INTERNATIONAL UNIVERSITY OF SARAJEVO
INTERNACIONALNI UNIVERZITET U SARAJEVU

# PROJECT REPORT

COURSE: CS303 DIGITAL DESIGN

STUDENT: Amina Kadić

Sarajevo, 2021

## Introduction

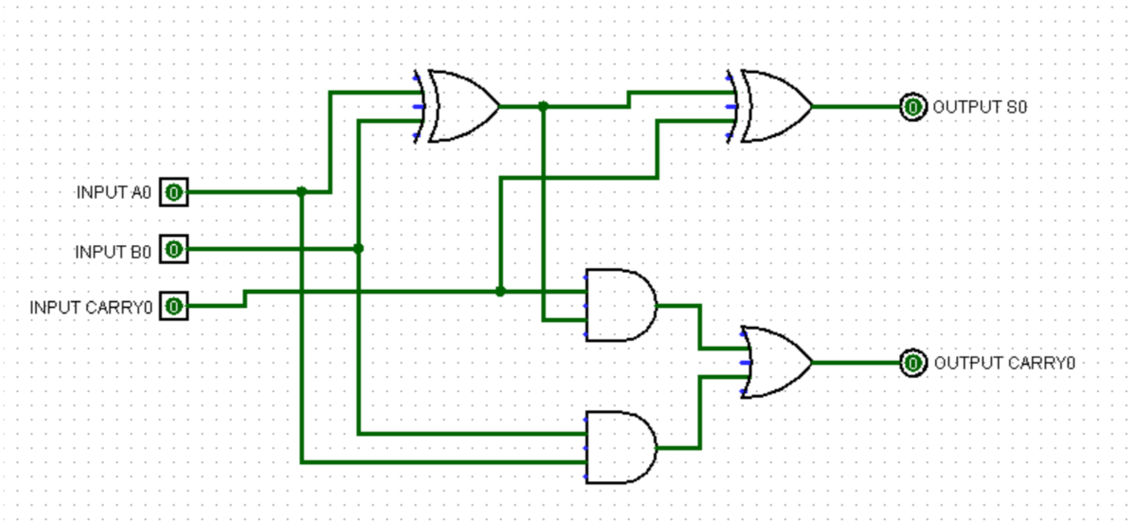
During one of our FPGA-focused laboratory practices we got a task to make several adders and connect them to displays. after making a 2-bit adder and a 4-bit adder we had no time to connect them to a display, therefore we decided to finish it outside the lab.

We started off with an adder which would display a binary result and a decimal result. Later on, we also made a subtractor which had the addition and subtraction displayed.

Both full-adder and full-adder with subtractor were made in Quartus Software for Altera FPGA using Block Design File.

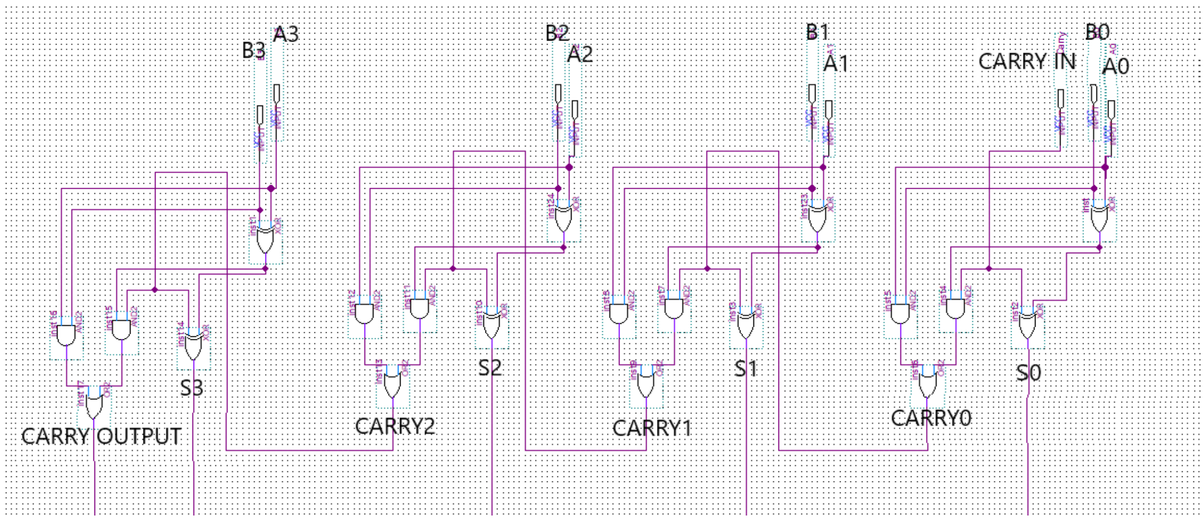
## Implementation of a 4-bit full adder

To implement 4 bit full-adder, firstly we needed to implement a 1 bit-full adder- which is accomplished by using two XOR gates, two ANDs and one OR-connected as shown in scheme. This adder has 3 inputs and 2 outputs. The three inputs represent the 2 bits we are adding up (A0, B0) and a carry bit leftover from previous addition (CARRY0). The outputs are the sum of the bits (S0) and a carry (CARRY0).



*I: 1-bit full-adder in Logisim*

Furthermore, to create a 4-bit adder we needed to use 4 1-bit adders, as shown previously, which are connected by the carries of each adder. The carry output of the first adder will be connected in place of the carry input of the second adder. The same logic applies for the third and fourth 1-bit adder. After connecting the 4 adders we will have a 4-bit adder.

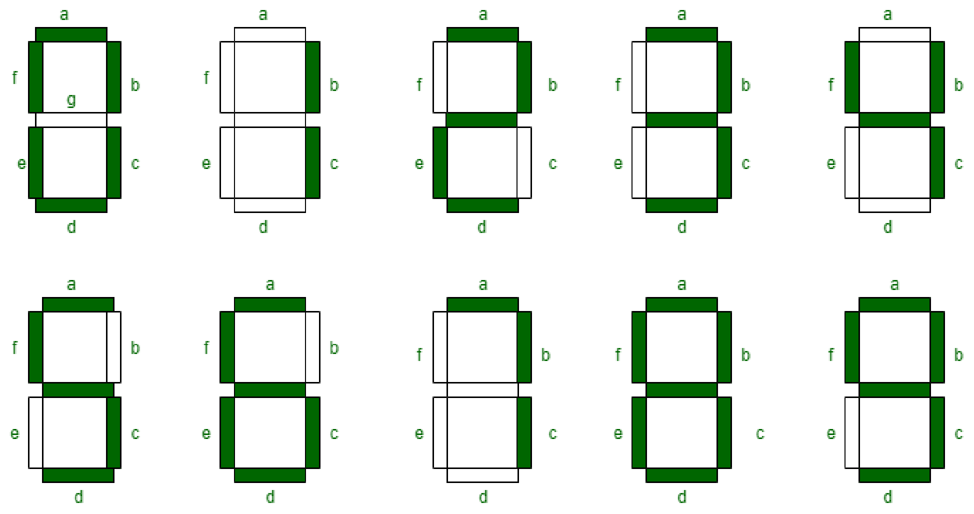


*II: 4-bit full-adder in Quartus using Block Diagram*

As shown in the scheme this adder has 9 inputs and 5 outputs. Eight inputs are 4-bit numbers A and B. Each of the bits has one input- A0, A1, A2 and A3. B also has 4 1-bit inputs B0, B1, B2, B3. The last input is a carry from a previous operation. The outputs are 4 sums and 1 final carry output. The 4 sums are represented as S1, S2, S3 and S4. Each sum is the sum of the corresponding bits, for example- the S1 (Sum1) corresponds to bits A1 and B2, S2 (Sum2) corresponds to A2 and B2, and so on. When we tested our 4-bit adder and confirmed that it works it was time to connect it to a display to show the output.

## Implementing seven-segment displays for a 4-bit full adder

After trying to connect the adder to a seven-segment display during laboratory hours, we figured out that we needed to use a different approach, which was creating our own decoders for the output results. Since the results were numbers up to 31 (given that had two 4-bit inputs and one 1-bit carry input) we needed two decoders for the two 7-segment displays. One of the displays would show 10's and the other one would display 1's. As I already mentioned our results go up to 31- so the 10's display would only need to show numbers 0, 1, 2 and 3.



*III: seven segment display representation of numbers 0 – 9*

To create the decoders we had to write down all possible combinations of inputs and outputs which looked like this:

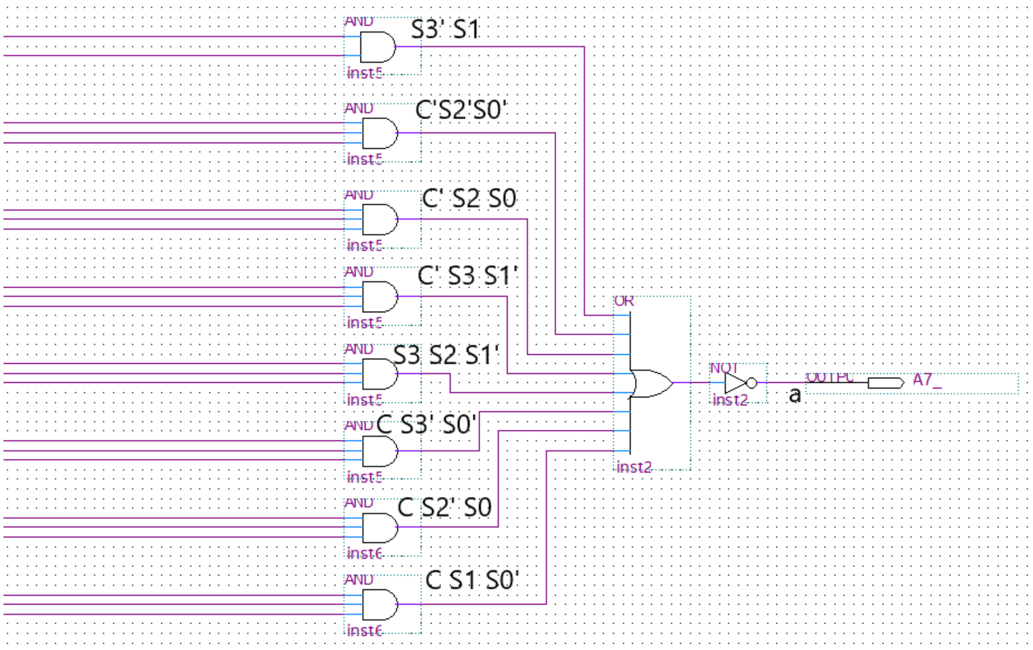
| C (carry) | S3 | S2 | S1 | S0 | a | b | c | d | e | f | g | output on 7SD |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 2 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 3 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 4 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 5 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 6 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 7 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 8 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 9 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 2 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 3 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 4 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 5 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 6 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 7 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 8 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 9 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 2 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 3 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 4 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 5 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 6 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 7 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 8 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 9 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |

*IV: truth table for seven segment display that shows ones*

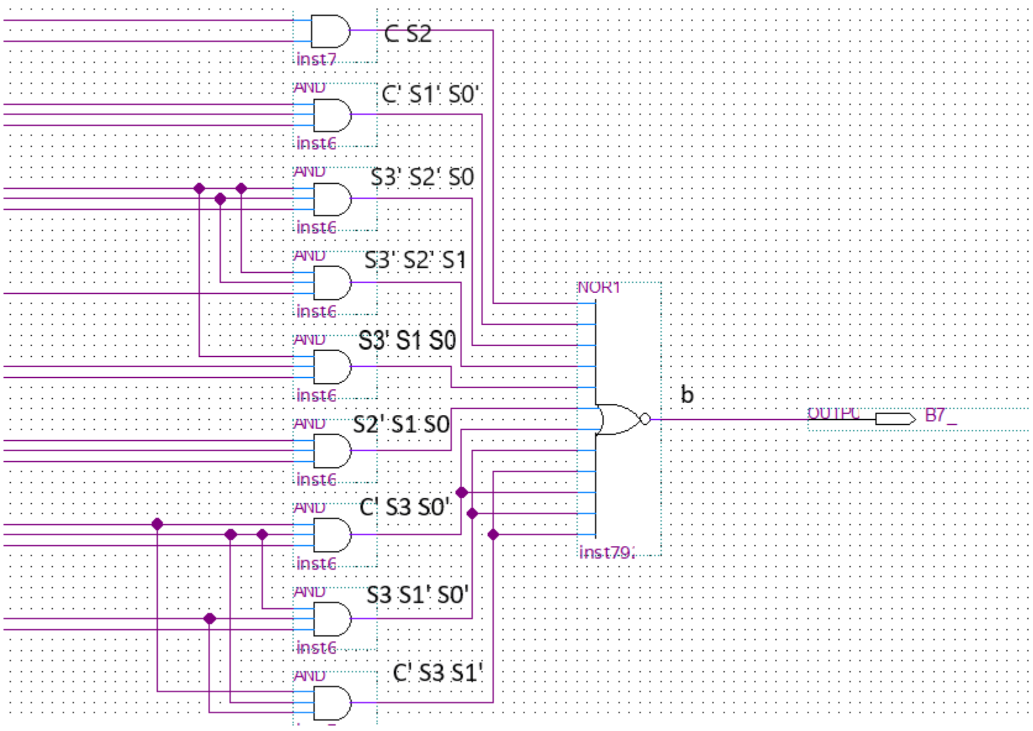
After deriving the equation for each segment of the 1's display using Karnaugh maps the logic was the following:

a = $S_3'S_1 + C'S_2'S_0' + C'S_2S_0 + C'S_3S_1' + S_3S_2S_1' + CS_3'S_0' + CS_2'S_0 + CS_1S_0'$



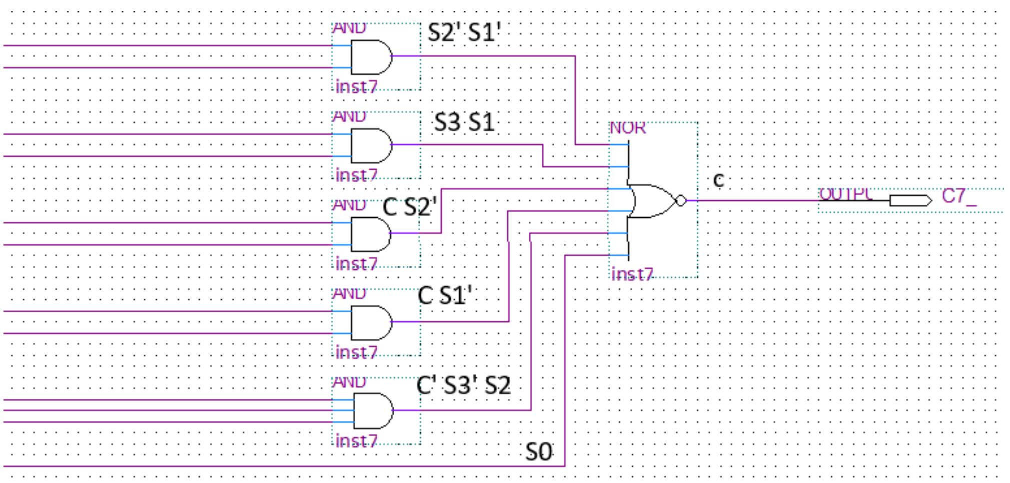
*V: logic for segment a*

b = $CS_2 + C'S_1'S_0' + S_3'S_2'S_0 + S_3'S_2'S_1 + S_3'S_1S_0 + S_2'S_1S_0 + C'S_3S_0' + S_3S_1'S_0' + C'S_3S_1'$



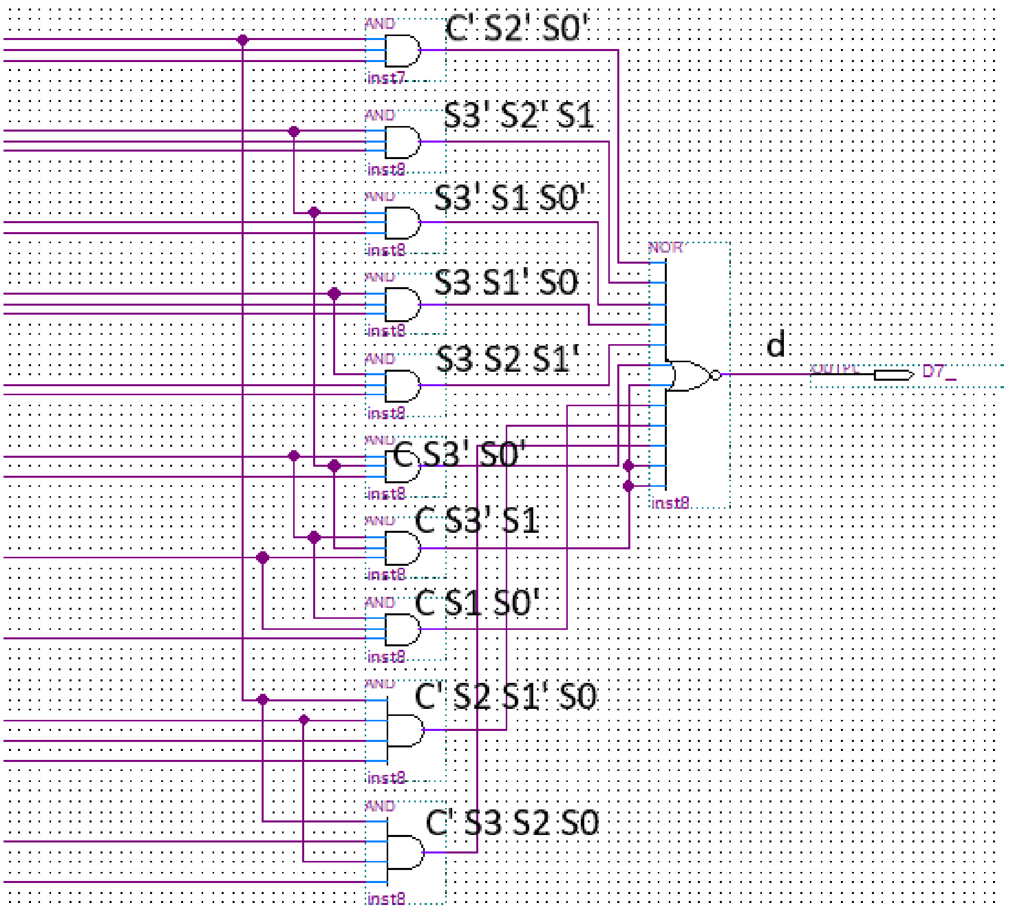
*VI: logic for segment b*

$c = S_0 + S_2'S_1' + S_3S_1 + CS_2' + CS_1' + C'S_3'S_2$



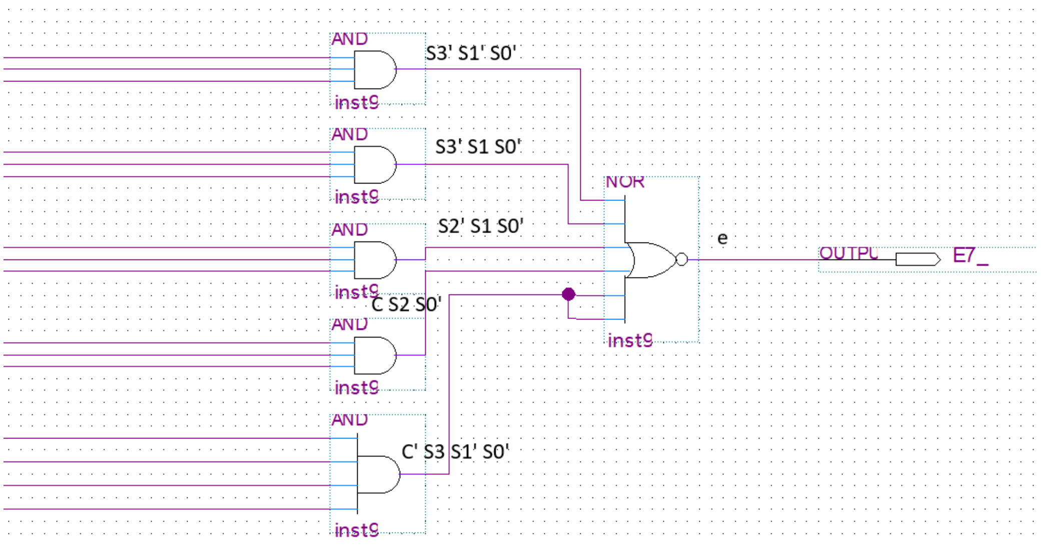
*VII: logic for segment c*

$d = C'S_2'S_0' + S_3'S_2'S_1 + S_3'S_1S_0' + S_3S_1'S_0 + S_3S_2S_1' + CS_3'S_0' + CS_3'S_1 + CS_1S_0' + C'S_2S_1'S_0 + C'S_3S_2S_0$



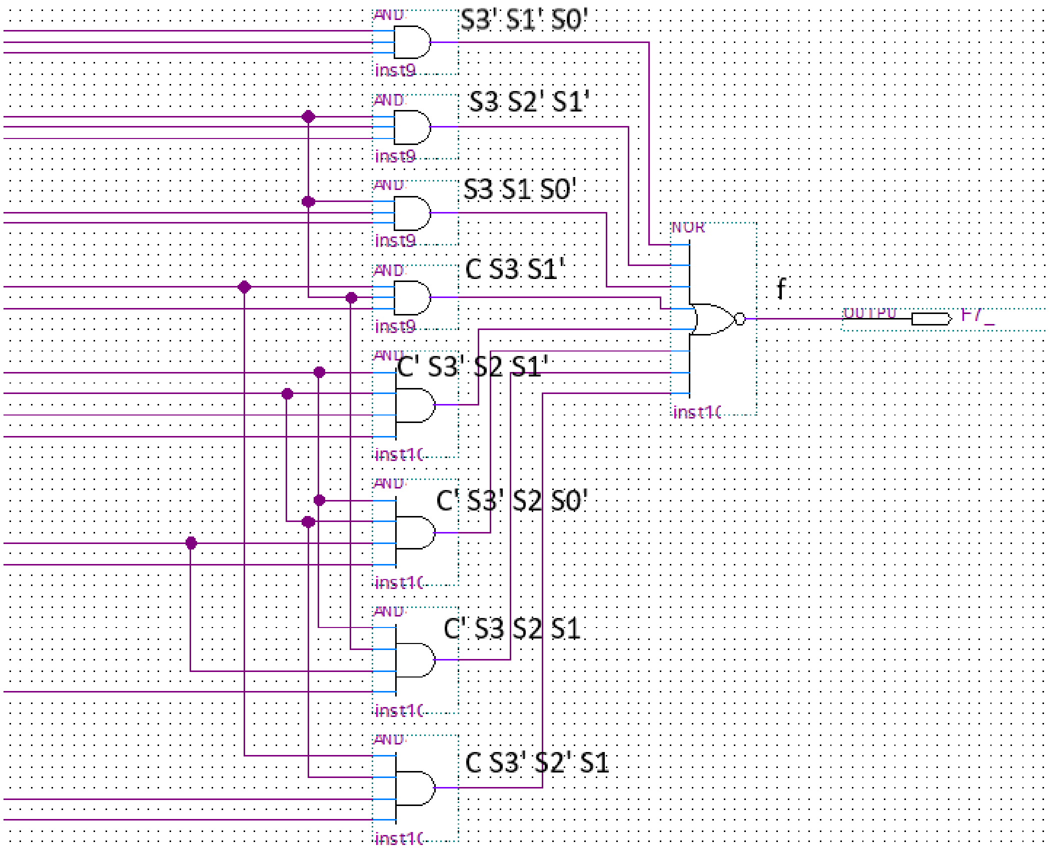
*VIII: logic for segment d*

$e = S_3'S_1'S_0' + S_3'S_1S_0' + S_2'S_1S_0' + CS_2S_0' + C'S_3S_1'S_0'$



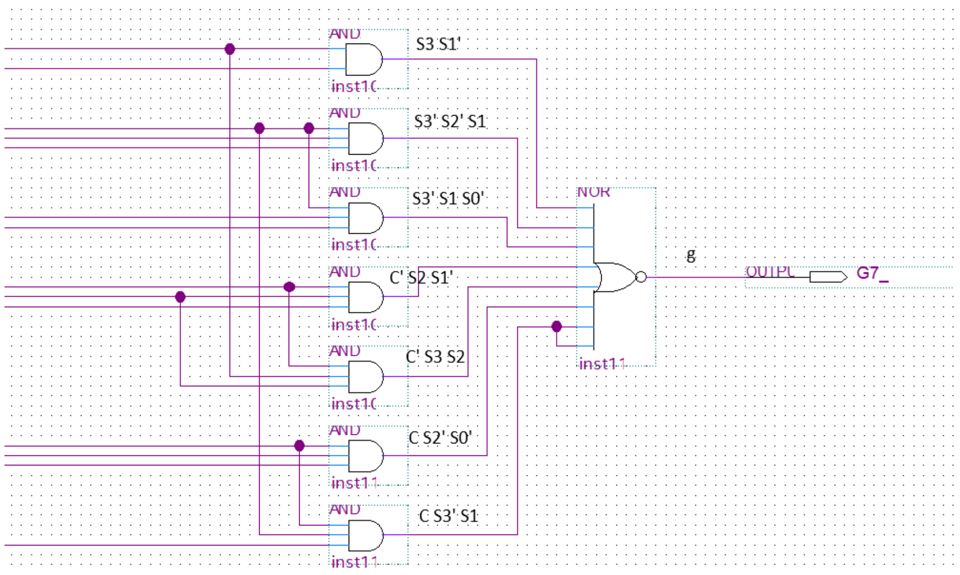
*IX: logic for segment e*

$f = S_3'S_1'S_0' + S_3S_2'S_1' + S_3S_1S_0' + CS_3S_1' + C'S_3'S_2S_1' + C'S_3'S_2S_0' + C'S_3S_2S_1 + CS_3'S_2'S_1$



*X: logic for segment f*

$g = S_3S_1' + S_3'S_2'S_1 + S_3'S_1S_0' + C'S_2S_1' + C'S_3S_2 + CS_2'S_0' + CS_3'S_1$



*XI: logic for segment g*

The next step was to create a truth table for the display that would represent the 10's:

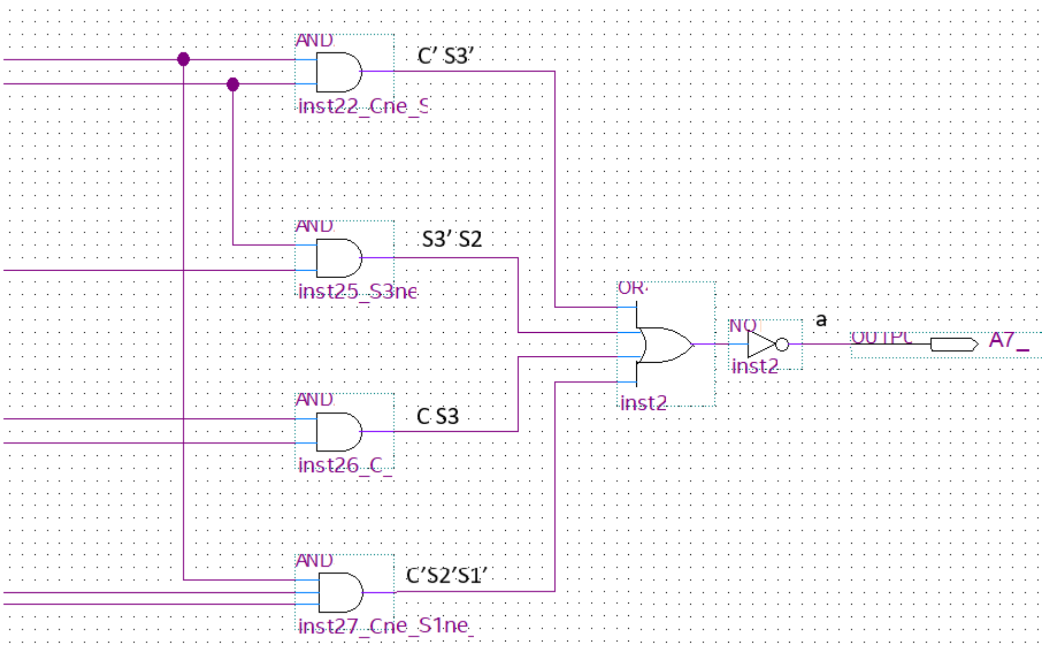
| CARRY | S0 | S1 | S2 | S3 | a | b | c | d | e | f | g | output on 7SD |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 2 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 2 |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 2 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 2 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 2 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 2 |
| 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 2 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 2 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 2 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 2 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 3 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 3 |

*XII: truth table for seven segment display that shows tens*

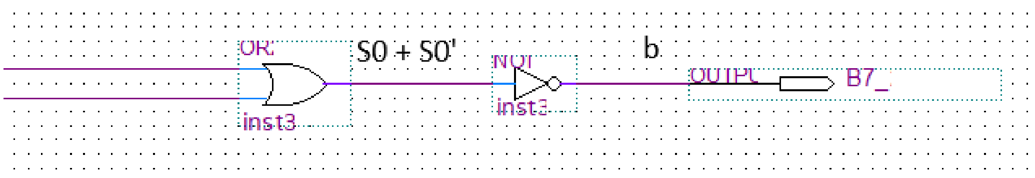
Again, using the Karnaugh maps we derived the equations for each segment of the 10's display:

$a = C'S_3' + S_3'S_2 + CS_3 + C'S_2'S_1'$



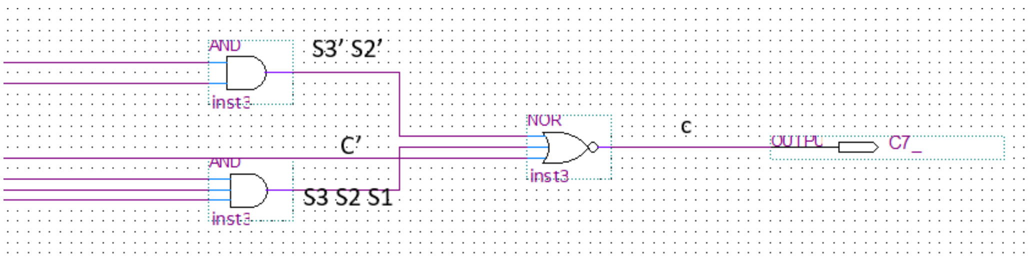
*XIII: logic for segment a*

b = 1



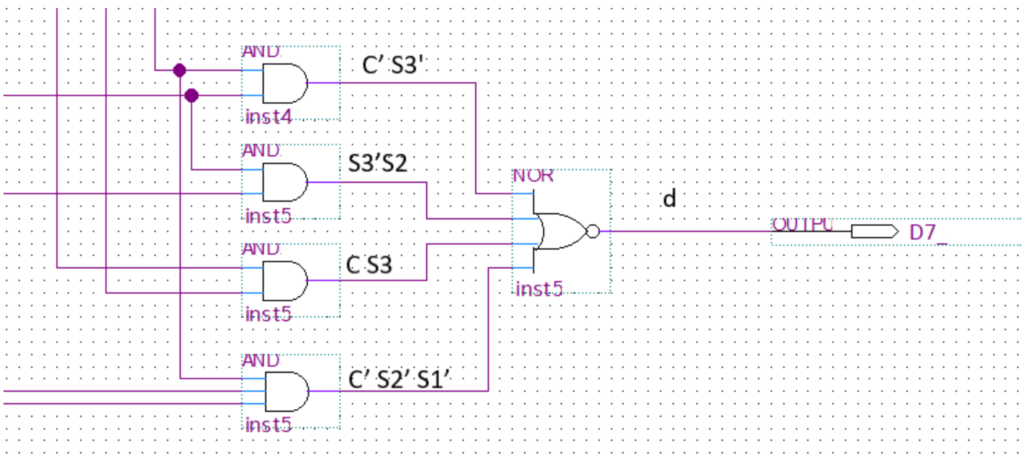
*XIV: logic for segment b*

c = C’ + $S_3$’$S_2$’ + $S_3S_2S_1$



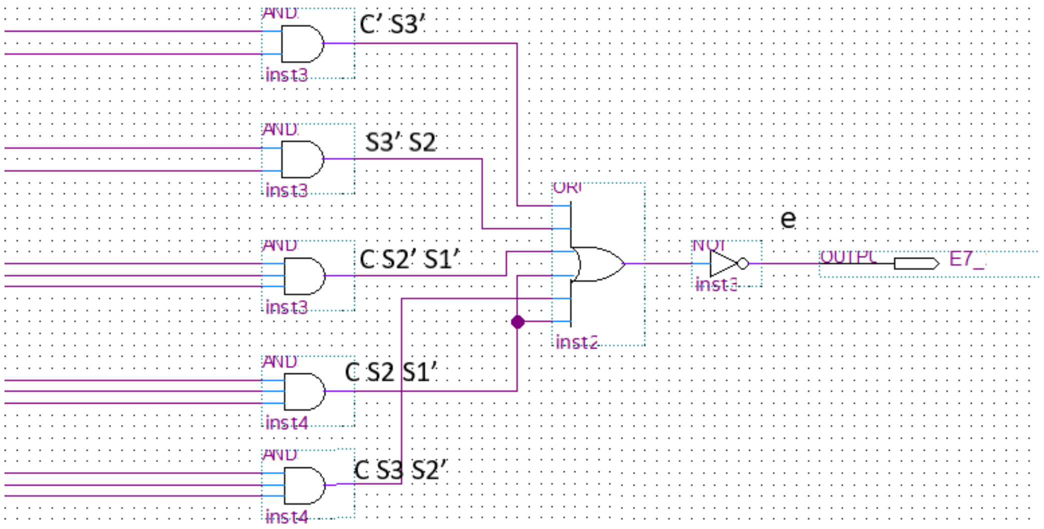
*XV: logic for segment c*

d = C’$S_3$’ + $S_3$’$S_2$ + C$S_3$ + C’$S_2$’$S_1$’



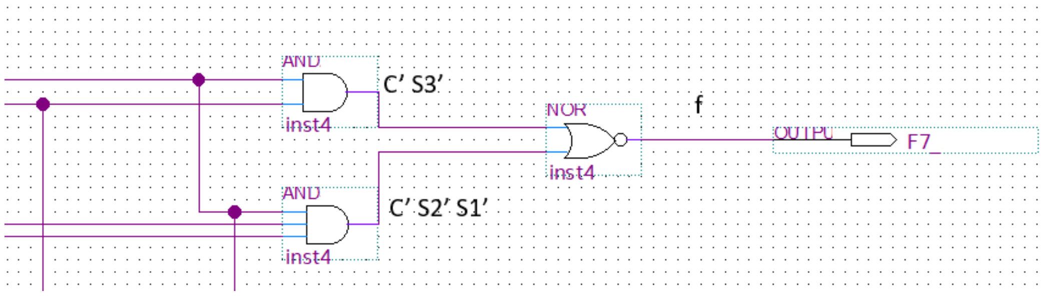
*XVI: logic for segment d*

$e = C'S_3' + S_3'S_2 + CS_2'S_1' + CS_2S_1' + CS_3S_2'$



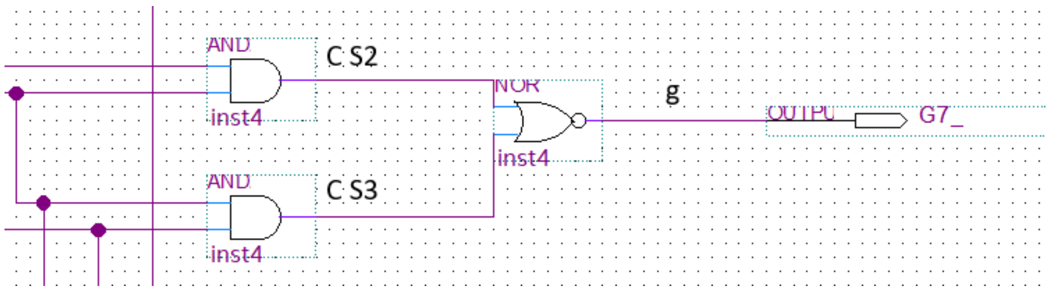
*XVII: logic for segment e*

$f = C'S_3' + C'S_2'S_1'$



*XVIII: logic for segment f*

$g = CS_2 + CS_3$



*XIX: logic for segment g*

We also wanted to represent the outputs with their original binary values, so we created five more decoders that would connect 5 more displays to out outputs. since the binary output needs only numbers 0 and 1, the truth table of all displays will have 2 possible values, and they will be the same.

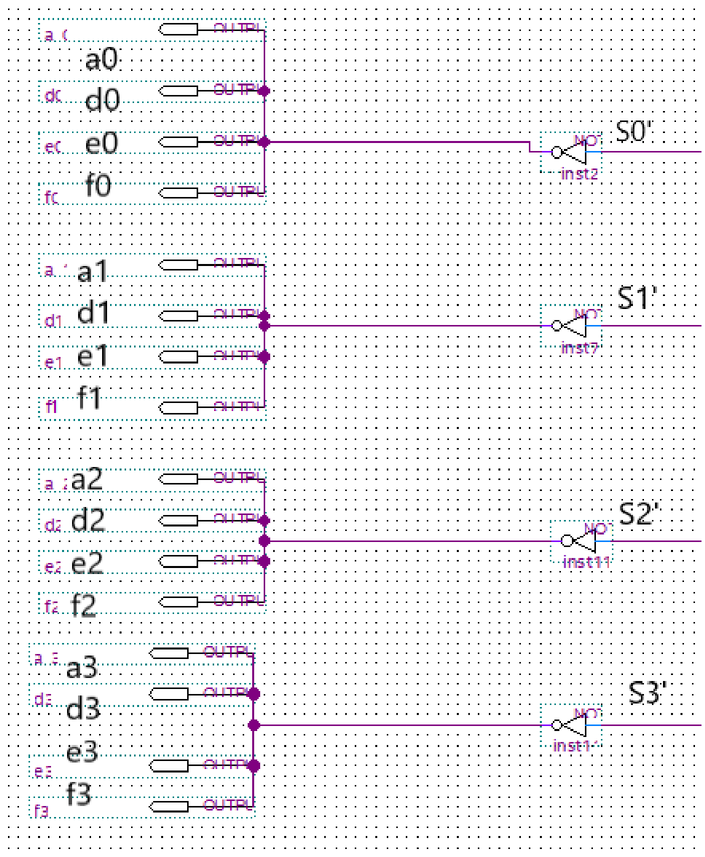
| S or C | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

*XX: table for S0, S1, S2, S3 and C 7SD*

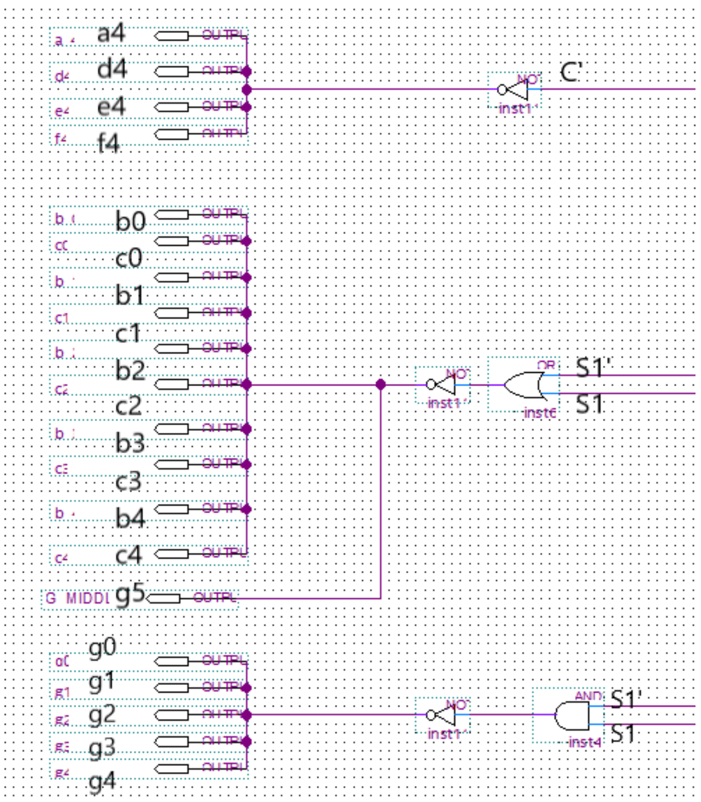
From the truth table we can conclude that logic for seven segment displays will be:

a, d, e, f = S0’ or S1’ or S2’ or S3’ or C’, depending on the output which the display is showing.

B and c segments will always be 1 and g segments will always be 0



*XXI: logic for other five seven segment display*



*XXII: logic for other five seven segment displays*

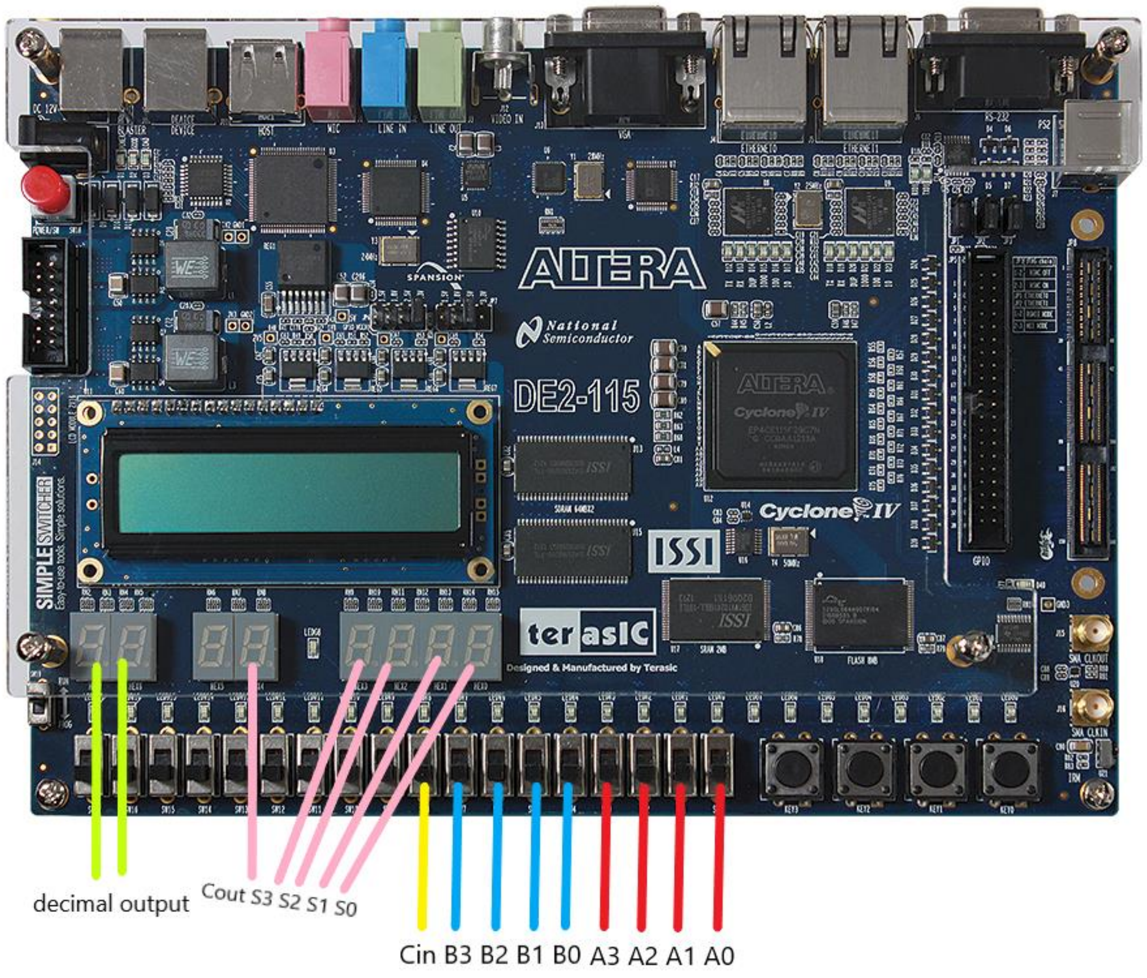
***g5 is for the sixth seven segment display, we used it to make a dash that separates decimal and binary output- for aesthetical purposes*

All our logic for seven segment display had to be inverted because seven segment displays on Altera FPGA are common anode.

As for the always on and always off segments (b's, c's and g's) we had to implements different logic. To create an output that is always 0 we connected two complement (opposite) inputs to an AND gate- for instance S0 and S0' in an and gate will always give us an output of 0. Similarly, to create an output that is always 1 we used two complement inputs in an OR gate.

## PIN assignments

The last thing left for our first project is to assign inputs and outputs to PINs on our FPGA. For our 9 inputs, we assigned 9 switches, from SW0 to SW8. First four switches were assigned for our first binary input A, and the second four switches for our second binary input B, ninth switch was assigned to the carry input. Our outputs were assigned to the seven-segment display, and one extra for the dash that was separating the decimal and binary outputs.



*XXIII: photo of FPGA with assigned PINs*

Inputs:

| Switches | PIN |
|---|---|
| A0 | PIN_AB28 |
| A1 | PIN_AC28 |
| A2 | PIN_AC27 |
| A3 | PIN_AD27 |
| B0 | PIN_AB27 |
| B1 | PIN_AC26 |
| B2 | PIN_AD26 |
| B3 | PIN_AB26 |
| C in | PIN_AC25 |

Seven segment displays:

| S0 output - Hex0 | PIN |
|---|---|
| a0 | PIN_G18 |
| b0 | PIN_F22 |
| c0 | PIN_E17 |
| d0 | PIN_L26 |
| e0 | PIN_L25 |
| f0 | PIN_J22 |
| g0 | PIN_H22 |

| S1 output - HEX1 | PIN |
|---|---|
| a1 | PIN_M24 |
| b1 | PIN_Y22 |
| c1 | PIN_W21 |
| d1 | PIN_W22 |
| e1 | PIN_W25 |
| f1 | PIN_U23 |
| g1 | PIN_U24 |

| S2 output – HEX2 | PIN |
|---|---|
| a2 | PIN_AA25 |
| b2 | PIN_AA26 |
| c2 | PIN_Y25 |
| d2 | PIN_W26 |
| e2 | PIN_Y26 |
| f2 | PIN_W27 |
| g2 | PIN_W28 |

| S3 output – HEX3 | PIN |
|---|---|
| a3 | PIN_V21 |
| b3 | PIN_U21 |
| c3 | PIN_AB20 |
| d3 | PIN_AA21 |
| e3 | PIN_AD24 |
| f3 | PIN_AF23 |
| g3 | PIN_Y19 |

| C output – HEX4 | PIN |
|---|---|
| a4 | PIN_AB19 |
| b4 | PIN_AA19 |
| c4 | PIN_AG21 |
| d4 | PIN_AH21 |
| e4 | PIN_AE19 |
| f4 | PIN_AF19 |
| g4 | PIN_AE18 |

| dash output – HEX5 | PIN |
|---|---|
| g5 | PIN_AH18 |

| decimal output for 1’s – HEX6 | PIN |
|---|---|
| a | PIN_AA17 |
| b | PIN_AB16 |
| c | PIN_AA16 |
| d | PIN_AB17 |
| e | PIN_AB15 |
| f | PIN_AA15 |
| g | PIN_AC17 |

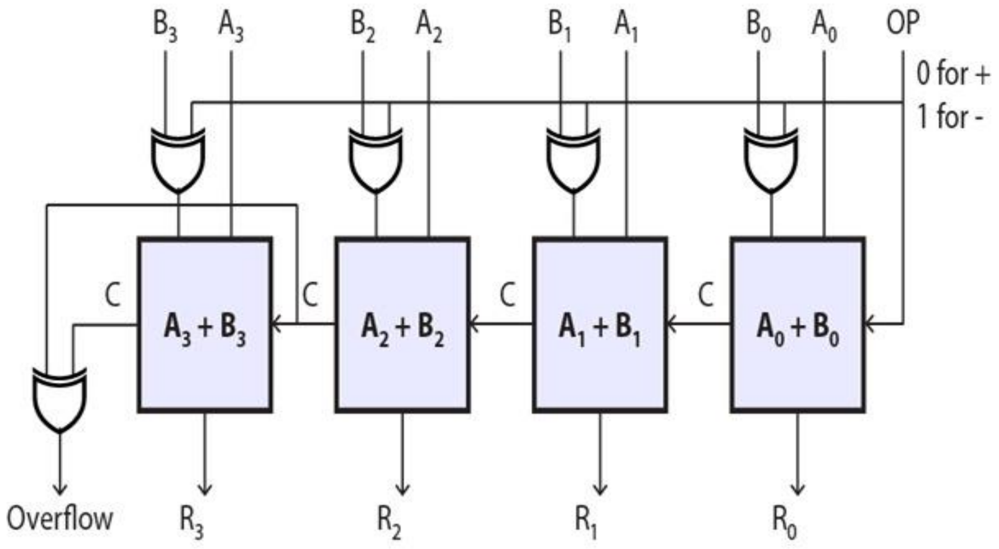
| decimal output for 10’s – HEX7 | PIN |
|---|---|
| a | PIN_AD17 |
| b | PIN_AE17 |
| c | PIN_AG17 |
| d | PIN_AH17 |
| e | PIN_AF17 |
| f | PIN_AG18 |
| g | PIN_AA14 |

## Implementation of a full adder with a subtractor

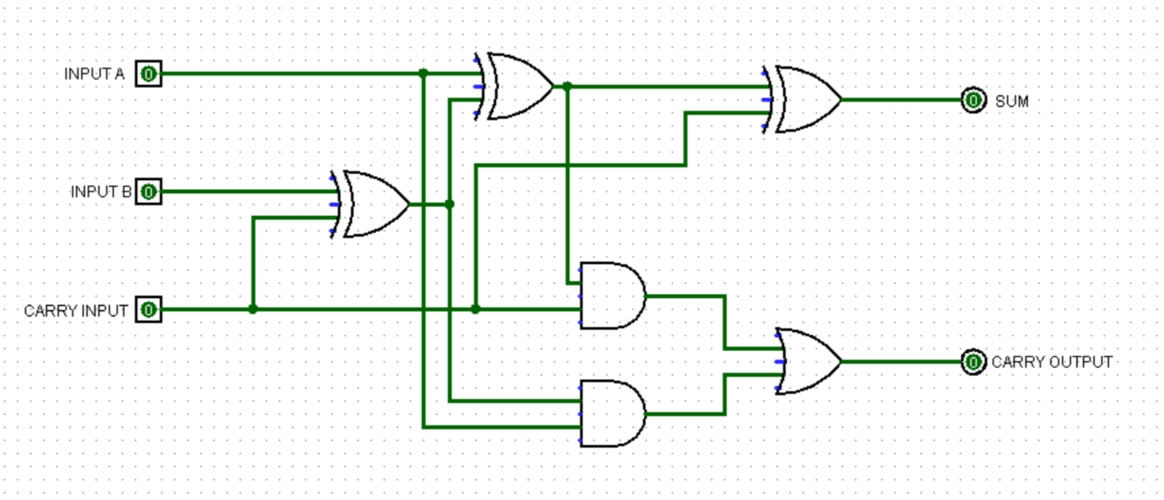
After implementing full-adder, we decided to make full-adder with subtractor. The idea we used for implementation is getting the sum and difference of two numbers based on whether the carry switch is ON or OFF. This time we had four seven segment displays, two of them were always showing the sum, and the other two were showing the sum when carry was OFF and difference when carry was ON.

The difference in logic of a subtractor is the change in the second 4-bit number. While all bits of A stay the same, we have to calculate the 2's complement of B (reversing all the values of 1 to 0 and vice versa, and adding 1 to the result).

Because of this, the second input B has one extra XOR logic gate on each bit, the XOR's inputs are B's bits B0, B1, B2 and B3 and Carry in.

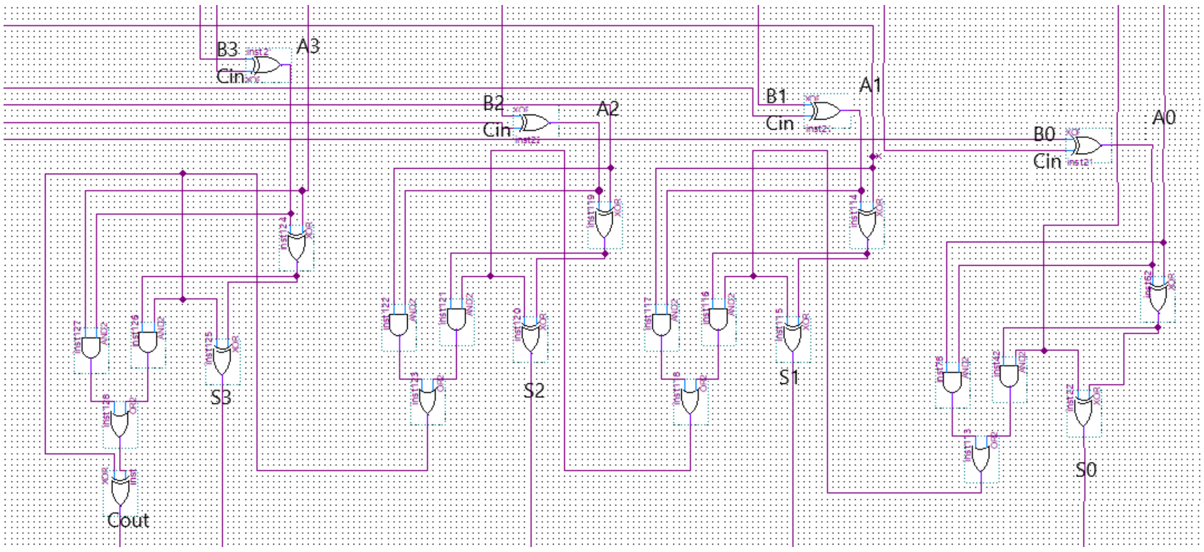


*XXIV: 4-bit full adder with subtractor*



*XXV: Logic we used for 1-bit full adder with subtractor*

The connections of the adder and the subtractor are almost the same, we create 4 1-bit adders and add them together (with the XOR gate for subtraction).

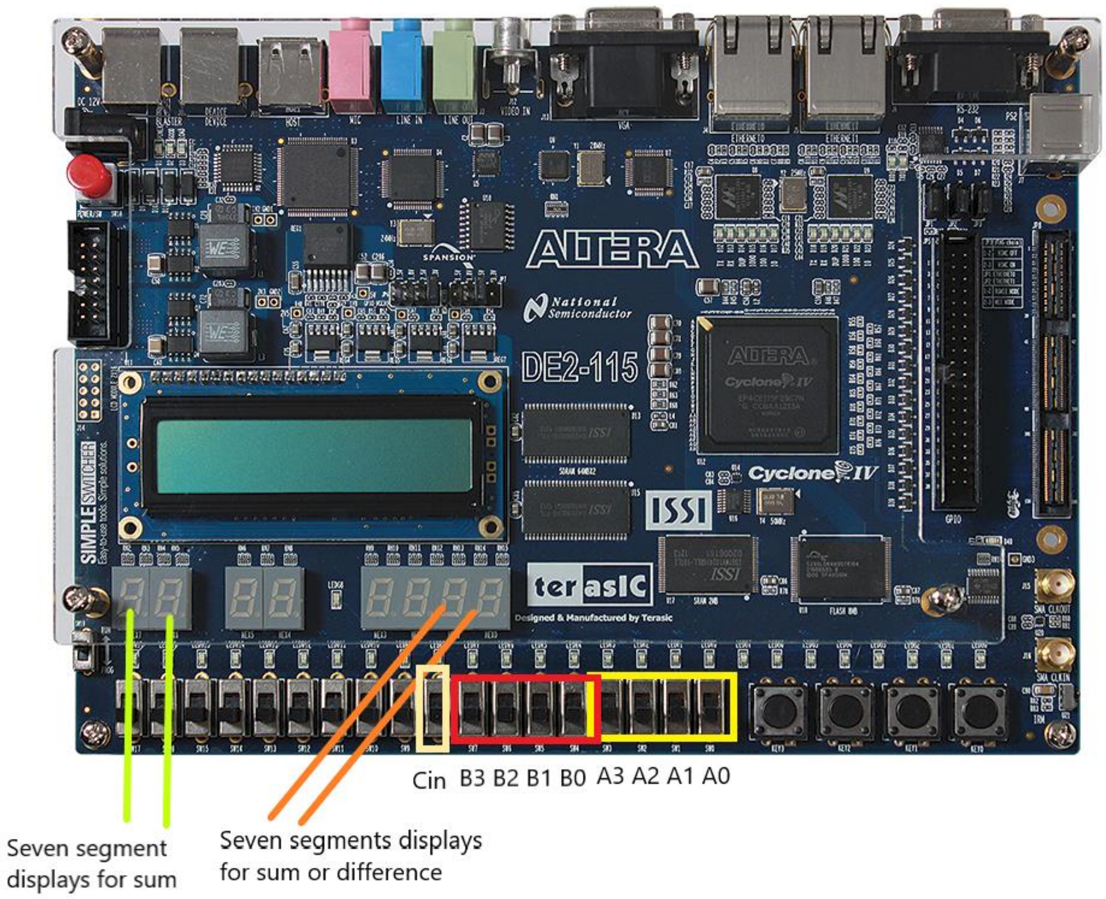


*XXVI: 4-bit full adder with subtractor in Quartus*

The implementation of the decoders for the 7 segment displays is completely the same as before- the only difference being that we do not have the binary outputs since our FPGA board didn’t have enough displays to show 5 more binary outputs. so, we agreed upon displaying the addition on 2 displays and sum or difference on 2 more. The logic for both seven-segment displays was the same as for the ordinary full-adder that we already explained above, so we will not repeat.

## PIN assignments

This time we used, as mentioned only four seven segment displays for output, and input stayed the same.


Cin
B3
B2
B1
B0
A3
A2
A1
A0
Seven segment displays for sum
Seven segments displays for sum or difference

*XXVII: photo of FPGA with used PINs*

### Inputs:

| Switches | PIN |
|---|---|
| A0 | PIN_AB28 |
| A1 | PIN_AC28 |
| A2 | PIN_AC27 |
| A3 | PIN_AD27 |
| B0 | PIN_AB27 |
| B1 | PIN_AC26 |
| B2 | PIN_AD26 |
| B3 | PIN_AB26 |
| C in | PIN_AC25 |

## Seven segment displays

### Displaying sum of two inputs:

| decimal output for 1's – HEX6 | PIN |
|---|---|
| a | PIN_AA17 |
| b | PIN_AB16 |
| c | PIN_AA16 |
| d | PIN_AB17 |
| e | PIN_AB15 |
| f | PIN_AA15 |
| g | PIN_AC17 |

| decimal output for 10's – HEX7 | PIN |
|---|---|
| a | PIN_AD17 |
| b | PIN_AE17 |
| c | PIN_AG17 |
| d | PIN_AH17 |
| e | PIN_AF17 |
| f | PIN_AG18 |
| g | PIN_AA14 |

### Displaying sum or difference depending on carry input:

| S0 output - Hex0 | PIN |
|---|---|
| a0 | PIN_G18 |
| b0 | PIN_F22 |
| c0 | PIN_E17 |
| d0 | PIN_L26 |
| e0 | PIN_L25 |
| f0 | PIN_J22 |
| g0 | PIN_H22 |

| S1 output - HEX1 | PIN |
|---|---|
| a1 | PIN_M24 |
| b1 | PIN_Y22 |
| c1 | PIN_W21 |
| d1 | PIN_W22 |
| e1 | PIN_W25 |
| f1 | PIN_U23 |
| g1 | PIN_U24 |

## Conclusion

In the end, this project gave me a lot of insight in troubleshooting bigger circuits, creating decoders for seven-segment displays, and FPGA boards in general.

However, that doesn’t speak for the difficulty of the project itself. With such a big scheme we had to work with it was almost impossible to find mistakes and figure out which connections needed to be changed. It was very time-consuming, and it took us a lot of tries to get it right.

Mid-project we concluded that the problem at hand would be a lot easier to do with some form of HDL. Still, it was a great learning experience, and I hope to re-do something similar in the future.