EE 457: Digital Integrated Circuits
Project #1 Report Cover Sheet
Due 2/27/2019

PROJECT TITLE: Ripple Carry Adder

Student Name: Sebastian Grygorczuk

Sections	GRADE
Section 1: Executive Summary	/5
Section 2: Introduction	/5
Section 3: Electric Schematic	/10
Section 4: LTSPICE simulations of Schematic	/10
Section 5: IRSIM simulations for Schematic	/10
Section 6: Electric Layout	/25
Section 7: LTSPICE simulations of Layout	/10
Section 8: IRSIM simulations for Layout	/10
Section 9: Summary of Measurements in a) Propagation of Gates and Entier I/O b) Total Chip Area in um^2	 /3 /2
Section 10. Comparison of Schematic and Layout	/5
Section 11. Conclusion	/5
TOTAL	**/100**

Executive Summary

Through this project I have learned how to create a Ripple Carry Adder on layout by designing five different schematic in hopes to create the most compact Full Adder. To do this I've gone over two schematics of a Full Adder and four different XOR schematics. The Introduction goes over a lot background information and process which I went through and the Schematic section shows off all the different design which lead to my final design. This showed me that there are many ways to design very simple components which can make a big difference when trying to transfer their ideas onto a layout. Performing the Rise and Fall time and the propagation analysis was an interesting challenge of figuring out how to follow a signal from the input to the output in a clear and easy manner and the IRSIM was a good although a cumbersome method of verifying if the calculations done by the RCA were correct.

Introduction

The first design was based on a nine NAND gate set up of two Half Adders to create the Full Adder shown in Figure 1, which I got from the Electronic Hub. To make this design work I had to have an operational NAND gate from which I would build a Half Adder and then use two Half Adders to create one Full Adder.

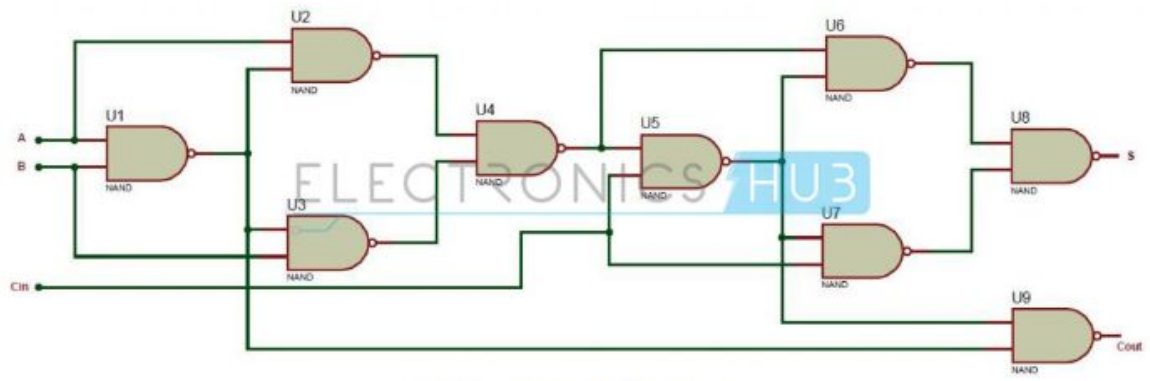


Figure 1. Full Adder Using NAND Gates as shown on Electrics Hub

The formula for NAND gate is F = (X AND Y)' and is shown in Table 1. This is a basic element that will help me build a XOR gate which is a part of a Half Adder.

Table 1. NAND Truth Table

X	Y	F

0	0	1
0	1	1
1	0	1
1	1	0

A XOR gate takes in two inputs and output high only if one of the inputs is high otherwise the output is zero. This allows us to perform a logic of addition such that 0+0 = 0, 1+0 = 1, 0+1 = 1 and 1+1 = 10 ~ 0, of course the last one can't be done and we'll need a second output for the carry out 1 which will be built into the Half Adder. The formula for a XOR gate is F = (A AND B' OR A' AND B). XOR gate using only NAND gates formula is F = ((X AND (X AND Y)')' AND (Y AND (X AND Y)')')' as shown in Figure 1.

Table 2. XOR Truth Table

X	Y	F
0	0	0
0	1	1
1	0	1
1	1	0

The Half Adder takes two inputs X and Y and uses a XOR gate for the addition and uses a AND Gate for Carry Out. The Carry Out saves the 1 that would result from the addition of 1+1 = 10 and takes that 1 to the next addition inside the Full Adder so that Sum = (X XOR Y) XOR (Carry In). Thus the Sum = X XOR Y or ((X AND (X AND Y)')' AND (Y AND (X AND Y)')')' and Carry Out = X AND Y or ((X AND Y)' AND (X AND Y)')'.

Table 3. Half Adder Truth Table

X	Y	Sum	Carry Out
0	0	0	0
0	1	1	0

1	0	1	0
1	1	0	1

A Full Adder is a combination of two Half Adders, where

Sum_0 = (X XOR Y)
Sum = Sum_0 XOR (Carry In)

Or

Sum_0 = ((X AND (X AND Y)')' AND (Y AND (X AND Y)')')'
Sum = ((Sum_0 AND (Sum_0 AND Carry_In)')' AND (Carry_In AND (Sum_0 AND Carry_In)')')'

Carry Out = (X AND Y) OR (X AND Carry In) OR (Y and CARRY IN).
Carry Out = ((X AND Y)' AND (Sum_0 AND Carry_In)')'

This allows us to add three inputs the initial X and Y and a Carry in which would be given from previous calculation so that a addition such as this is possible

4 Bit Addition Example

9 = 1001 +5 = 0101 ---------- 14 = 1110	X0 = 1 Y0 = 1 Carry In = 0 Sum = 0 Carry Out = 1 X1 = 0 Y1 = 0 Carry In = 1 Sum = 1 Carry Out = 0 X2 = 0 Y2 = 1 Carry In = 0 Sum = 1 Carry Out = 0 X3 = 1 Y2 = 0 Carry In = 0 Sum = 1 Carry Out = 0

Tabel 4. Full Adder Truth Table

Carry In	Input X	Input Y	Sum	Carry Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0

1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Once that was that design was complete I found that the logic for Carry Out proved to be very cumbersome to implement in layout form. Thus I created a new design which separated the Carry Out from using the outputs of the inner components of the XOR gates shown in Figure 2 which I learned about on the GeeksforGeeks website but since we are using NAND Gates I've turned it into what is seen in Figure 3.

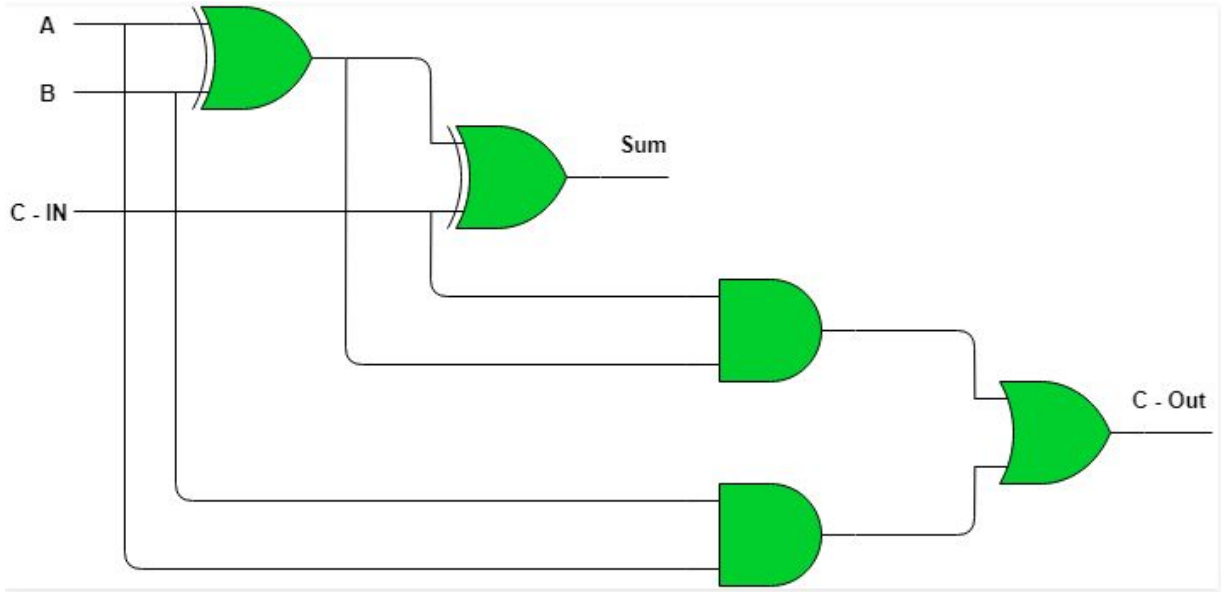


Figure 2. Full Adder Carry Out Version 2

This makes the layout a bit simpler since the output aren't coming from inside the XOR gates giving this as the new boolean expression for carry out
Sum_0 = A XOR B = ((X AND (X AND Y)')' AND (Y AND (X AND Y)')')'
Carry Out = (A AND B) Or (Carry In AND Sum_0)
Or
Carry Out = ((A AND B)' AND (Carry In AND Sum_0)')'

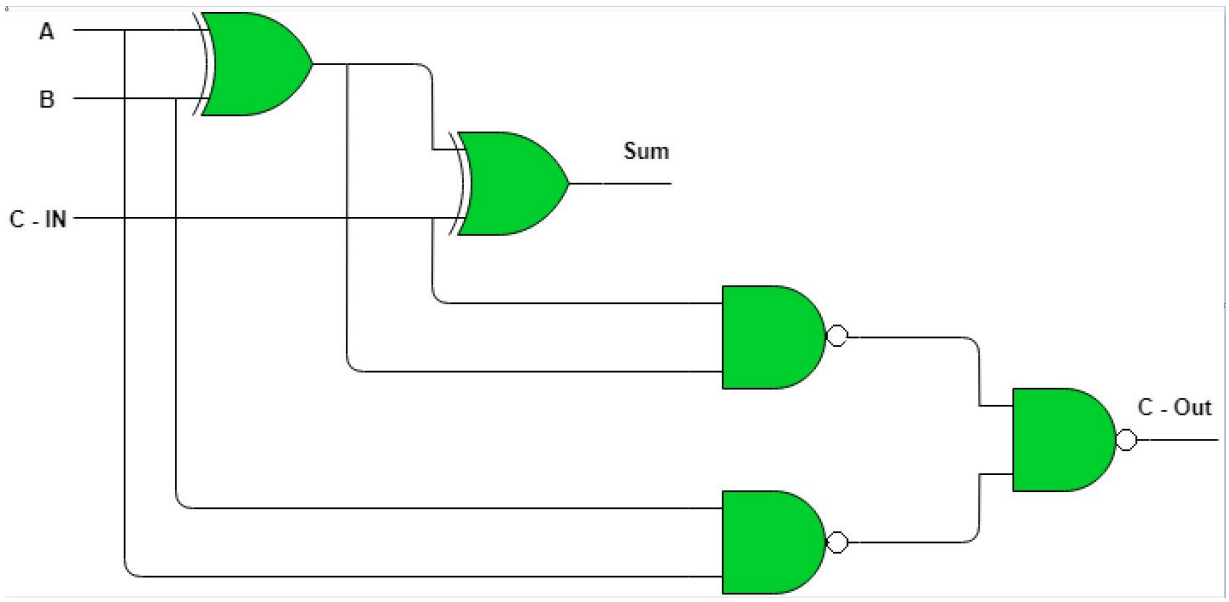


Figure 3. Full Adder Carry Out NAND Gates

This proves to be easier to implement however it requires eleven NAND gates to do so making a very long layout. To resolve this I turned my attention to making the XOR gate a bit smaller in layout form. I started off with the new design for the XOR gate shown in Figure 4, that I found from VLSI Universe.

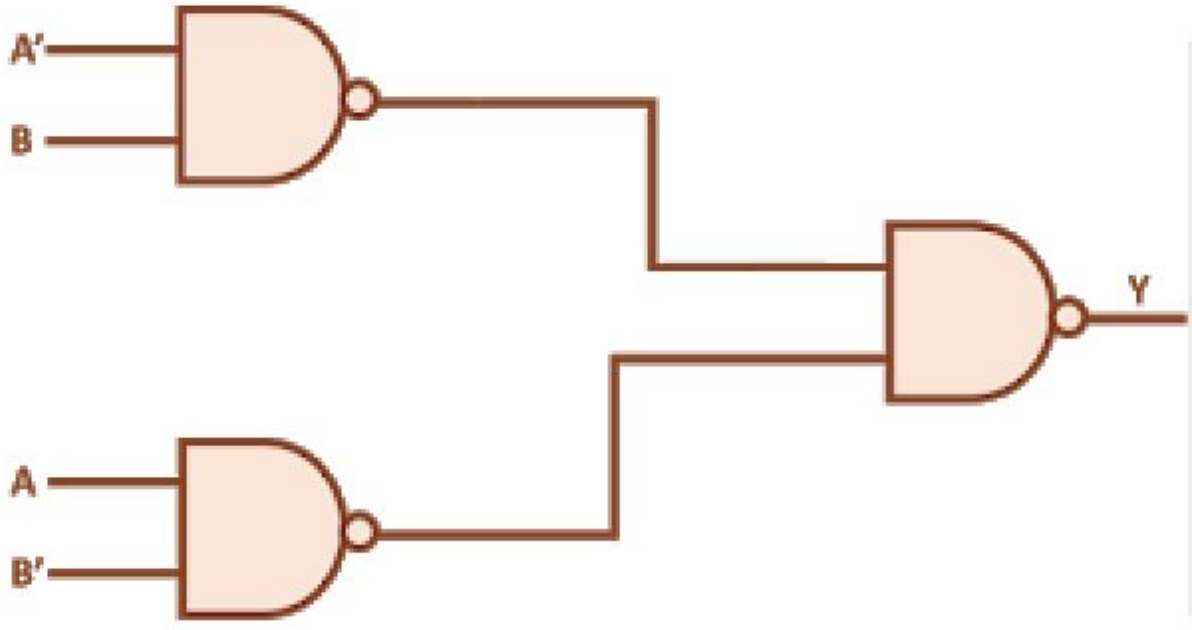


Figure 4. XOR Version 2

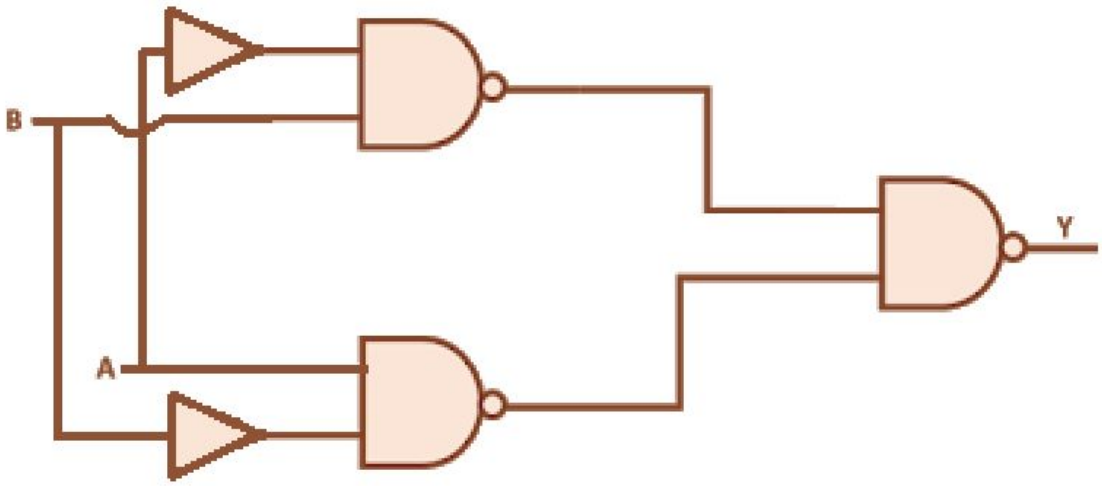


Figure 5. Full Design of the XOR gate.

This design requires the implementation of two inverters and three NAND gates as shown in Figure 5 and the use of inverters who's truth table is shown in Table 5. This approach was a step in the right direction but didn't do much for compressing the XOR gate. Taking a step back from looking at the schematic and focusing on boolean expression for the XOR gate, F = (A AND B' OR A' AND B), shows that it can be simplified using techniques we learned in the class. Thus I've come up with a new design that shown in Figure 6.

Table 5. Inverter

X	F
0	1
1	0

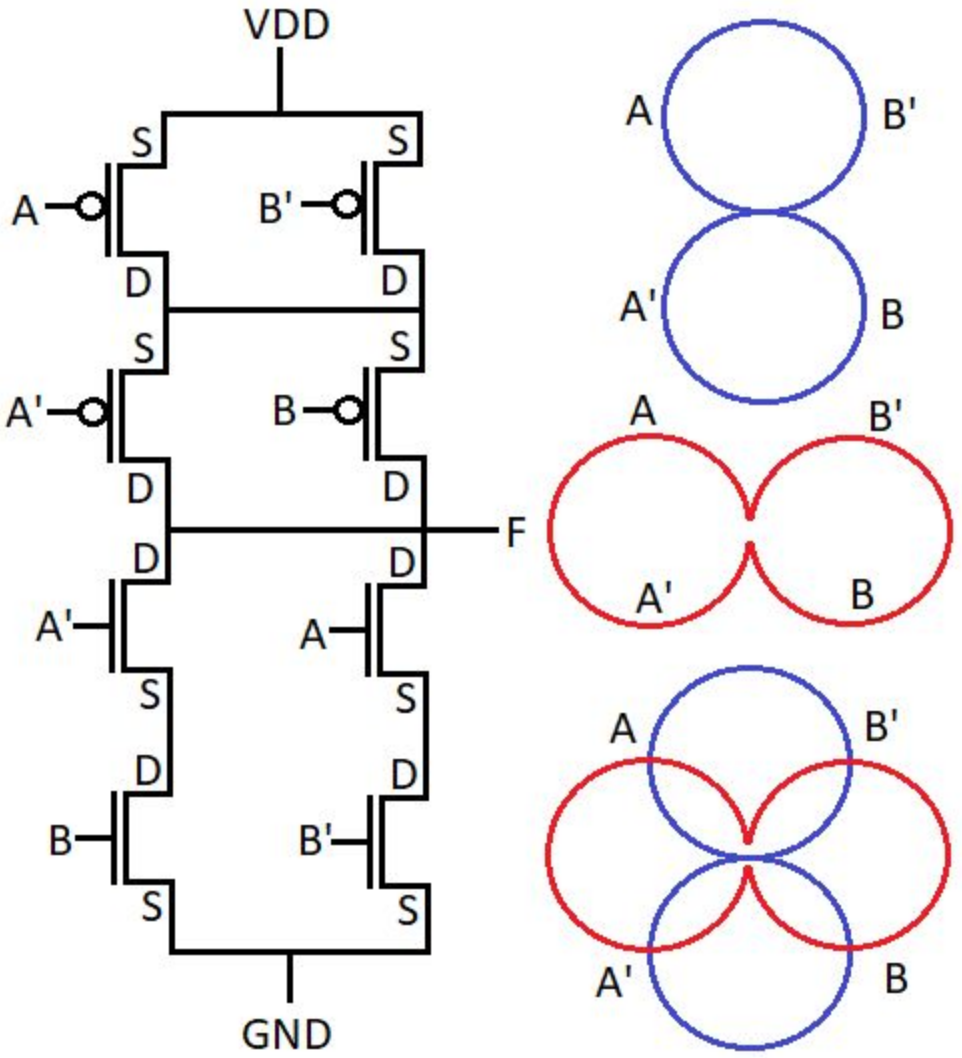
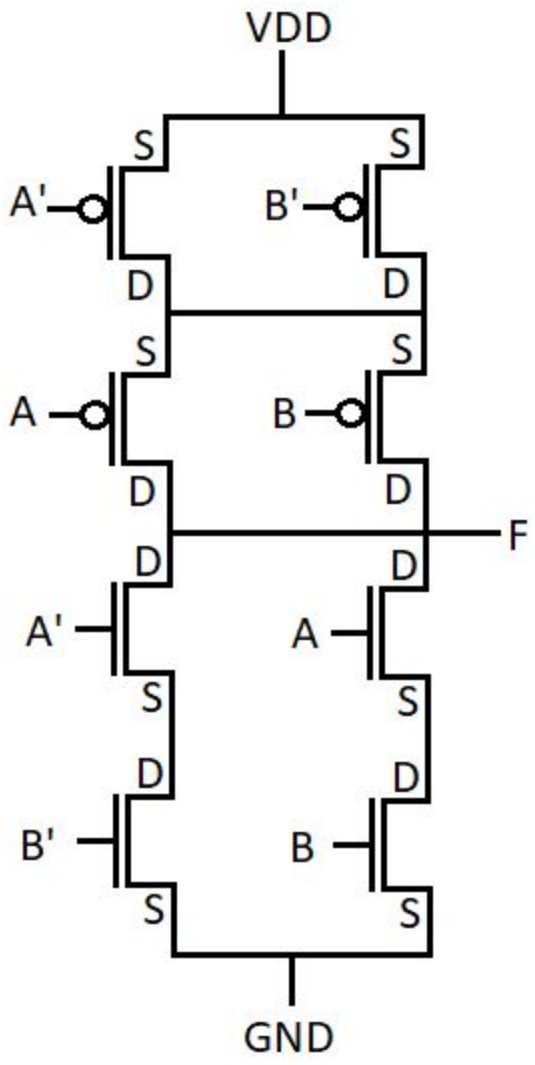


Figure 6. XOR gate Version 3 with Euler's Path

This was another step in the right direction however, when design it I forgot that the F output will be inverse or whatever I created thus this network gives out F = (A AND B' OR A' AND B)' which requires two inverters for A and B and one additional inverter after F. To rectify that I created the final design for the XOR gate which was that of a XNOR, thus F = ((A' AND B') +(A AND B'))' giving me the correct output without the need for an third inverter shown in Figure 7.

Table 6. XNOR Truth Table

X	Y	F
0	0	1
0	1	0
1	0	0
1	1	1



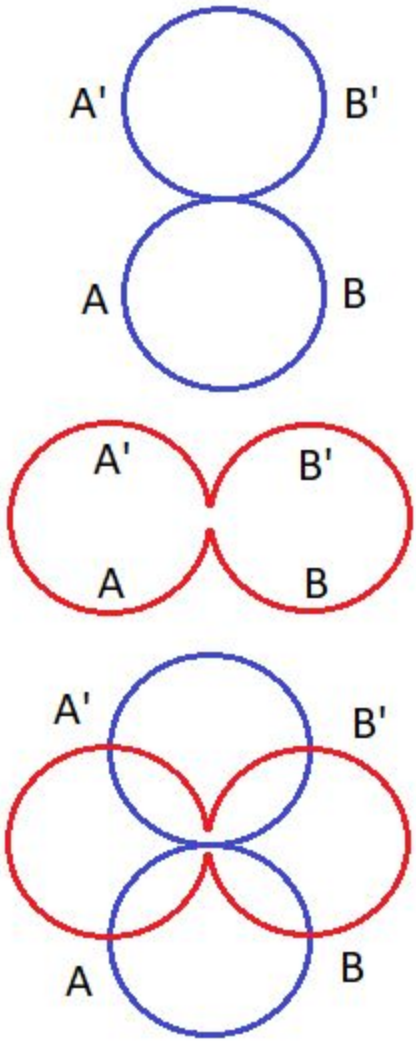


Figure 7. XNOR gate Version 3 with Euler's Path

This setup gives me the following Euler's Paths

AA’B’B, ABB’A’, A’B’BA, A’ABB’, B’BAA’ B’A’AB, BAA’B’ BB’A’A

I chose the B’BAA’ because I planned to set the inverter for B right before the XOR gate on the layout. Figure 8 shows the Stick Diagram for the XOR gate.

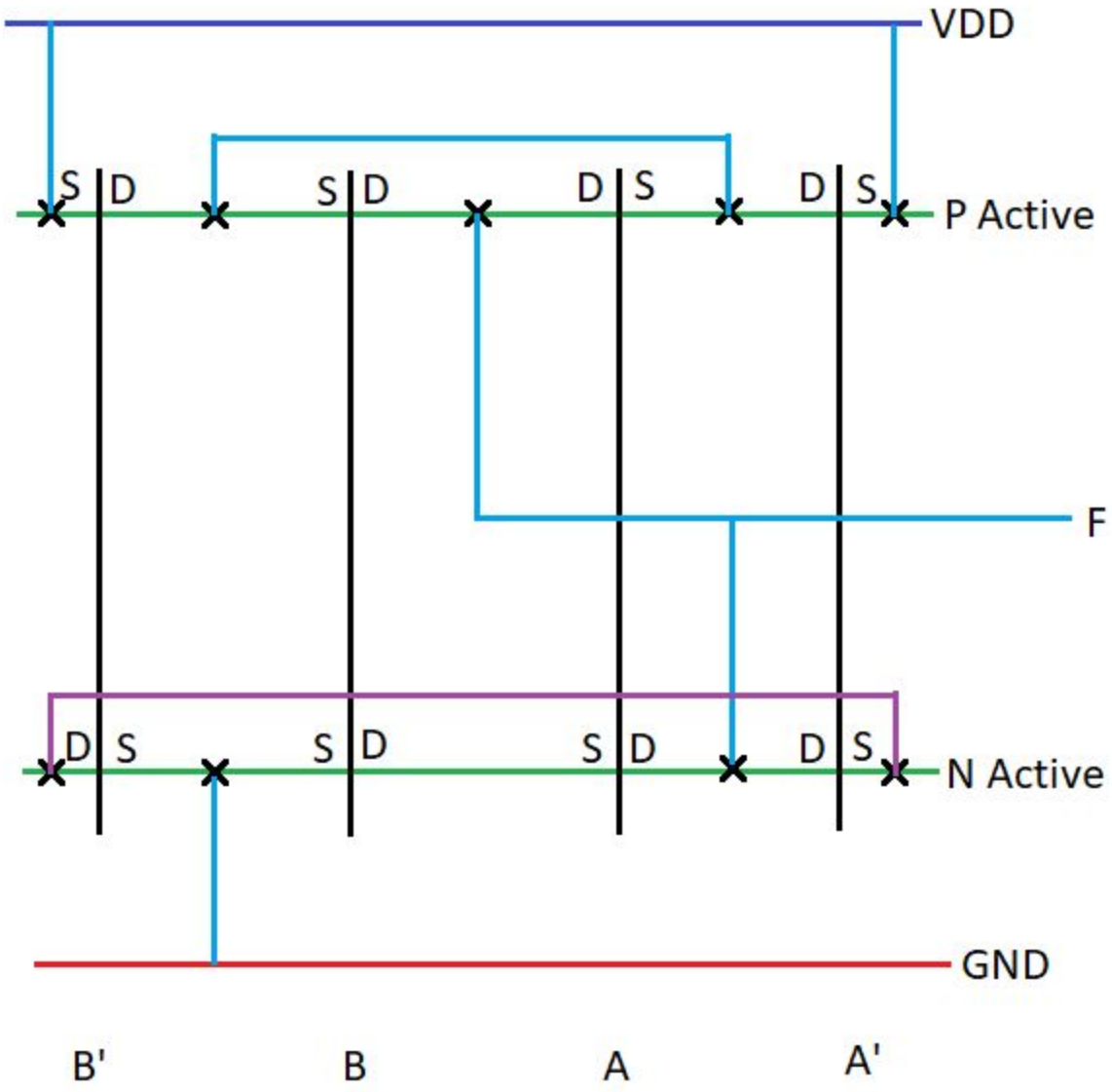


Figure 8. Stick Diagram of XOR gate

This design does require a second level of wiring, this could have been avoided by choosing a different path however due to the fact that two inverters have to be connected to the design a second level of wiring will be necessary regardless. With this XOR I go back to the Full Adder design shown in Figure 3 and placing eight of these sequentially creates the 8 Bit Ripple Adder that we desire shown in Figure 9.

Schematic

In Introduction I went over the five different designs I created through this project. I don't think it's necessary to go over again however I will add images of the Schematics that came from them to have a comparison between them. Below are the basic components NAND gate and the Half Adder I used in my initial design. It may not be visible in many of the images but the sizes of the PMOS and NMOS are all set to be 5 width and 2 length.

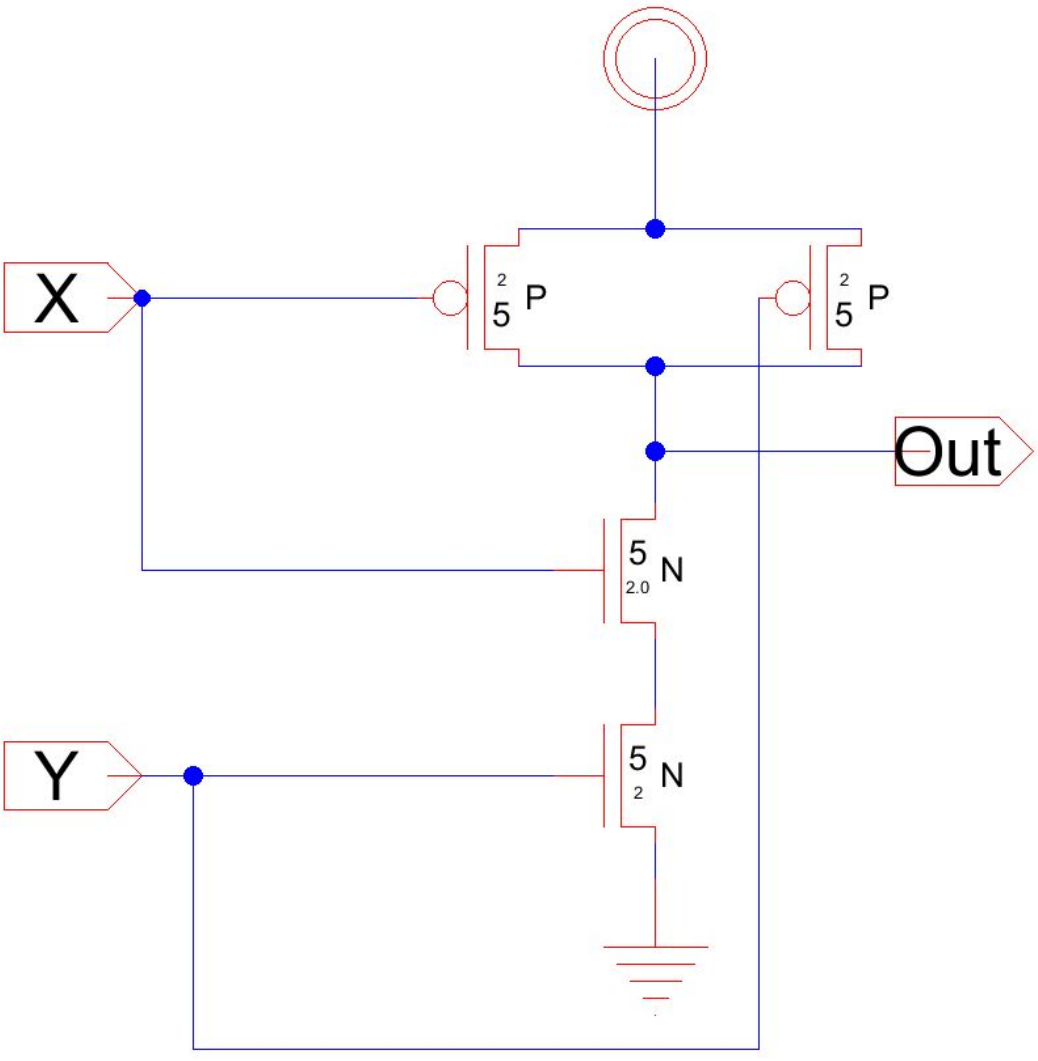


Figure 11. NAND

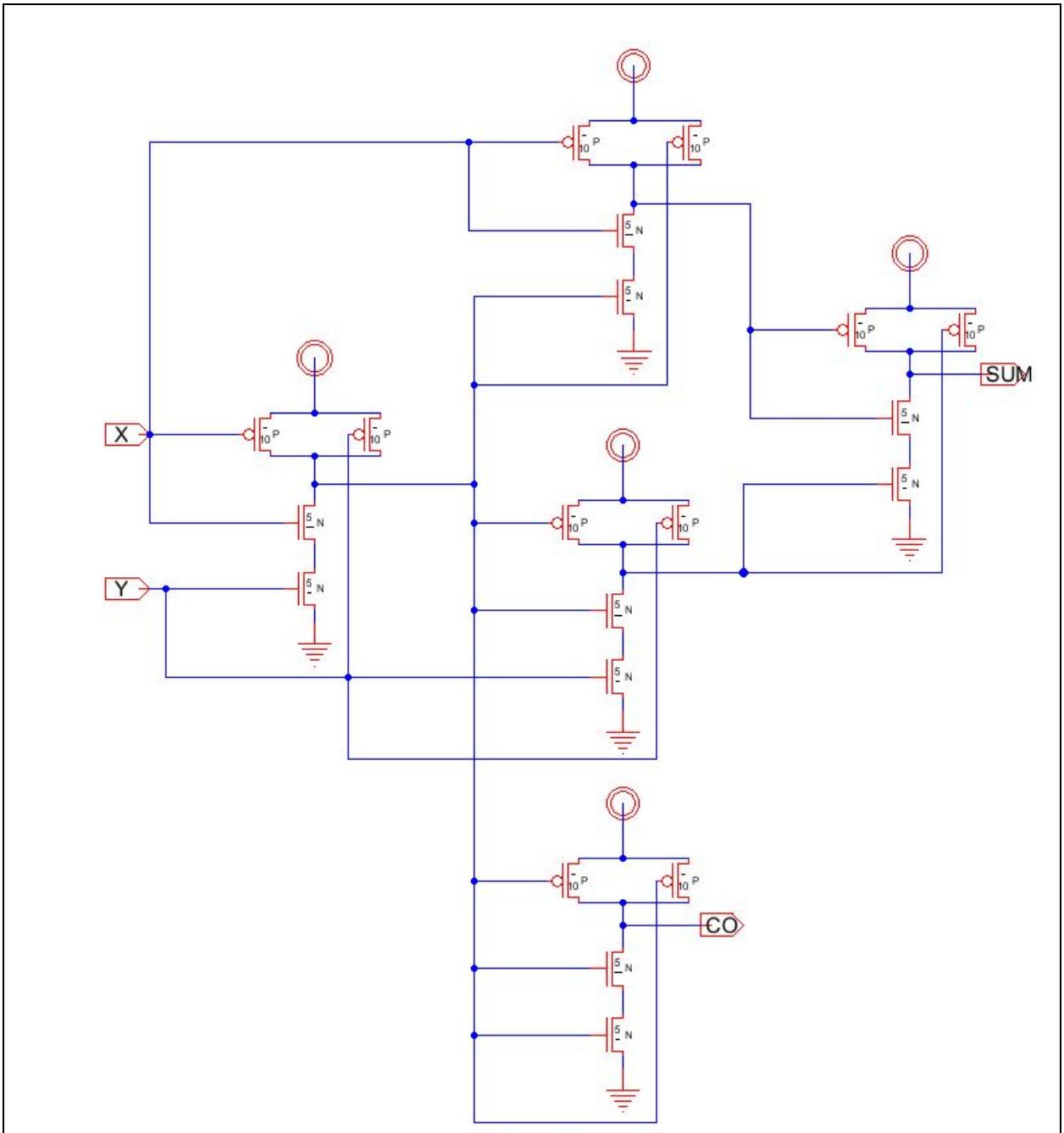


Figure 12. Half Adder

Below are the four XOR gate designs I ended up making through this project, Figure 13.1 uses four NAND gates to create a XOR gate, Figure 13.2 uses two inverters and three NAND gates, Figure 13.3 uses three inverters and a XOR component, andFigure 13.4 which is the final version uses two inverters and a XNOR.

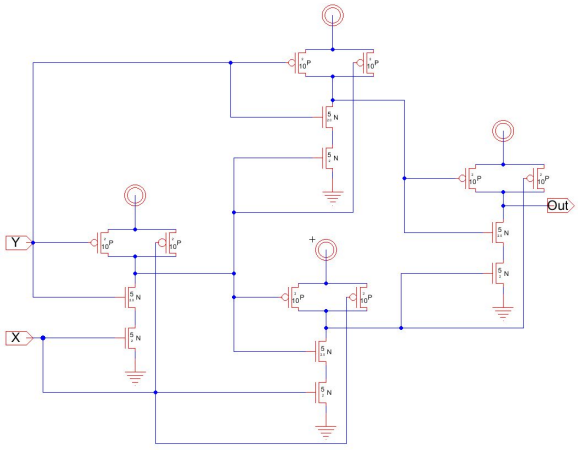


Figure 13.1 XOR Gate Version 1

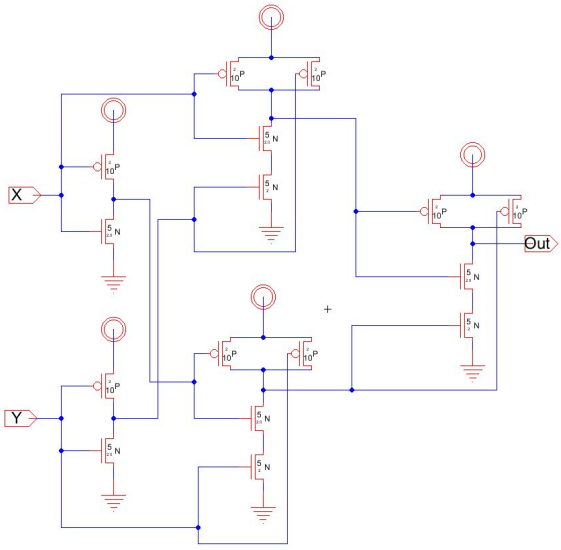


Figure 13.2 XOR Gate Version 2

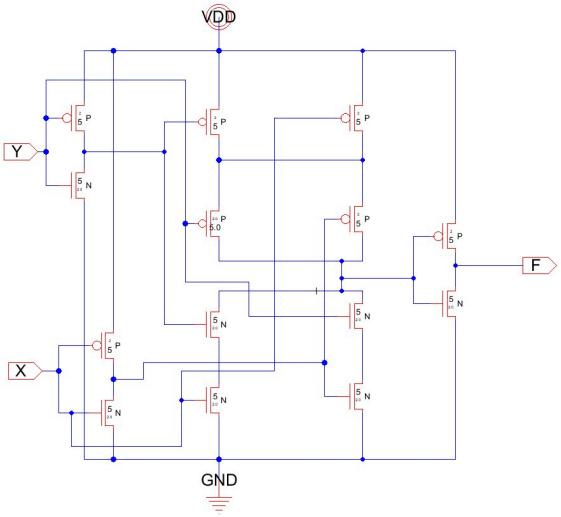


Figure 13.3 XOR Gate Version 3

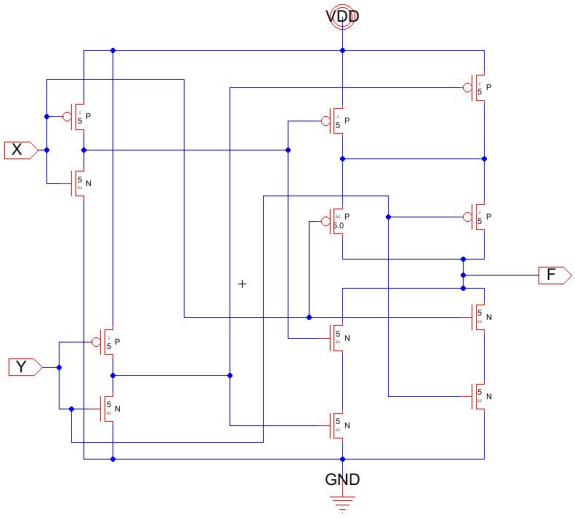


Figure 13.4 XOR Gate Final Version

Below is the close up of the final version of the XOR gate, as mentioned in the in the Introduction this uses a XNOR gate because F will always be inverted of whatever design we create. The boolean expression for it is F = ((A’ AND B’) +(A AND B’))’.

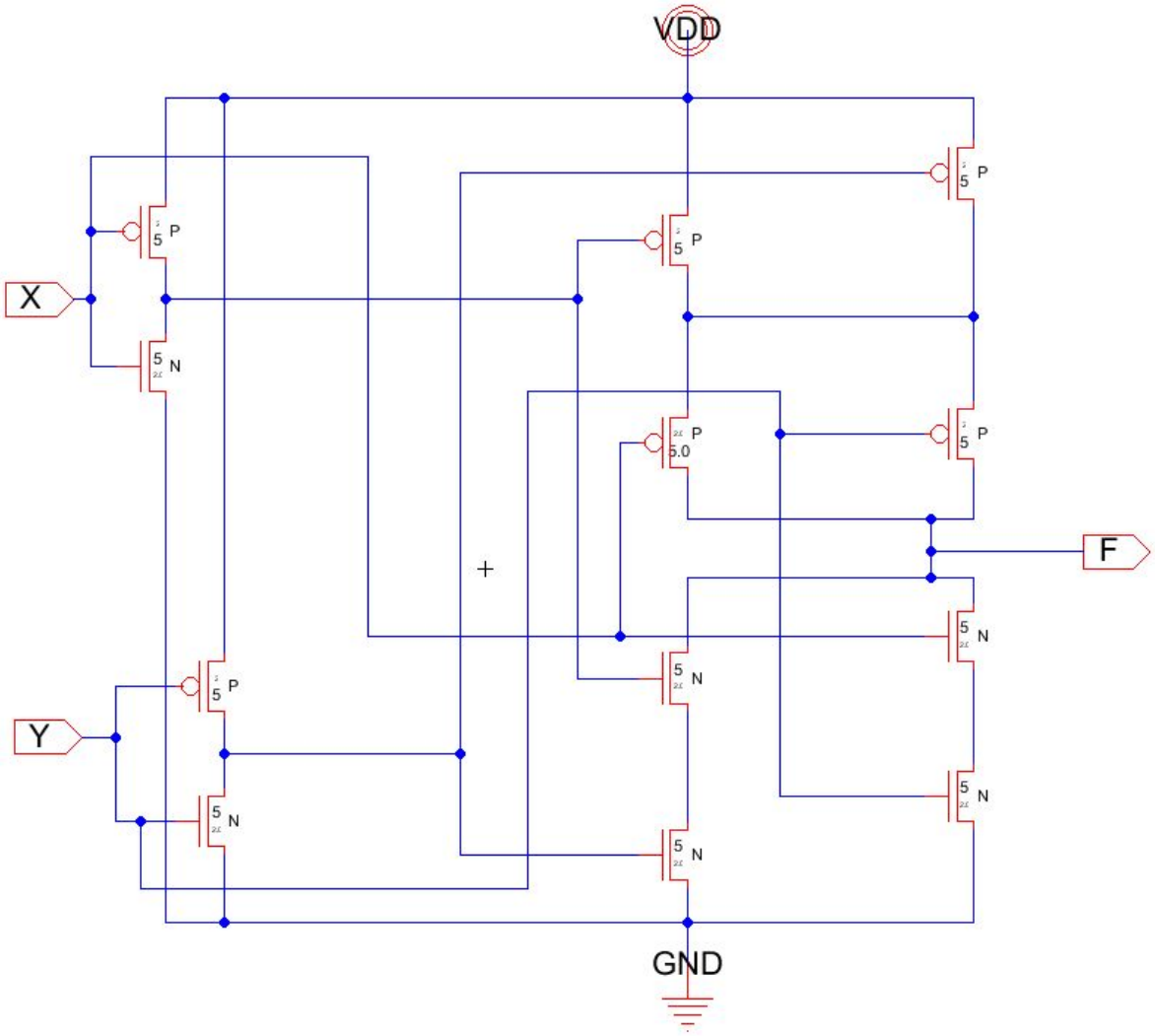


Figure 14. Close up of the Final Version of XOR Gate

Below are the five Full Adder designs that I've created for this project, Figure 15.1 shows the design using the Half Adder shown in Figure 12. Figure 15.2 shows the Full Adder as two XOR gates shown in Figure 13.1 XOR gate and the Carry Out being separated from the XOR internal logic. Figure 15.3 shows the Full Adder using 13.2 XOR gate, Figure 15.4 shows the Full Adder using 13.3 XOR gate, Figure 15.5 shows the final version Full Adder using 14 XOR gate.

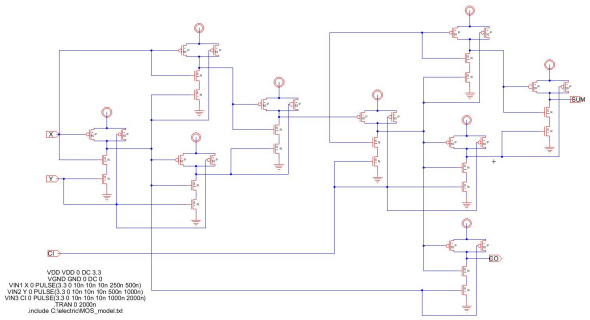


Figure 15.1 Full Adder Version 1

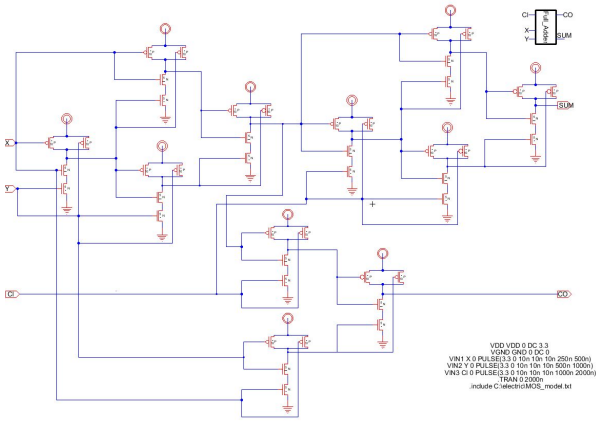


Figure 15.2 Full Adder Version 2

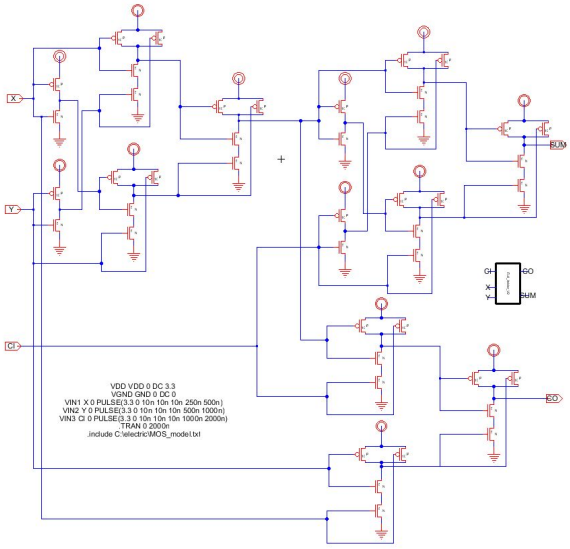


Figure 15.3 Full Adder Version 3

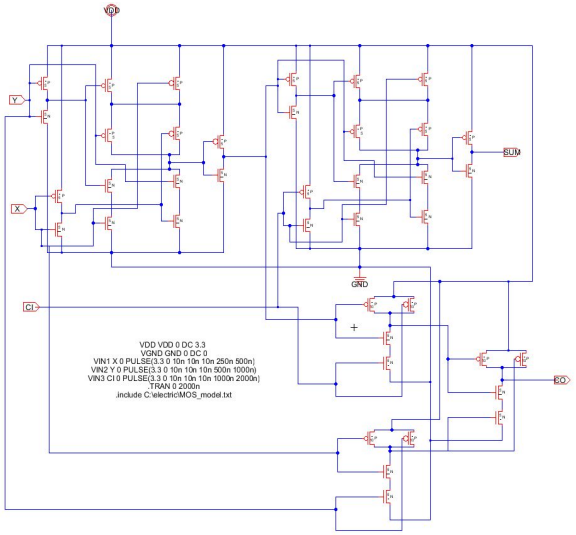


Figure 15.4 Full Adder Version 4

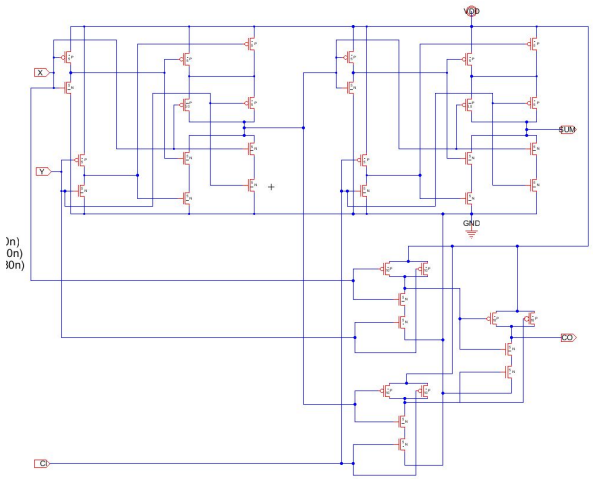


Figure 15.5 Full Adder Final Version

Belows is the close up of the final version of the Full Adder, on top of the image there are two XOR gates shown in Figure 14, the first one takes in the inputs of X and Y, the second one takes the output of the first XOR gate and Carry In (CI) and gives us the Sum. The bottom shows three NAND gates that calculate the Carry Out (CO).

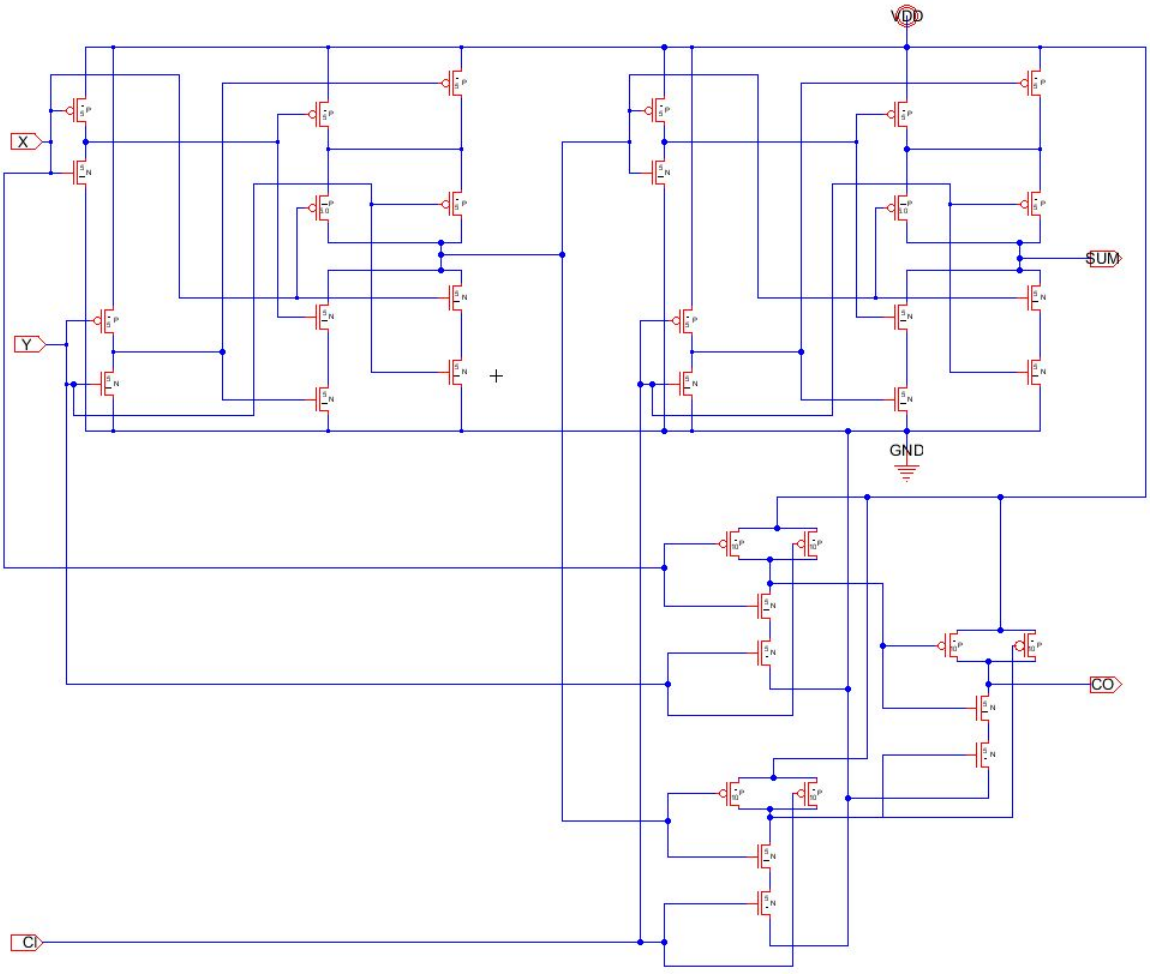


Figure 16. Close up of the final version of the Full Adder.

```
================================36================================
Checking schematic cell 'Full_Adder_V4{sch}'
   No errors found
0 errors and 0 warnings found (took 0.0 secs)
```

Figure 17. DRC

Below is the 8 Bit Ripple Carry Adder, it's just eight Full Adders connected sequentially given 16 inputs and 8 outputs. It has a Carry In input however that will be left to be zero, it is only one if we are subtracting and this component requires a bit more operations to be able to perform that.

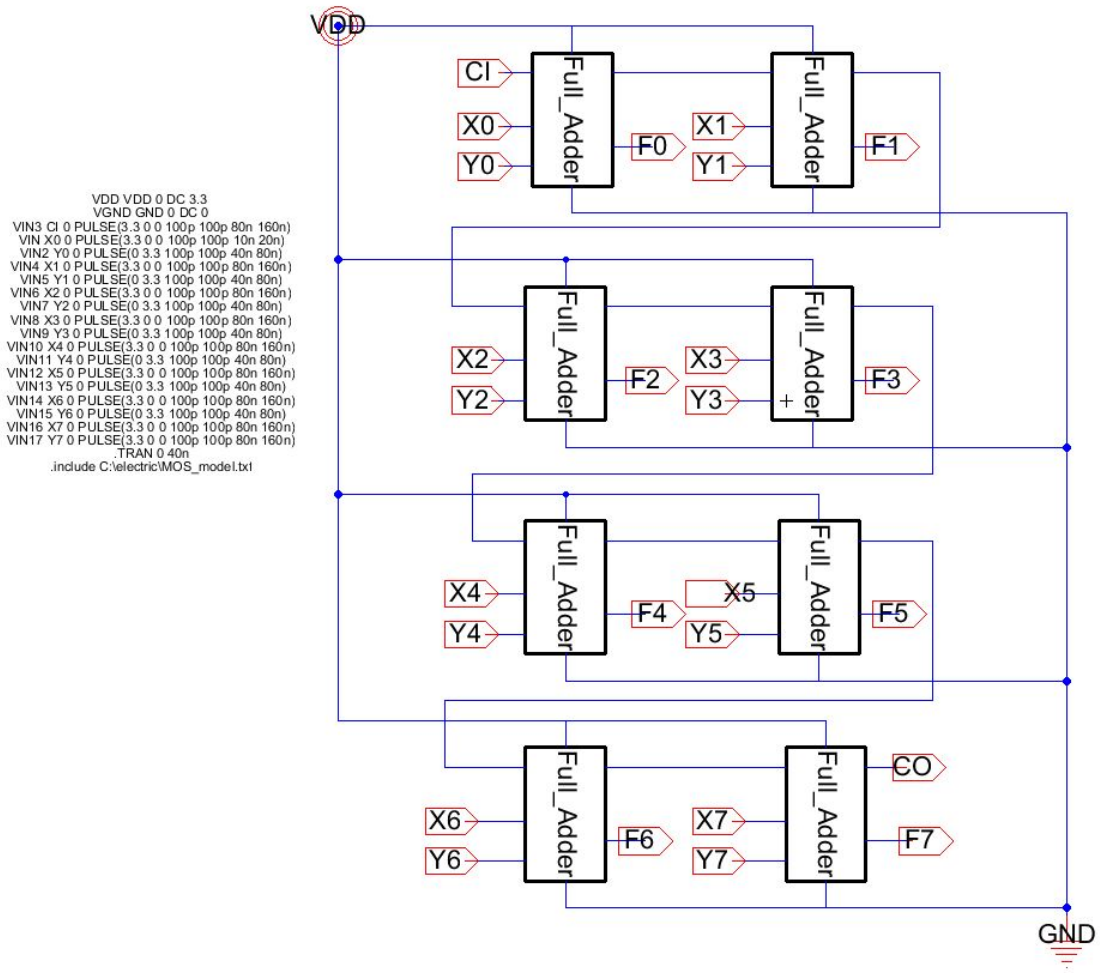


Figure 17. 8 Bit Ripple Carry Adder

Schematic LTspice

In LTspice we are trying to see the change in Rise and Fall Times from our initial data to what comes out and the propagation, how long it takes for the data to pass through the whole system. Here I have to check Sum at the end of the 8 Bit Ripple Adder. To check the result for Sum I passed in X = 0000 0001 and Y = 0111 1111 which will result in 1000 0000 that can be seen in Figure 18. From here I can measure the Rise and Fall time between X0 input and F7 output.

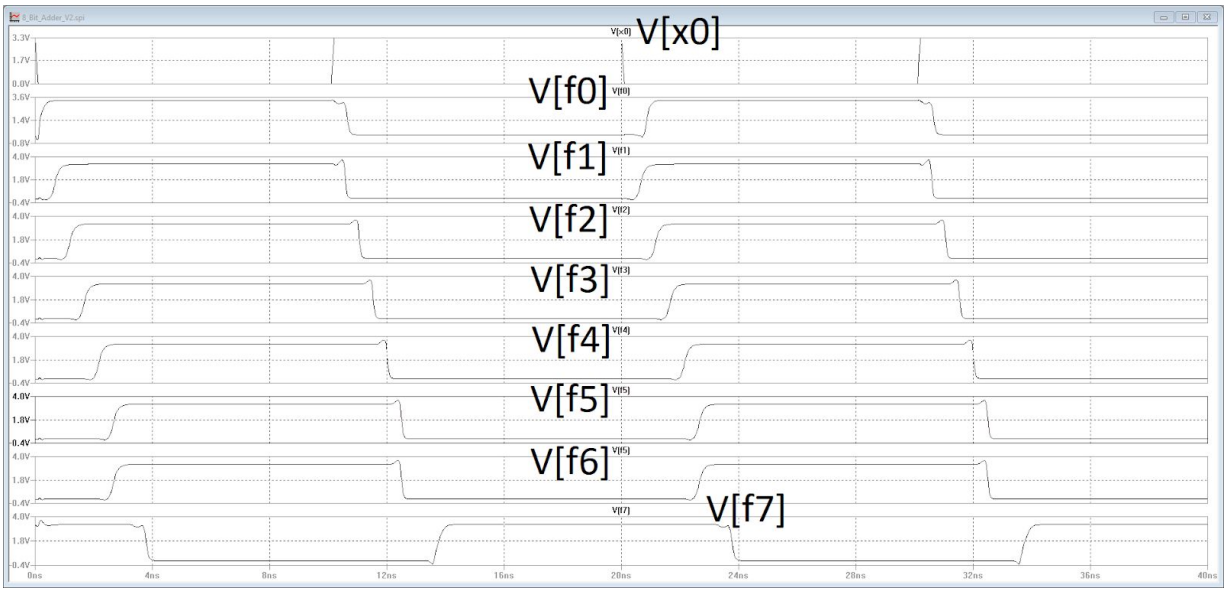


Figure 18. Movement of the Signal from input to last output

In Figure 19 shows the Rise Time of the Input V[0x] which is 0.10ns.

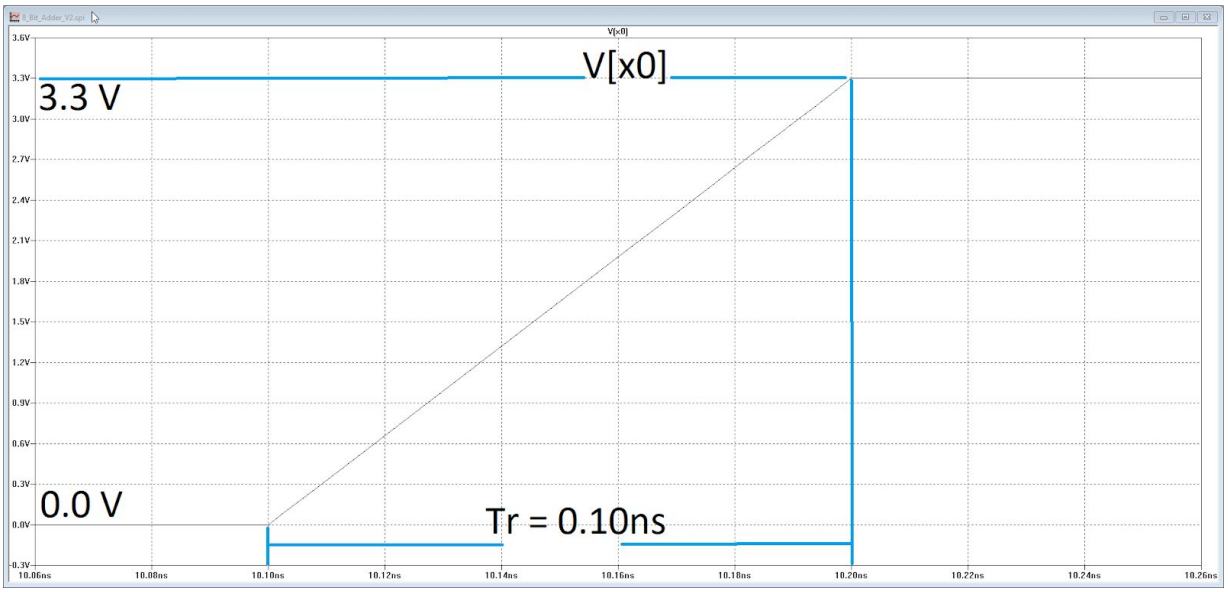


Figure 19. Rise Time X0

In Figure 20 we see the Rise Time of the F7, last output of sum in the schematic RCA both of the times fall in the middle of their time stamps so I took an average and the Rise time is approximately 0.36ns.

$$Tr = \frac{14.04ns+13.95ns}{2} - \frac{13.68ns+13.59ns}{2} = 0.36ns$$

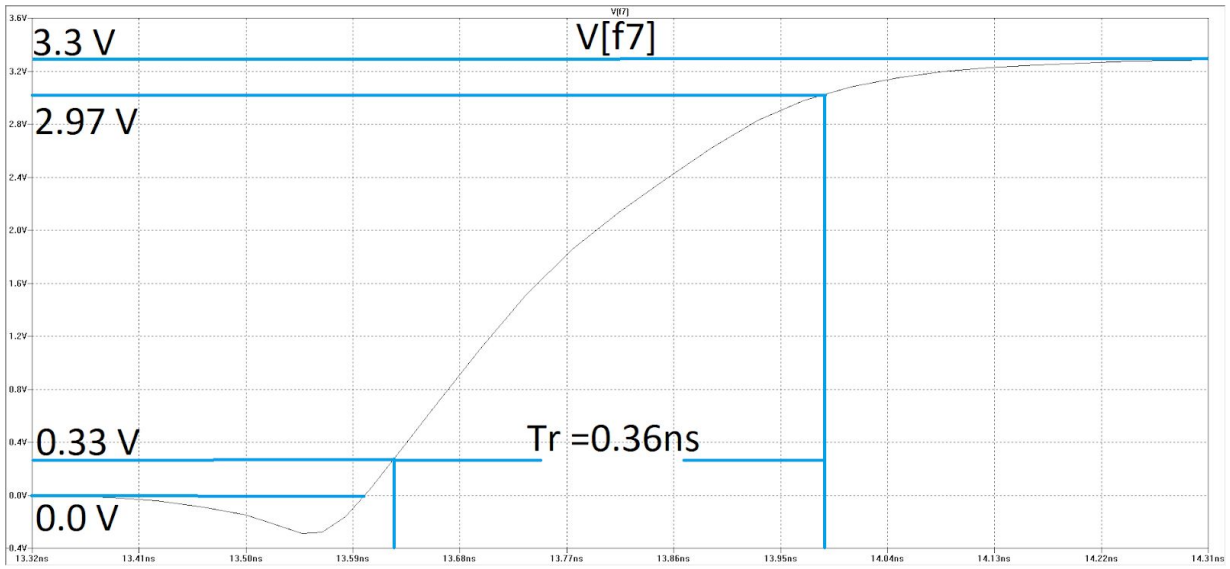


Figure 20 Rise Time F7

In Figure 21 shows the Fall Time of the Input V[0x] which is 0.10ns.

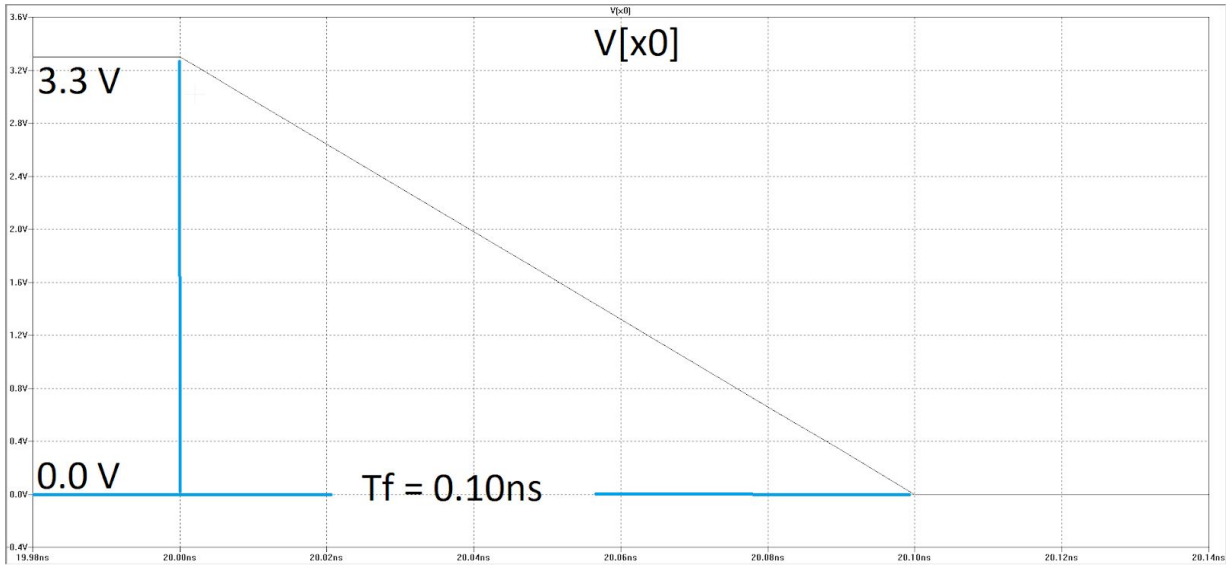


Figure 21. Fall Time X0

In Figure 22 shows the Fall Time of the output V[f7] which is 0.20ns.

$$Tf = 23.9ns - 23.7ns = 0.20ns$$

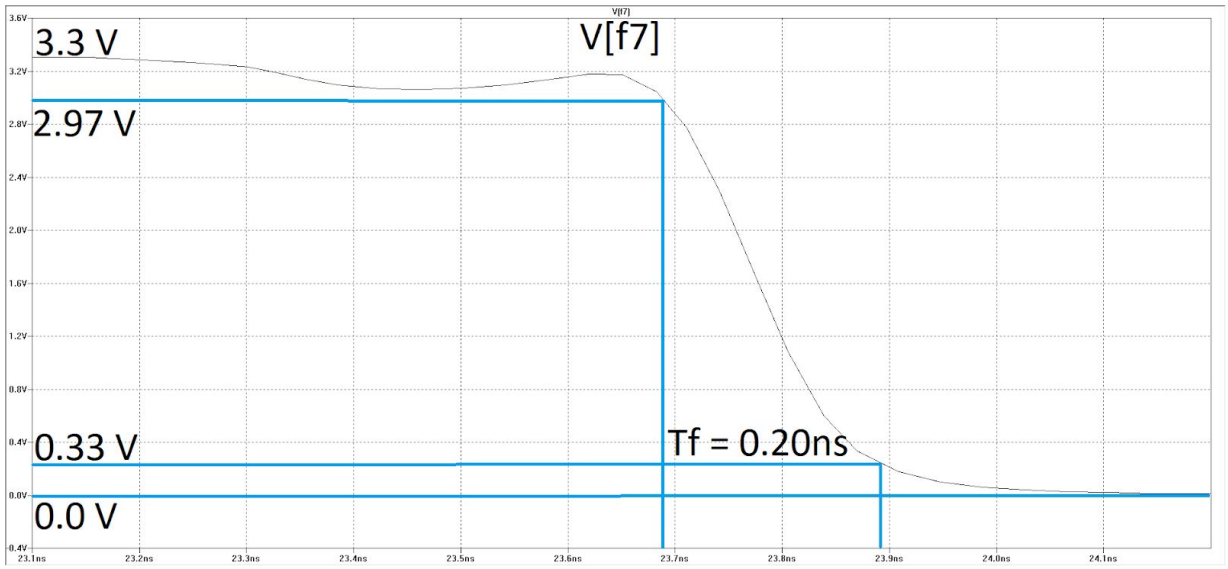


Figure 22 Fall Time F7

In Figure 23 shows the tHL of between the input V[x0] and last output V[f7] which is 3.59ns

$$tHL = 13.74ns - 10.15ns = 3.59ns$$

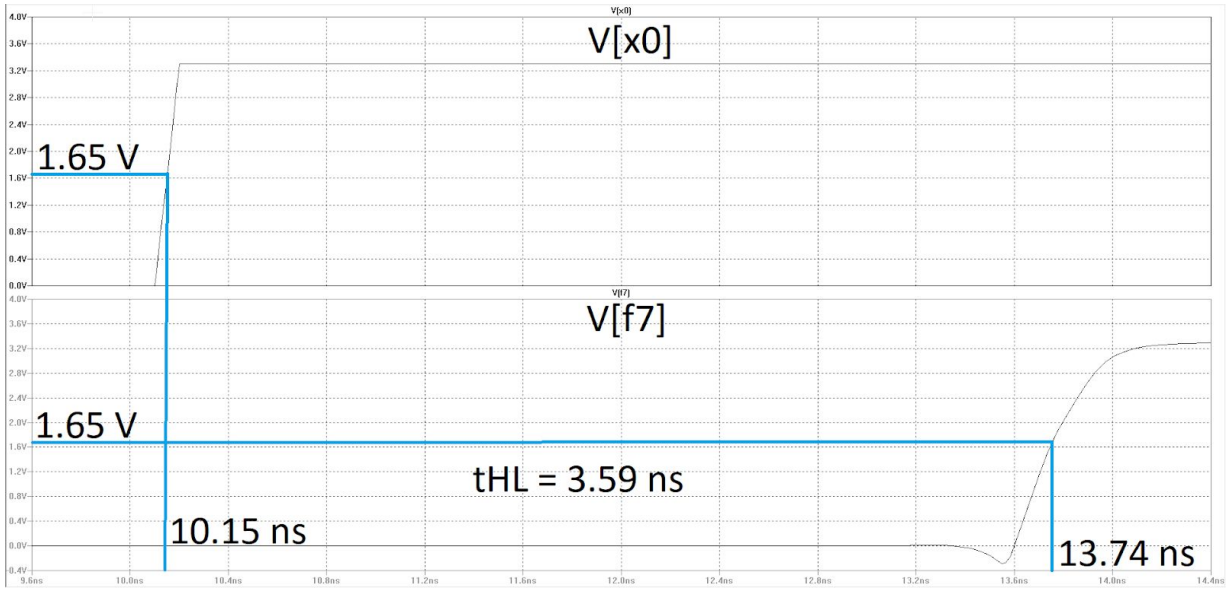


Figure 23. tHL

In Figure 24 shows the tLH of between the input V[x0] and last output V[f7] which is 3.73ns

$$tLH = 23.78ns - 20.05ns = 3.73ns$$

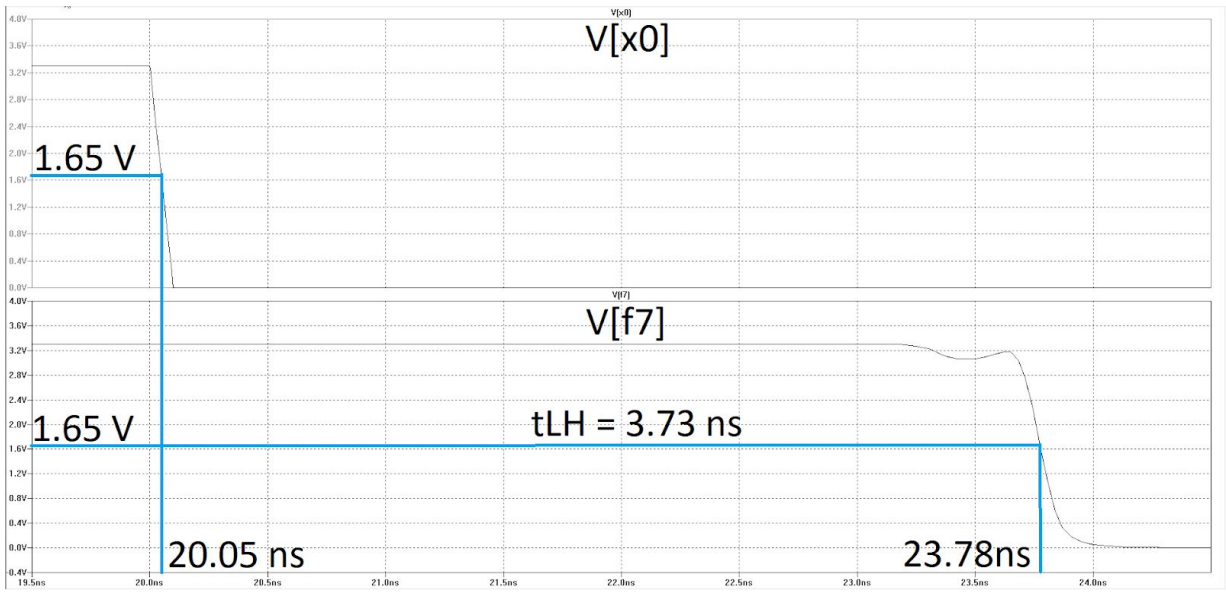


Figure 24. tLH

Taking the average of the two delays we see that the propagation is 3.66ns. Below that we see the code output that LTspice provides during simulation.

$$tp = \frac{tLH + tHL}{2} = \frac{3.59ns + 3.73ns}{2} = 3.66ns$$

```
*** SPICE deck for cell 8_Bit_Adder_V2{sch} from library Project-2
*** Created on Fri Apr 05, 2019 17:49:54
*** Last revised on Sun Apr 07, 2019 18:54:04
*** Written on Sun Apr 07, 2019 18:54:11 by Electric VLSI Design System, version 9.07
*** Layout tech: mocmos, foundry MOSIS
*** UC SPICE *** , MIN_RESIST 4.0, MIN_CAPAC 0.1FF

*** SUBCIRCUIT Project-2__Full_Adder_V4 FROM CELL Full_Adder_V4{sch}
.SUBCKT Project-2__Full_Adder_V4 CI CO gnd SUM vdd X Y
** GLOBAL gnd
** GLOBAL vdd
Mnmos@0 net@10 net@14 net@7 gnd N L=0.7U W=1.75U
Mnmos@1 net@10 X net@32 gnd N L=0.7U W=1.75U
Mnmos@2 net@7 net@15 gnd gnd N L=0.7U W=1.75U
Mnmos@3 net@32 Y gnd gnd N L=0.7U W=1.75U
Mnmos@4 net@14 X gnd gnd N L=0.7U W=1.75U
Mnmos@5 net@15 Y gnd gnd N L=0.7U W=1.75U
Mnmos@6 SUM net@98 net@91 gnd N L=0.7U W=1.75U
Mnmos@7 SUM net@10 net@116 gnd N L=0.7U W=1.75U
Mnmos@8 net@91 net@99 gnd gnd N L=0.7U W=1.75U
Mnmos@9 net@116 CI gnd gnd N L=0.7U W=1.75U
Mnmos@10 net@98 net@10 gnd gnd N L=0.7U W=1.75U
Mnmos@11 net@99 CI gnd gnd N L=0.7U W=1.75U
Mnmos@12 CO net@178 net@237 gnd N L=0.7U W=1.75U
Mnmos@13 net@237 net@183 gnd gnd N L=0.7U W=1.75U
Mnmos@14 net@178 X net@250 gnd N L=0.7U W=1.75U
Mnmos@15 net@250 Y gnd gnd N L=0.7U W=1.75U
```

```
Mnmos@16 net@183 net@10 net@233 gnd N L=0.7U W=1.75U
Mnmos@17 net@233 CI gnd gnd N L=0.7U W=1.75U
Mpmos@0 net@1 net@14 vdd vdd P L=0.7U W=1.75U
Mpmos@1 net@1 net@15 vdd vdd P L=0.7U W=1.75U
Mpmos@2 net@10 X net@1 vdd P L=0.7U W=1.75U
Mpmos@3 net@10 Y net@1 vdd P L=0.7U W=1.75U
Mpmos@4 net@14 X vdd vdd P L=0.7U W=1.75U
Mpmos@5 net@15 Y vdd vdd P L=0.7U W=1.75U
Mpmos@6 net@85 net@98 vdd vdd P L=0.7U W=1.75U
Mpmos@7 net@85 net@99 vdd vdd P L=0.7U W=1.75U
Mpmos@8 SUM net@10 net@85 vdd P L=0.7U W=1.75U
Mpmos@9 SUM CI net@85 vdd P L=0.7U W=1.75U
Mpmos@10 net@98 net@10 vdd vdd P L=0.7U W=1.75U
Mpmos@11 net@99 CI vdd vdd P L=0.7U W=1.75U
Mpmos@12 CO net@178 vdd vdd P L=0.7U W=3.5U
Mpmos@13 CO net@183 vdd vdd P L=0.7U W=3.5U
Mpmos@14 net@178 X vdd vdd P L=0.7U W=3.5U
Mpmos@15 net@178 Y vdd vdd P L=0.7U W=3.5U
Mpmos@16 net@183 net@10 vdd vdd P L=0.7U W=3.5U
Mpmos@17 net@183 CI vdd vdd P L=0.7U W=3.5U
.ENDS Project-2__Full_Adder_V4

.global gnd vdd

*** TOP LEVEL CELL: 8_Bit_Adder_V2{sch}
XFull_Add@0 CI C00 gnd F0 vdd X0 Y0 Project-2__Full_Adder_V4
XFull_Add@1 C00 C01 gnd F1 vdd X1 Y1 Project-2__Full_Adder_V4
XFull_Add@2 C01 C02 gnd F2 vdd X2 Y2 Project-2__Full_Adder_V4
XFull_Add@3 C02 C03 gnd F3 vdd X3 Y3 Project-2__Full_Adder_V4
XFull_Add@4 C03 C04 gnd F4 vdd X4 Y4 Project-2__Full_Adder_V4
XFull_Add@5 C04 C05 gnd F5 vdd X5 Y5 Project-2__Full_Adder_V4
XFull_Add@6 C05 C06 gnd F6 vdd X6 Y6 Project-2__Full_Adder_V4
XFull_Add@7 C06 CO gnd F7 vdd X7 Y7 Project-2__Full_Adder_V4

* Spice Code nodes in cell cell '8_Bit_Adder_V2{sch}'
VDD VDD 0 DC 3.3
VGND GND 0 DC 0
VIN3 CI 0 PULSE(3.3 0 0 100p 100p 80n 160n)
VIN X0 0 PULSE(3.3 0 0 100p 100p 10n 20n)
VIN2 Y0 0 PULSE(0 3.3 100p 100p 40n 80n)
VIN4 X1 0 PULSE(3.3 0 0 100p 100p 80n 160n)
VIN5 Y1 0 PULSE(0 3.3 100p 100p 40n 80n)
VIN6 X2 0 PULSE(3.3 0 0 100p 100p 80n 160n)
VIN7 Y2 0 PULSE(0 3.3 100p 100p 40n 80n)
VIN8 X3 0 PULSE(3.3 0 0 100p 100p 80n 160n)
VIN9 Y3 0 PULSE(0 3.3 100p 100p 40n 80n)
VIN10 X4 0 PULSE(3.3 0 0 100p 100p 80n 160n)
VIN11 Y4 0 PULSE(0 3.3 100p 100p 40n 80n)
VIN12 X5 0 PULSE(3.3 0 0 100p 100p 80n 160n)
VIN13 Y5 0 PULSE(0 3.3 100p 100p 40n 80n)
VIN14 X6 0 PULSE(3.3 0 0 100p 100p 80n 160n)
VIN15 Y6 0 PULSE(0 3.3 100p 100p 40n 80n)
VIN16 X7 0 PULSE(3.3 0 0 100p 100p 80n 160n)
VIN17 Y7 0 PULSE(3.3 0 0 100p 100p 80n 160n)
.TRAN 0 40n
.include C:\electric\MOS_model.txt
.END
```

Figure 25. LTspice Code

Schematic IRSIM

To check if out system works correctly we are going to run it through IRSIM and see if the logic of it adds our numbers correctly, below are the four examples we are going to run through the 8 Bit Ripple Adder. Below we see the additions that we want to perform and what their correspond binary representations are.

(1) -32+107=75,

-32	1110 0000
107	0110 1011
75	0100 1011

(2) -28+16= -12

-28	1110 0100
16	0001 0000
-12	1111 0100

(3) 33+70= 103

33	0010 0001
70	0100 0110
103	0110 0111

(4) –71+(-10) = -81

-71	1011 1001
-10	1111 0110
-81	1010 1111

Below we see four images that show the additions of 2 bits at a time, Carry In is set to be always 0 and we have four columns, the first one is example (1) second column is example (2) and so on. The way to read is is X0 + Y0 = F0 along the column. There is a bit of a delay on the output of F0, I don't know why I tried to fix IRSIM but it always had propagation on it. In the descriptions I have the expected outcome such as (4) 1010 1111 and then current outcome shown as xxxx xx11 changing the xx to number of those bits so it's easier to track the updates.

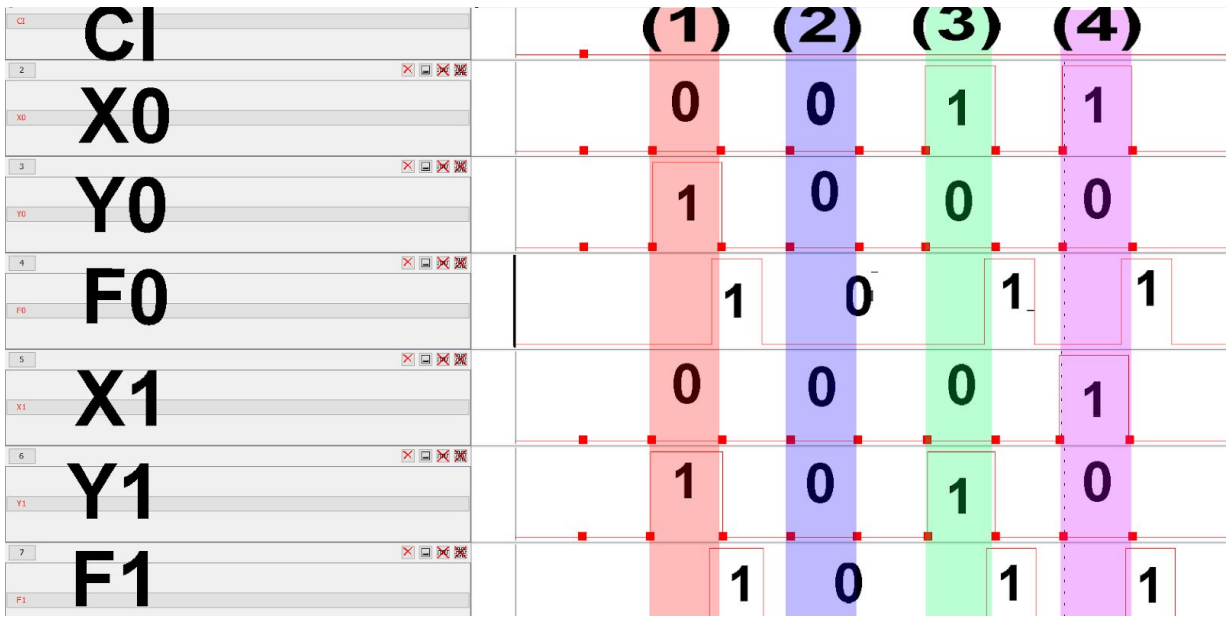


Figure 26. 0th and 1st Inputs and Output
(1) F = 0100 1011 - xxxx xx11(2) F =1111 0100 - xxxx xx00 (3) F = 0110 0111 - xxxx xx11 (4) F = 1010 1111 - xxxx xx11

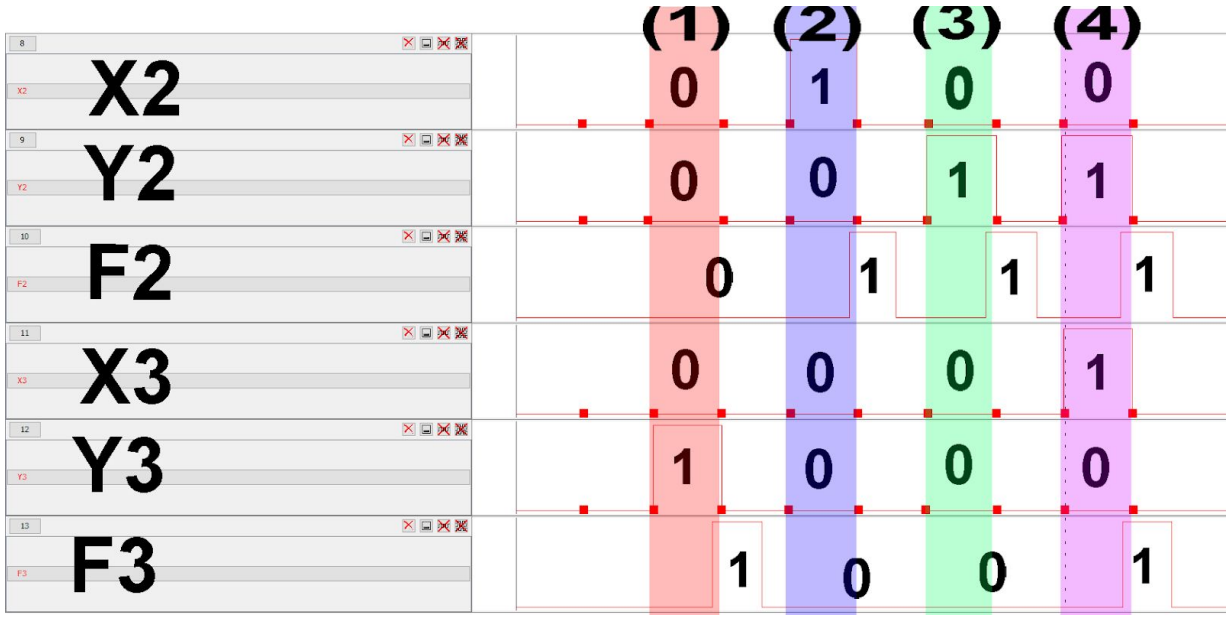


Figure 27. 2nd and 3rd Inputs and Output
(1) F = 0100 1011 - xxxx 1011(2) F =1111 0100 - xxxx 0100 (3) F = 0110 0111 - xxxx 0111 (4) F = 1010 1111 - xxxx 1111

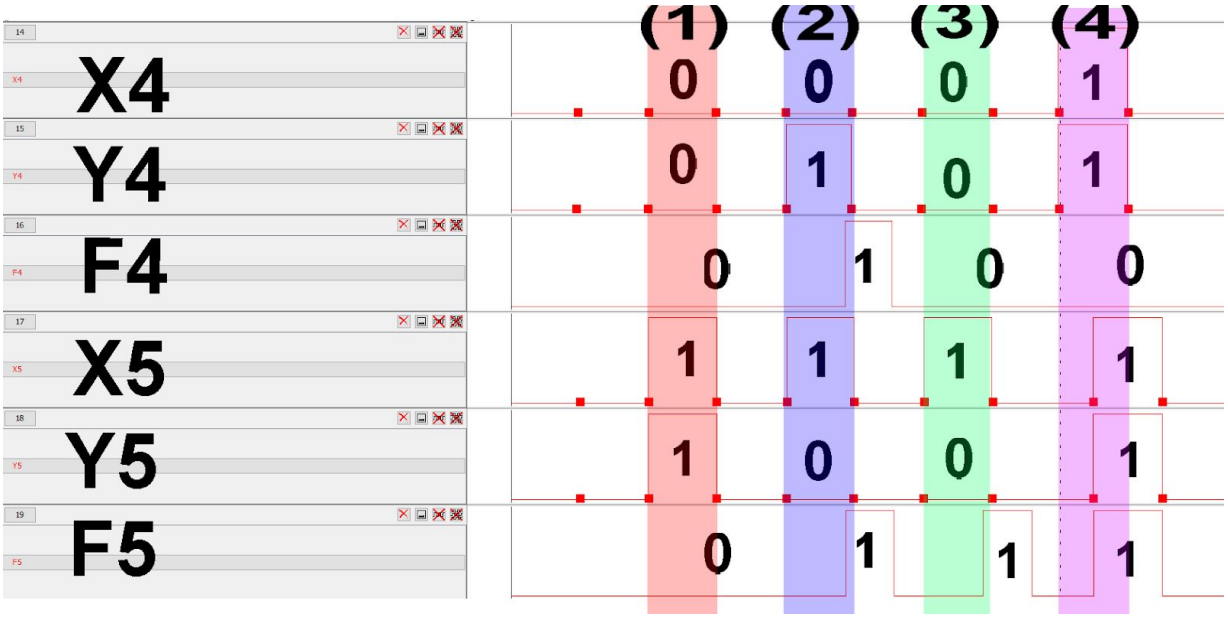


Figure 28. 4th and 5th Inputs and Output
(1) F = 0100 1011 - xx00 1011(2) F =1111 0100 - xx11 0100 (3) F = 0110 0111 - xx10 0111
(4) F = 1010 1111 - xx10 1111

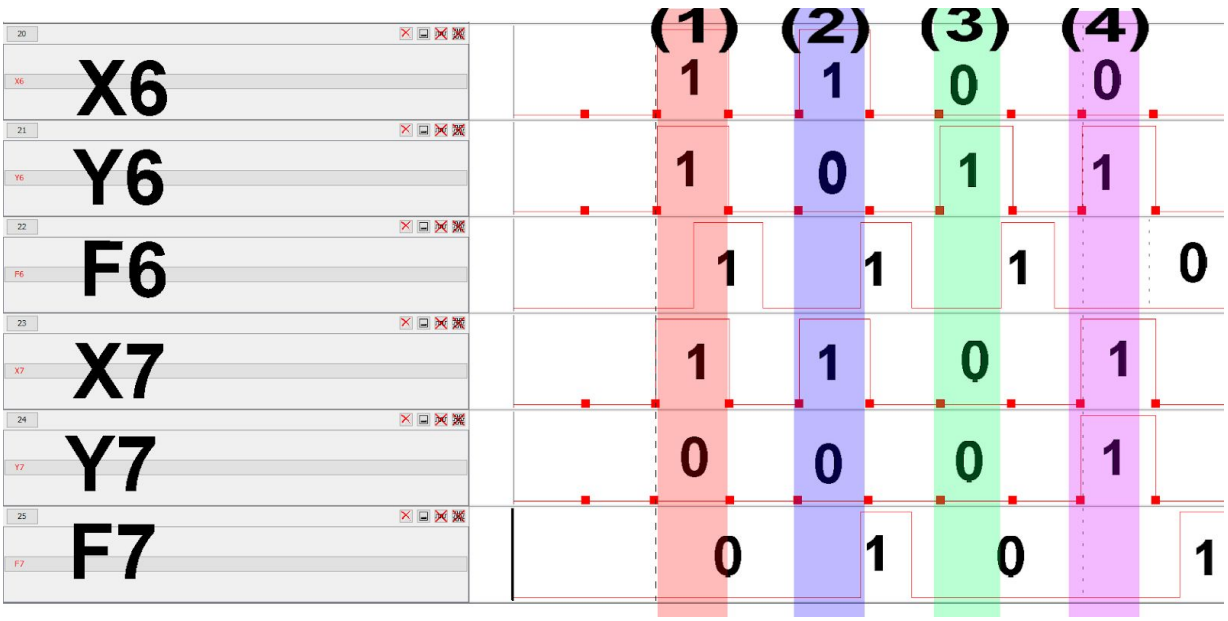


Figure 29. 6th and 7th Inputs and Output
(1) F = 0100 1011 - 0100 1011(2) F =1111 0100 - 1111 0100 (3) F = 0110 0111 - 0110 0111
(4) F = 1010 1111 - 1010 1111

As we can see all the additions match up and the RCA works as intended.

Layout

When it comes to the Layout I only implemented two of the XOR gates shown in the schematics and only built the adder using the final XOR gate design. Below are the two layouts I did for the XOR gate, Figure has the XOR implementation which requires a inverter at the end of the XOR gate. Figure shows the XNOR layout that made the layout smaller.

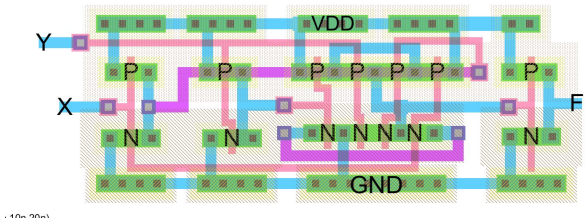


Figure 30.1 XOR gate version 3

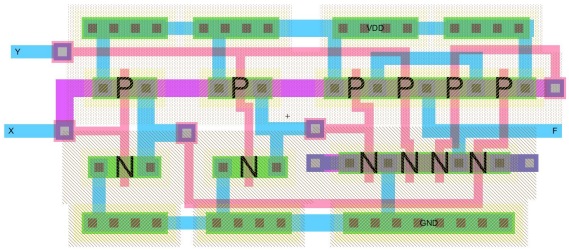


Figure 30.2 XOR gate final version

Below is the Stick Diagram used to create the final version of the XOR gate and the XOR gate before I connected inverters to it as we can see they are the same.

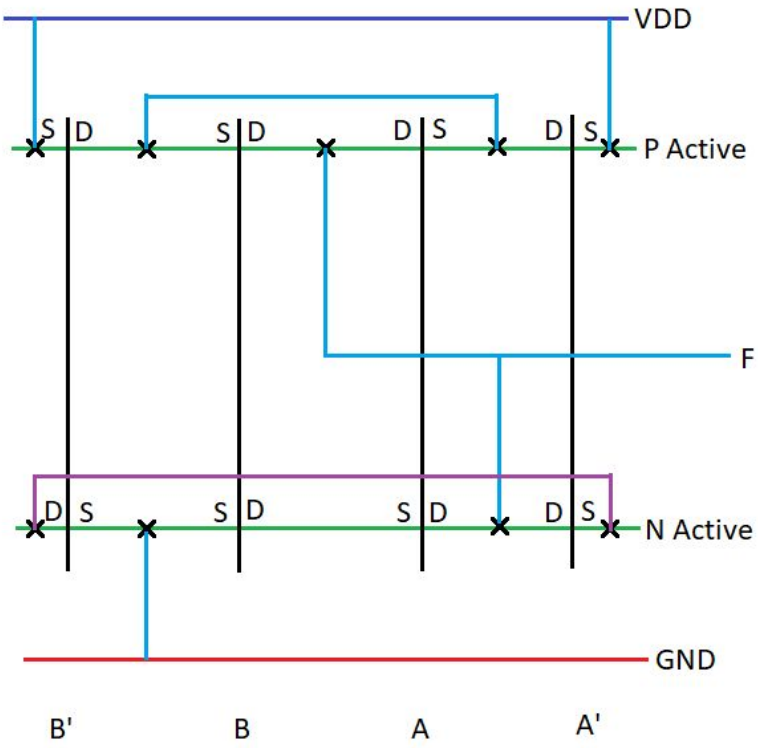


Figure 31.1 Stick Diagram of the XOR gate

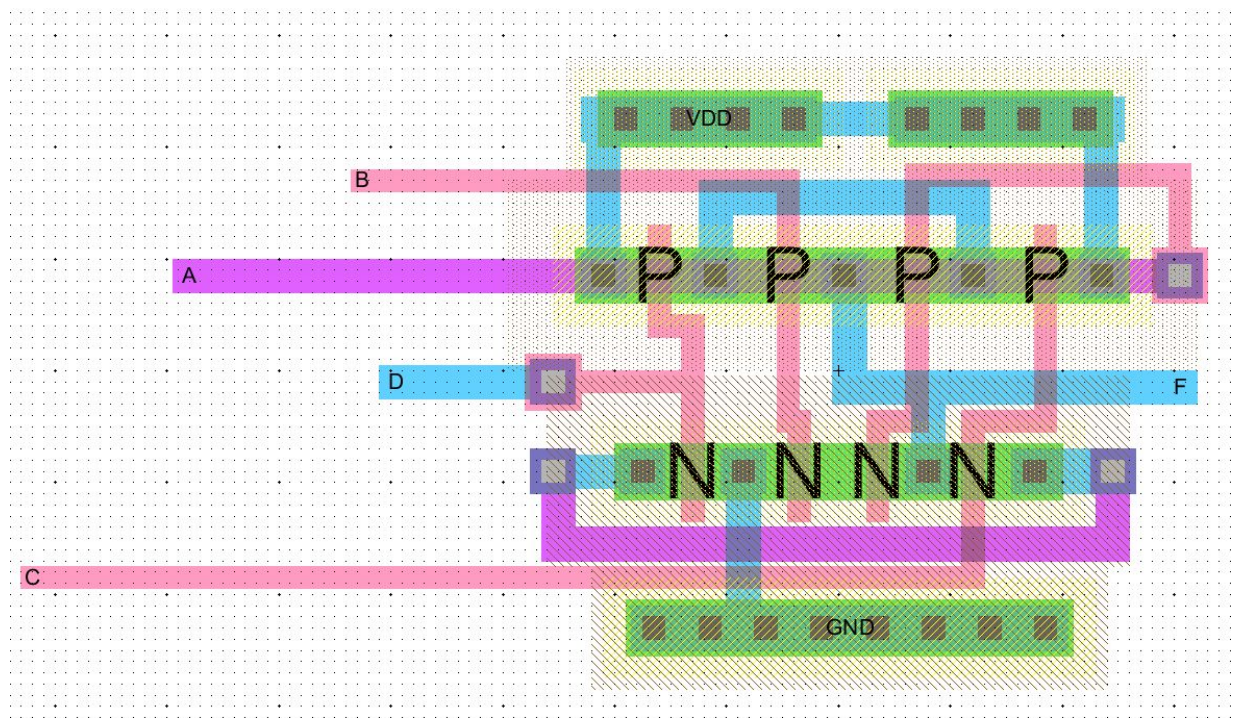


Figure 31.2 XOR gate based off of the Stick Diagram.

Below is the final version of the XOR gate it starts off with two inverters the first one being the X and the second one being Y, and as planned the XOR takes in the B' or in this case Y' in directly.

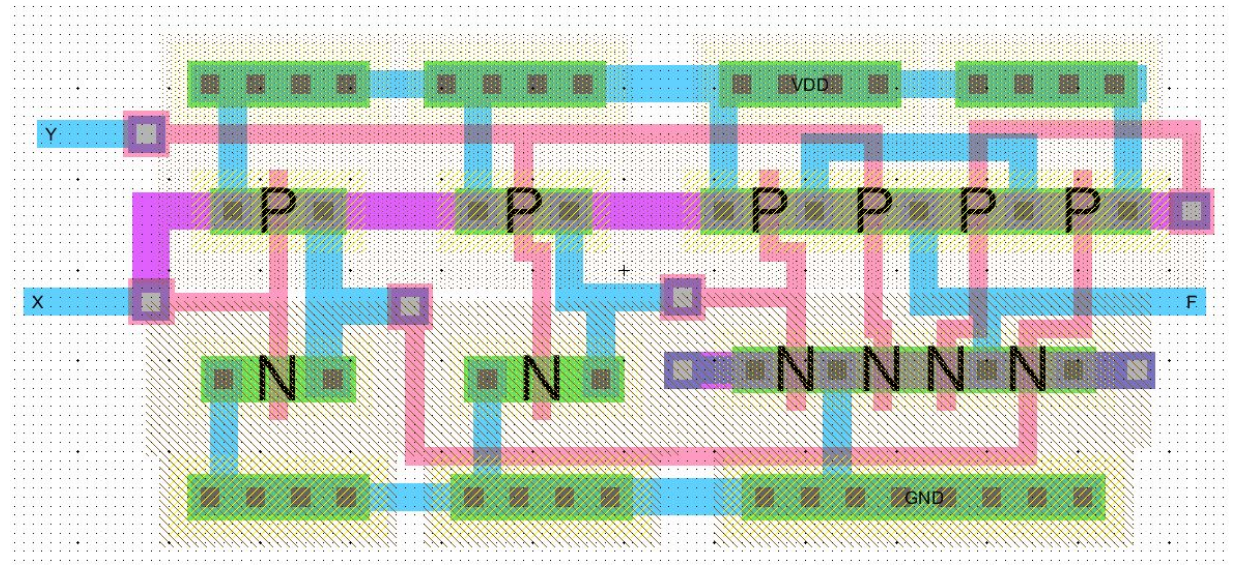


Figure 32. Close up of the final version of the XOR gate

Below is the Full Adder layout and close ups on individual parts of it. As Figure shows the layout start with a XOR gate into which I input X and Y, the next close up shows the three NAND gates necessary to get the Carry Out, they take X, Y, Carry In and output of the first XOR gate as inputs and the final close up shows the second XOR gate which takes in the input of Carry In and output of the first XOR gate.

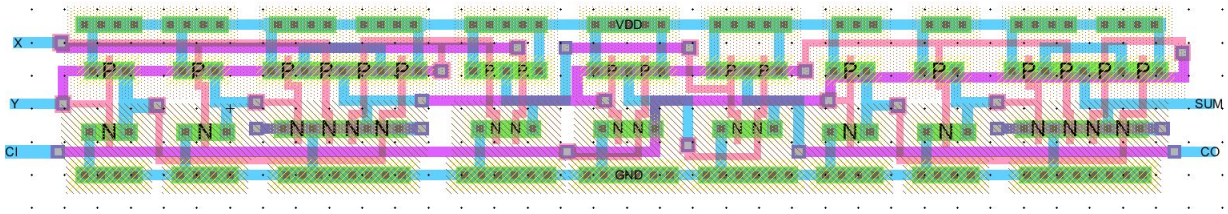


Figure 33. Full Adder Layout

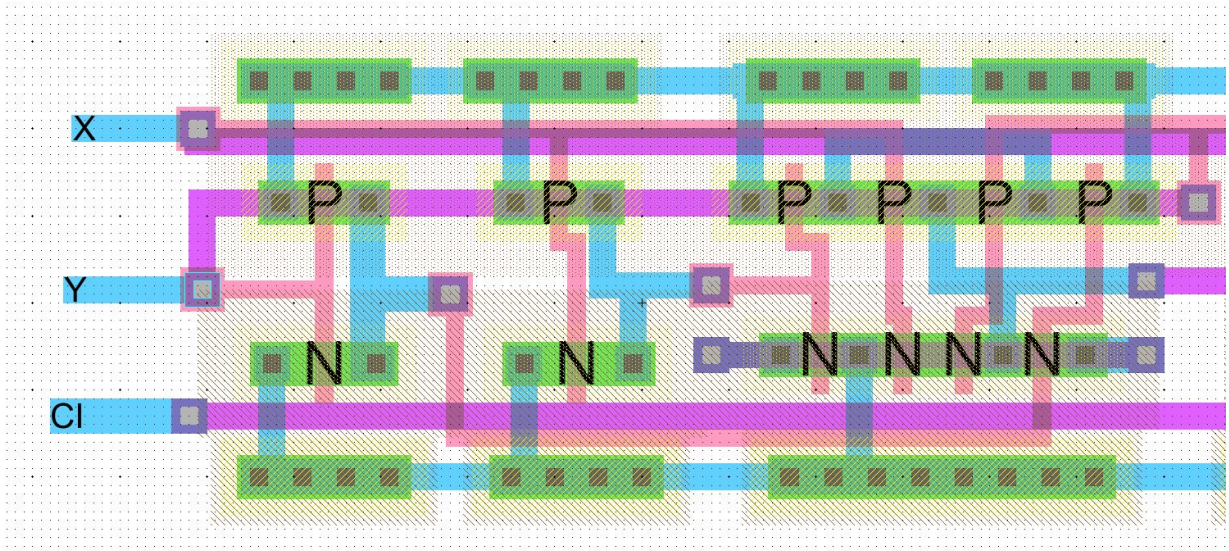


Figure 34. Close up of the first XOR gate and input

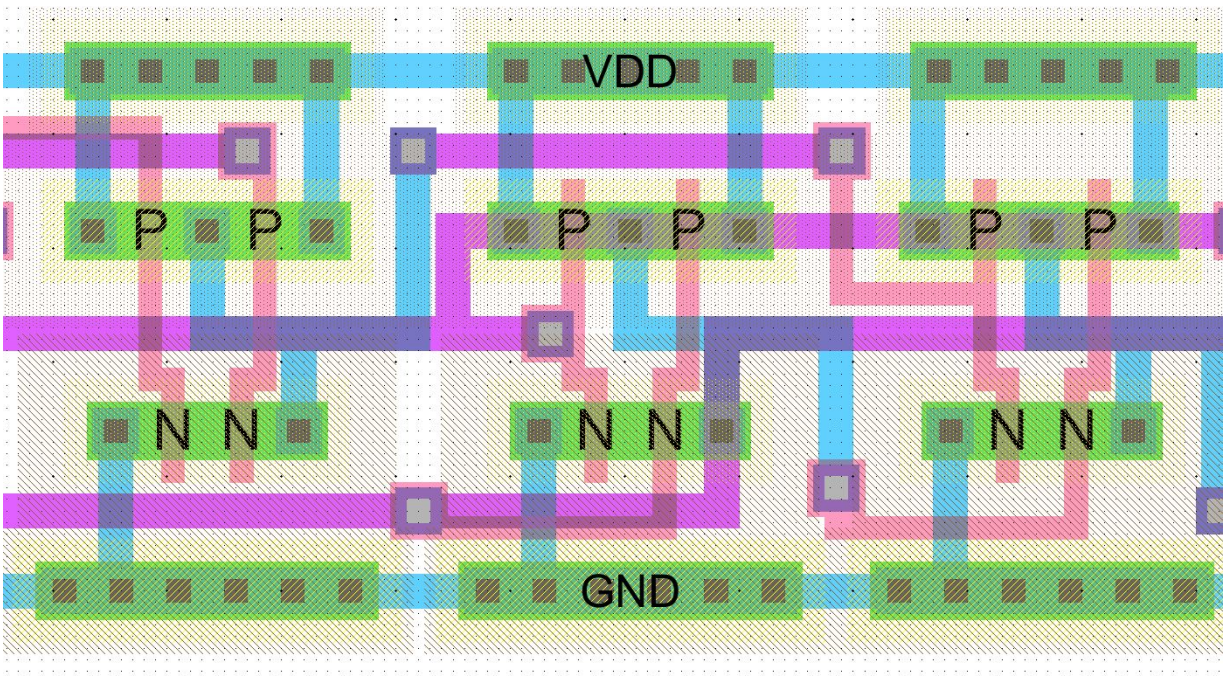


Figure 35. Close up of the 3 NAND gates

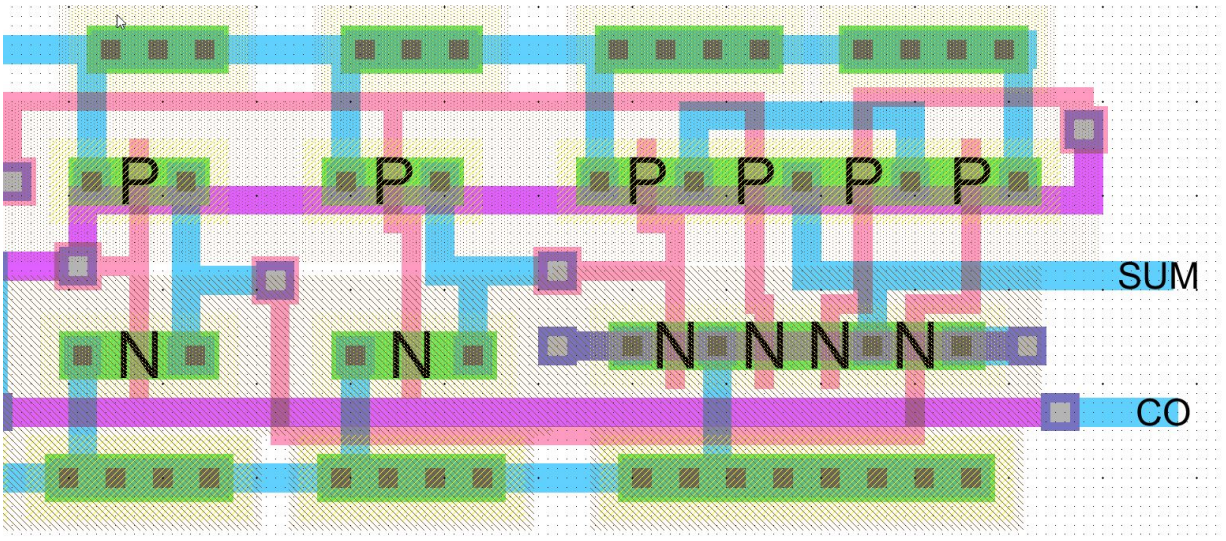


Figure 36. Close up of the second XOR gate and output

```
================================57================================
Running DRC with area bit on, extension bit on, Mosis bit
Checking again hierarchy .... (0.001 secs)
Found 60 networks
0 errors and 0 warnings found (took 0.005 secs)
================================58================================
Checking Wells and Substrates in 'Project-2:Full_Adder_V4{lay}' ...
   Geometry collection found 163 well pieces, took 0.01 secs
   Geometry analysis used 4 threads and took 0.0 secs
NetValues propagation took 0.0 secs
Checking short circuits in 20 well contacts
   Additional analysis took 0.0 secs
No Well errors found (took 0.01 secs)
```

Figure 37. DRC and Well Check

Below we see the 8 Bit Ripple Carry Adder made in layout, I made the VDD and GND really large so that it connects to everything easily. Below that are the sequential close ups of the RCA.

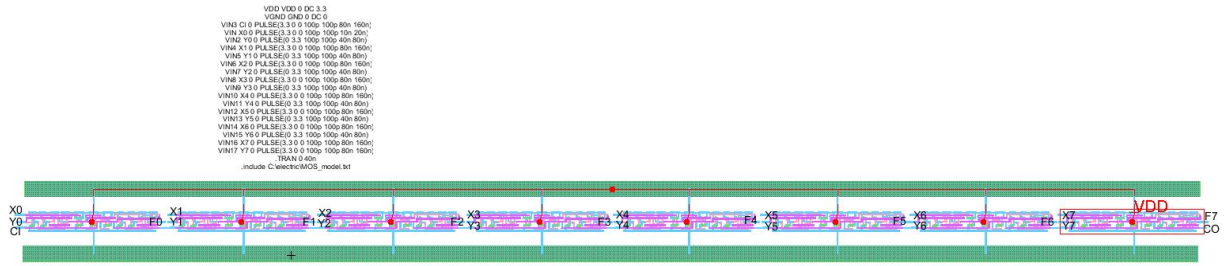


Figure 38. 8 Bit Ripple Adder Layout

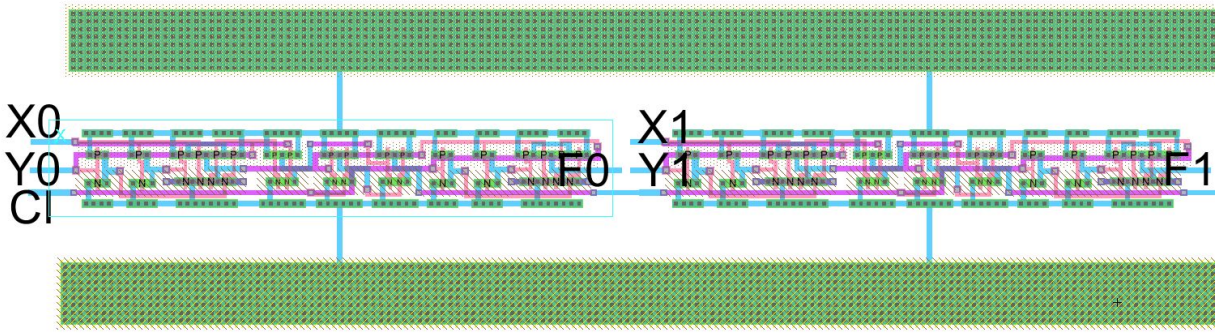


Figure 39.1 Close Up #1

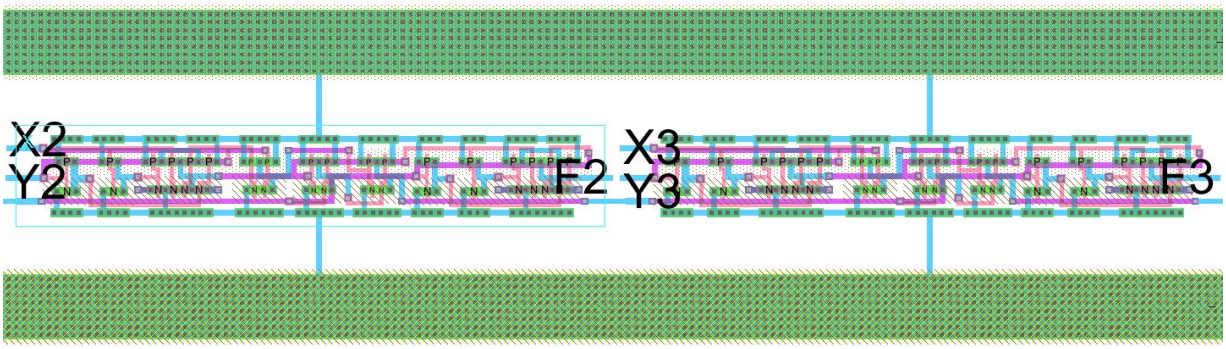


Figure 39.2 Close Up #2

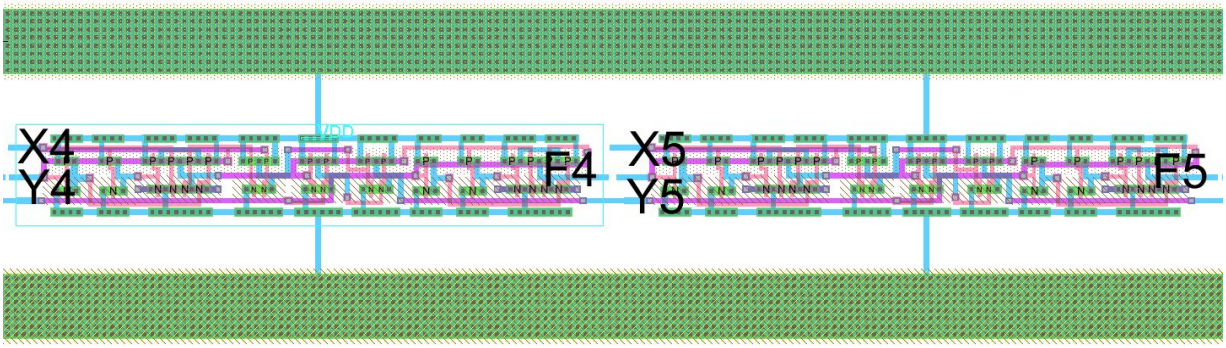


Figure 39.3 Close Up #3

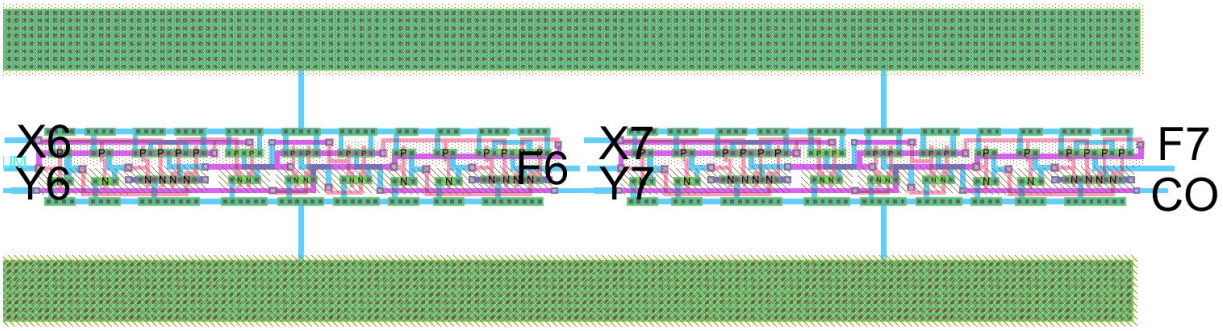


Figure 39.4 Close Up #4

Layout LTspice

For the LTspice for layout I take the same approach I did for the schematic I send in the signal of X = 0000 0001 and Y = 0111 1111 and add them to see the difference between input V[x0] and V[f7]. As figure 40 shows the propagation is much longer this time around.

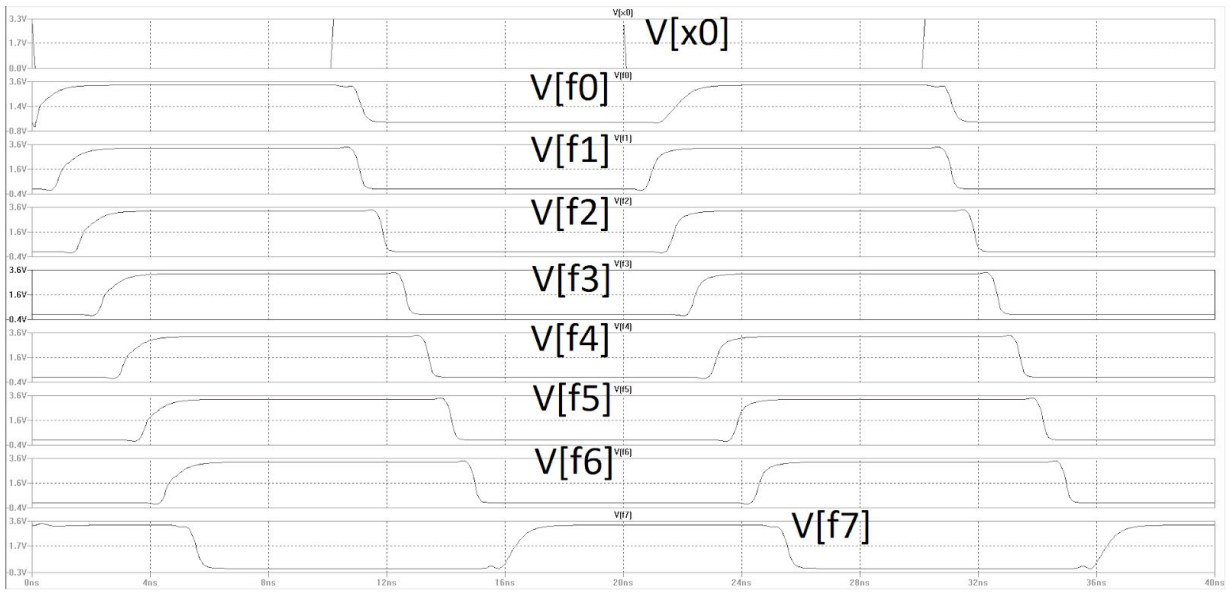


Figure 40 Movement of the Signal from input to last output

The Rise and Fall times for the input are the same as shown in 19 and 21 meaning Rise Time and Fall Time of the Input V[0x] which is 0.10ns.

In Figure 41 shows the Fall Time of the output V[f7] which is 1.0ns.

$$Tr = 17.0ns - 16.0ns = 1.0ns$$

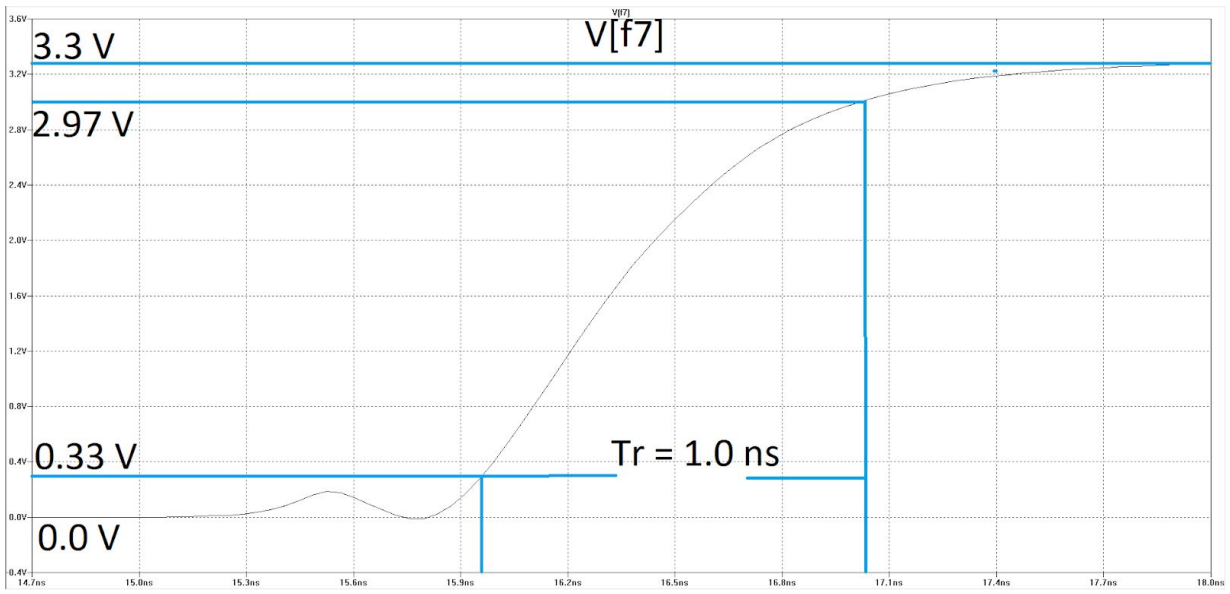


Figure 41 Rise Time of V[f7] layout

In Figure 22 shows the Fall Time of the output V[f7] which is 0.50ns.

$$Tf = 25.8ns - 25.3ns = 0.50ns$$

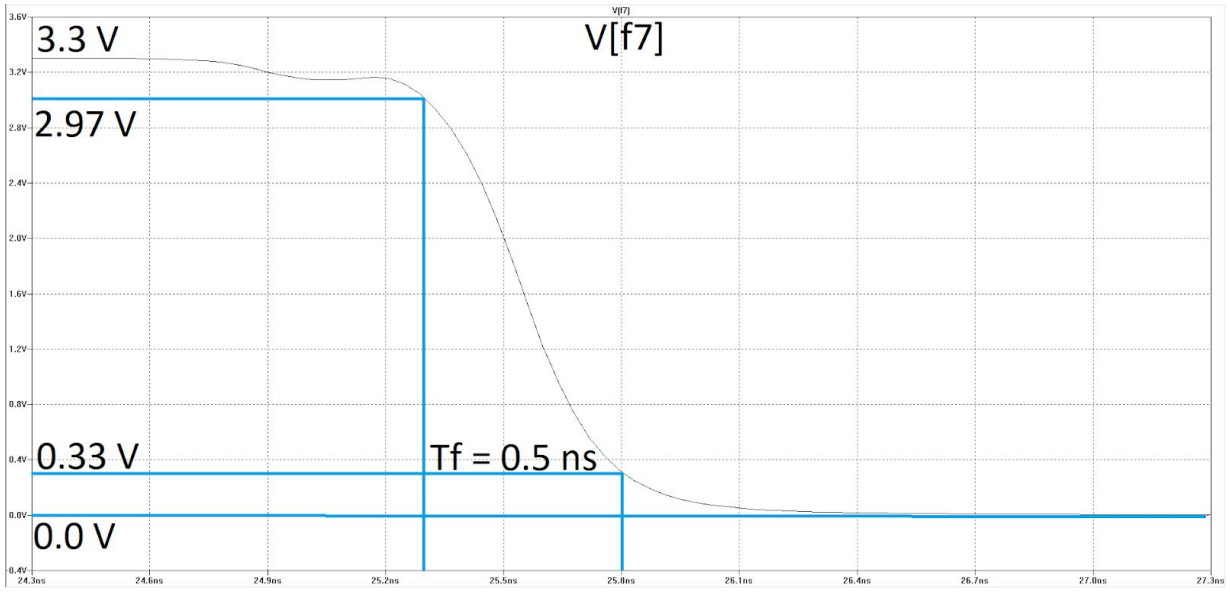


Figure 42 Fall Time of V[f7] layout

In Figure 43 shows the tHL of between the input V[x0] and last output V[f7] which is 6.15ns

$$tHL = 16.30ns - 10.15ns = 6.15ns$$

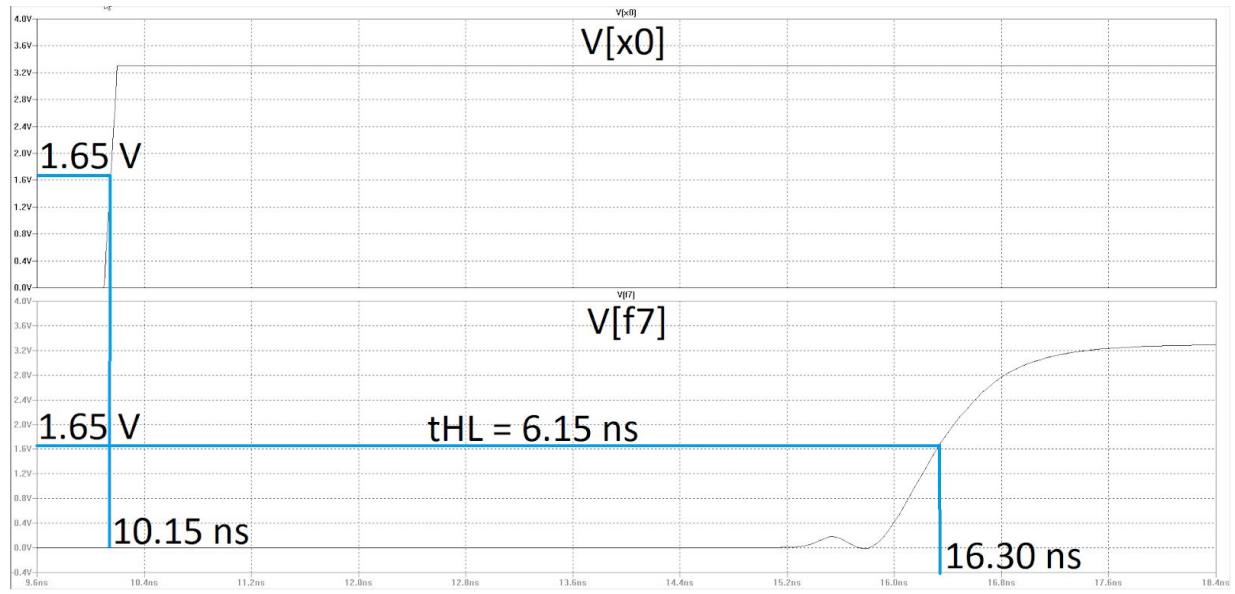


Figure 43 tHL V[f7] layout

In Figure 44 shows the tLH of between the input V[x0] and last output V[f7] which is 5.45ns

$$tLH = 25.50ns - 20.05ns = 5.45ns$$

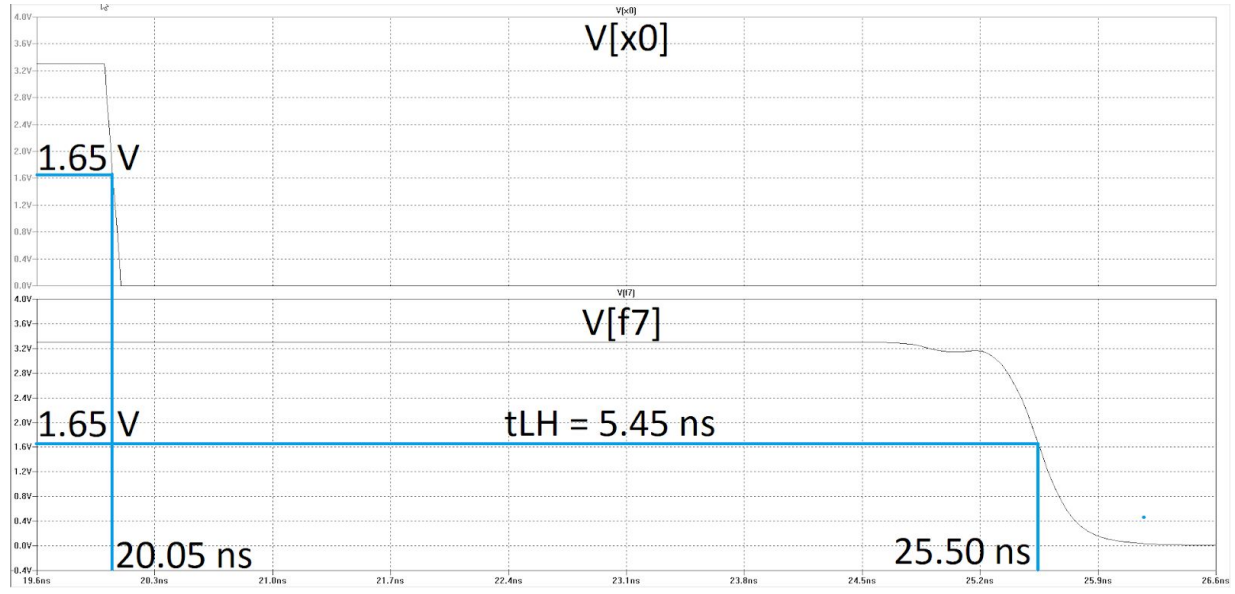


Figure 44 tLH V[f7] layout

Taking the average of the two delays we see that the propagation is 5.8ns. Below that we see the code output that LTspice provides during simulation.

$$tp = \frac{tLH + tHL}{2} = \frac{6.15ns + 5.45ns}{2} = 5.8ns$$

```
*** SPICE deck for cell 8_Bit_Adder_V2{lay} from library Project-2
*** Created on Sun Apr 07, 2019 15:14:48
*** Last revised on Sun Apr 07, 2019 20:20:15
*** Written on Sun Apr 07, 2019 21:01:21 by Electric VLSI Design System, version 9.07
*** Layout tech: mocmos, foundry MOSIS
*** UC SPICE *** , MIN_RESIST 4.0, MIN_CAPAC 0.1FF

*** SUBCIRCUIT Project-2__Full_Adder_V4 FROM CELL Full_Adder_V4{lay}
.SUBCKT Project-2__Full_Adder_V4 CI CO gnd SUM vdd X Y
Mnmos@0 net@10 net@13 gnd gnd N L=0.7U W=1.75U AS=17.512P AD=3.828P PS=25.741U PD=7.875U
Mnmos@1 gnd X net@27 gnd N L=0.7U W=1.75U AS=1.531P AD=17.512P PS=3.5U PD=25.741U
Mnmos@2 net@27 Y net@29 gnd N L=0.7U W=1.75U AS=2.527P AD=1.531P PS=4.637U PD=3.5U
Mnmos@3 net@29 net@0 net@10 gnd N L=0.7U W=1.75U AS=3.828P AD=2.527P PS=7.875U PD=4.637U
Mnmos@4 net@0 Y gnd gnd N L=0.7U W=1.75U AS=17.512P AD=4.288P PS=25.741U PD=8.4U
Mnmos@5 net@13 X gnd gnd N L=0.7U W=1.75U AS=17.512P AD=4.441P PS=25.741U PD=8.575U
Mnmos@6 net@94 net@97 gnd gnd N L=0.7U W=1.75U AS=17.512P AD=3.828P PS=25.741U PD=7.875U
Mnmos@7 gnd net@29 net@112 gnd N L=0.7U W=1.75U AS=1.531P AD=17.512P PS=3.5U PD=25.741U
Mnmos@8 net@112 CI SUM gnd N L=0.7U W=1.75U AS=2.527P AD=1.531P PS=4.637U PD=3.5U
Mnmos@9 SUM net@92 net@94 gnd N L=0.7U W=1.75U AS=3.828P AD=2.527P PS=7.875U PD=4.637U
Mnmos@10 net@92 CI gnd gnd N L=0.7U W=1.75U AS=17.512P AD=4.288P PS=25.741U PD=8.4U
Mnmos@11 net@97 net@29 gnd gnd N L=0.7U W=1.75U AS=17.512P AD=4.441P PS=25.741U PD=8.575U
Mnmos@16 net@240 Y gnd gnd N L=0.7U W=1.75U AS=17.512P AD=1.317P PS=25.741U PD=3.325U
Mnmos@17 net@238 X net@240 gnd N L=0.7U W=1.75U AS=1.317P AD=2.96P PS=3.325U PD=5.717U
Mnmos@18 net@283 net@29 gnd gnd N L=0.7U W=1.75U AS=17.512P AD=1.317P PS=25.741U PD=3.325U
Mnmos@19 net@281 CI net@283 gnd N L=0.7U W=1.75U AS=1.317P AD=2.96P PS=3.325U PD=5.717U
Mnmos@20 net@308 net@238 gnd gnd N L=0.7U W=1.75U AS=17.512P AD=1.317P PS=25.741U PD=3.325U
Mnmos@21 CO net@281 net@308 gnd N L=0.7U W=1.75U AS=1.317P AD=2.96P PS=3.325U PD=5.717U
Mpmos@0 net@17 net@13 vdd vdd P L=0.7U W=1.75U AS=13.803P AD=2.909P PS=22.025U PD=5.075U
Mpmos@1 vdd net@0 net@17 vdd P L=0.7U W=1.75U AS=2.909P AD=13.803P PS=5.075U PD=22.025U
Mpmos@2 net@29 X net@17 vdd P L=0.7U W=1.75U AS=2.909P AD=2.527P PS=5.075U PD=4.637U
Mpmos@3 net@17 Y net@29 vdd P L=0.7U W=1.75U AS=2.527P AD=2.909P PS=4.637U PD=5.075U
Mpmos@4 net@0 Y vdd vdd P L=0.7U W=1.75U AS=13.803P AD=4.288P PS=22.025U PD=8.4U
Mpmos@5 net@13 X vdd vdd P L=0.7U W=1.75U AS=13.803P AD=4.441P PS=22.025U PD=8.575U
Mpmos@6 net@101 net@97 vdd vdd P L=0.7U W=1.75U AS=13.803P AD=2.909P PS=22.025U PD=5.075U
Mpmos@7 vdd net@92 net@101 vdd P L=0.7U W=1.75U AS=2.909P AD=13.803P PS=5.075U PD=22.025U
Mpmos@8 SUM net@29 net@101 vdd P L=0.7U W=1.75U AS=2.909P AD=2.527P PS=5.075U PD=4.637U
Mpmos@9 net@101 CI SUM vdd P L=0.7U W=1.75U AS=2.527P AD=2.909P PS=4.637U PD=5.075U
Mpmos@10 net@92 CI vdd vdd P L=0.7U W=1.75U AS=13.803P AD=4.288P PS=22.025U PD=8.4U
Mpmos@11 net@97 net@29 vdd vdd P L=0.7U W=1.75U AS=13.803P AD=4.441P PS=22.025U PD=8.575U
Mpmos@16 net@238 Y vdd vdd P L=0.7U W=1.75U AS=13.803P AD=2.96P PS=22.025U PD=5.717U
Mpmos@17 vdd X net@238 vdd P L=0.7U W=1.75U AS=2.96P AD=13.803P PS=5.717U PD=22.025U
Mpmos@18 net@281 net@29 vdd vdd P L=0.7U W=1.75U AS=13.803P AD=2.96P PS=22.025U PD=5.717U
Mpmos@19 vdd CI net@281 vdd P L=0.7U W=1.75U AS=2.96P AD=13.803P PS=5.717U PD=22.025U
Mpmos@20 CO net@238 vdd vdd P L=0.7U W=1.75U AS=13.803P AD=2.96P PS=22.025U PD=5.717U
Mpmos@21 vdd net@281 CO vdd P L=0.7U W=1.75U AS=2.96P AD=13.803P PS=5.717U PD=22.025U
.ENDS Project-2__Full_Adder_V4

*** TOP LEVEL CELL: 8_Bit_Adder_V2{lay}
XFull_Add@0 CI net@0 gnd F0 vdd X0 Y0 Project-2__Full_Adder_V4
XFull_Add@1 net@54 net@9 gnd F2 vdd X2 Y2 Project-2__Full_Adder_V4
XFull_Add@2 net@0 net@54 gnd F1 vdd X1 Y1 Project-2__Full_Adder_V4
XFull_Add@3 net@9 net@55 gnd F3 vdd X3 Y3 Project-2__Full_Adder_V4
XFull_Add@8 net@55 net@17 gnd F4 vdd X4 Y4 Project-2__Full_Adder_V4
XFull_Add@9 net@67 net@11 gnd F6 vdd X6 Y6 Project-2__Full_Adder_V4
XFull_Add@10 net@17 net@67 gnd F5 vdd X5 Y5 Project-2__Full_Adder_V4
XFull_Add@11 net@11 CO gnd F7 vdd X7 Y7 Project-2__Full_Adder_V4
```

```
*** TOP LEVEL CELL: 8_Bit_Adder_V2{lay}
XFull_Add@0 CI net@0 gnd F0 vdd X0 Y0 Project-2__Full_Adder_V4
XFull_Add@1 net@54 net@9 gnd F2 vdd X2 Y2 Project-2__Full_Adder_V4
XFull_Add@2 net@0 net@54 gnd F1 vdd X1 Y1 Project-2__Full_Adder_V4
XFull_Add@3 net@9 net@55 gnd F3 vdd X3 Y3 Project-2__Full_Adder_V4
XFull_Add@8 net@55 net@17 gnd F4 vdd X4 Y4 Project-2__Full_Adder_V4
XFull_Add@9 net@67 net@11 gnd F6 vdd X6 Y6 Project-2__Full_Adder_V4
XFull_Add@10 net@17 net@67 gnd F5 vdd X5 Y5 Project-2__Full_Adder_V4
XFull_Add@11 net@11 CO gnd F7 vdd X7 Y7 Project-2__Full_Adder_V4

* Spice Code nodes in cell cell '8_Bit_Adder_V2{lay}'
VDD VDD 0 DC 3.3
VGND GND 0 DC 0
VIN3 CI 0 PULSE(3.3 0 0 100p 100p 80n 160n)
VIN X0 0 PULSE(3.3 0 0 100p 100p 10n 20n)
VIN2 Y0 0 PULSE(0 3.3 100p 100p 40n 80n)
VIN4 X1 0 PULSE(3.3 0 0 100p 100p 80n 160n)
VIN5 Y1 0 PULSE(0 3.3 100p 100p 40n 80n)
VIN6 X2 0 PULSE(3.3 0 0 100p 100p 80n 160n)
VIN7 Y2 0 PULSE(0 3.3 100p 100p 40n 80n)
VIN8 X3 0 PULSE(3.3 0 0 100p 100p 80n 160n)
VIN9 Y3 0 PULSE(0 3.3 100p 100p 40n 80n)
VIN10 X4 0 PULSE(3.3 0 0 100p 100p 80n 160n)
VIN11 Y4 0 PULSE(0 3.3 100p 100p 40n 80n)
VIN12 X5 0 PULSE(3.3 0 0 100p 100p 80n 160n)
VIN13 Y5 0 PULSE(0 3.3 100p 100p 40n 80n)
VIN14 X6 0 PULSE(3.3 0 0 100p 100p 80n 160n)
VIN15 Y6 0 PULSE(0 3.3 100p 100p 40n 80n)
VIN16 X7 0 PULSE(3.3 0 0 100p 100p 80n 160n)
VIN17 Y7 0 PULSE(3.3 0 0 100p 100p 80n 160n)
.TRAN 0 40n
.include C:\electric\MOS_model.txt
.END
```

Figure 45 LTspice Code

Layout IRSIM

For IRSIM for Layout I do the same thing I've done for the schematic look at 2 Bits at a time with all four calculations happening sequentially, each column corresponds to examples (1) (2) and so on that were shown in the the Schematic IRSIM section and similarly to that section I follow the expected outcome to the xxxx xx11 bit rule to be able to track the updates as they occur every 2 bits.

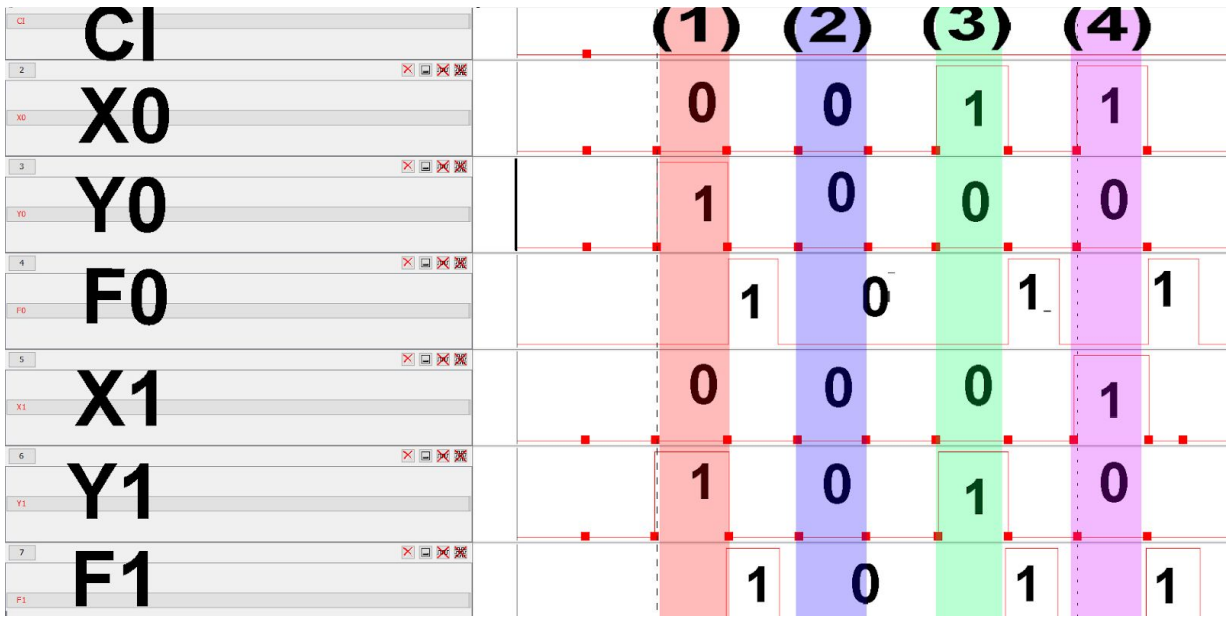


Figure 46. 0th and 1st Inputs and Output
(1) F = 0100 1011 - xxxx xx11(2) F =1111 0100 - xxxx xx00 (3) F = 0110 0111 - xxxx xx11 (4) F = 1010 1111 - xxxx xx11

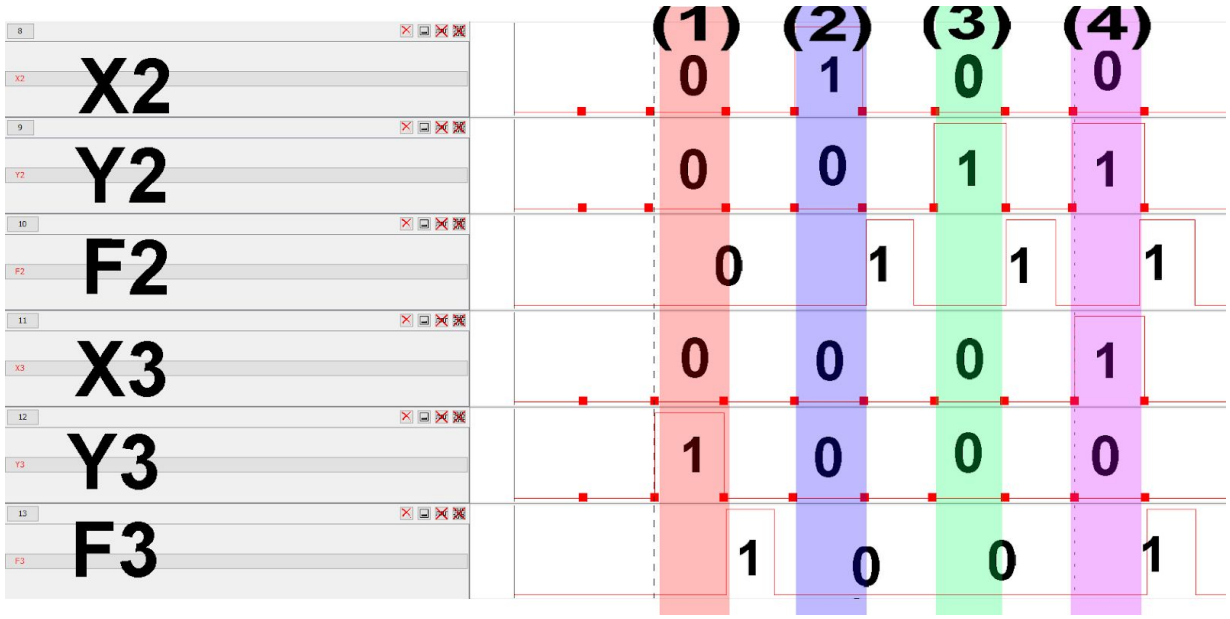


Figure 47. 2nd and 3rd Inputs and Output
(1) F = 0100 1011 - xxxx 1011(2) F =1111 0100 - xxxx 0100 (3) F = 0110 0111 - xxxx 0111 (4) F = 1010 1111 - xxxx 1111

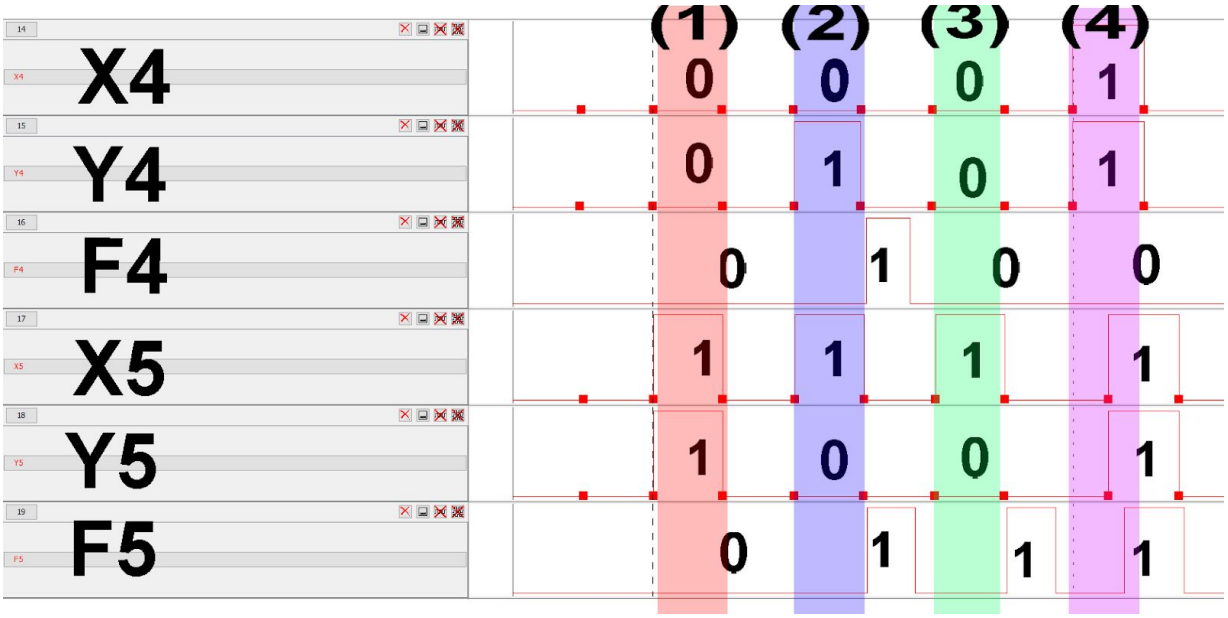


Figure 48. 4th and 5th Inputs and Output
(1) F = 0100 1011 - xx00 1011(2) F =1111 0100 - xx11 0100 (3) F = 0110 0111 - xx10 0111
(4) F = 1010 1111 - xx10 1111

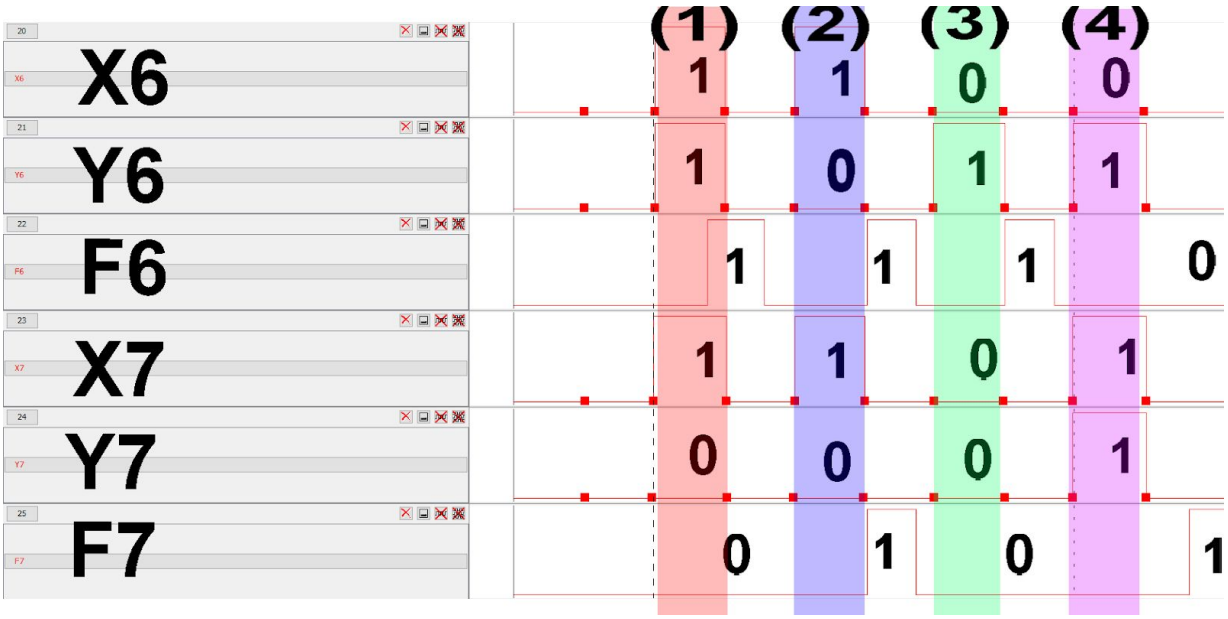


Figure 49. 6th and 7th Inputs and Output
(1) F = 0100 1011 - 0100 1011(2) F =1111 0100 - 1111 0100 (3) F = 0110 0111 - 0110 0111
(4) F = 1010 1111 - 1010 1111

As we can see the Layout works just as the schematic did, the only differences is that the propagation is longer so the outcomes appear later than they did in the schematic.

Summary of Measurements

Here is the compilation of all the measurements from the schematic and layout.

Propagation

Tabel 7. Propagation Measurements

	Schematic	Layout
Rise Time	0.36 ns	1.0 ns
Fall Time	0.2 ns	0.50 ns
tHL	3.59 ns	6.15 ns
tLH	3.73 ns	5.45 ns
tp	3.66 ns	5.8 ns

Total Chip Area

Using the measure tool I measured the length and width of the and as given that the chip is 2989.5 units long by 56.5 units as shown in Figures 51 and 52, we know that one unit is shown to be 350 nm as shown in Figure 50 therefore the area of the chip is

$$Area = 2989.5 * 56.5 * 350nm * 350nm = 2.06910769 \times 10^{6} um^{2}$$

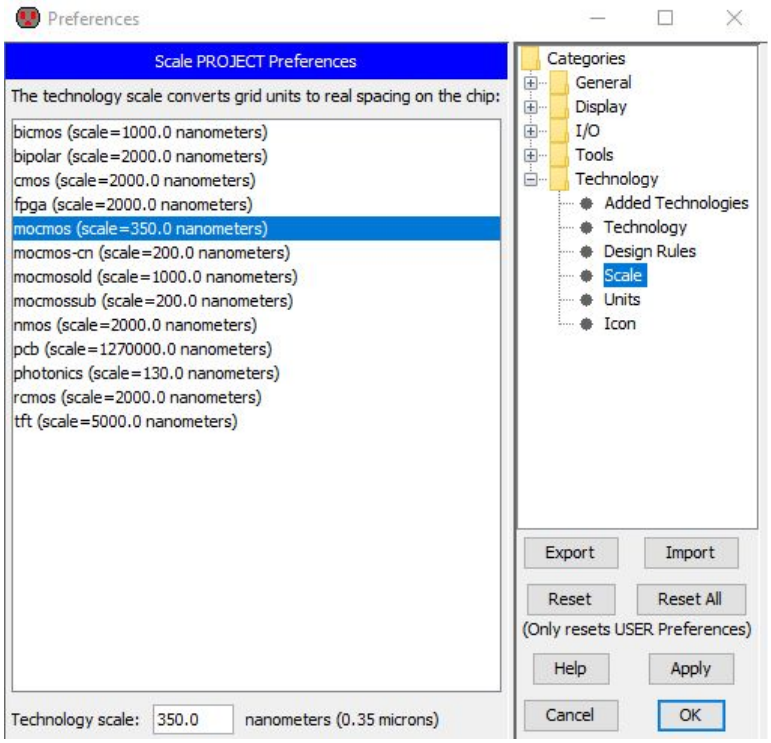


Figure 50. Scale Set

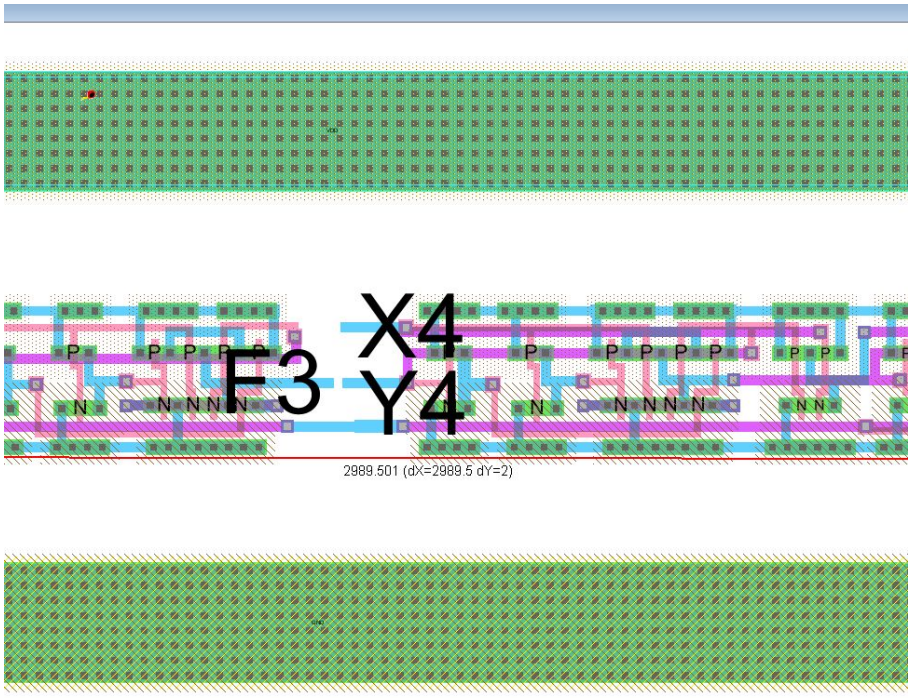


Figure 51. Length of the chip

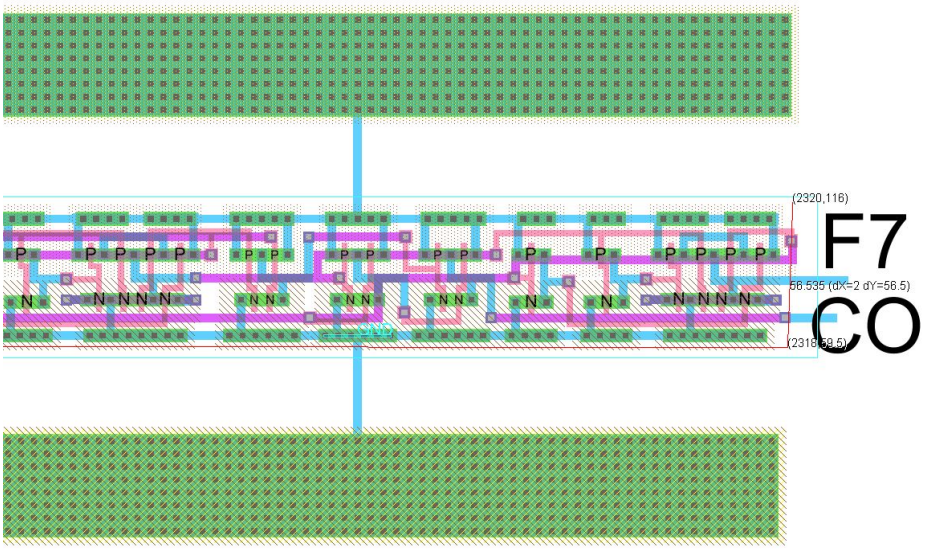


Figure 52. Width of the chip

Comparison of Schematic and Layout

Looking at Tabel 7, we can see that most of the measurements are around 1.5 times as longer on the layout over the schematic. This makes sense and honestly I was expecting a much bigger change as this is around the same amount of increase that was seen in a NAND gate on it's own when transferring from schematic to layout. It is possible my method for checking the propagation was a wrong approach but Figures 18 and 40 which show the movement of the signal lead me to believe that it is correct.

Conclusion

This project showed me that there are many ways to create a simple Full Adder, with me going over five different designs and considering how their implementation would go onto a layout was an interesting challenge as I tried to make the most condense layout as possible. The change in time between schematic and layout was surprisingly small especially with how much parasitic components must have been created along the way. Dealing with IRSIM was cumbersome task as the order in which you put signals in affects changes how the signals are drawn and the program doesn't seem to want to update drawn outputs so the process of obtaining the additions was a very slow and tedious process as you have to reload points and redraw them in correct order. Overall I feel like I learned a decent amount through the process of designing and redesigning the Full Adder for sake of having a small layout.

References

Abirr, et al. “Half Adder and Full Adder Circuits Using NAND Gates.” *Electronics Hub*, 24 Dec. 2017, www.electronicshub.org/half-adder-and-full-adder-circuits/.

“Full Adder | Digital Electronics.” *GeeksforGeeks*, 11 Dec. 2018, www.geeksforgeeks.org/full-adder-digital-electronics/.

How to Build an XOR Gate Using NAND Gates.” *How to Build an XOR Gate Using NAND Gates*, vlsiuniverse.blogspot.com/2016/10/xor-using-nand.html