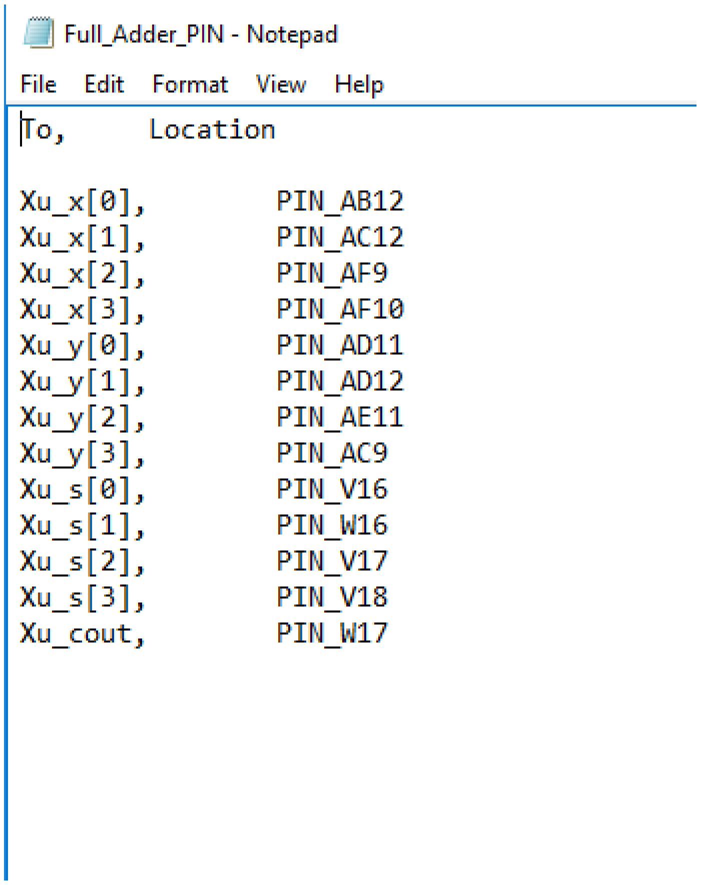
Figure 37: PIN assignment for seven-segment decoder.



```
To,     Location

Xu_A,   PIN_AB12
Xu_B,   PIN_AC12
Xu_C,   PIN_AF9
Xu_D,   PIN_AF10
Xu_a1,  PIN_AE26
Xu_b1,  PIN_AE27
Xu_c1,  PIN_AE28
Xu_d1,  PIN_AG27
Xu_e1,  PIN_AF28
Xu_f1,  PIN_AG28
Xu_g1,  PIN_AH28
Xu_a2,  PIN_AJ29
Xu_b2,  PIN_AH29
Xu_c2,  PIN_AH30
Xu_d2,  PIN_AG30
Xu_e2,  PIN_AF29
Xu_f2,  PIN_AF30
Xu_g2,  PIN_AD27
```

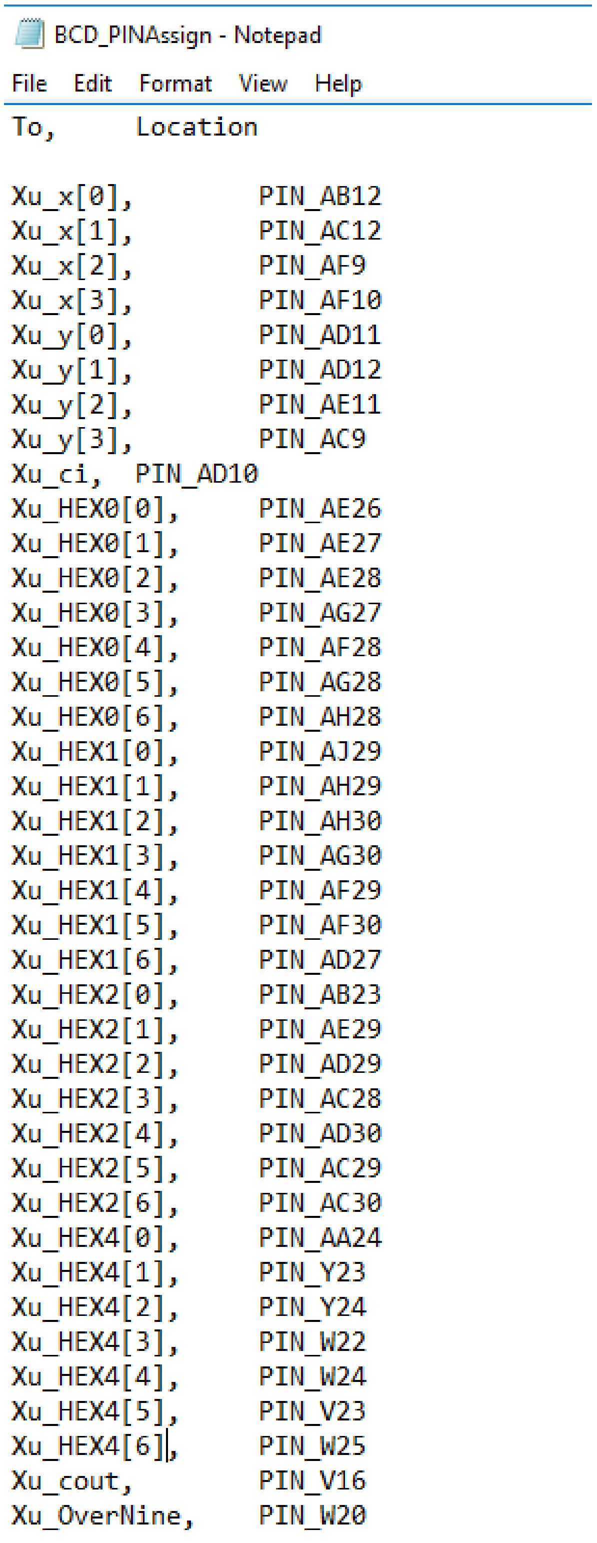
Figure 38: PIN assignment for Binary to Decimal.



```
To,      Location

Xu_x[0],        PIN_AB12
Xu_x[1],        PIN_AC12
Xu_x[2],        PIN_AF9
Xu_x[3],        PIN_AF10
Xu_y[0],        PIN_AD11
Xu_y[1],        PIN_AD12
Xu_y[2],        PIN_AE11
Xu_y[3],        PIN_AC9
Xu_s[0],        PIN_V16
Xu_s[1],        PIN_W16
Xu_s[2],        PIN_V17
Xu_s[3],        PIN_V18
Xu_cout,        PIN_W17
```

Figure 39: PIN assignment for 4-bit full adder using bus tool.



```
To,     Location

Xu_x[0],        PIN_AB12
Xu_x[1],        PIN_AC12
Xu_x[2],        PIN_AF9
Xu_x[3],        PIN_AF10
Xu_y[0],        PIN_AD11
Xu_y[1],        PIN_AD12
Xu_y[2],        PIN_AE11
Xu_y[3],        PIN_AC9
Xu_ci,  PIN_AD10
Xu_HEX0[0],     PIN_AE26
Xu_HEX0[1],     PIN_AE27
Xu_HEX0[2],     PIN_AE28
Xu_HEX0[3],     PIN_AG27
Xu_HEX0[4],     PIN_AF28
Xu_HEX0[5],     PIN_AG28
Xu_HEX0[6],     PIN_AH28
Xu_HEX1[0],     PIN_AJ29
Xu_HEX1[1],     PIN_AH29
Xu_HEX1[2],     PIN_AH30
Xu_HEX1[3],     PIN_AG30
Xu_HEX1[4],     PIN_AF29
Xu_HEX1[5],     PIN_AF30
Xu_HEX1[6],     PIN_AD27
Xu_HEX2[0],     PIN_AB23
Xu_HEX2[1],     PIN_AE29
Xu_HEX2[2],     PIN_AD29
Xu_HEX2[3],     PIN_AC28
Xu_HEX2[4],     PIN_AD30
Xu_HEX2[5],     PIN_AC29
Xu_HEX2[6],     PIN_AC30
Xu_HEX4[0],     PIN_AA24
Xu_HEX4[1],     PIN_Y23
Xu_HEX4[2],     PIN_Y24
Xu_HEX4[3],     PIN_W22
Xu_HEX4[4],     PIN_W24
Xu_HEX4[5],     PIN_V23
Xu_HEX4[6],     PIN_W25
Xu_cout,        PIN_V16
Xu_OverNine,    PIN_W20
```

Figure 40: PIN assignment for BCD Adder