

## CE100 Lab Report 2

3-bit adder with carry-in

Student Name: Armando Silva

Tutor: Dawn Hustig-Schultz

Lab Sec: TTH 2:00pm-4:00pm

Date: January 22th, 2016

## Description:

The purpose of this lab was to get familiar with multiplexers and creating Full Adders to carry one bit to the next full adder. The design of the full adder comes from the logic with combines three bits (a, b, cin) into two (cout, s) bits. Then using these full adders, I constructed 3-bit full adder symbol; which sends its outputs to the 7-segment display. This 7-segment display turns on or off the 7-segment LEDs to display the values of (n3, n2, n1, n0) as a hex digit. When the design was finished we used our ISE simulator ISim to display the times of the inputs and outputs of the top level schematic.

## Part A- Adder Design

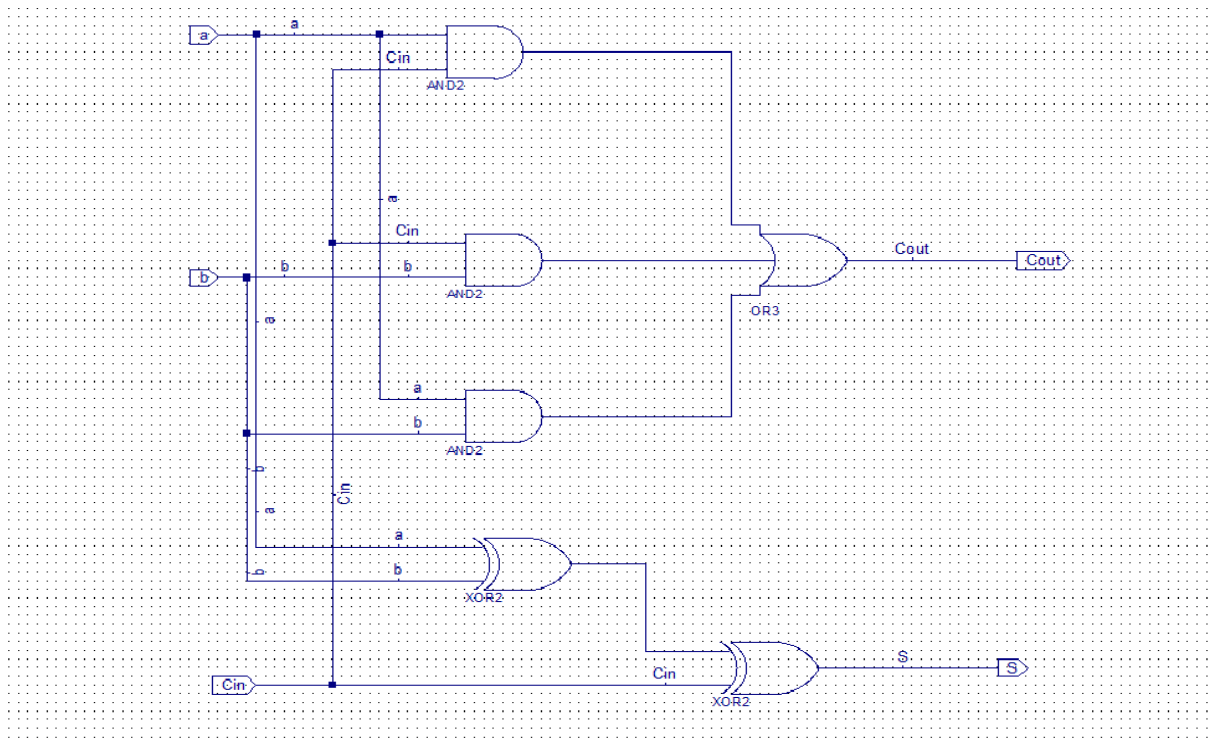


Figure 1: The Full Adder

As we are provided the logic inputs of our full adder (a, b, cin), I wrote out the truth table corresponding to these inputs. Then based upon the truth table, I got the minterms and created a minimum sum of products for the outputs (cout, s). From figure 1, we can determine the longest path from any input to any output in the full adder. As we notice that there are three inputs at the first stage of the logic gates, therefore that is our n. Now we check the longest path. Choosing the input, A, it is three to the cout stage, however it needs to go from cin to s after that, and to get to cin counts as two. Ultimately ending with,  $3 + 2$  which is 5 for the longest path. For n-bit adder, to determine the length of the longest path from any input to any output, we do exactly what I derived for the longest path. Which is: count one for each input of a logic gate to cout, then count two for cin to s and sum those numbers up.

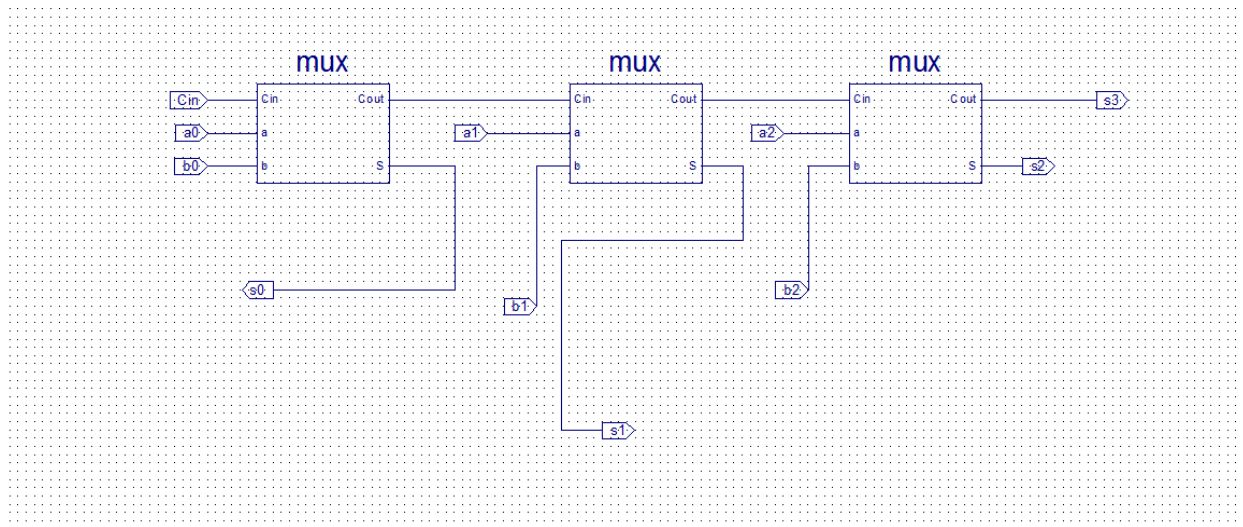


Figure 2: Is the Top Level Full Adder

## Results:

A more through explanation of the configuration of Figure 2 is to pass along the carry out of each full adder ("mux") and input it to Cin. Note that for the very first Full Adder, we will have cin connecting to ground or zero, because we start with no carry in. From these two figures we can determine that our full adder is properly symbolize and ready for the 7-segment display to take in while this symbol has the three inputs and four outputs.

## Part B- 7-Segment Display

This part of the lab helped me familiarize myself with the Basys2 Board and how to turn on the 7-segment LEDs to display the value I wanted as a hex digit. We were to read about the 7-Segment Display in the Basys2 Board Reference Manual to determine which FPGA pin numbers were used for the 7-segment display controls. To create the schematic of Figure 3, we needed to derive the minimalist sum of product for each hex bit from 0 to F. To do this, I wrote a truth table, that corresponded the pins turned on for the display, I had my truth table inverted. Therefore, it was a little confusing to read, however my 1's was 0's and 0's was 1's. On the right of the truth table was the inputs N0, N1, N2, N3, which corresponded to 16 inputs. At the end of this, we were to obtain the schematic of Figure 3 as shown.

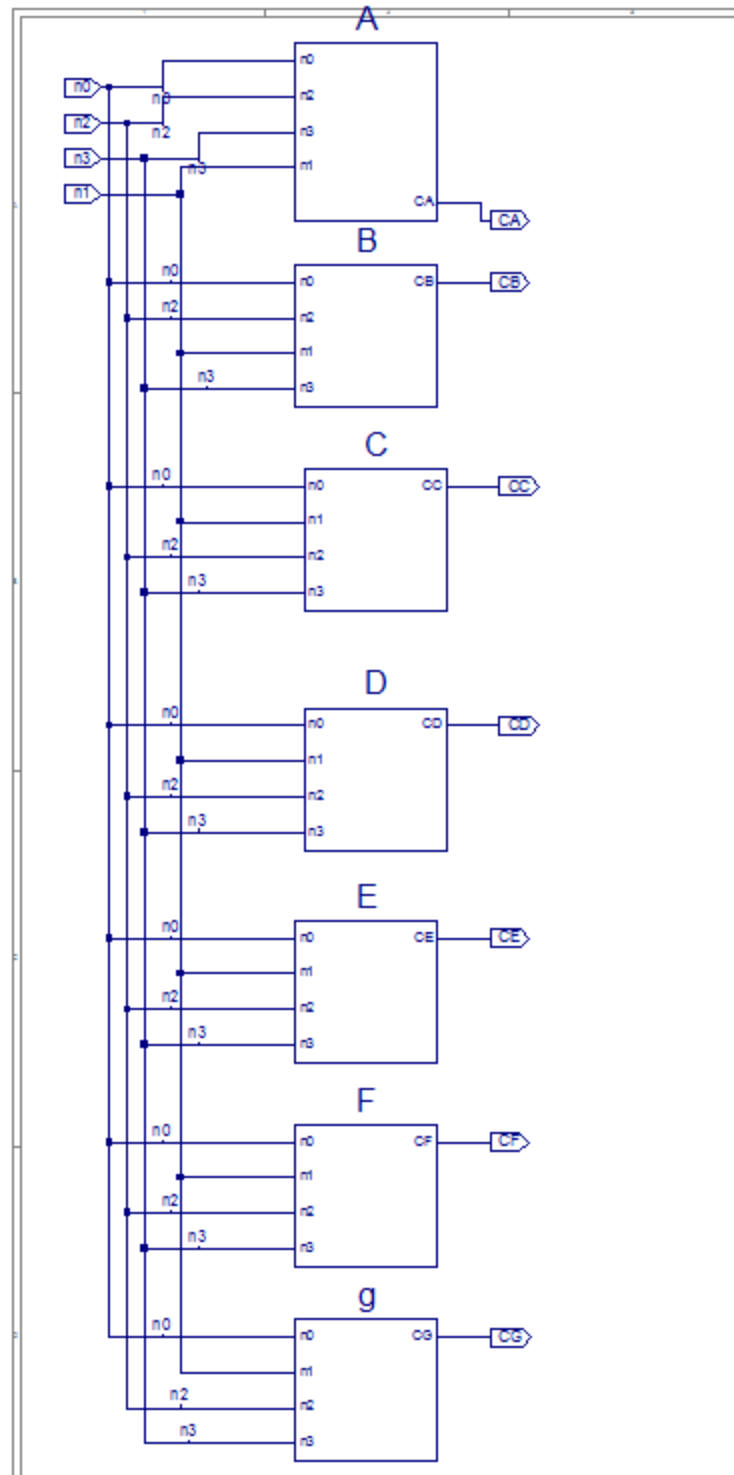


Figure 3: The 7-Segment Display

Results:

Great, after we have derived the 7-Segment Display, we can now use the outputs from the Full adder and tie them to the input them into our Segment Display. For further in depth of each letter symbol, you can find each of the

The schematic diagram illustrates the hardware implementation of the proposed algorithm, organized into three main functional blocks: **Add**, **Segment**, and **Thisan**.

**Add Block:** This block performs an 8-bit addition. It features two 4-bit input buses,  $a[3:0]$  and  $b[3:0]$ , each composed of four 1-bit inputs ( $a_1, a_2, a_3, a_0$  and  $b_1, b_2, b_3, b_0$ ). These inputs are connected to a central 8-bit adder block. The adder has two 4-bit output buses,  $s[2:0]$  and  $s[3:0]$ , which are connected to the **Segment** block. The adder also includes several control inputs:  $sw2$  (LOC=F3),  $sw3$  (LOC=B4),  $sw5$  (LOC=F3),  $sw0$  (LOC=P11),  $sw1$  (LOC=L3),  $sw4$  (LOC=E2), and  $sw6$  (LOC=G3). Each control input is connected to a 1-bit input bus (IBUF) and a 1-bit output bus (OBUF).

**Segment Block:** This block processes the 8-bit output from the adder. It has four 1-bit inputs ( $n_0, n_2, n_3, n_1$ ) and four 1-bit outputs ( $CA, CB, CC, CD, CE, CF, CG$ ). The inputs are connected to a central 4-bit segment block. The outputs are connected to four 1-bit output buses (OBUF).

**Thisan Block:** This block contains three comparators ( $Thisan1, Thisan2, Thisan3$ ) that compare the output of the adder with a reference value. The comparators have three inputs:  $LOC=H12$ ,  $LOC=M13$ , and  $LOC=K14$ . The outputs are connected to three 1-bit output buses (OBUF). The comparators also have control inputs:  $VCC$  and  $GND$ .

We lastly connect all the switches to their appropriate inputs of the full adder and note that our An1, 2, 3 are high (meaning that they will be off). This is so that the three other LED displays will be off and only An0 will be on. Furthermore, these are the symbols for the 7-Segment muxes (A-G):

```

graph LR
    n1[n1] --- AND4B1
    n2[n2] --- AND4B1
    n3[n3] --- AND4B1
    n1 --- AND4B2
    n2 --- AND4B2
    n3 --- AND4B2
    n1 --- AND4B3
    n2 --- AND4B3
    n3 --- AND4B3
    n1 --- AND4
    n2 --- AND4
    n3 --- AND4
    AND4B1 --> OR4
    AND4B2 --> OR4
    AND4B3 --> OR4
    AND4 --> OR4
    OR4 --> Out[Output]
  
```

Figure 8: Symbol for CD:

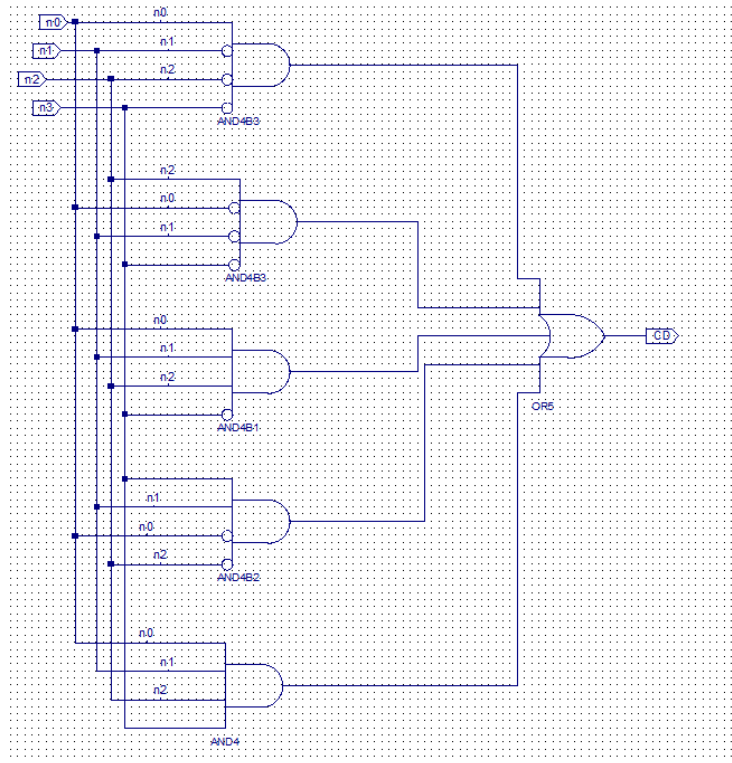


Figure 9: Symbol for CE:

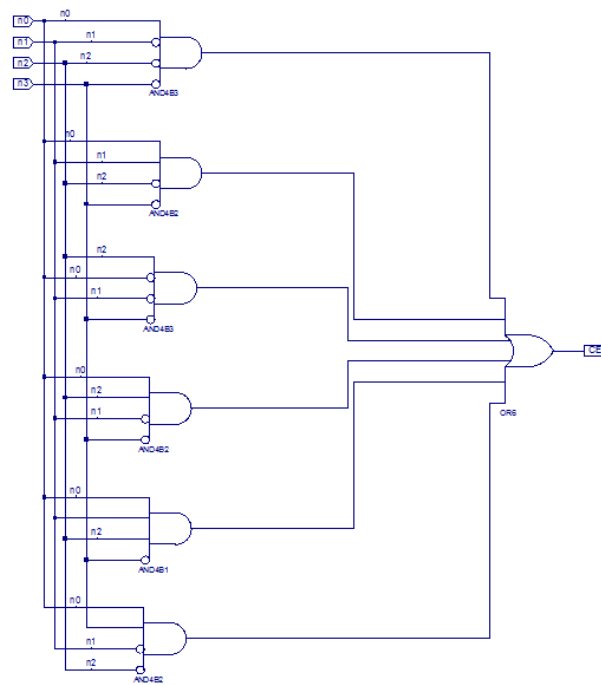


Figure 10: Symbol for CF

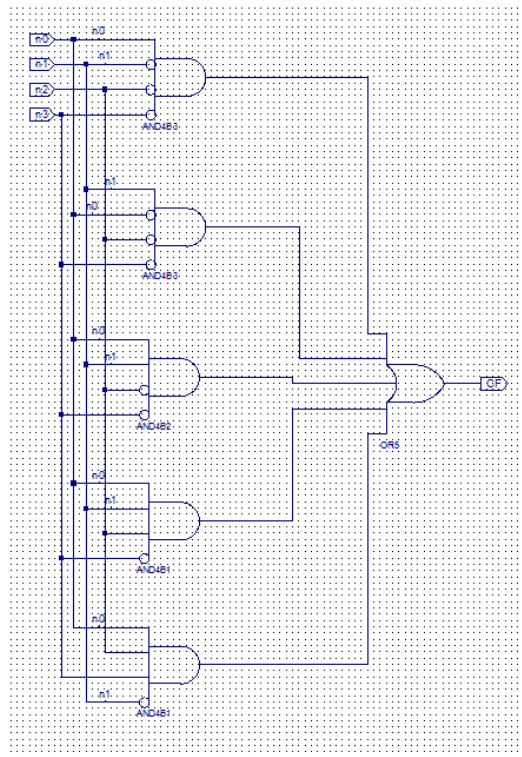
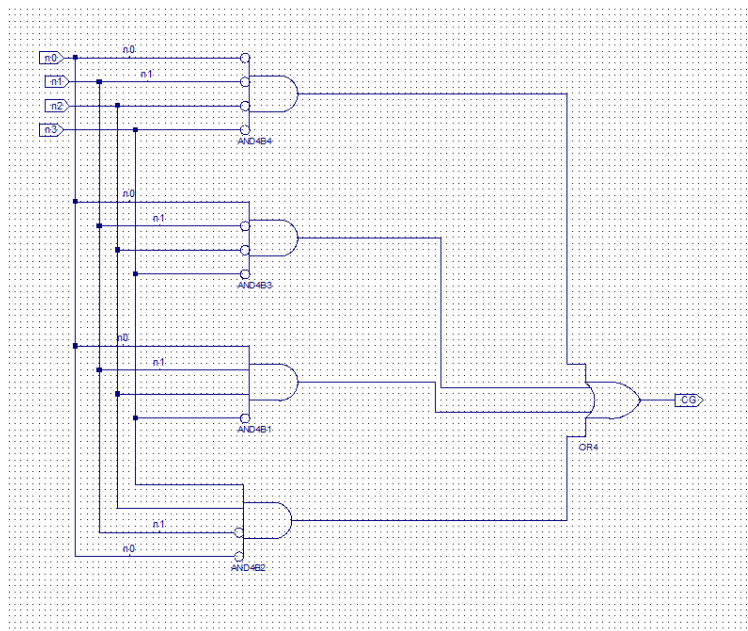


Figure 11: Symbol for CG:





## Waveform

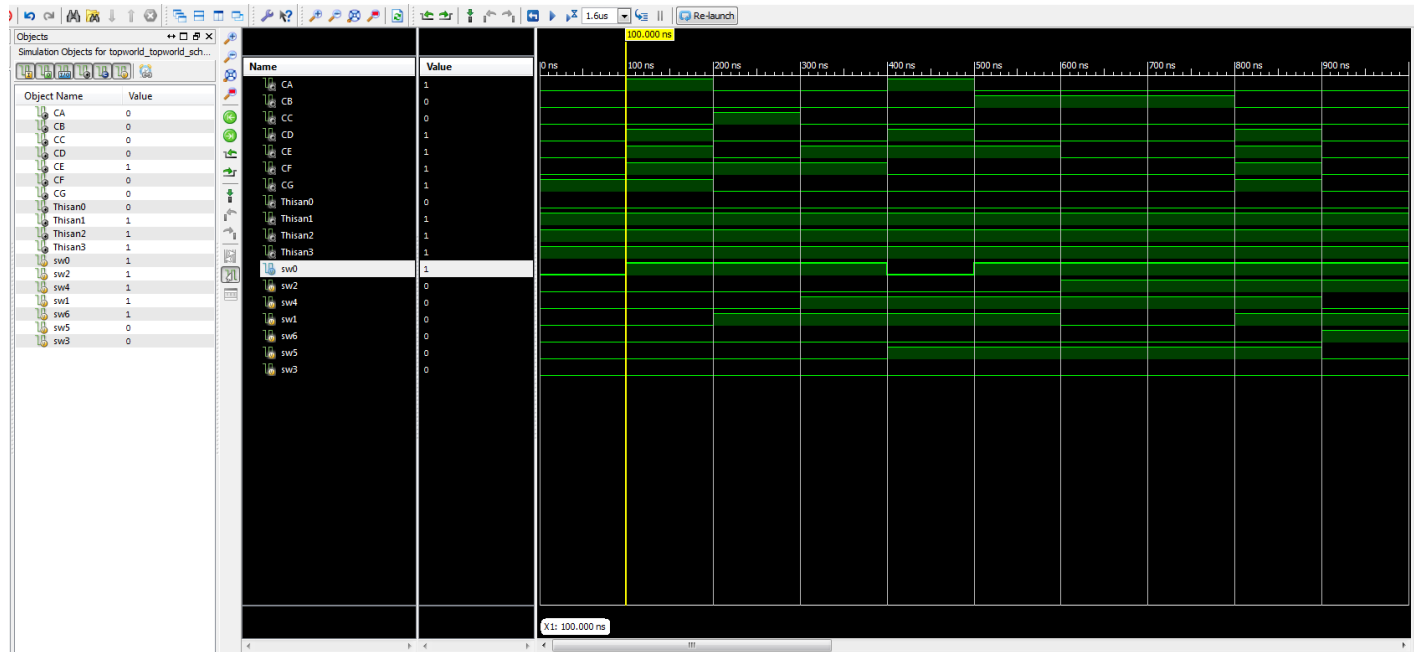


Figure 11: Simulation for the Top Level

The number of possible input values in a 7-bit for my adder 7 choose 3 (3 inputs), totaling 35 different ways. We tested only 4 choose 3, which was 4 different ways, totaling to  $4/35 = 11\%$  of test simulations.

## Conclusion:

In this lab, we used full adder multipliers to minimize the logic design of our three-bit adder. Using the Basys2 Board, we were able to display correct hexadecimal output along with Figure 11 simulator. Ultimately, this verified our logic sums of products and helped with the understanding of full adders. The following is the result of my lab notebook:

# Lab 2

Switch	SW6	SW5	SW4	SW3	SW2	SW1	SW0
Input	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>	C <sub>in</sub>

a	b	C <sub>in</sub>	C <sub>out</sub>	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

$$C_{out} = \bar{a} b C_{in} + a \bar{b} C_{in} + a b \bar{C}_{in} + a b C_{in}$$

$$C_{in}(\bar{a} b + a \bar{b}) + a b(\bar{C}_{in} + C_{in})$$

$$C_{in}(\bar{a} b + a \bar{b}) + a b$$

$$C_{in}(\bar{a} + b) + a b$$

$$C_{out} = a C_{in} + b C_{in} + a b$$

$$S = \bar{a} \bar{b} C_{in} + \bar{a} b \bar{C}_{in} + a \bar{b} \bar{C}_{in} + a b C_{in}$$

$$\bar{a}(\bar{b} C_{in} + b \bar{C}_{in}) + a(\bar{b} \bar{C}_{in} + b C_{in})$$

$$S = (a \oplus b) \oplus C_{in}$$

Figure 12: Truth Table

This picture is the truth table for the 3 inputs (a,b,cin) and taking the sum of products to minimize my expression.



Figure 13: The 7-Segment display

This is the truth table for 7 segment display. My numbers corresponding to the letters are inverted; therefore, a 0 is a 1 and 1 a 0. On the right side, is NOT an output rather the input (N0, N1, N2, N3); which are 16 possibilities. Apologizes for the confusion. This is documentation of which pins of the FPGA I used. They were connected through outputs of the Full Adder which took in the switches, check Figure 4.



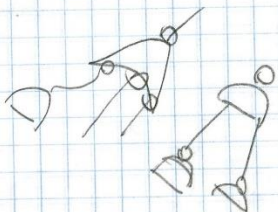
$$\begin{aligned}
 \downarrow A &= \overline{n_3} \overline{n_2} \overline{n_1} \overline{n_0} + \overline{n_3} \overline{n_2} \overline{n_1} \overline{n_0} + \overline{n_3} \overline{n_2} \overline{n_1} \overline{n_0} + \overline{n_3} \overline{n_2} \overline{n_1} \overline{n_0} \\
 B &= (\overline{n_3} + \overline{n_2} + \overline{n_1} + \overline{n_0}) (\overline{n_3} + \overline{n_2} + \overline{n_1} + \overline{n_0}) (\overline{n_3} + \overline{n_2} + \overline{n_1} + \overline{n_0}) (\overline{n_3} + \overline{n_2} + \overline{n_1} + \overline{n_0}) \\
 &\quad (\overline{n_3} + \overline{n_2} + \overline{n_1} + \overline{n_0}) (\overline{n_3} + \overline{n_2} + \overline{n_1} + \overline{n_0}) \\
 C &= (\overline{n_3} \odot \overline{n_2} \odot \overline{n_1} \odot \overline{n_0}) (\overline{n_3} \odot \overline{n_2} \odot \overline{n_1} \odot \overline{n_0}) (\overline{n_3} \odot \overline{n_2} \odot \overline{n_1} \odot \overline{n_0}) \\
 &\quad (\overline{n_3} \odot \overline{n_2} \odot \overline{n_1} \odot \overline{n_0}) \\
 D &= (\overline{n_3} + \overline{n_2} + \overline{n_1} + \overline{n_0}) (\overline{n_3} + \overline{n_2} + \overline{n_1} + \overline{n_0}) (\overline{n_3} + \overline{n_2} + \overline{n_1} + \overline{n_0}) \\
 &\quad (\overline{n_3} + \overline{n_2} + \overline{n_1} + \overline{n_0}) (\overline{n_3} + \overline{n_2} + \overline{n_1} + \overline{n_0}) \\
 * E &= (\overline{n_3} + \overline{n_2} + \overline{n_1} + \overline{n_0}) (\overline{n_3} + \overline{n_2} + \overline{n_1} + \overline{n_0}) (\overline{n_3} + \overline{n_2} + \overline{n_1} + \overline{n_0}) \\
 &\quad (\overline{n_3} + \overline{n_2} + \overline{n_1} + \overline{n_0}) (\overline{n_3} + \overline{n_2} + \overline{n_1} + \overline{n_0}) (\overline{n_3} + \overline{n_2} + \overline{n_1} + \overline{n_0}) \\
 F &= (\overline{n_3} + \overline{n_2} + \overline{n_1} + \overline{n_0}) (\overline{n_3} + \overline{n_2} + \overline{n_1} + \overline{n_0}) (\overline{n_3} + \overline{n_2} + \overline{n_1} + \overline{n_0}) \\
 &\quad (\overline{n_3} + \overline{n_2} + \overline{n_1} + \overline{n_0}) (\overline{n_3} + \overline{n_2} + \overline{n_1} + \overline{n_0}) \\
 \downarrow G &= (\overline{n_3} + \overline{n_2} + \overline{n_1} + \overline{n_0}) (\overline{n_3} + \overline{n_2} + \overline{n_1} + \overline{n_0}) (\overline{n_3} + \overline{n_2} + \overline{n_1} + \overline{n_0}) + \\
 &\quad (\overline{n_3} + \overline{n_2} + \overline{n_1} + \overline{n_0})
 \end{aligned}$$


Figure 14: The sum of products for each letter

Originally, I had mixed up sum of products with product of sums but at the end I figured out that each gate should be matching letter A, where AND all four inputs and OR them with other expressions.