EE457: Digital IC Design

Project #1 SPRING SEMESTER 2024

Report Cover Sheet*

Due 3/15/2024 by 8PM (-2 pts per hour up to 5 points/day)

PROJECT TITLE: Design of 8-Bit Binary Ripple Carry Adder

Student's Name: Paniz Peiravani

Topics (Do not change orders of section)	GRADES
Section1: Executive Summary (1/2 page)	Required
Section 2: Introduction and Background	/5
Section 3: Electric Circuit Schematics	/10
Section 4: LTSPICE Simulation for Schematic	/10
Section 5: IRSIM for Schematic	/10
Section 6: Electric Layouts (landscape mode and legible to see transistors)	/25
Section 7: LTSPICE Simulation for Layouts	/10
Section 8: IRSIM for Layout	/10
Section 9: Compare LTSPICE Measurements for Schematic and Layout (must provide comparisons between the two in table format)	/10
Section 10: Layout Measurements of chip area, number of transistors for the layout (provide a table to compare)	/10
Section 11: Conclusions and References	Required
Penalty: 5pts/day for Late Submission, Zero after 3/20/24 -10 points for no Electric schematic and layout files on BB	
TOTAL SCORE:	**/100**

*Penalty rules: A) -5pts per day for late submission. After five days, a score of zero will be given, but you are still required to complete the report. B) -2pts for any violations and delays submitted after 8PM on 3/15/24.

Table of Contents

Section1: Executive Summary

The purpose of this project is to design an 8-bit ripple carry adder. An 8-bit ripple carry adder is a circuit that we use to perform the addition of two 8-bit numbers. For instance, it is a sequence of 8 1-bit full adders. In other words, it is constructed by cascading eight 1-bit full adders with the carry-out from each full adder connected to the carry-in of the next full adder. This action will create a ripple effect for the carry bit which moves through the adders from the least significant bit to the most significant bit. Once we build out the 8-bit adder schematic and layout using Electric, we will use LTSpice and IRSIM to check the accuracy of our adder. Also, we will add different 8-bit numbers to see if the result of our addition is accurate for double-checking. For this project based on the requirement we used we used 175nm or $\boldsymbol{\lambda}$=1 for our design.

Section 2: Introduction and Background

To design our 8-bit ripple carry adder, first, we designed the full adder. This method will help us when we want to design our 8-bit ripple carry adder since it consists of eight 1-bit full adders. We know that a 1-bit full adder includes two XOR gates, two AND gates, and one OR gate, which will aid us in designing our schematic for the 8-bit full adder. To build our 1-bit full adder, we need to refer to the 1-bit full adder truth table and then derive the equations to design our CMOS circuit.

Table 1 - 1bit Truth Table

A	B	Cin	Sum	Cout
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

Based on the truth table, we can determine the Boolean equations for the Sum and the Cout or carry-out outputs. For instance, a 1-bit full adder has three inputs, which are A, B, and Cin or carry-in, and it produces two outputs: Sum and Cout or carry-out, which are determined by those inputs.

Sum: $\bar{A}\bar{B}Cin + \bar{A}B\overline{Cin} + A\bar{B}\overline{Cin} + ABCin \rightarrow$ As you can see, we have 30 transistors here.

For instance, we have 6 transistors for $A, \bar{A}, B, \bar{B}, C, \bar{C}$ and we have 4 products of 3 in parallel which makes it 12 so we have 12 for PUN and 12 for PDN. Hence, 6+24 = 30 transistors.

$Cout$: $\bar{A}BCin + A\bar{B}Cin + AB\overline{Cin} + ABCin = \bar{A}BCin + A\bar{B}Cin + AB(\overline{Cin} + Cin) = \bar{A}BCin + A\bar{B}Cin + AB = Cin(\bar{A}B + A\bar{B}) + AB \rightarrow$ As you can see, we have 10 transistors here. For isntance we have 5 PUN and 5 PDN which makes it 10 transistors.

Based on our equations we have 40 transistors in total. We can reduce the number of transistors by implement Sum output as a function of A, B, Cin, and $\overline{Cout}$. Therefore, our new truth table will be:

Table 2 - Updated Truth Table for less number of transistors

A	B	Cin	$\overline{Cout}$	Sum
0	0	0	1	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	0	0
1	1	1	0	1

The new equation for Sum will be:

Sum: $Cin\overline{Cout} + B\overline{Cout} + A\overline{Cout} + ABCin = ABC + \overline{Cout}(Cin + B + A)$

To change $\overline{Cout}$ to Cout, we will need an invertor.

With new equations we have 28 transistors in total which is better than 48 transistors. Hence, the final Cout and Sum equation will be:

Sum: $ABC + \overline{Cout}(Cin + B + A)$, and invertor to change $\overline{Cout}$ to Cout

$Cout$: $AB + Cin(A + B)$

For Cout we have:

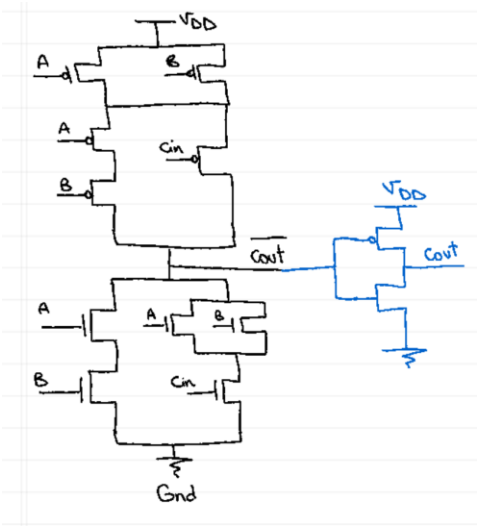


Figure 2 - CMOS Drawing for Cout

For Sum we have:

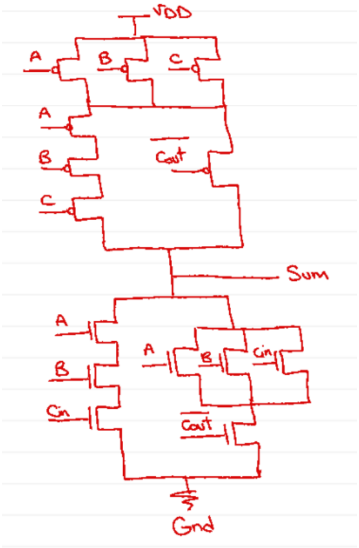


Figure 1 - CMOS Drawing for Sum

So the final CMOS design for our 1-bit full adder will be which has 28 transistor in total:

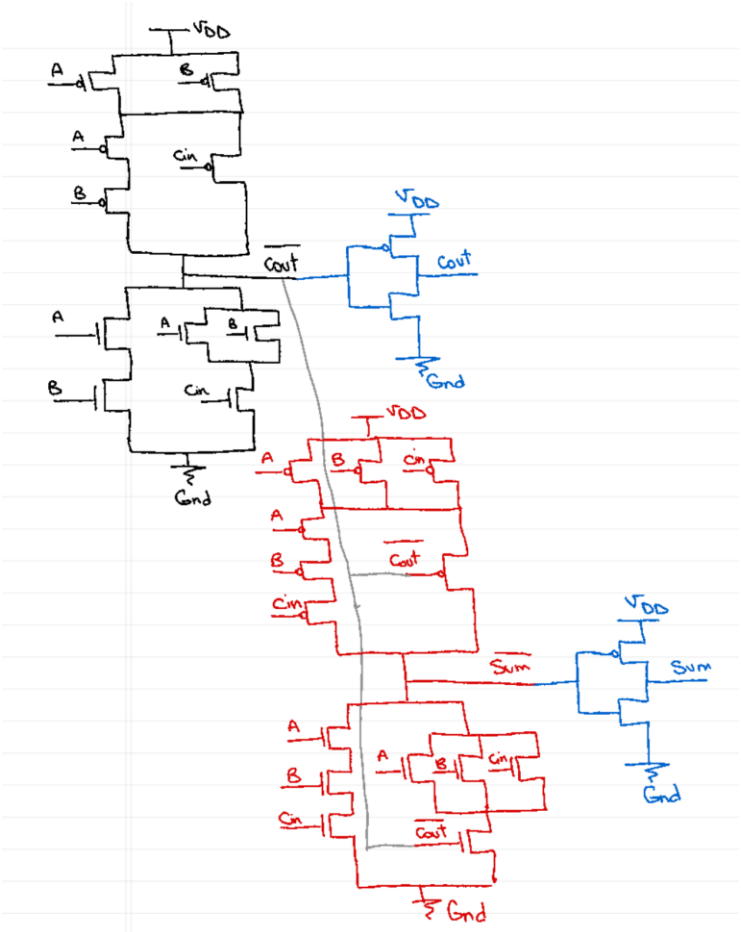


Figure 3 - CMOS for full Adder

Section 3: Electric Circuit Schematics

Now we can move on to design our circuit on Electric. To do that first, we will design our 1bit full adder and then change it to our final design to 8bit ripple adder.

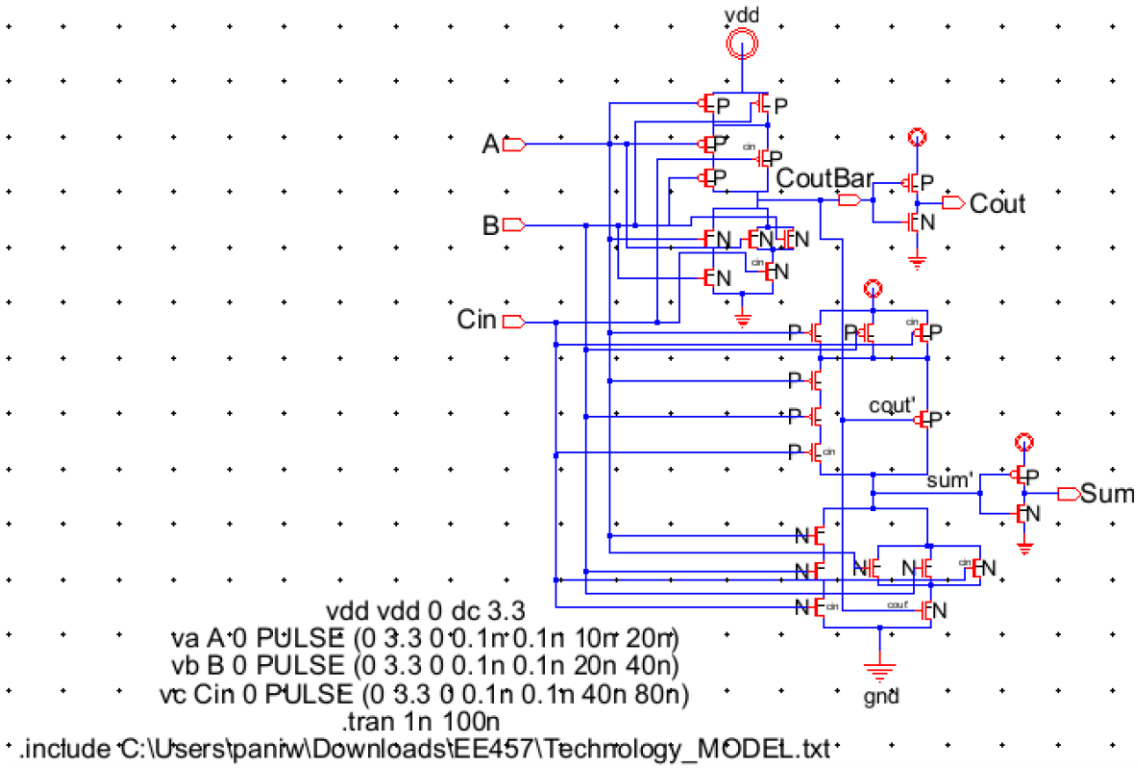


Figure 4 - 1bit Full Adder Schematic

Now, we will check to see if we have correct wave using LTSPICE and IRSIM. If we compare the waveform from truth table, we can see that we have a correct output for Cout and Sum.

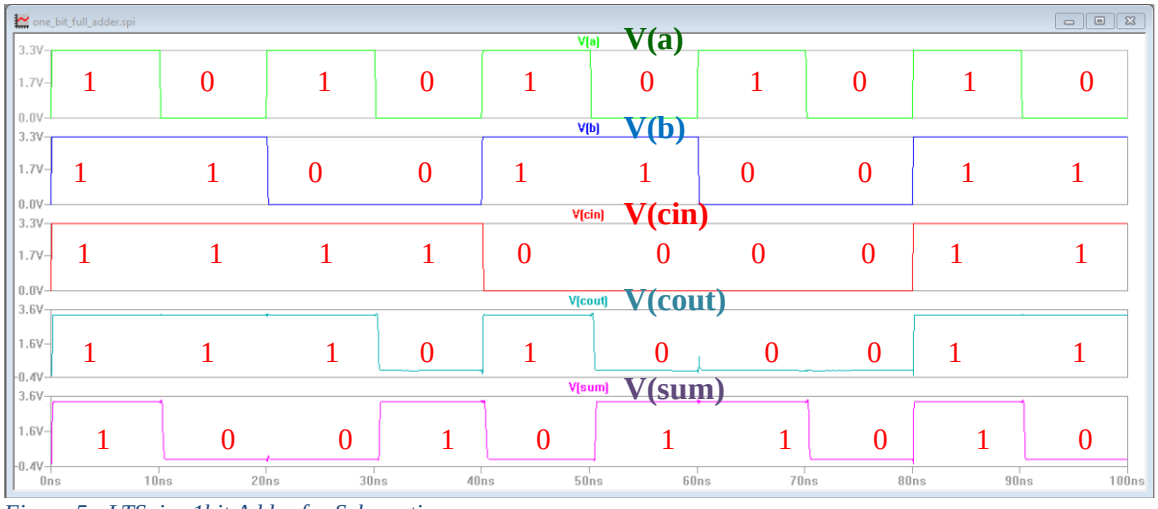


Figure 5 - LTSpice 1bit Adder for Schematic

As you can see based on the truth table I do have correct output using IRSIM as well. I just have a small delay.

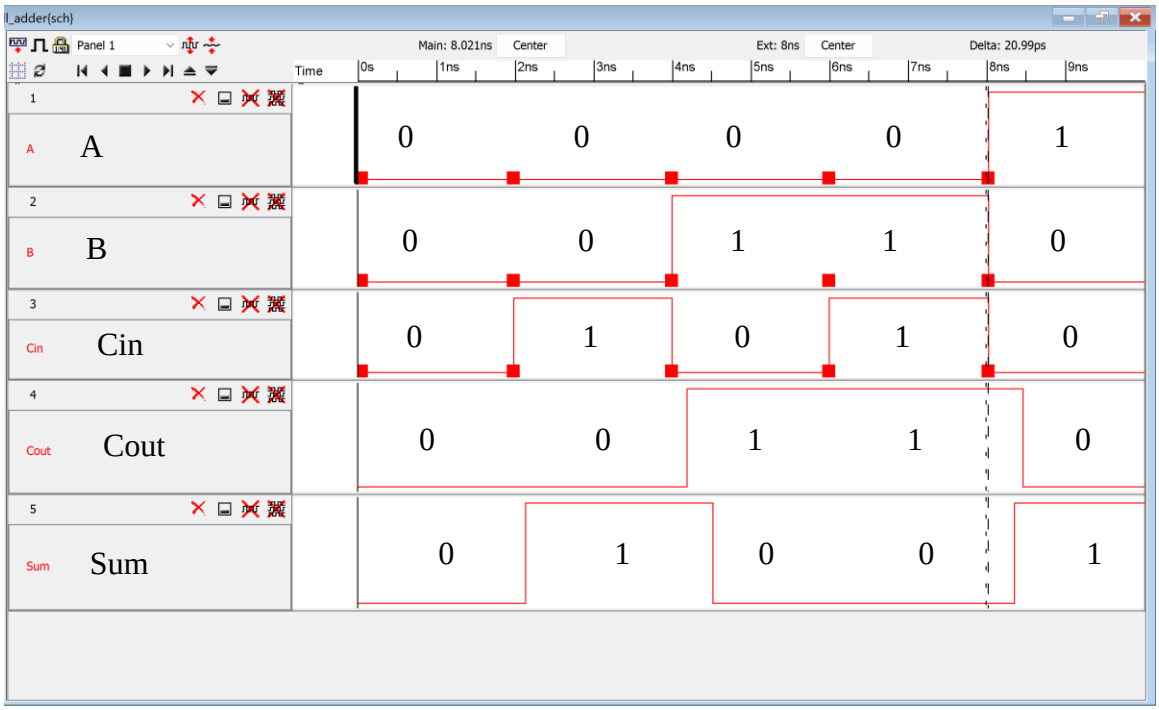


Figure 6 - IRSIM 1bit adder for Schematic

Now that we know our 1bit full adder works, we can move to create our 8bit full adder.

Below, you can see the schematic of our 8bit full adder.

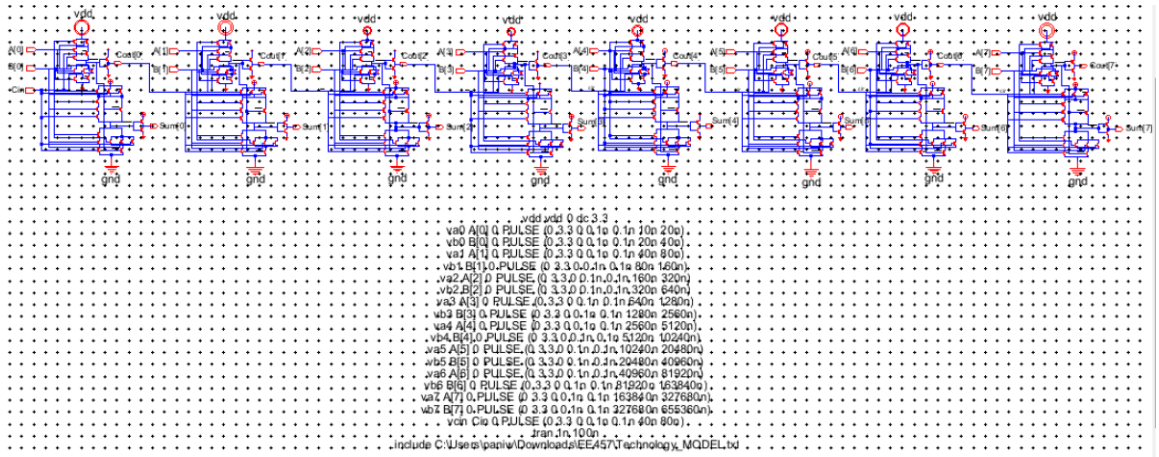


Figure 7 - 8bit carry in Adder Schematic

We have A[0:7] and B[0:1], and Cin for our input and Cout[0:7] to Sum[0:7] for our outputs.

To make sure that our schematic works properly, we use DRC check. As you can see from the figure below, we do not have any DRC error.

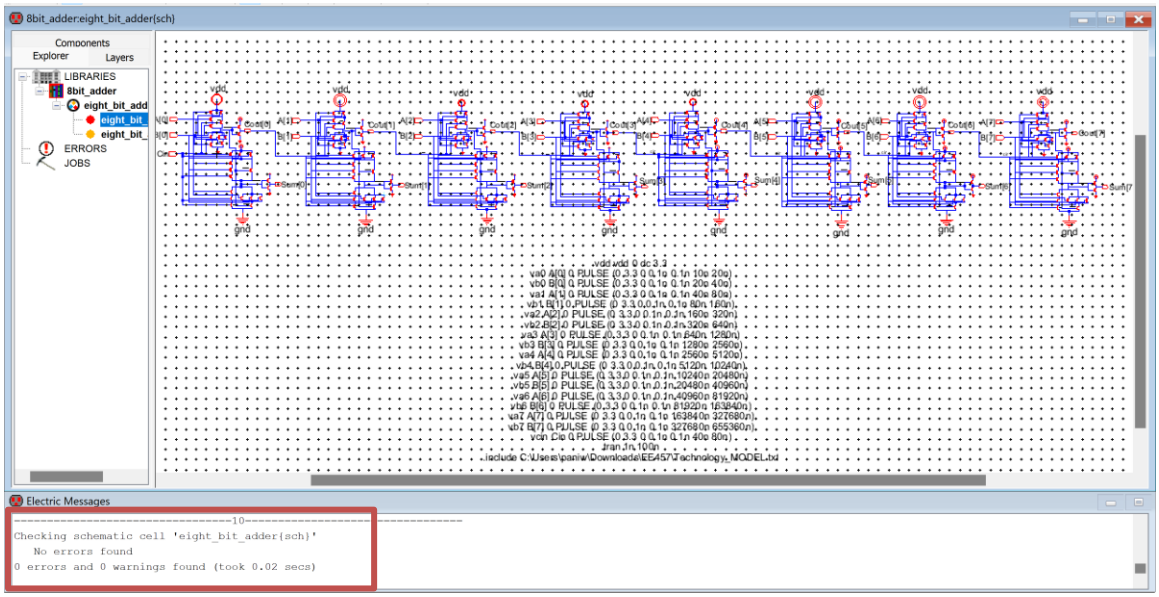


Figure 8 – DRC Check for 8bit adder Schematic

Section 4: LTSPICE Simulation for Schematic

Now that we have our schematic, we will move to simulate our schematic using LTSpice. You can see the Spice Code that we use below.

```
* Spice Code nodes in cell cell 'eight_bit_adder{sch}'
vdd vdd 0 dc 3.3
va0 A[0] 0 PULSE (0 3.3 0 0.1n 0.1n 10n 20n)
vb0 B[0] 0 PULSE (0 3.3 0 0.1n 0.1n 20n 40n)
va1 A[1] 0 PULSE (0 3.3 0 0.1n 0.1n 40n 80n)
vb1 B[1] 0 PULSE (0 3.3 0 0.1n 0.1n 80n 160n)
va2 A[2] 0 PULSE (0 3.3 0 0.1n 0.1n 160n 320n)
vb2 B[2] 0 PULSE (0 3.3 0 0.1n 0.1n 320n 640n)
va3 A[3] 0 PULSE (0 3.3 0 0.1n 0.1n 640n 1280n)
vb3 B[3] 0 PULSE (0 3.3 0 0.1n 0.1n 1280n 2560n)
va4 A[4] 0 PULSE (0 3.3 0 0.1n 0.1n 2560n 5120n)
vb4 B[4] 0 PULSE (0 3.3 0 0.1n 0.1n 5120n 10240n)
va5 A[5] 0 PULSE (0 3.3 0 0.1n 0.1n 10240n 20480n)
vb5 B[5] 0 PULSE (0 3.3 0 0.1n 0.1n 20480n 40960n)
va6 A[6] 0 PULSE (0 3.3 0 0.1n 0.1n 40960n 81920n)
vb6 B[6] 0 PULSE (0 3.3 0 0.1n 0.1n 81920n 163840n)
va7 A[7] 0 PULSE (0 3.3 0 0.1n 0.1n 163840n 327680n)
vb7 B[7] 0 PULSE (0 3.3 0 0.1n 0.1n 327680n 655360n)
vcin Cin 0 PULSE (0 3.3 0 0.1n 0.1n 40n 80n)
.tran 1n 100n
.include C:\Users\paniw\Downloads\EE457\Technology_MODEL.txt
.END
```

Figure 9 - Spice Code for 8bit carry in Adder for Schematic

As you can see on *Figure 10* and *Figure 11*, we have the correct waveform output based on our truth table. To make sure our output is correct, I double check the result multiple times with different inputs and outputs.

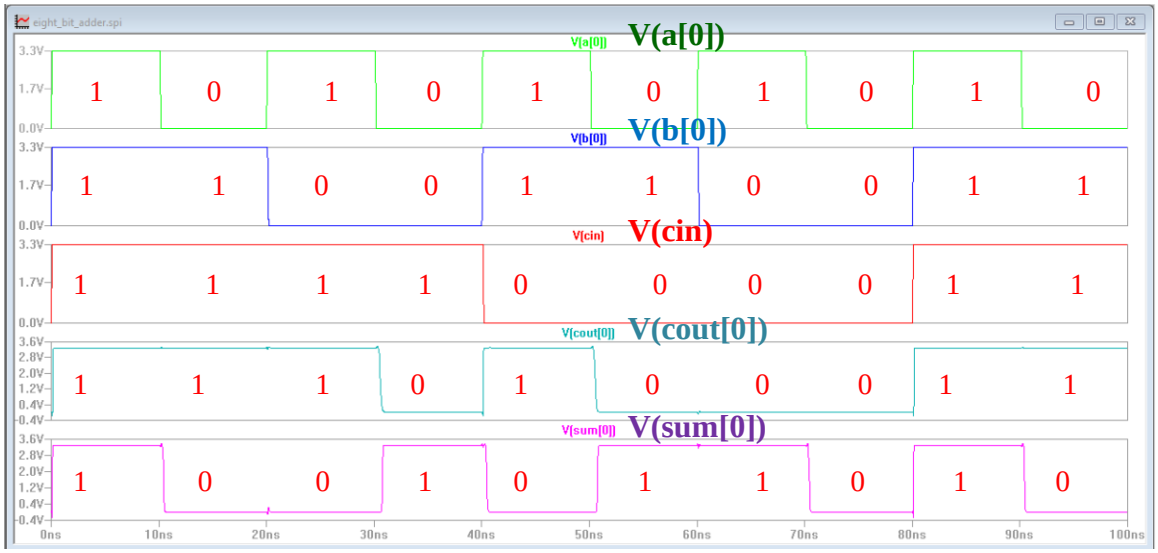


Figure 10 - LTSpice 8bit arry in Adder Waveform for Schematic

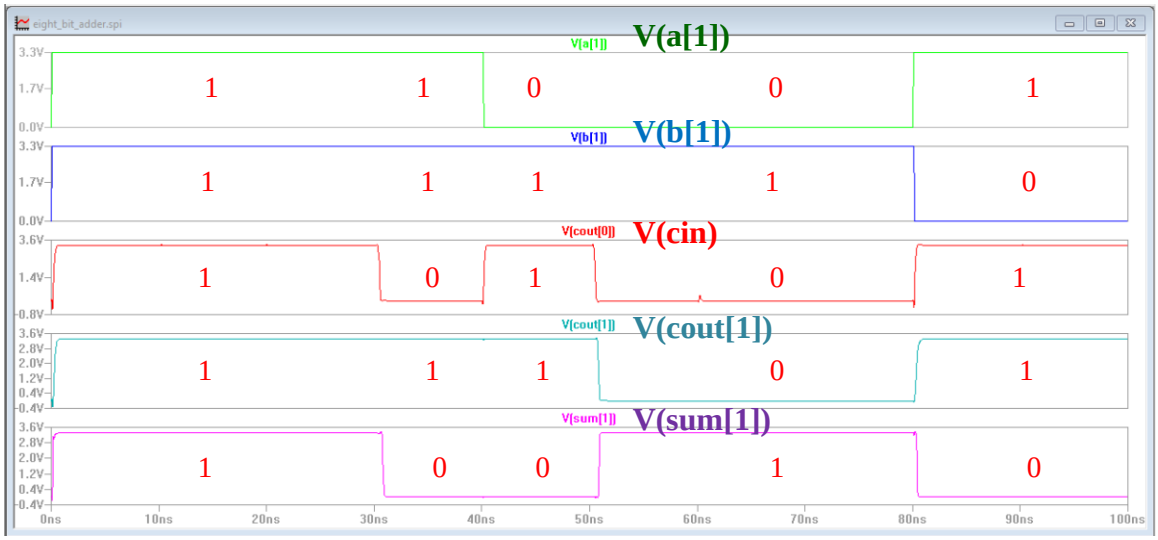


Figure 11 - LTSpice 8bit carry in Adder Waveform for Schematic

Section 5: IRSIM for Schematic

Now that we checked the LTSpice for our 8bit schematic, we can check the IRSIM. As you can see, based on the truth table our IRSIM has a correct waveform and similar to LTSpice waves; however, IRSIM has a slight delay of 0.4ns.

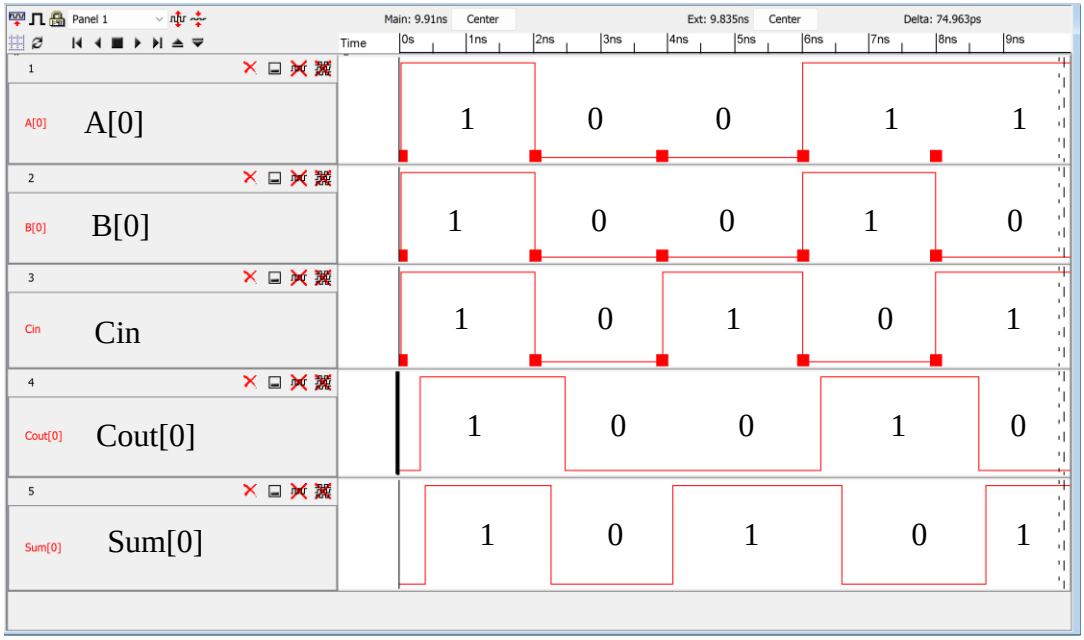


Figure 12 - IRSIM 8bit carry in Ader Waveform for Schematic

To make sure the schematic works perfectly, we try another inputs and outputs to double check the waveform. For instance, we tried A[1], B[1], Cout[0] which is our Cin for second adder, Cout[1], and Sum[1].

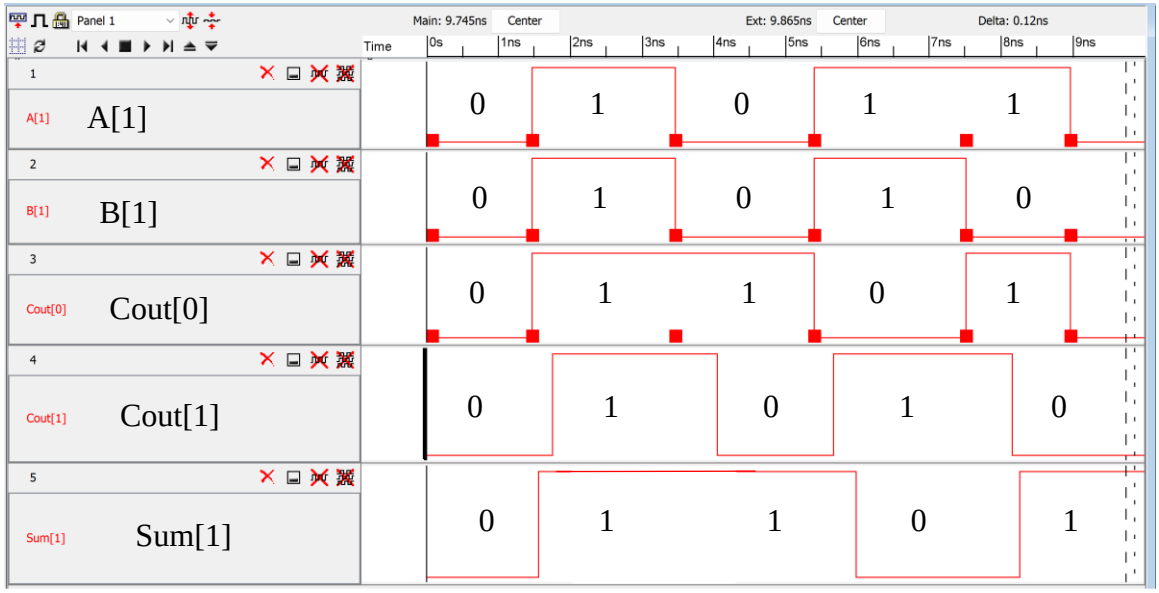


Figure 13 – IRSIM 8bit carry in Adder Waveform for Schematic

Now we can move on to calculate some binary numbers using IRSIM for the schematic to double check our schematic for 8bit ripple carry. Below you can see the table of binary numbers that we are going to calculate.

Table 3 - First Addition

-32+107 = 75								
11100000 + 01101011 = (1) 01001011								
A[0:7]	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
11100000	0	0	0	0	0	1	1	1
B[0:7]	B[0]	B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]
01101011	1	1	0	1	0	1	1	0
Cin	0							
Cout[0:7]	Cout[0]	Cout[1]	Cout[2]	Cout[3]	Cout[4]	Cout[5]	Cout[6]	Cout[7]
00000111	0	0	0	0	0	1	1	1
Sum[0:7]	Sum[0]	Sum[1]	Sum[2]	Sum[3]	Sum[4]	Sum[5]	Sum[6]	Sum[7]
01001011	1	1	0	1	0	0	1	0

Table 4 - Second Addition

-28-16 = -44								
11100100 + 11110000 = (1)11010100								
A[0:7]	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
11100100	0	0	1	0	0	1	1	1
B[0:7]	B[0]	B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]
11110000	0	0	0	0	1	1	1	1
Cin	0							
Cout[0:7]	Cout[0]	Cout[1]	Cout[2]	Cout[3]	Cout[4]	Cout[5]	Cout[6]	Cout[7]
00000111	0	0	0	0	0	1	1	1
Sum[0:7]	Sum[0]	Sum[1]	Sum[2]	Sum[3]	Sum[4]	Sum[5]	Sum[6]	Sum[7]
11010100	0	0	1	0	1	0	1	1

Table 5 - Third Addition

+112+9 = 121								
01110000 + 00001001 = 1111001								
A[0:7]	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
01110000	0	0	0	0	1	1	1	0
B[0:7]	B[0]	B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]
00001001	1	0	0	1	0	0	0	0
Cin	0							
Cout[0:7]	Cout[0]	Cout[1]	Cout[2]	Cout[3]	Cout[4]	Cout[5]	Cout[6]	Cout[7]
00000000	0	0	0	0	0	0	0	0
Sum[0:7]	Sum[0]	Sum[1]	Sum[2]	Sum[3]	Sum[4]	Sum[5]	Sum[6]	Sum[7]
01111011	1	1	0	1	1	1	1	0

Table 6 - Fourth Addition

-71-29 = -100								
10111001 + 11100011 = (1)10011100								
A[0:7]	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
10111001	0	0	0	0	0	1	1	1
B[0:7]	B[0]	B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]
11100011	1	1	0	1	0	1	1	0
Cin	0							
Cout[0:7]	Cout[0]	Cout[1]	Cout[2]	Cout[3]	Cout[4]	Cout[5]	Cout[6]	Cout[7]
11100011	1	1	0	0	0	1	1	1
Sum[0:7]	Sum[0]	Sum[1]	Sum[2]	Sum[3]	Sum[4]	Sum[5]	Sum[6]	Sum[7]
10011100	0	0	1	1	1	0	0	1

As you can see below we applied our binary number to A[0:7] and B[0:7] and we get correct hexadecimal value based our Schematic IRSIM. For instance,

For A[0:7] we have:
0xE0 = -32
0xE4 = -28
0x70 = +112
0xB9 = -71

For B[0:7] we have:
0x6B = +107
0xF0 = -16
0x09 = +9
0xE3 = -29

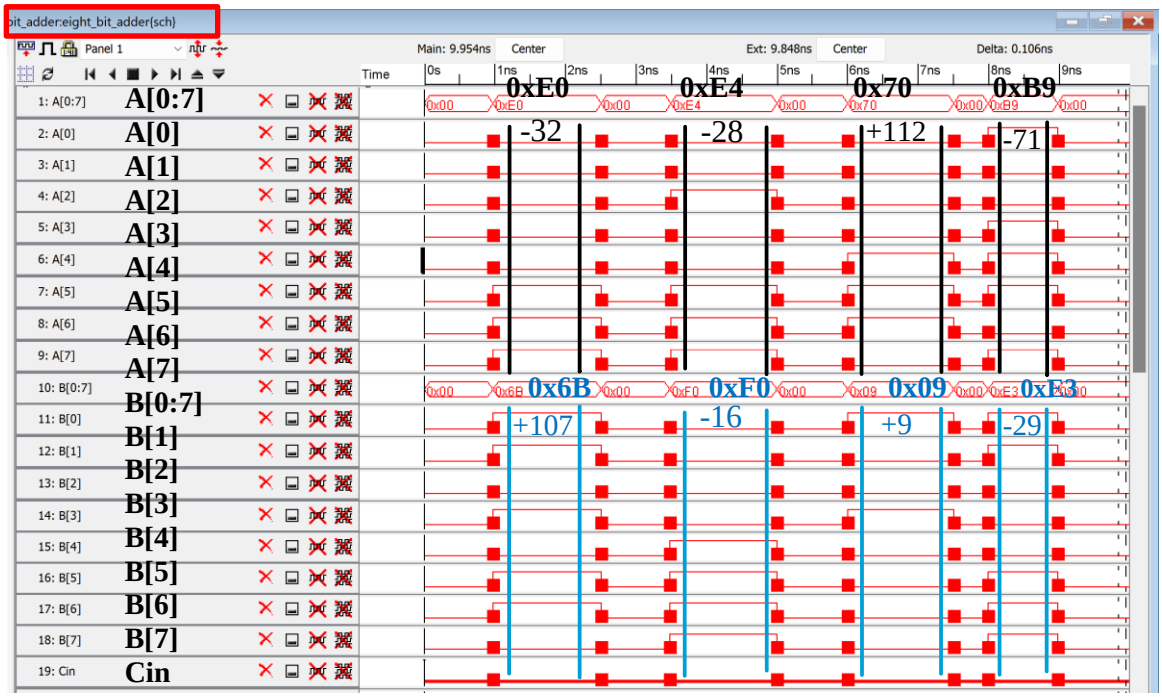


Figure 14 - Addition on IRSIM for A[0:7] and B[0:7] for Schematic

For the first addition, -32+107 = 75, we used the IRSIM and as you can see in *Figure 15*, we got the correct result based on the Cout and Sum. You can compare the result from *Figure 15* to the content of *Table 3* to match the results.

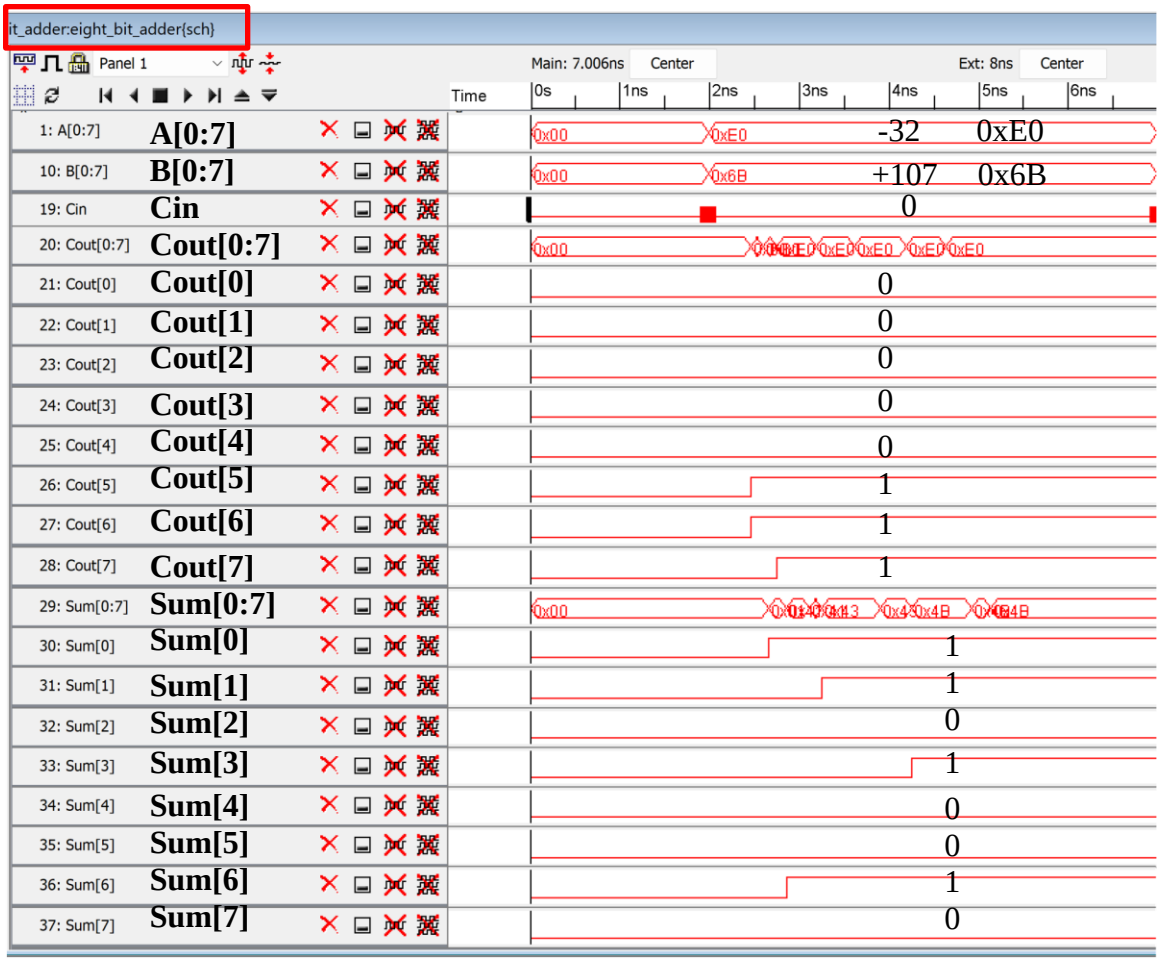


Figure 15 - (-32+107=75) for schematic

For the second addition, -28-16 = -44, we used the IRSIM and as you can see in *Figure 16*, we got the correct result based on the Cout and Sum. You can compare the result from *Figure 16* to the content of *Table 4* to match the results.

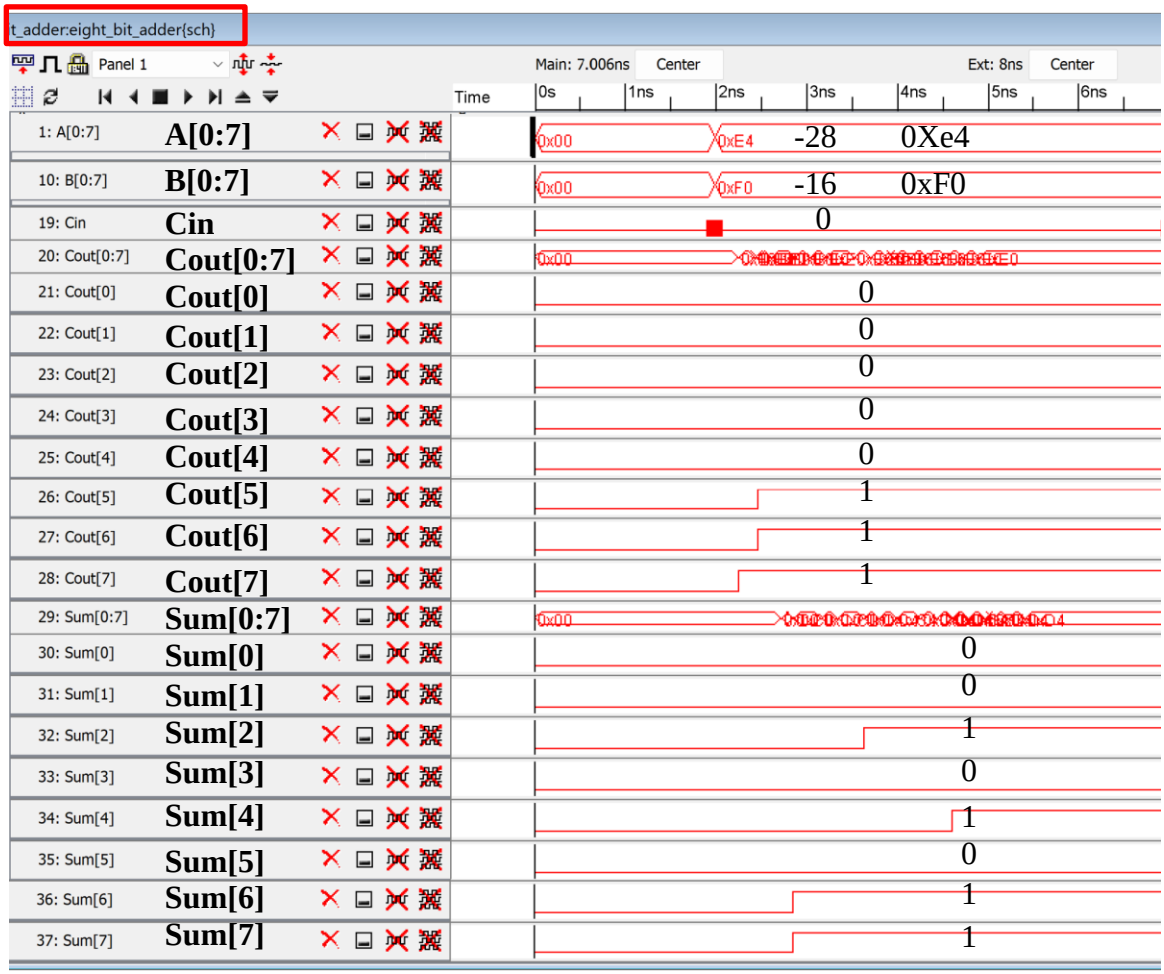


Figure 16 - (-28-16=-44) for schematic

For the third addition, +112+9 = +121, we used the IRSIM and as you can see on *Figure 17*, we got the correct result based on the Cout and Sum. You can compare the result from *Figure 17* to the content of *Table 5* to match the results.

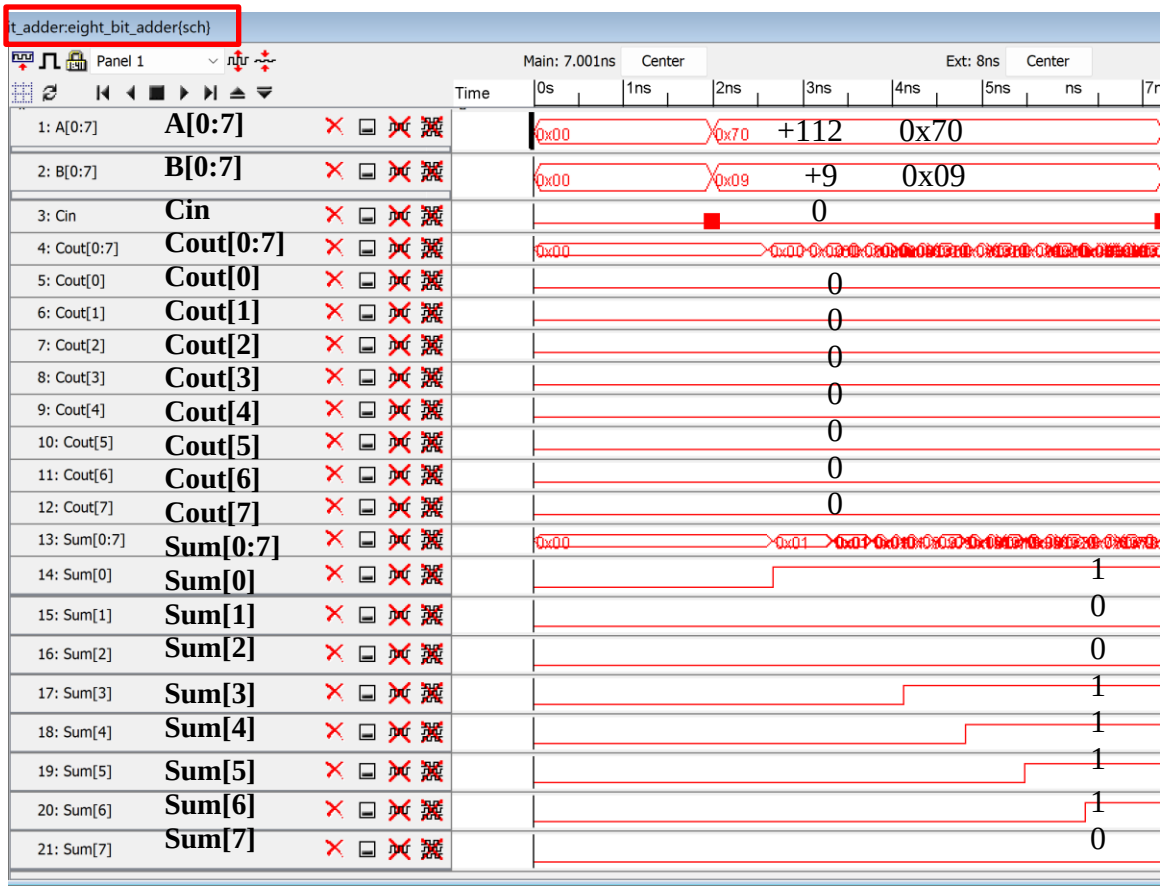


Figure 17 – (+112+9 = +121) for schematic

For the fourth addition, -79-20=-100, we used the IRSIM and as you can see on *Figure 18*, we got the correct result based on the Cout and Sum. You can compare the result from *Figure 18* to the content of *Table 6* to match the results.

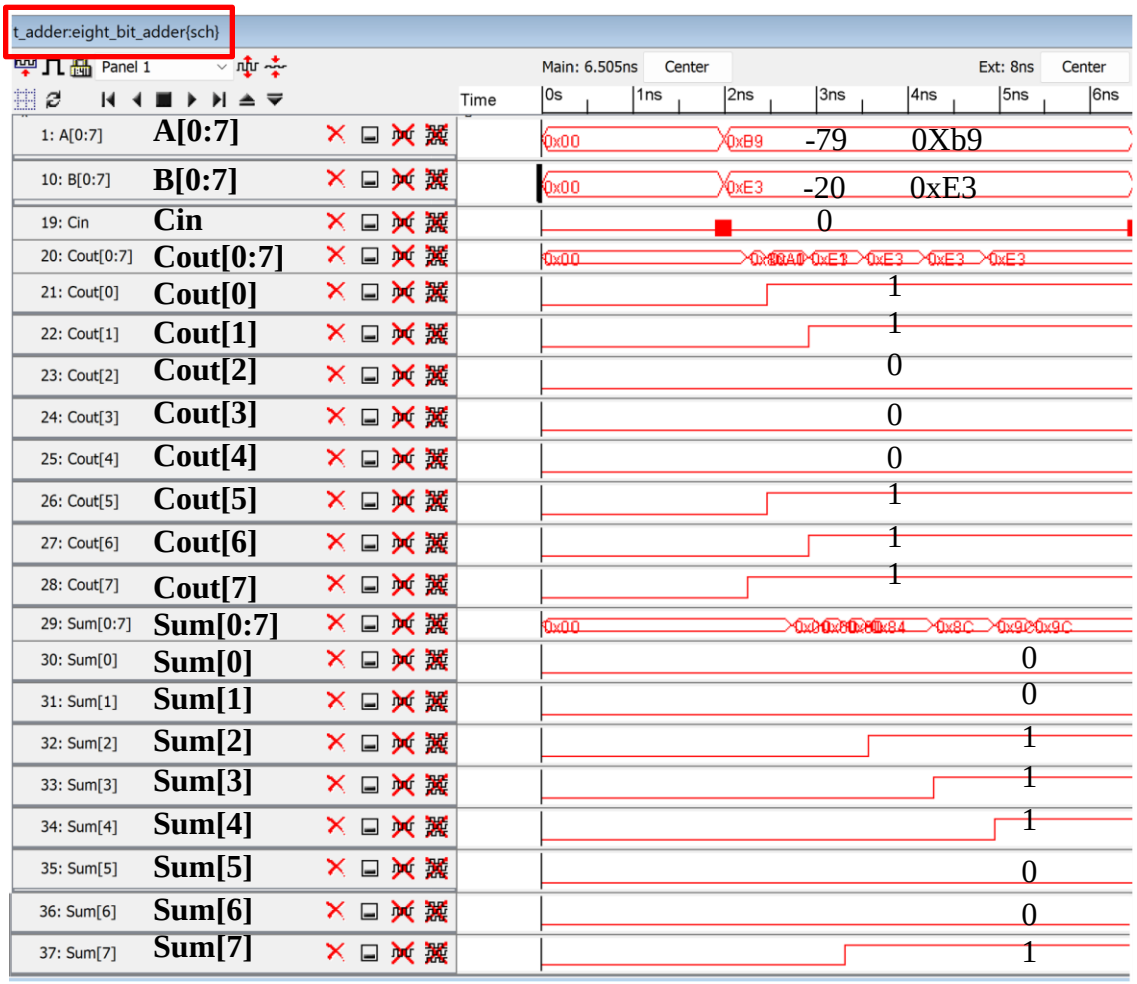


Figure 18 - (-79-20=-100) for schematic

As you can see from *Figure 15 to 18*, all our calculations output is correct. Hence, our schematic works perfectly.

Section 6: Electric Layouts

To build the layout, first we need to find the Euler's Path for both $\overline{Cout}$ and $\overline{Sum}$ so we be able to draw the stick diagram.
Based on the circuit, we can see the Euler's path for $\overline{Cout}$ is:

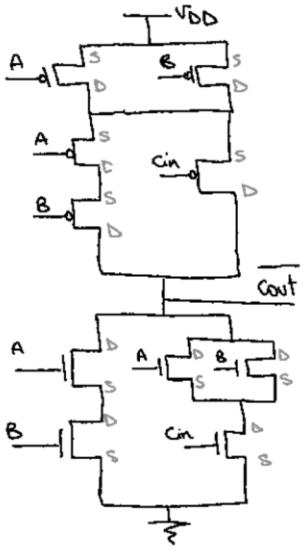
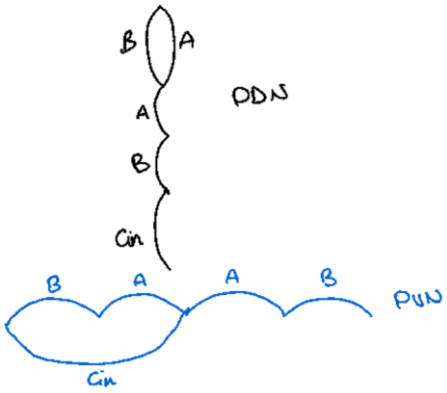


Figure 19 - Euler's Path for Cout'



Hence, based on the Euler's Path we can see our path for stick diagram is:

B, A, A, B, Cin

Now for $\overline{Sum}$ we have:

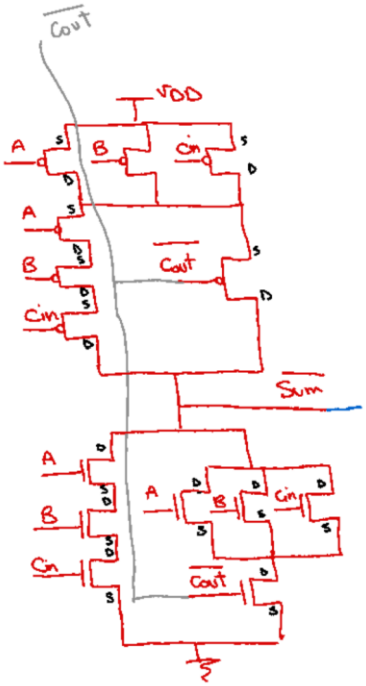
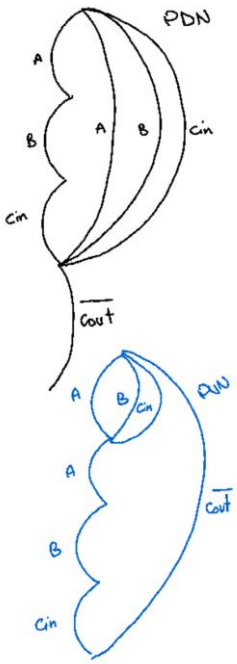


Figure 20 - Euler's Path for Sum'



Hence, based on the Euler's path, we can see out path for stick diagram is:
$\overline{Cout}$, Cin, B, A, A, B, Cin

Now that we know our Euler's path, we can move to draw our stick diagram,
Stick diagram for out $\overline{Cout}$ is:

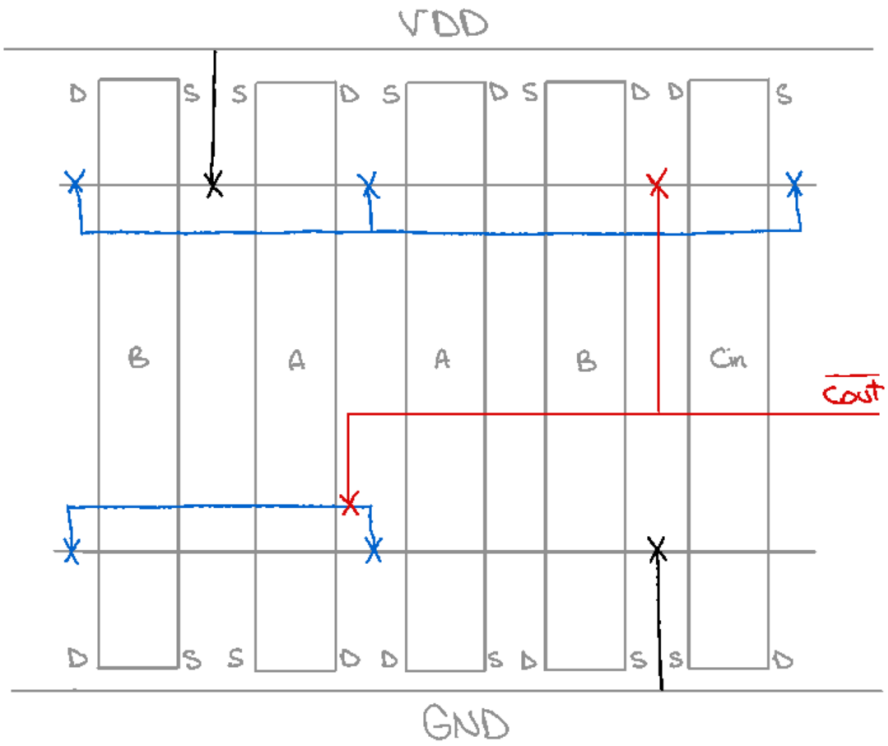


Figure 21 - Cout' Stick Diagram

Stick diagram for out $\overline{Sum}$ is:

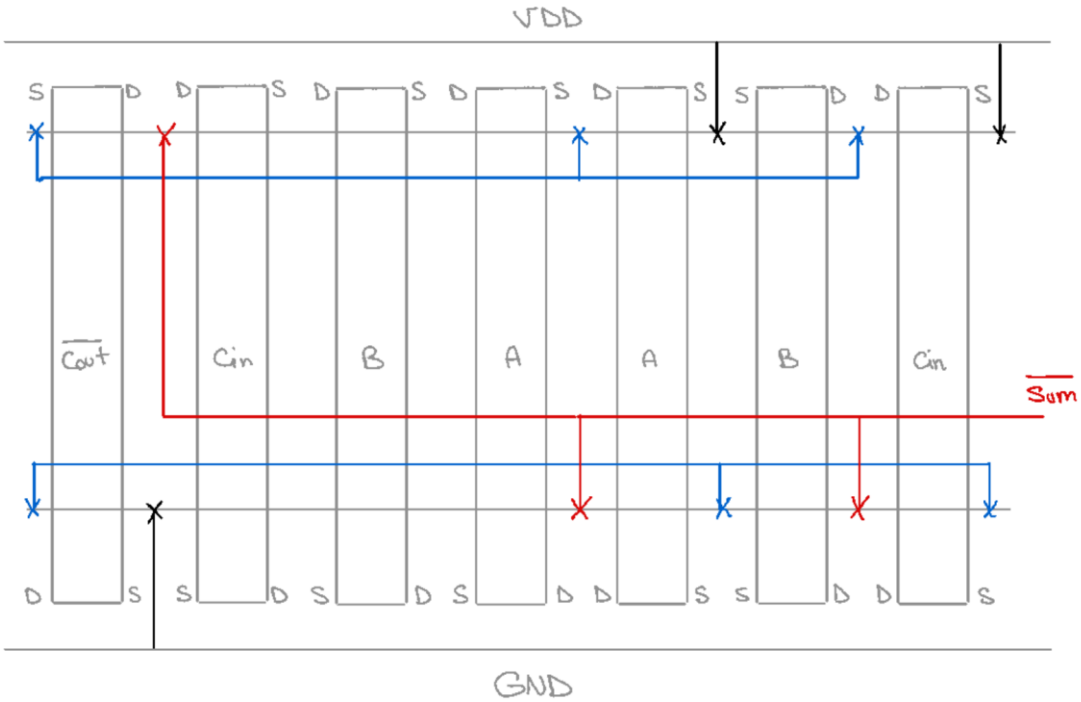


Figure 22 - Sum' Stick Diagram

After drawing our stick diagram, we can move to build layout for 1bit full adder and then change it to our 8-bit ripple carry adder.

Below you can see a layout for our 1bit full adder:

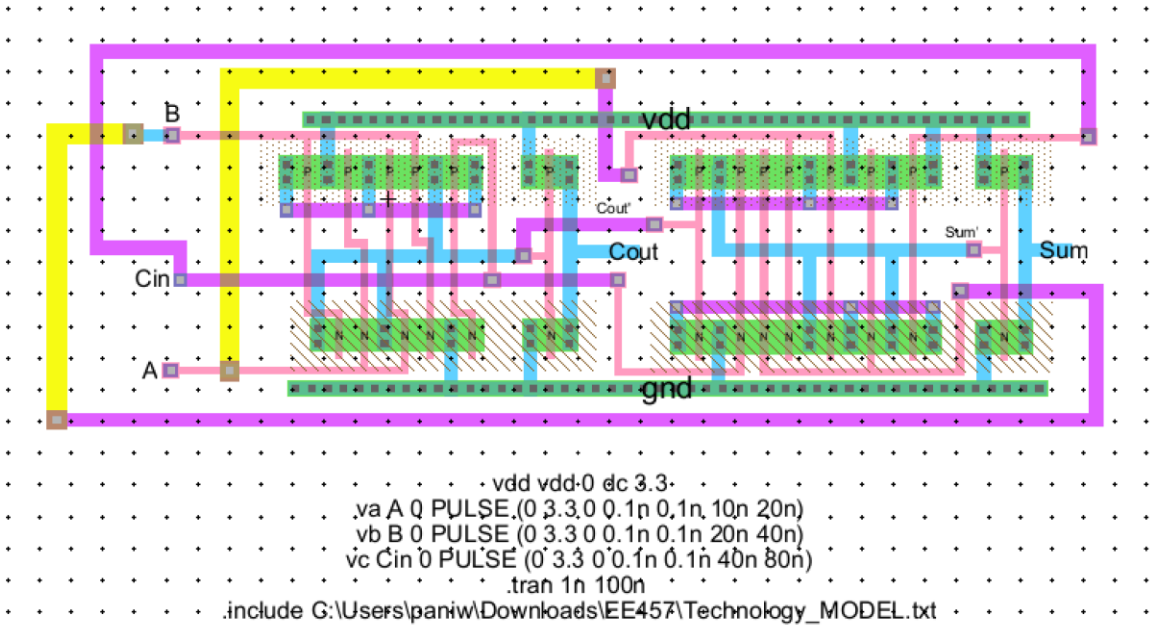


Figure 23 - 1bit Full Adder Layout

To make sure our design works properly, we tried the LTSpice and IRSIM. As you can see on the waveform for LTSpice, *Figure 24*, and IRSIM, *Figure 25*, below we have correct outputs based on the truth table.

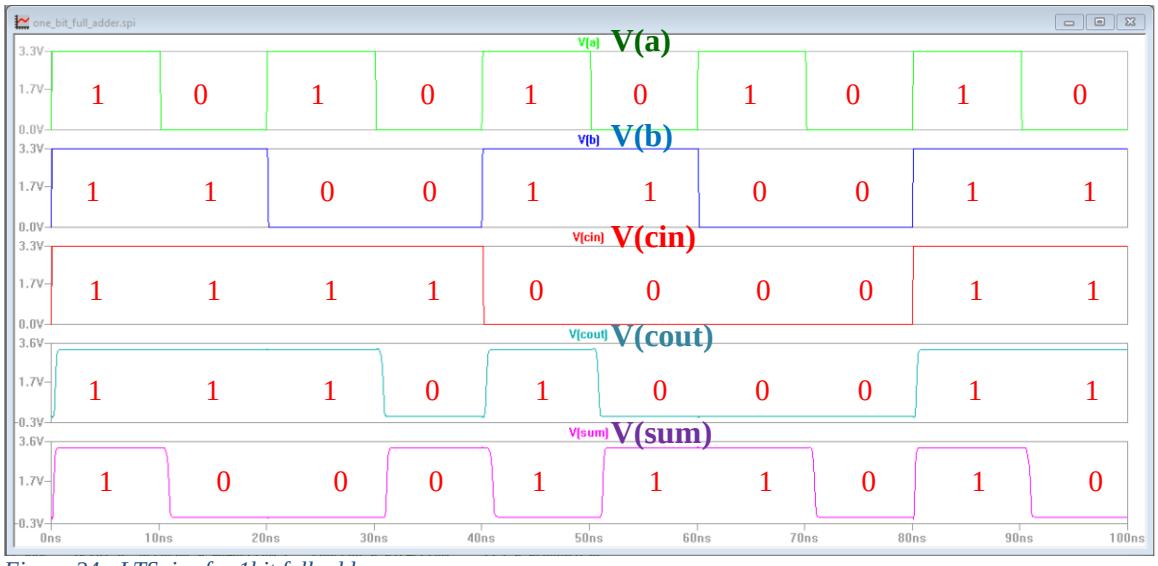


Figure 24 - LTSpice for 1bit full adder

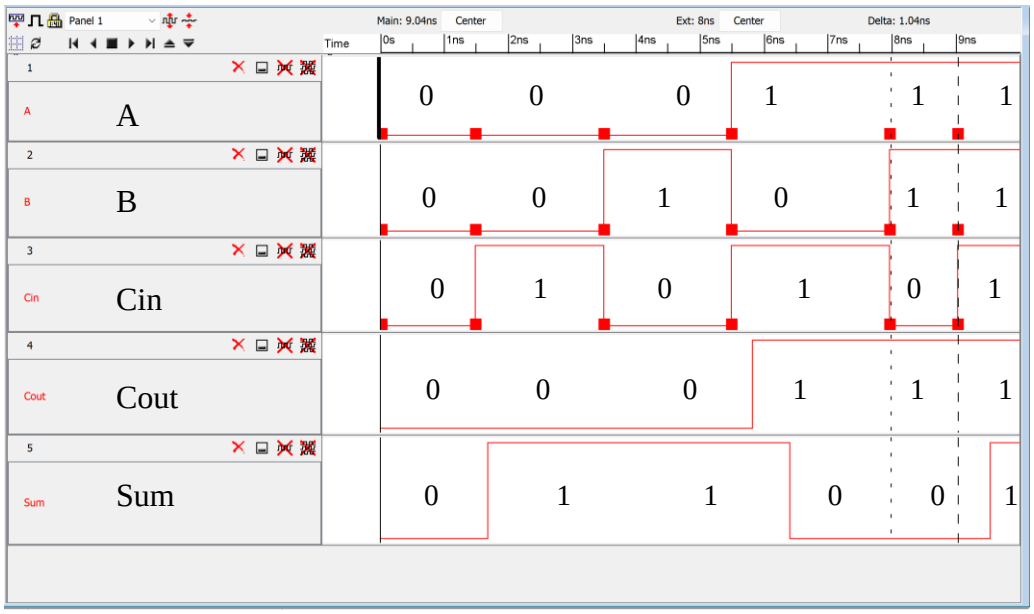


Figure 25 - IRSIM for 1bit Adder

Now that we have our layout for the 1-bit adder we can extend it to be 8it ripple carry adder. The Grid is on; however, due to the resolution, the grid is not available. However, if we go closer to the layout, we can see the Grid.

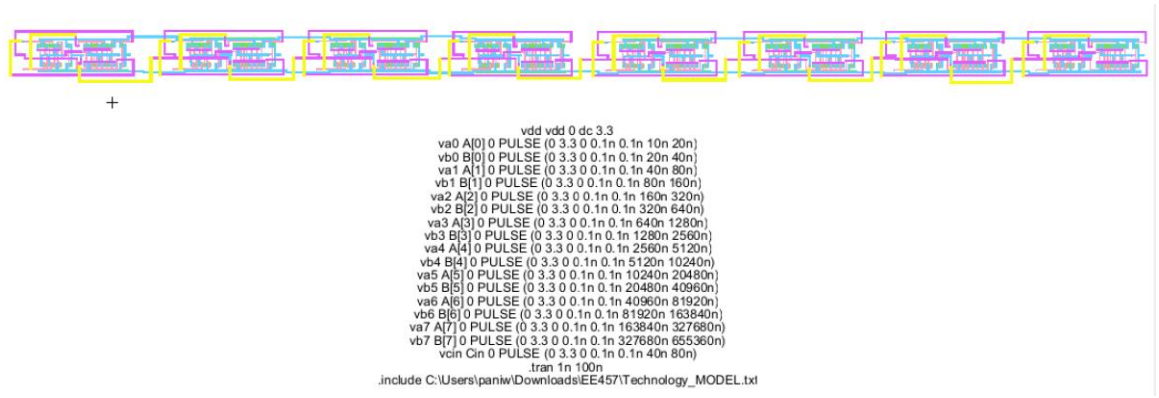


Figure 26 - 8bit ripple carry adder layout

In *Figure 27*, you can see a closer look at two adders that are connected to each other. 8 of these are connected to each other and make an 8-bit ripple adder at the end which you can see on *Figure 26*.

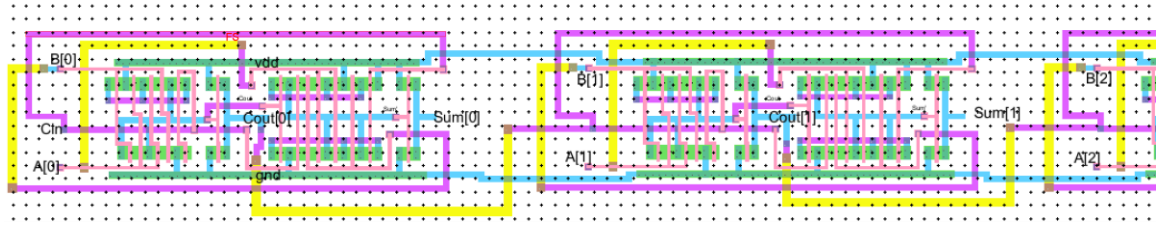


Figure 27 - Closer look for 8bit ripple adder

To make it more clear, our 8-bit ripple carry adder has A[0:7], B[0:7], Cin, Cout[0:7], and Sum[0:7]

To make sure our layout does not have any error, we check the DRC and Well Check as well.

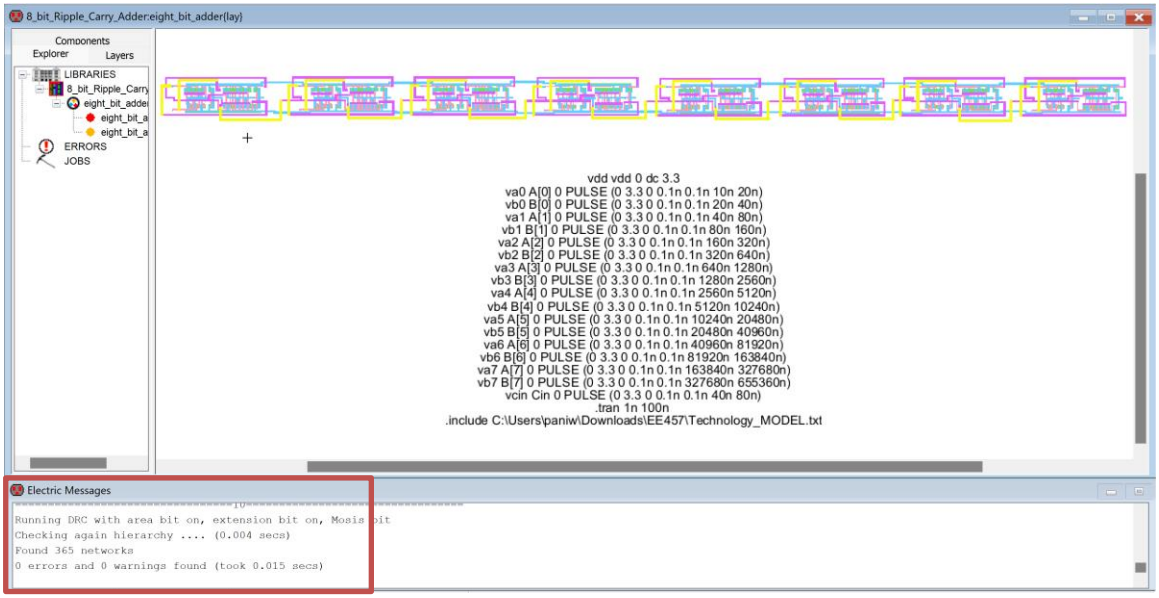


Figure 28 - DRC Check

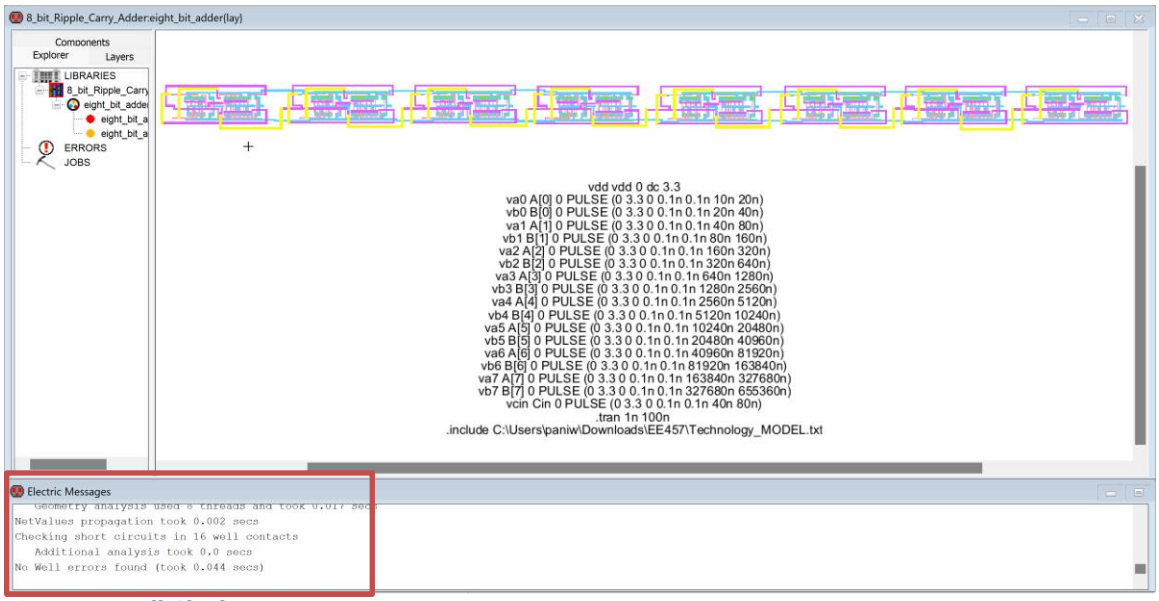


Figure 29 - Well Check

Section 7: LTSPICE Simulation for Layouts

Now that we have our layout, we will move to simulate our schematic using LTSpice. You can see the Spice Code that we use below.

```
* Spice Code nodes in cell cell 'eight_bit_adder{lay}'
vdd vdd 0 dc 3.3
va0 A[0] 0 PULSE (0 3.3 0 0.1n 0.1n 10n 20n)
vb0 B[0] 0 PULSE (0 3.3 0 0.1n 0.1n 20n 40n)
va1 A[1] 0 PULSE (0 3.3 0 0.1n 0.1n 40n 80n)
vb1 B[1] 0 PULSE (0 3.3 0 0.1n 0.1n 80n 160n)
va2 A[2] 0 PULSE (0 3.3 0 0.1n 0.1n 160n 320n)
vb2 B[2] 0 PULSE (0 3.3 0 0.1n 0.1n 320n 640n)
va3 A[3] 0 PULSE (0 3.3 0 0.1n 0.1n 640n 1280n)
vb3 B[3] 0 PULSE (0 3.3 0 0.1n 0.1n 1280n 2560n)
va4 A[4] 0 PULSE (0 3.3 0 0.1n 0.1n 2560n 5120n)
vb4 B[4] 0 PULSE (0 3.3 0 0.1n 0.1n 5120n 10240n)
va5 A[5] 0 PULSE (0 3.3 0 0.1n 0.1n 10240n 20480n)
vb5 B[5] 0 PULSE (0 3.3 0 0.1n 0.1n 20480n 40960n)
va6 A[6] 0 PULSE (0 3.3 0 0.1n 0.1n 40960n 81920n)
vb6 B[6] 0 PULSE (0 3.3 0 0.1n 0.1n 81920n 163840n)
va7 A[7] 0 PULSE (0 3.3 0 0.1n 0.1n 163840n 327680n)
vb7 B[7] 0 PULSE (0 3.3 0 0.1n 0.1n 327680n 655360n)
vcin Cin 0 PULSE (0 3.3 0 0.1n 0.1n 40n 80n)
.tran 1n 100n
.include C:\Users\paniw\Downloads\EE457\Technology_MODEL.txt
.END
```

Figure 30 - Layout Spice Code

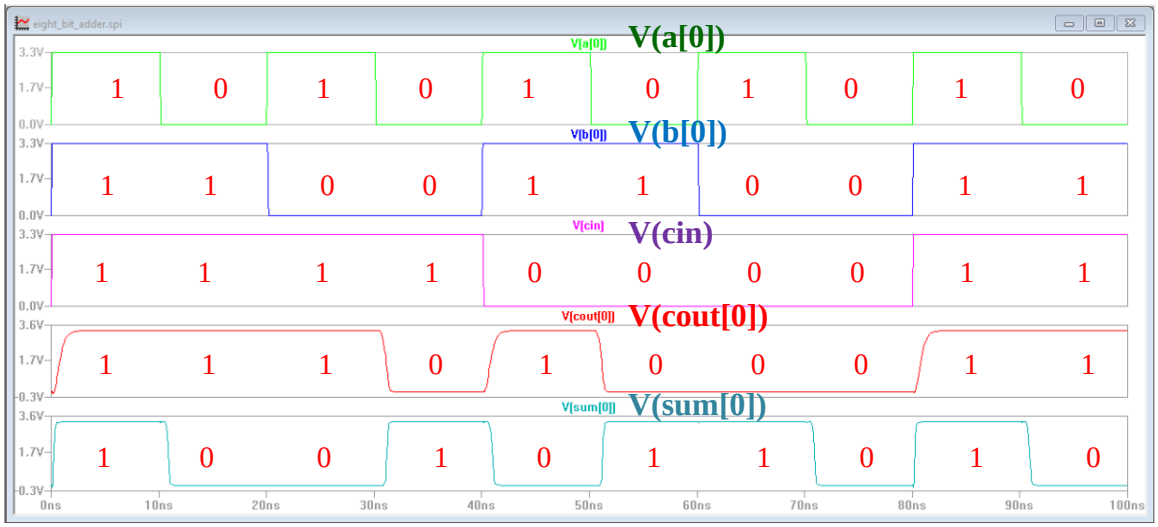


Figure 31 – LTSpice for 8bit ripple adder

To make sure our adder works correctly, we see the waveform of other inputs and outputs as well.

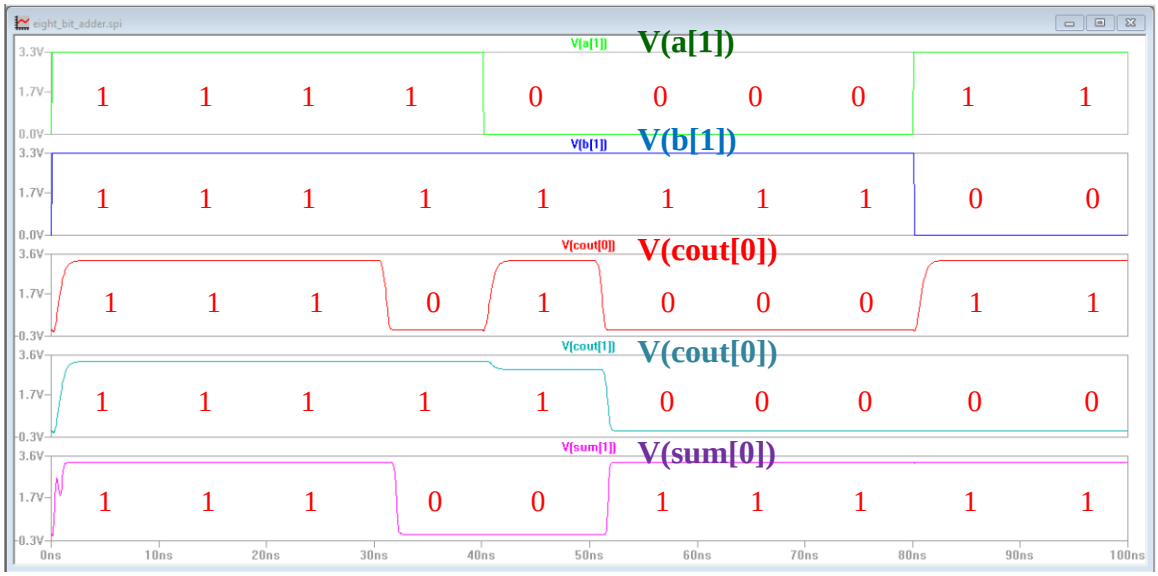


Figure 32 - LTSpice 8bit ripple adder

Section 8: IRSIM for Layout

Now that we checked the LTSpice for our 8-bit layout, we can check the IRSIM.

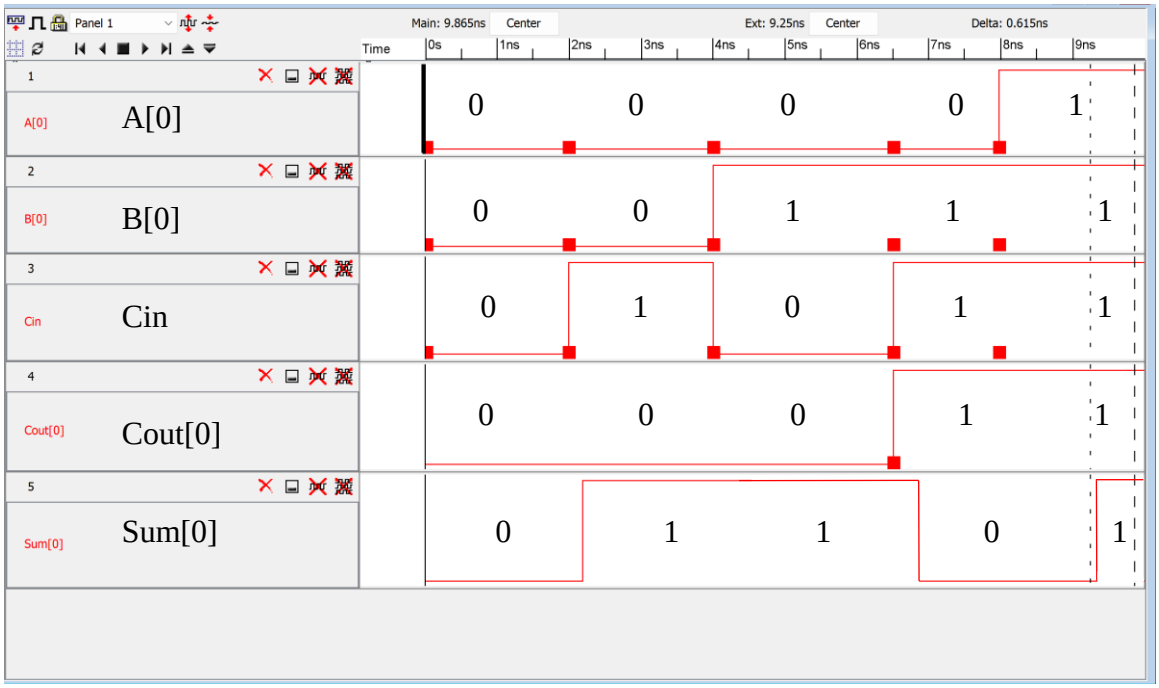


Figure 33 - IRSIM 8bit layout

To make sure the layout works perfectly, we try another inputs and outputs to double check the waveform. For instance, we tried A[1], B[1], Cout[0] which is our Cin for second adder, Cout[1], and Sum[1].

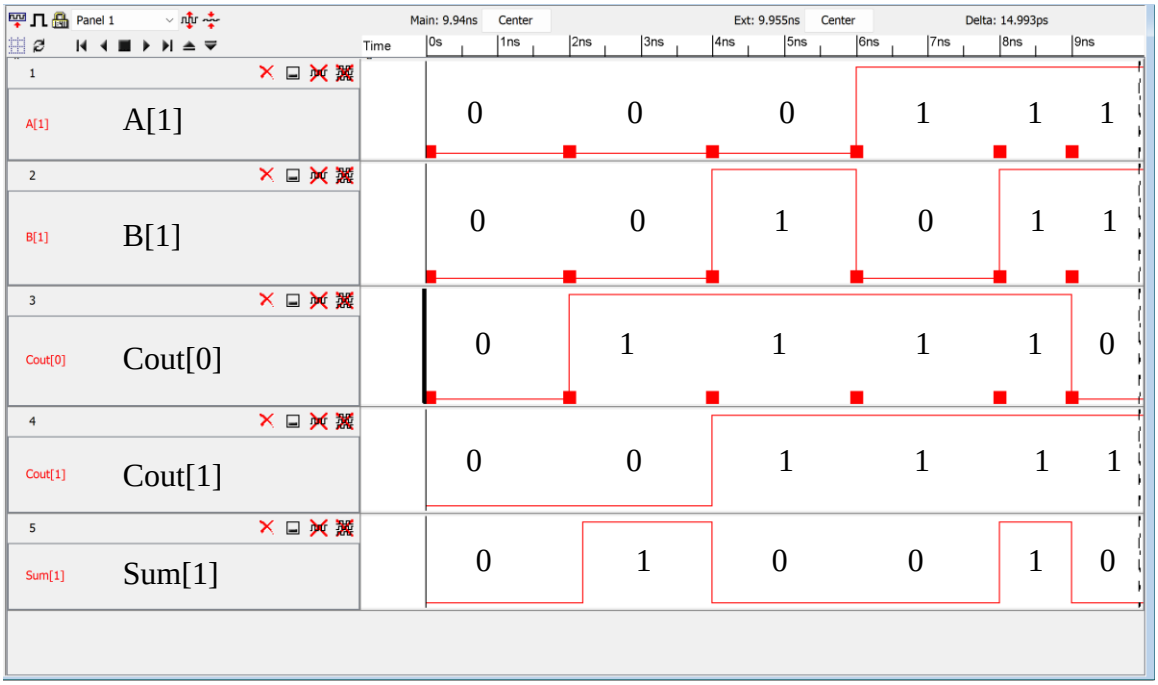


Figure 34 - IRSIM for 8th ripple adder layout

Now we can move on to calculate some binary numbers using IRSIM for the layout to double check our layout for 8-bit ripple carry adder. Below you can see the table of binary numbers that we are going to calculate.

Table 7 - First Addition

-32+107 = 75								
11100000 + 01101011 = (1) 01001011								
A[0:7]	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
11100000	0	0	0	0	0	1	1	1
B[0:7]	B[0]	B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]
01101011	1	1	0	1	0	1	1	0
Cin	0							
Cout[0:7]	Cout[0]	Cout[1]	Cout[2]	Cout[3]	Cout[4]	Cout[5]	Cout[6]	Cout[7]
00000111	0	0	0	0	0	1	1	1
Sum[0:7]	Sum[0]	Sum[1]	Sum[2]	Sum[3]	Sum[4]	Sum[5]	Sum[6]	Sum[7]
01001011	1	1	0	1	0	0	1	0

Table 8 - Second Addition

-28-16 = -44								
11100100 + 11110000 = (1)11010100								
A[0:7]	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
11100100	0	0	1	0	0	1	1	1
B[0:7]	B[0]	B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]
11110000	0	0	0	0	1	1	1	1
Cin	0							
Cout[0:7]	Cout[0]	Cout[1]	Cout[2]	Cout[3]	Cout[4]	Cout[5]	Cout[6]	Cout[7]
00000111	0	0	0	0	0	1	1	1
Sum[0:7]	Sum[0]	Sum[1]	Sum[2]	Sum[3]	Sum[4]	Sum[5]	Sum[6]	Sum[7]
11010100	0	0	1	0	1	0	1	1

Table 9 - Third Addition

+112+9 = 121								
01110000 + 00001001 = 1111001								
A[0:7]	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
01110000	0	0	0	0	1	1	1	0
B[0:7]	B[0]	B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]
00001001	1	0	0	1	0	0	0	0
Cin	0							
Cout[0:7]	Cout[0]	Cout[1]	Cout[2]	Cout[3]	Cout[4]	Cout[5]	Cout[6]	Cout[7]
00000000	0	0	0	0	0	0	0	0
Sum[0:7]	Sum[0]	Sum[1]	Sum[2]	Sum[3]	Sum[4]	Sum[5]	Sum[6]	Sum[7]
01111011	1	1	0	1	1	1	1	0

Table 10 - Fourth Addition

-71-29 = -100								
10111001 + 11100011 = (1)10011100								
A[0:7]	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
10111001	0	0	0	0	0	1	1	1
B[0:7]	B[0]	B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]
11100011	1	1	0	1	0	1	1	0
Cin	0							
Cout[0:7]	Cout[0]	Cout[1]	Cout[2]	Cout[3]	Cout[4]	Cout[5]	Cout[6]	Cout[7]
11100011	1	1	0	0	0	1	1	1
Sum[0:7]	Sum[0]	Sum[1]	Sum[2]	Sum[3]	Sum[4]	Sum[5]	Sum[6]	Sum[7]
10011100	0	0	1	1	1	0	0	1

As you can see below we applied our binary number to A[0:7] and B[0:7] and we get the correct hexadecimal value based on our Layout IRSIM. For instance,

For A[0:7] we have:
0xE0 = -32
0xE4 = -28
0x70 = +112
0xB9 = -71

For B[0:7] we have:
0x6B = +107
0xF0 = -16
0x09 = +9
0xE3 = -29

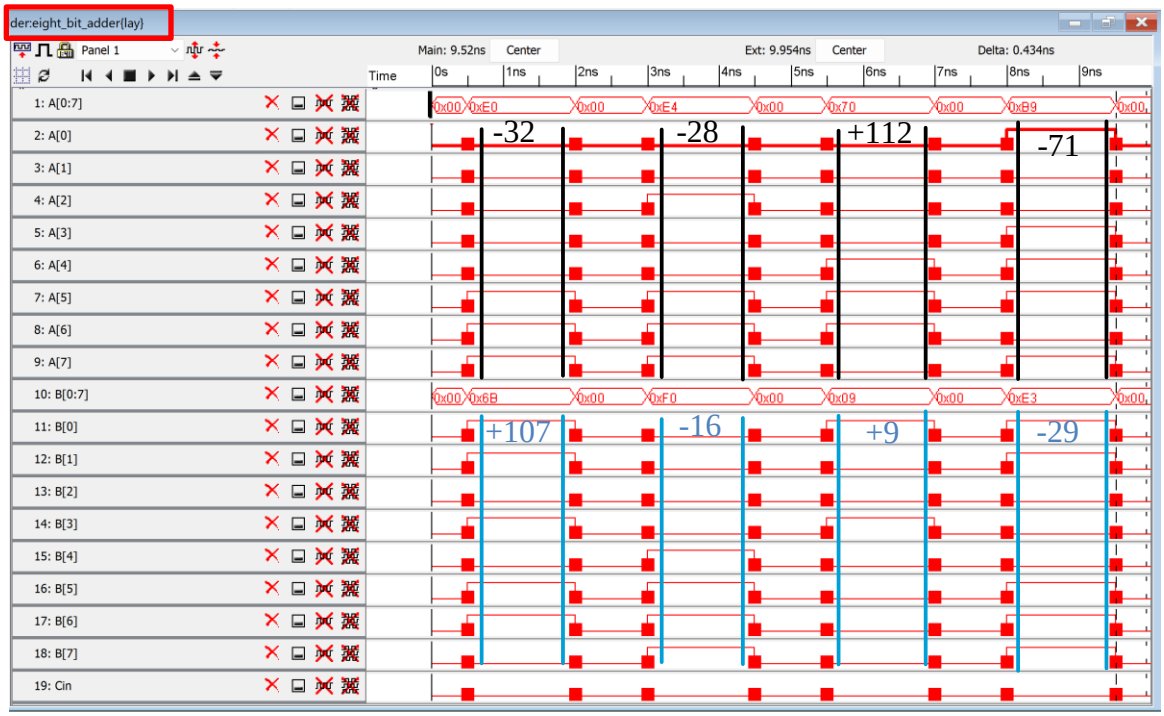


Figure 35 - Addition on IRSIM for A[0:7] and B[0:7] for Layout

For the first addition, -32+107 = 75, we used the IRSIM and as you can see in *Figure 36*, we got the correct result based on the Cout and Sum. You can compare the result from *Figure 36* to the content of *Table 7* to match the results.

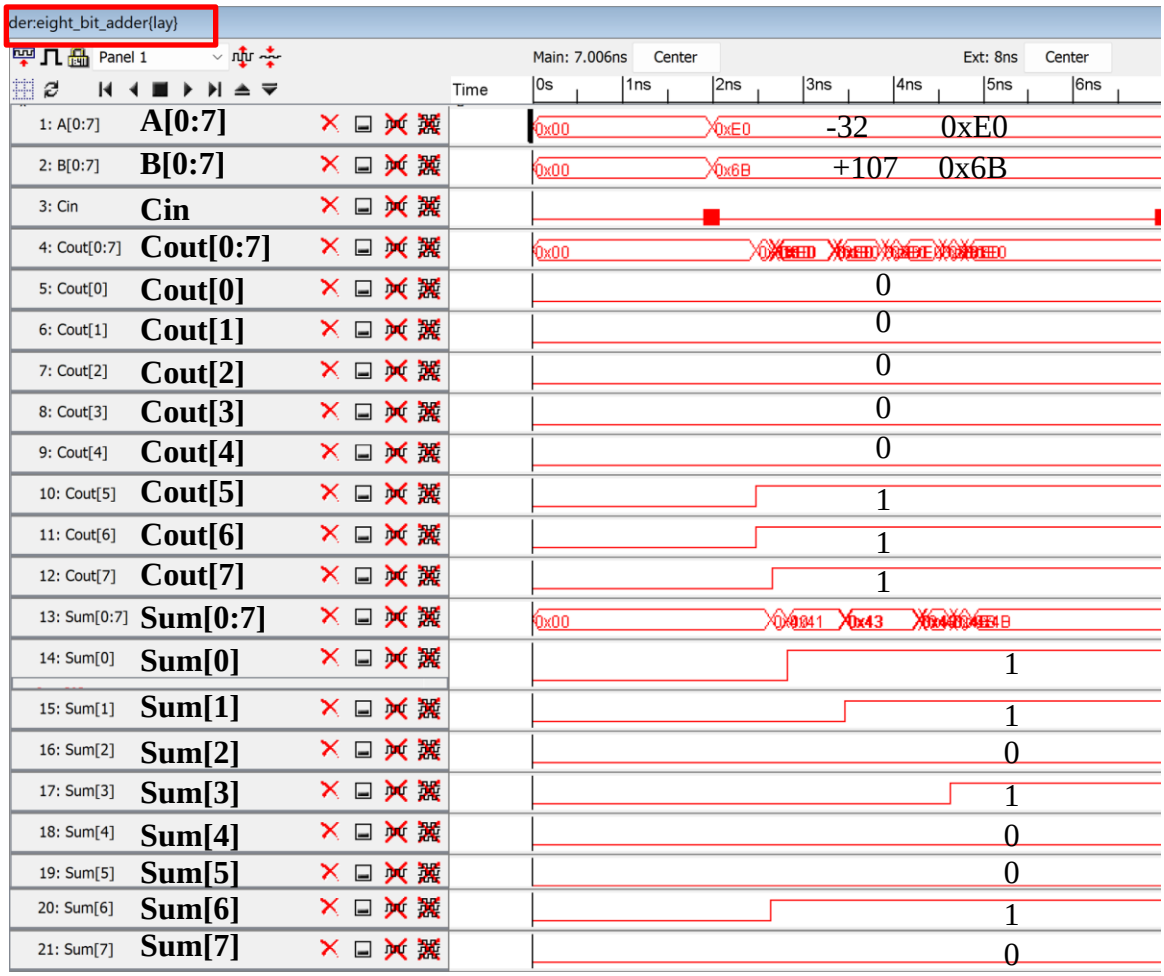


Figure 36 - (-32+107=75) for layout

For the second addition, -28-16 = -44, we used the IRSIM and as you can see in *Figure 37*, we got the correct result based on the Cout and Sum. You can compare the result from *Figure 37* to the content of *Table 8* to match the results.

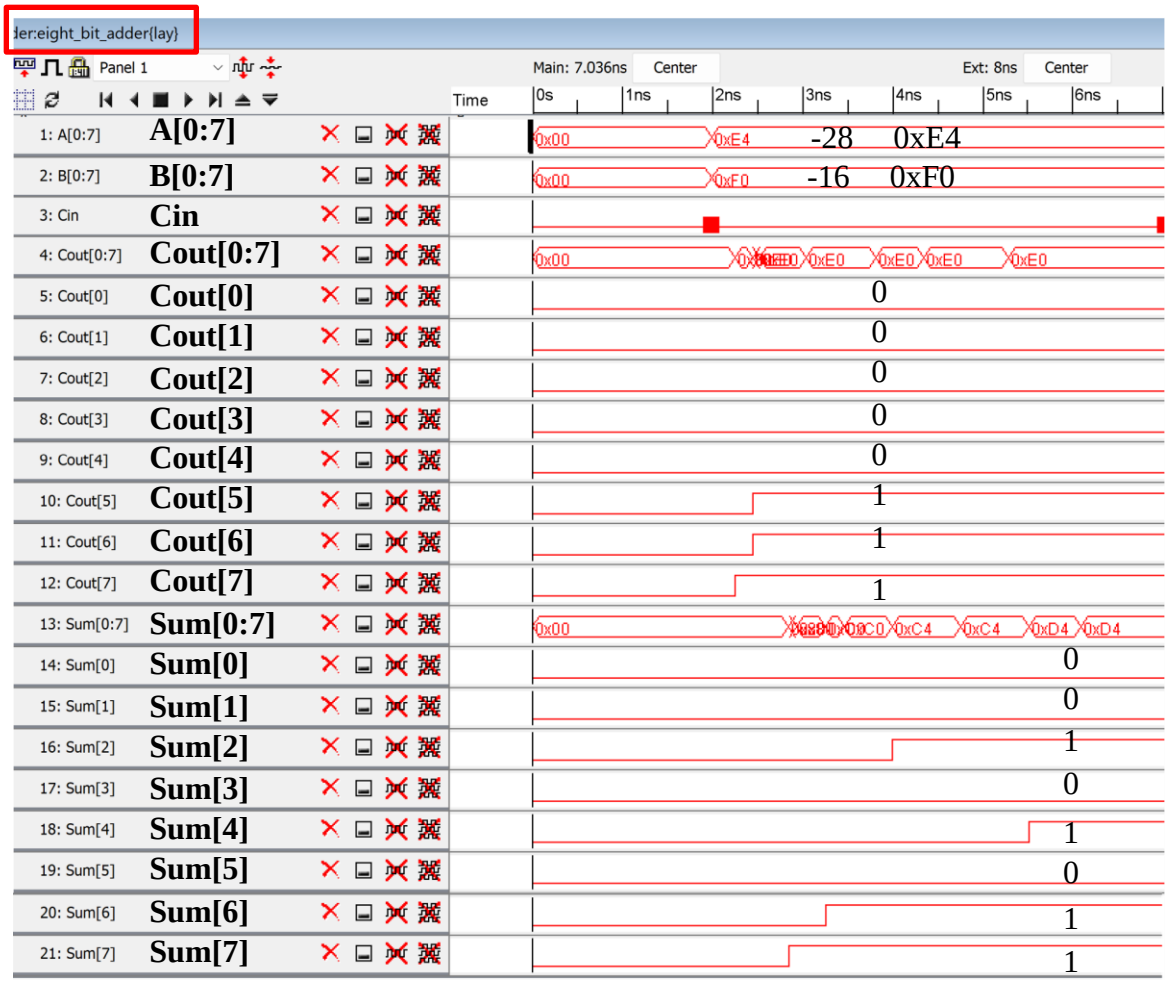


Figure 37- (-28-16=-44) for layout

For the third addition, +112+9 = +121, we used the IRSIM and as you can see in *Figure 38*, we got the correct result based on the Cout and Sum. You can compare the result from *Figure 38* to the content of *Table 9* to match the results.

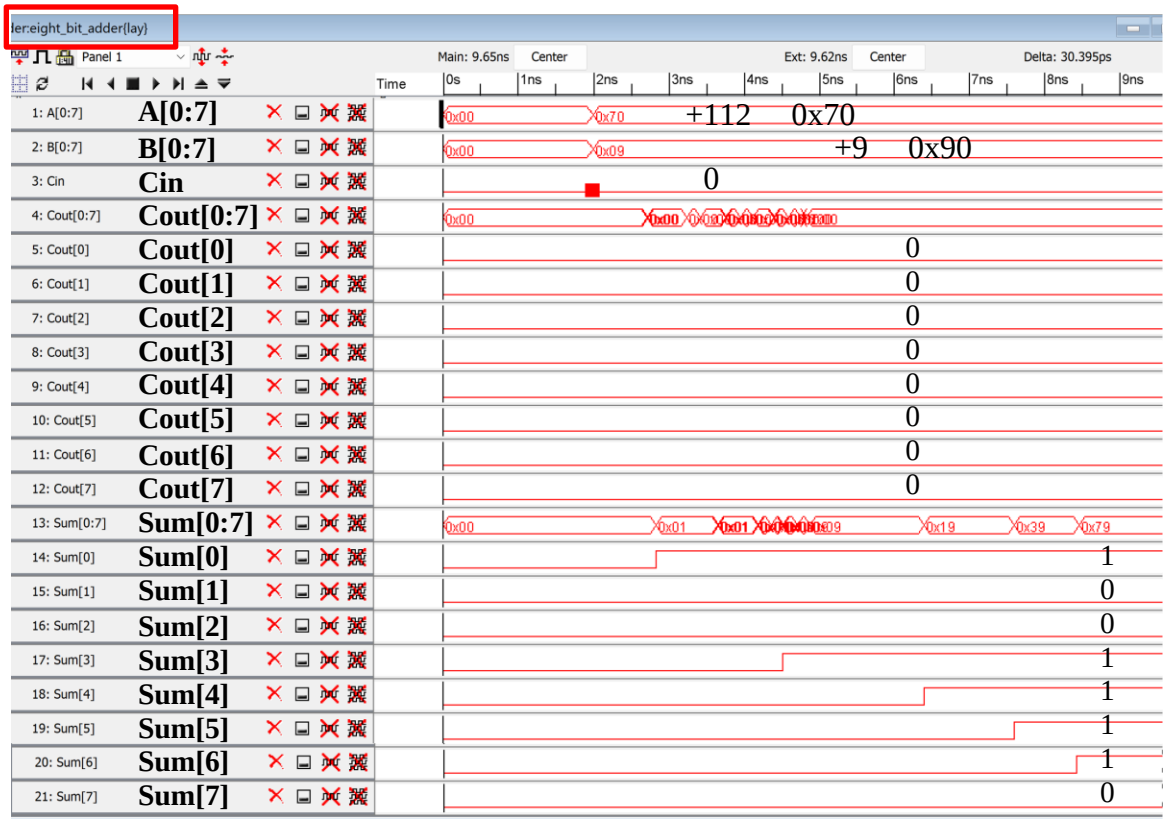


Figure 38 - (+112+9 = +121) for layout

For the fourth addition, -79-20=-100, we used the IRSIM and as you can see in *Figure 39*, we got the correct result based on the Cout and Sum. You can compare the result from *Figure 39* to the content of *Table 10* to match the results.

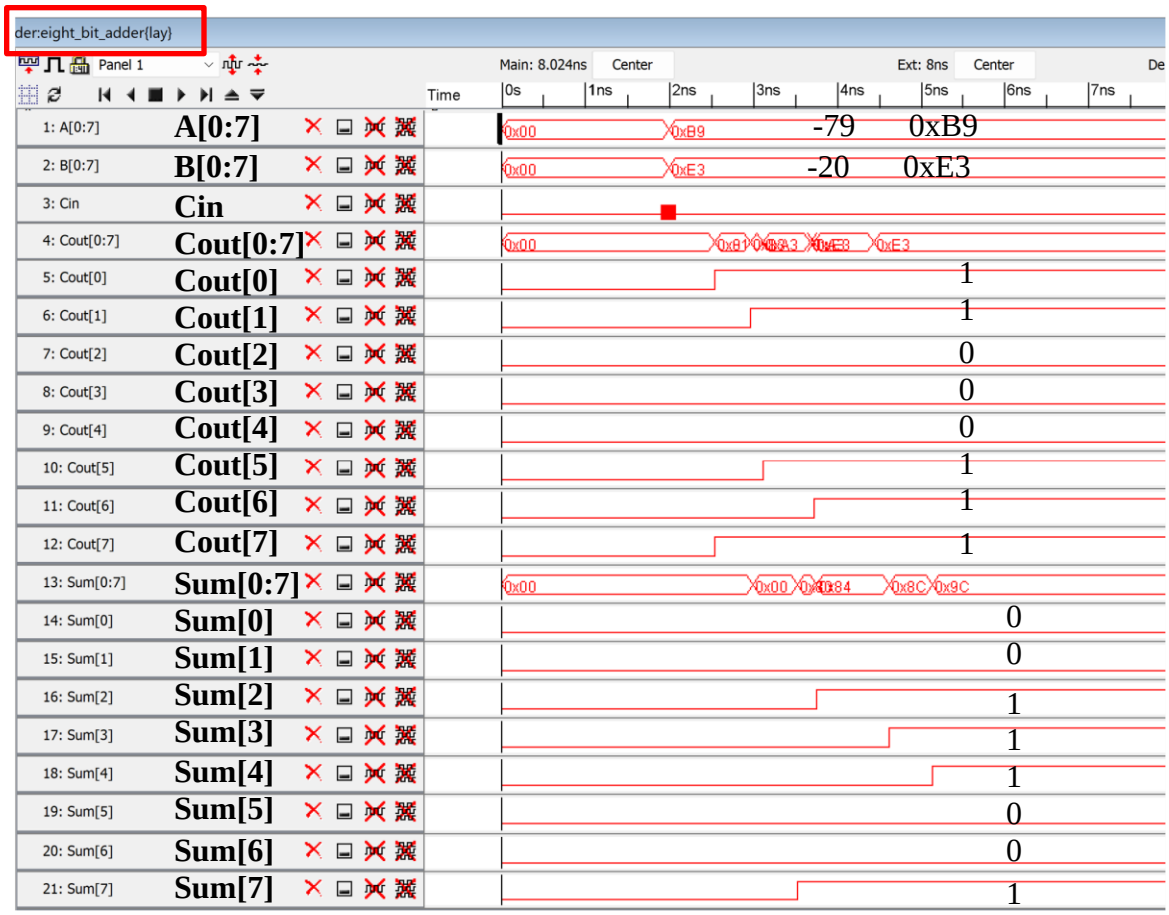


Figure 39 - (-79-20=-100) for layout

Section 9: Compare LTSPICE Measurements for Schematic and Layout

For this section, we will find fall time, rise tie, TPHL, TPLH. We are also going to find the Propagation Delay, TP. The formula to find propagation delay is:

$$tp = \frac{TPHL + TPLH}{2}$$

After we find the values, we will compare the values in a table for both schematic and layout.

To find the values we updated our spice code. You can see the updated Spice code in *Figure 40*.

```
vdd vdd 0 dc 3.3
va0 A[0] 0 PULSE (0 3.3 0 0.1n 0.1n 10n 20n)
vb0 B[0] 0 PULSE (0 3.3 0 0.1n 0.1n 20n 40n)
va1 A[1] 0 PULSE (0 3.3 0 0.1n 0.1n 40n 80n)
vb1 B[1] 0 PULSE (0 3.3 0 0.1n 0.1n 80n 160n)
va2 A[2] 0 PULSE (0 3.3 0 0.1n 0.1n 160n 320n)
vb2 B[2] 0 PULSE (0 3.3 0 0.1n 0.1n 320n 640n)
va3 A[3] 0 PULSE (0 3.3 0 0.1n 0.1n 640n 1280n)
vb3 B[3] 0 PULSE (0 3.3 0 0.1n 0.1n 1280n 2560n)
va4 A[4] 0 PULSE (0 3.3 0 0.1n 0.1n 2560n 5120n)
vb4 B[4] 0 PULSE (0 3.3 0 0.1n 0.1n 5120n 10240n)
va5 A[5] 0 PULSE (0 3.3 0 0.1n 0.1n 10240n 20480n)
vb5 B[5] 0 PULSE (0 3.3 0 0.1n 0.1n 20480n 40960n)
va6 A[6] 0 PULSE (0 3.3 0 0.1n 0.1n 40960n 81920n)
vb6 B[6] 0 PULSE (0 3.3 0 0.1n 0.1n 81920n 163840n)
va7 A[7] 0 PULSE (0 3.3 0 0.1n 0.1n 163840n 327680n)
vb7 B[7] 0 PULSE (0 3.3 0 0.1n 0.1n 327680n 655360n)
vcin Cin 0 PULSE (0 3.3 0 0.1n 0.1n 40n 80n)
.tran 1n 100n
.include C:\Users\paniw\Downloads\EE457\Technology_MODEL.txt
.meas rs TRIG v(Cout[0]) = 0.18 TD = 0 rise = 2 TARG v(Cout[0]) = 1.62 TD = 0 rise = 2
.meas ft TRIG v(Cout[0]) = 1.62 TD = 0 fall = 2 TARG v(Cout[0]) = 0.18 TD = 0 fall = 2
.meas TPHL TRIG v(Cin) = 0.9 TD = 0 rise = 1 TARG v(Cout[0]) = 0.9 TD = 0 rise = 2
.meas TPLH TRIG v(Cin) = 0.9 TD = 0 fall = 1 TARG v(Cout[0]) = 0.9 TD = 0 fall = 2
.meas propdelay param = (TPHL + TPLH)/2
.END
```

Figure 40 - Updated Spice Code for measurement

Table 11 - Measurement Comparison

	Rise Time	Fall Time	TPHL	TPLH	Propagation Delay
Schematic	0.0134117 ns	0.0191945 ns	40.0592 ns	40.1151ns	40.08715 ns
Layout	0.0751973 ns	0.0838327 ns	40.07389 ns	40.887 ns	40.480445 ns

Section 10: Layout Measurements of chip area, number of transistors

To measure the chip area of our layout first we need to find width and length of our layout. Based on the requirement we used 175nm or $\boldsymbol{\lambda}$=1 for our design. Electric has a measurement tool that helps us to find the width and length of our design. For instance, below you can see the width and length of our 1bit adder and then for our 8bit ripple carry design.

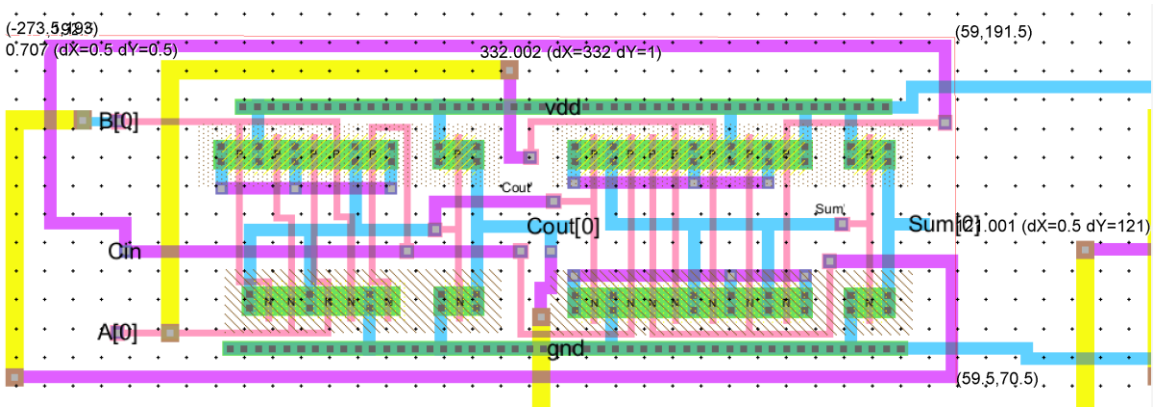


Figure 41 - Measurement for our 1bit adder

For 1bit we have:

Width: 121 $\boldsymbol{\lambda}$
Length: 332 $\boldsymbol{\lambda}$

$$121 \times 175nm = 21175nm = \frac{21175}{1000um} = 21.175um$$

$$332 \times 175nm = 58100nm = \frac{58100}{1000um} = 58.1um$$

Total Chip Area: $21.175um \times 58.1um = 1230.2675um^2$

Now for our 8bit ripple carry chip area:

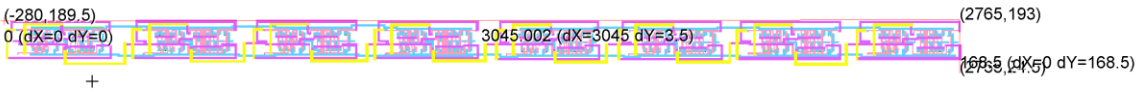


Figure 42 - Measurement for our 8bit carry adder

Width: 168 $\boldsymbol{\lambda}$
Length: 3045 $\boldsymbol{\lambda}$

$$168 \times 175nm = 29400nm = \frac{29400}{1000um} = 29.4um$$

$$3045 \times 175nm = 532875nm = \frac{532875}{1000um} = 532.875um$$

Total Chip Area: $29.4um \times 532.875um = 15666.525um^2$

Table 12 – Chip Area and Number of Transistor Measurement

	Number of Transistor	Chip Area
CMOS Schematic for 1bit Adder	28	X
SMOC Layout for 1bit Adder	28	$1{,}230.2675um^2$
CMOS Schematic for 8bit Ripple Carry Adder	224	X
CMOC Layout for 8bit Ripple Carry Adder	224	$15{,}666.525um^2$

Conclusion

In this project, we designed and analyzed an 8-bit binary ripple carry adder, a fundamental component in the realm of digital electronics. Through the processes, first, we designed a 1-bit adder and then designed an 8-bit adder by cascading eight 1-bit full adders.

To design a 1-bit adder, we tried some optimization to reduce the number of transistors. For instance, we reduced the number of transistors for each bit to 28 which for an 8-bit ripple carry adder is 224. For more information, you can check section two of this report. Once we have the design for our layout and schematic, we cascade it 8 times to make an 8-bit ripple carry adder. To be clearer, to cascade the adder the Cout of the previous adder was going to be Cin of the next adder. In order to design the layout for our 8-bit adder, we had to draw our stick diagram to see how many P-active and N-active we needed for the layout and how to wire them to have the correct functionality which you can see in section *6* of this report.

To check the functionality of our design, we used LTSpice and IRSIM to check the waveform based on the truth table. We also made some additions on IRSIM to see if our output bit for Cout and Sum is accurate. After attempting, we saw that the wave and output came out as it was expected; hence, our designed work as it should be.

After that, we started to measure the Rise Time, Fall Time, TPHL (Time Propagation High to Low), TPLH (Time Propagation Low to High), and Propagation Delay. To find these measurements, we added more details to our spice code and after simulating our design using LTSpice we checked the Spice Error Log. Once we had the results, we compared them on *Table 11*. As you can see in *Table 11*, the layout exhibited slightly higher rise and fall times. We also saw that the TPHL and TPLH realized that the layout's propagation delay is slightly higher than the schematic which can be acceptable due to limits for high-speed digital circuits. This minor increase can be attributed to the

additional physical factors encountered in a real-world environment, which are not present in the schematic simulation.

After that, we created a table to show the chip area for the 1-bit adder and 8-bit ripple carry adder layout and the number of transistors for the 1-bit and 8-bit ripple carry adder schematic and layout.

Overall, this project not only reinforced the theoretical understanding of the digital circuit design of an 8-bit ripple carry adder but also provided practical, hands-on experience in the creation of a digital circuit.

References

[1] GATEBOOK VIDEO LECTURES, "Ripple carry adder for N-Bits," YouTube, https://www.youtube.com/watch?v=koZbKyUikYw (accessed Feb. 24, 2024).

[2] ALL ABOUT ELECTRONICS, "Ripple carry adder explained (with solved example) | working and limitation of Ripple Carry Adder," YouTube, https://www.youtube.com/watch?v=b70ZQwci5sY (accessed Feb. 24, 2024).

[3] Explore the way, "Design of CMOS full adder || explore the way," YouTube, https://www.youtube.com/watch?v=i18ob8IdZ2o (accessed Feb. 29, 2024

[4] S. J. Shewale and S. A. Shirsath, "Design and analysis of CMOS full adder," International Journal of Advanced Research in Science, Communication and Technology (IJARSCT), https://ijarsct.co.in/Paper1902.pdf (March 1, 2024)

[5] A. Batool, "Design and implementation of Half Adder, full adder and CMOS full adder," Scribd, https://www.scribd.com/document/689240359/Design-and-Implementation-of-Half-Adder-Full-Adder-and-CMOS-Full-Adder (accessed Mar. 5, 2024).