Project 2 Final
Abel Chacko
ESE 370
November 3, 2020

1. Baseline

1.1. Design Schematics

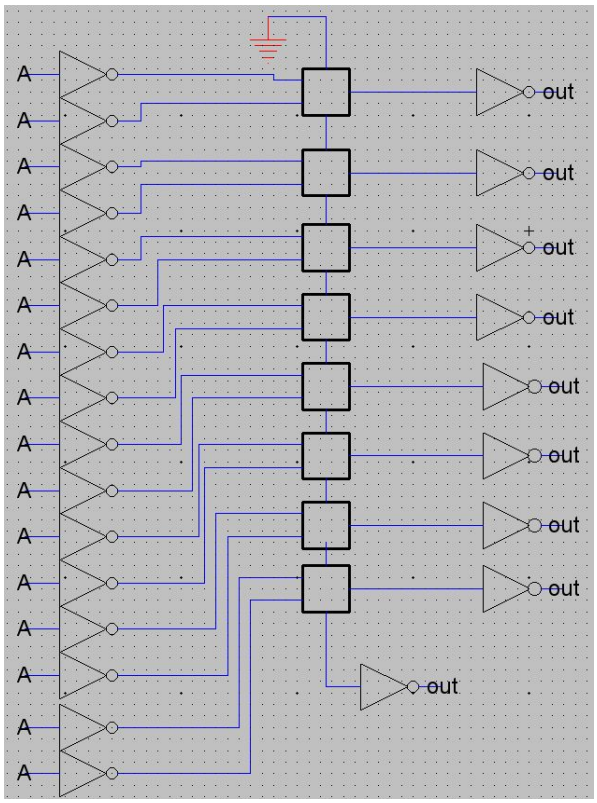


Figure 1.1

8-bit adder made from chaining the carry's of 8 1-bit full-adders. With inverters used as the input and output loads

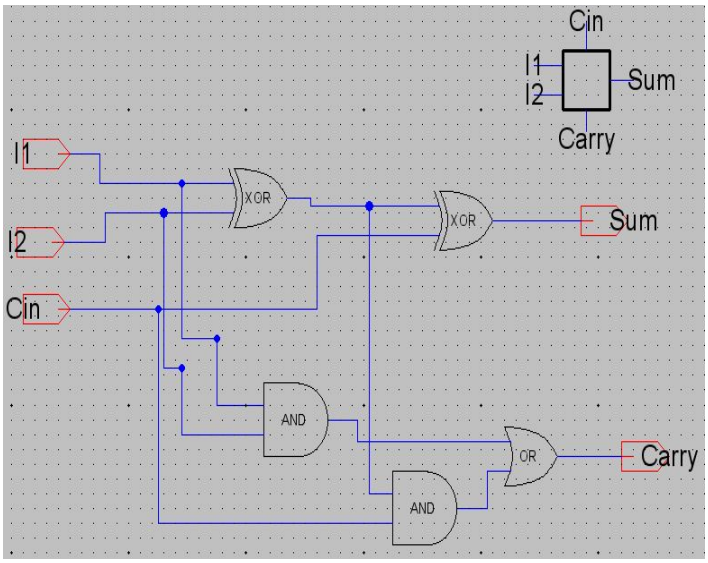


Figure 1.2: The construction of a single full-adder cell

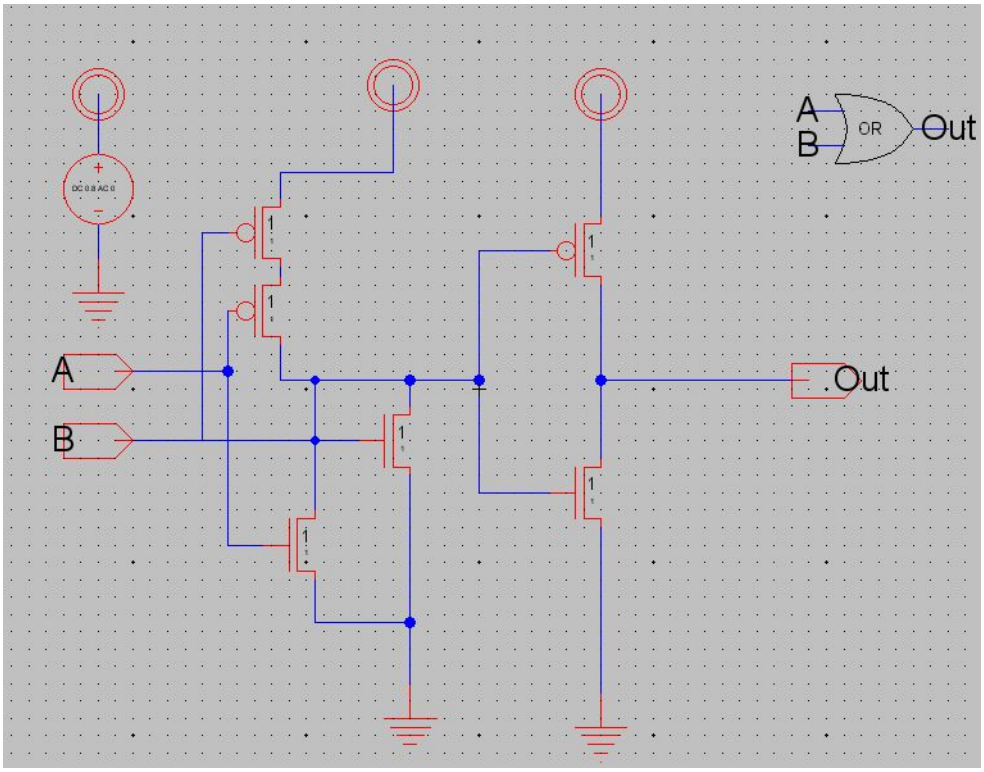


Figure 1.3: OR gate at minimum sizing

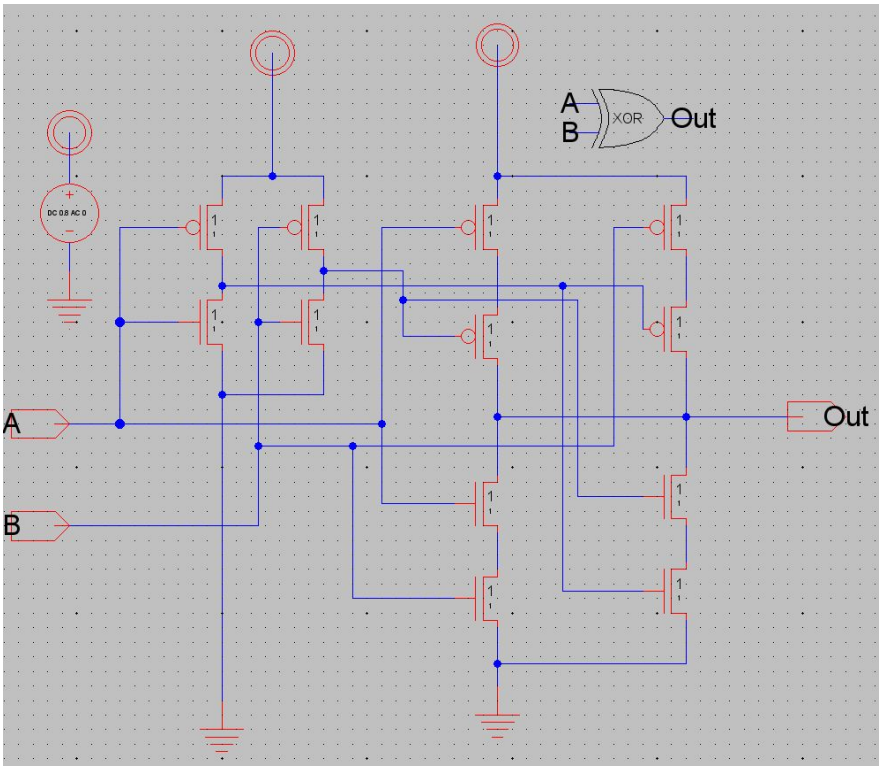


Figure 1.4: XOR gate at minimum sizing

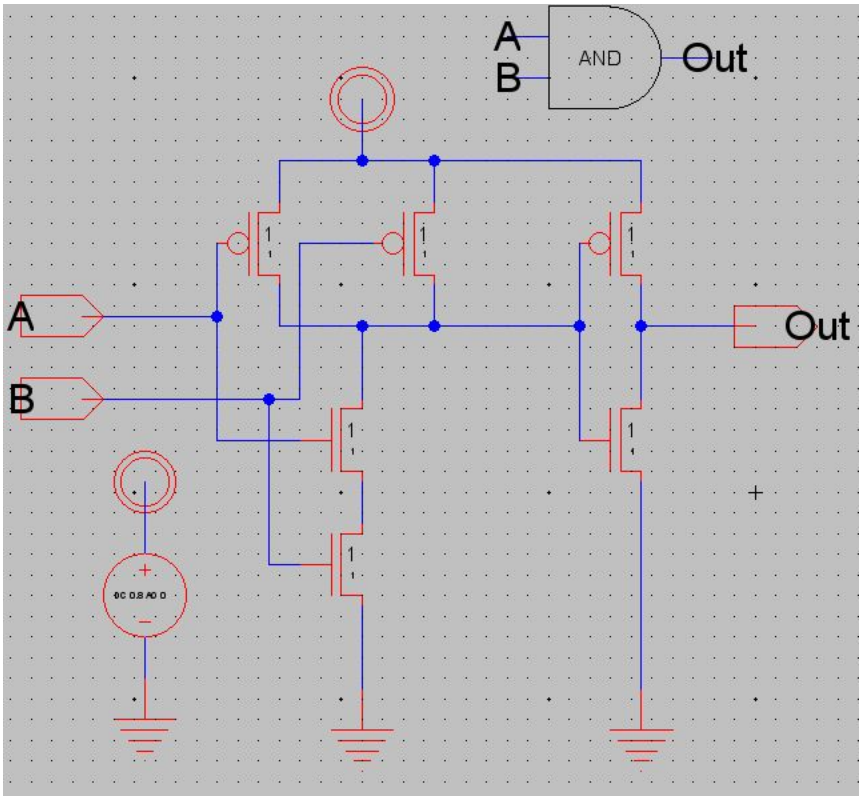


Figure 1.5: AND gate at minimum sizing

1.2. Description of Logic and Operation

In1	In2	Cin	Sum	Carry
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

Figure 1.6: Truth table for a full adder

The adder unit itself consists of 3 inputs that are used to generate 2 outputs. The inputs are the respective single bits of the inputs and the carry value from the last operation. The outputs are the sum and the carry.

For the sum, the inputs are XORed together to evaluate the sum as the only way to get the sum as a 1 would be for there to be a single high input or all high inputs, The XOR chain in the case of all 3 inputs would set the output of the first XOR to low making the output of the second XOR high because when the inputs to an XOR are high and low the output is high.
The carry is evaluated with 2 ANDs in parallel which are ORed together. One AND gate checks if In1 and In2 are both high which would give a carry. The other takes the output from the first XOR which tells if a either In1 or In2 was high and is evaluated with the Cin (the carry from the last bit) to see if both are on such that for all cases of either 3 or 2 bits of the input are high there will be a high for the carry output.

1.3. Delay Evaluation Using τ

First in the table below, there will be a breakdown of the load capacitances, for a single input, and maximum R_{on} of each of the gates used within the full adder. As gates like the AND and OR are made from NAND and NOR being inverted, in the table, they will be considered separately for the ease of evaluation.
Single Adder Break Down of Delay

Gate	C_{in} (Load Capacitance)	Max R_{on}
XOR	6	2/3
NAND	1	2
INV	2	1
NOR	2	2

Figure 1.7: Table providing capacitances and resistances of gates used

Now the worst case delay for the single adder can be found. This is the path going through the first XOR, taking its output into an AND gate then into the single OR gate to find output for the carry.
τ estimate evaluation:

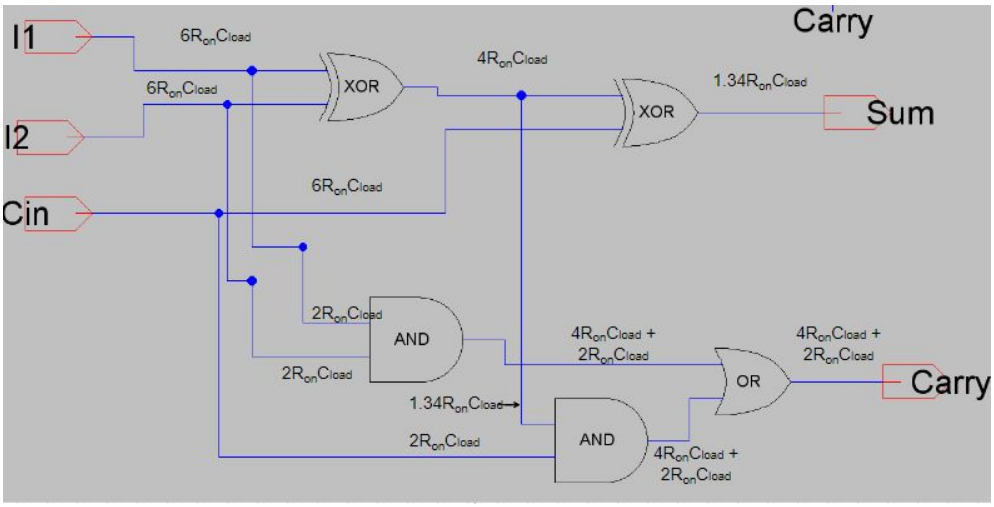


Figure 1.8: Annotated full adder with delays on each input

*Inputs from inverter

First XOR inputs:

$1R_{on} * 6C_{load} = 6\ R_{on}C_{load}$

Second XOR inputs:

First XOR:

$.67\ R_{on} * 6C_{load} = 4R_{on}C_{load}$

C_{in}:

$1R_{on} * 6C_{load} = 6R_{on}C_{load}$

Second XOR output:

$.67\ R_{on} * 2C_{load} = 1.34R_{on}$

I1 AND I2 inputs:

$1R_{on} * 2C_{load} = 2R_{on}C_{load}$

INV: $2R_{on} * 2C_{load} = 4R_{on}C_{load}$

C_{in} AND First XOR inputs:

First XOR:

$.67R_{on} * 2C_{load} = 1.34R_{on}C_{load}$

C_{in}:

$1R_{on} * 2C_{load} = 2R_{on}C_{load}$

OR inputs:

Setup:

INV: $1R_{on} * 2C_{load} = 2R_{on}C_{load}$

NAND: $2R_{on} * 2C_{load} = 4R_{on}C_{load}$

AND: NAND + INV = $6R_{on}C_{load}$

OR output

NOR: $R_{on} * 2C_{load} = 2R_{on}C_{load}$

OR: NOR + INV = $4R_{on}C_{load}$

Maximum Delay:

First XOR + (C_{in} AND First XOR) + OR = 6 $R_{on}C_{load}$ + 1.34$R_{on}C_{load}$ + 6$R_{on}C_{load}$ + 4$R_{on}C_{load}$ + 2$R_{on}C_{loa}$= **19.34$R_{on}C_{load}$**

Max delay of single full adder = 19.34$R_{on}C_{load}$

Thus for carry to be evaluated through all 8 bit slices, 154.72$R_{on}C_{load}$ (single delay * 8) is the total delay to the final carry

This delay can occur when the first two inputs are 1 and the rest have a single 1 for the addition (11111111 + 00000001). This will produce a delay which is about equal to what is calculated above. The first adder will not give the worst delay for the first XOR as in order to get a carry and the XOR to have worst case delay there would need to be a carry in from the previous adder. That is why a carry out is being assured. Thus after the first bit slice, the rest of the additions will have high coming out of the first XOR so that the path is delayed through the XOR into the AND with a high carry in. This will turn on the nMOS with the AND gate in series so the R_{drive} will be $2R_{on}$. This however does limit us to a single transistor for the NOR gate as the drive. However, as we are able to go through the XOR and NAND at maximum delay we will produce the greatest delay of any of the inputs.

1.4. Validation of Logical Correctness

First for a check, the single full adder will be tested such that all types of output are tested ((S=0, C=0), (S=0, C=1),(S=1, C=0),(S=1, C=1)). A 1 represents a high output Vdd = .8 V ideally. A 0 is a low at 0 V ideally.

```
ngspice 1 -> op
```
to conduct the DC solver

```
print sum carry
```
to measure the values at sum and carry

Test Case In_1= 0, In_2= 0, C_{in} = 0:

Ideal Expected Output: Sum=0, Carry=0

Measured Value:
```
sum = 5.335946e-05
carry = 2.324112e-05
```

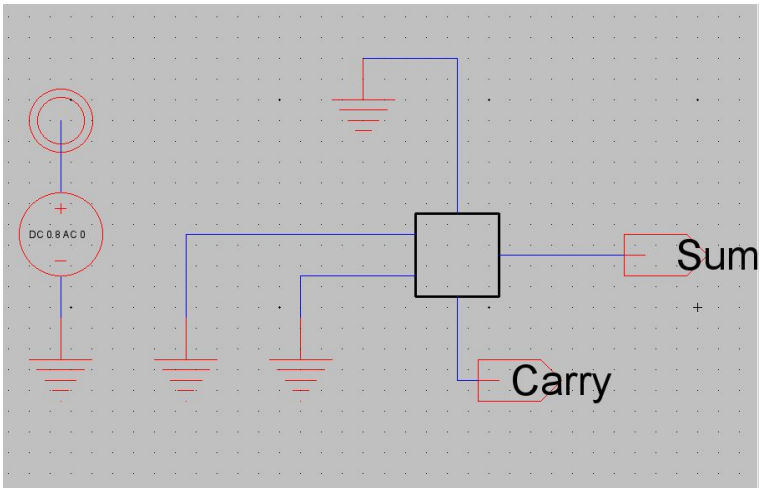


Figure 1.9: Circuit for test case of all 0 inputs into full adder

Test Case In_1= 0, In_2= 0, C_{in} = 1:

Ideal Expected Output: Sum=1, Carry=0

Measured Value:

```
sum = 7.999746e-01
carry = 2.324112e-05
```

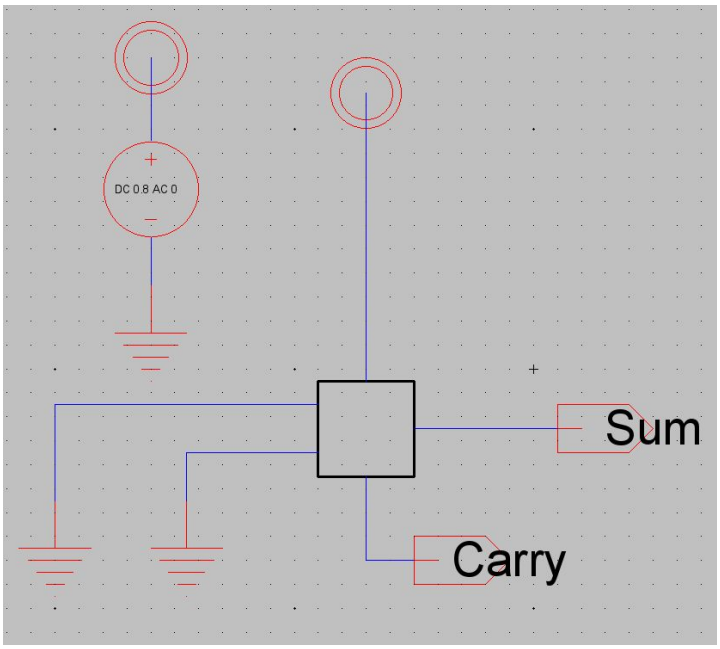


Figure 1.10: Circuit for test case of In_1= 0, In_2= 0, C_{in} = 1

Test Case In_1= 0, In_2= 1, C_{in} = 1:

Ideal Expected Output: Sum=0, Carry=1

Measured Value:

```
sum = 1.436952e-05
carry = 7.999488e-01
```

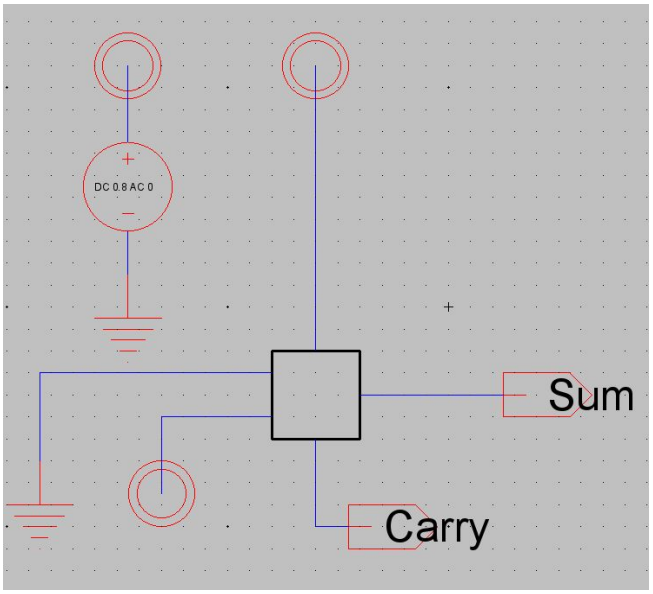


Figure 1.11: Circuit for test case of In_1= 0, In_2= 1, C_{in} = 1

Test Case In_1= 1, In_2= 1, C_{in} = 1:

Ideal Expected Output: Sum=1, Carry=1

Measured Value:

```
sum = 7.998723e-01
carry = 7.999488e-01
```

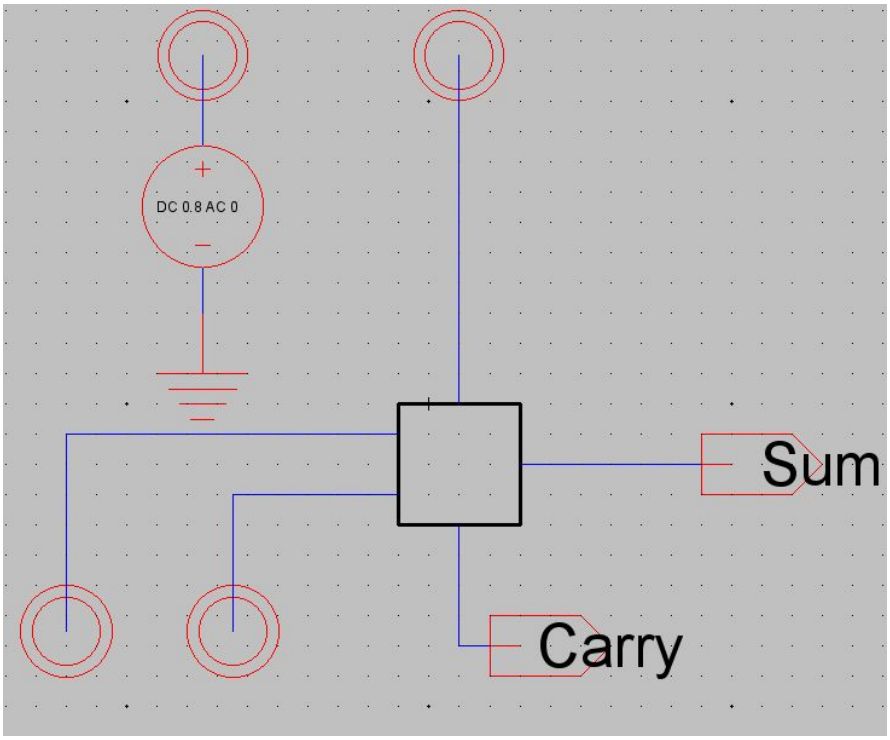


Figure 1.12: Circuit for test case of In_1= 1, In_2= 1, C_{in} = 1

It is possible to test all the inputs in a single case through the use of the transient simulation and observing the plot of the carry and sum based on the inputs

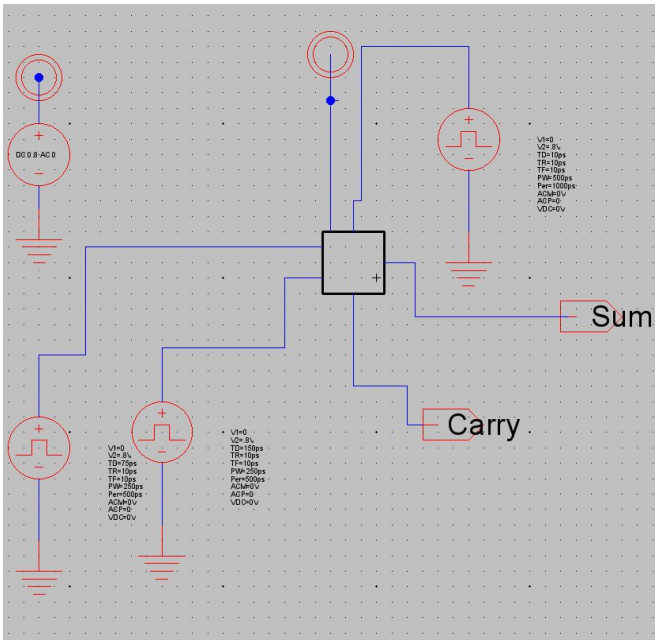


Figure 1.12.1: Using 3 voltage pulse generators at different pulse length all 8 inputs can tested in a single run for the 1-bit full adder

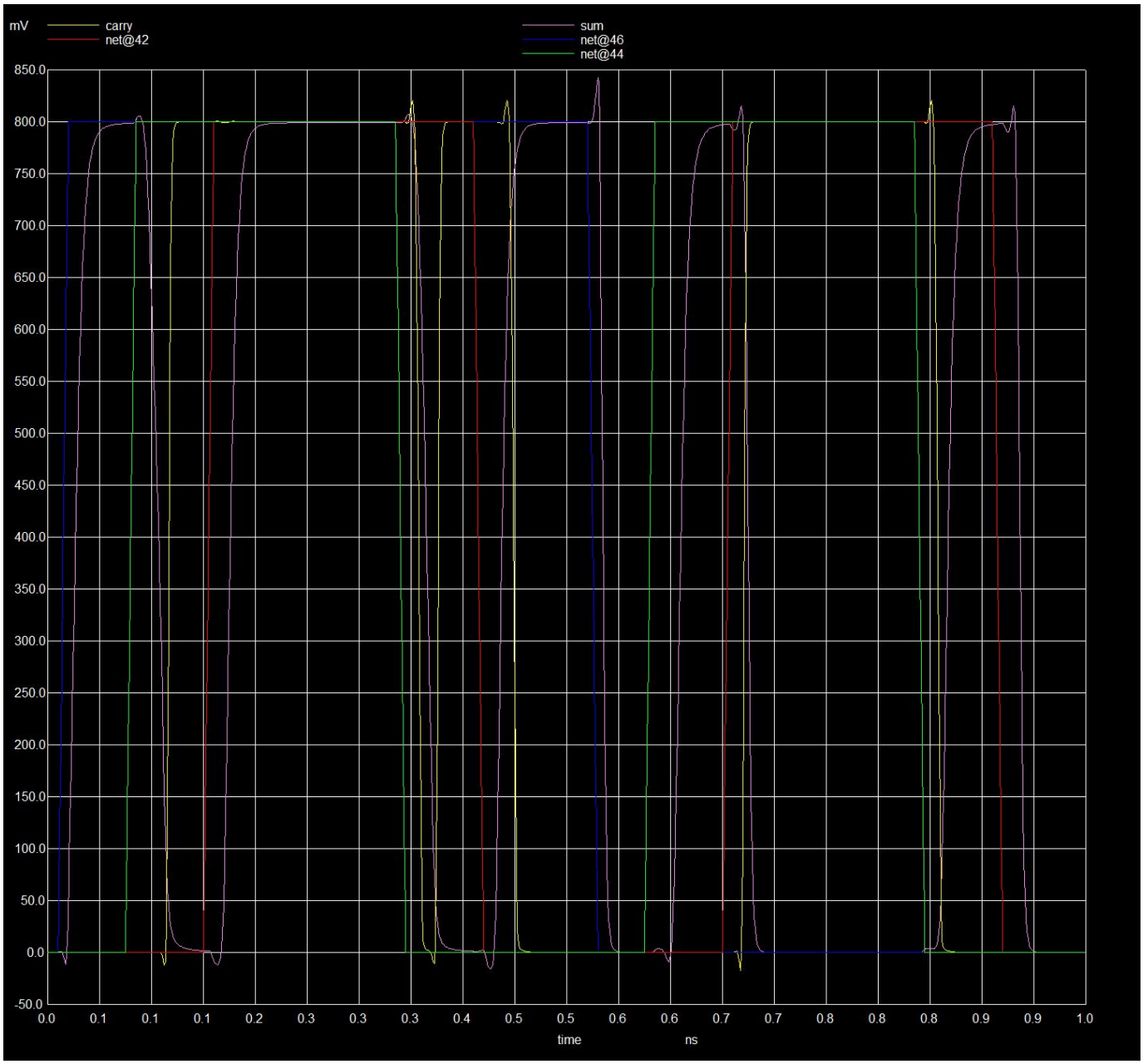


Figure 1.12.1: A plot of all inputs, the sum and the carry

Now it is time to test the 8-bit addition:

Once again will use the op command in ngspice to solve for the DC output.

As it is unreasonable to test all 256 possible, thus it is best to select a few test cases that can cover the general scope of the 8-bit adder. The test cases being an addition of all 0's on both inputs ($In_1 = 0$, $In_2 = 0$), the test of 0's on one input and 1's on the other (($In_1 = 255$, $In_2 = 0$), ($In_1 = 0$, $In_2 = 255$)), and the addition of 1's on all inputs ($In_1 = 255$, $In_2 = 255$).

Command used to print out all output bits:

```
print  Bit1  Bit2  Bit3  Bit4  Bit5  Bit6  Bit7  Bit8  Bit9
```

Test Case In_1= 0, In_2= 0:

Ideal Expected Output: 0 (All outputs low)

Measured Output:

```
bit1 = 9.234667e-05
bit2 = 9.234670e-05
bit3 = 9.234670e-05
bit4 = 9.234670e-05
bit5 = 9.234670e-05
bit6 = 9.234670e-05
bit7 = 9.234670e-05
bit8 = 9.234670e-05
bit9 = 2.325417e-05
```

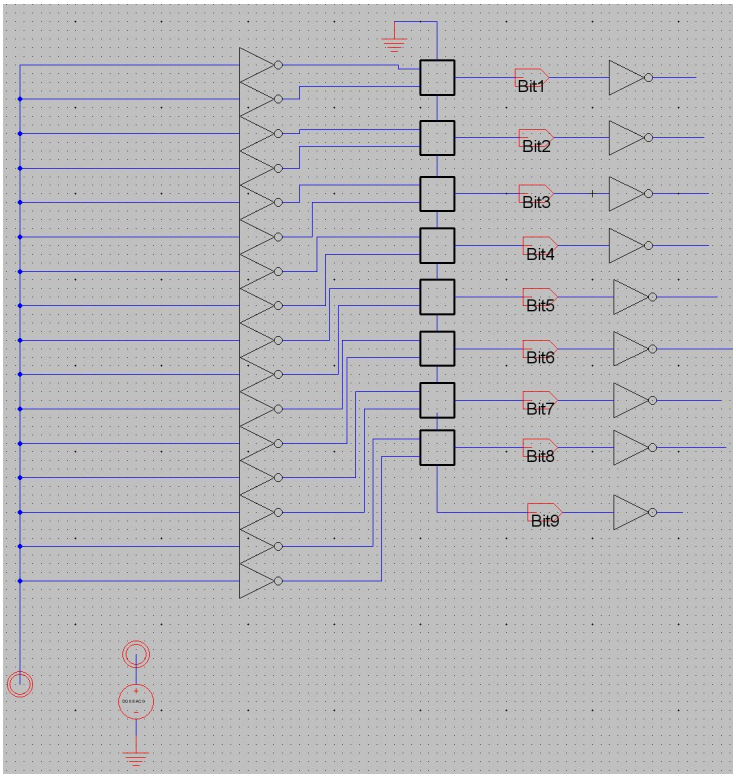


Figure 1.13: Circuit for test case of all inputs 0

Test Case In_1= 255, In_2= 0:

Ideal Expected Output: 255 (All outputs high except for final carry)

Measured Output:

```
bit1 = 7.998718e-01
bit2 = 7.998717e-01
bit3 = 7.998717e-01
bit4 = 7.998717e-01
bit5 = 7.998717e-01
bit6 = 7.998717e-01
bit7 = 7.998717e-01
bit8 = 7.998717e-01
bit9 = 2.325417e-05
```

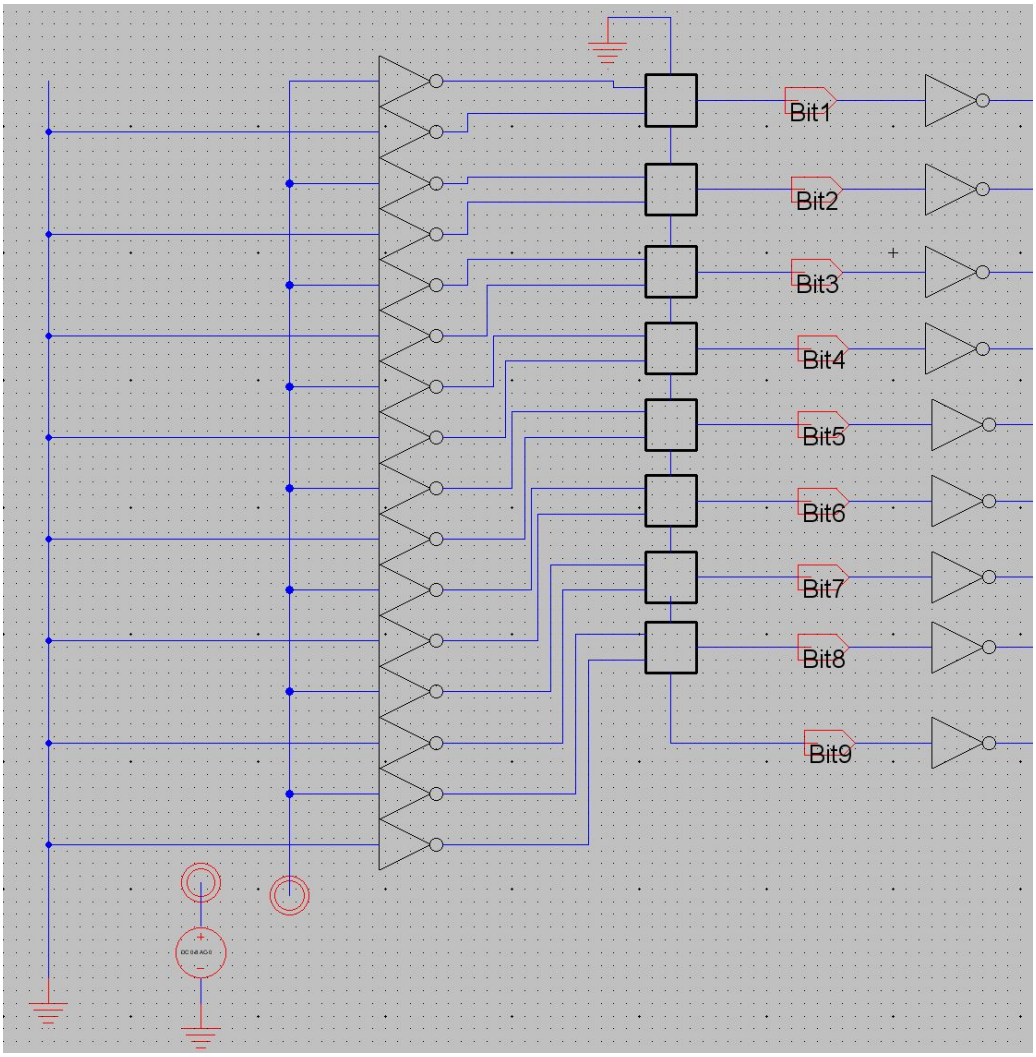


Figure 1.14: Circuit for test case of all inputs 1

Test Case In_1= 0, In_2= 255:

Ideal Expected Output: 255 (All outputs high except for final carry)

Measured Output:

```
bit1 = 7.998718e-01
bit2 = 7.998717e-01
bit3 = 7.998717e-01
bit4 = 7.998717e-01
bit5 = 7.998717e-01
bit6 = 7.998717e-01
bit7 = 7.998717e-01
bit8 = 7.998717e-01
bit9 = 2.325417e-05
```

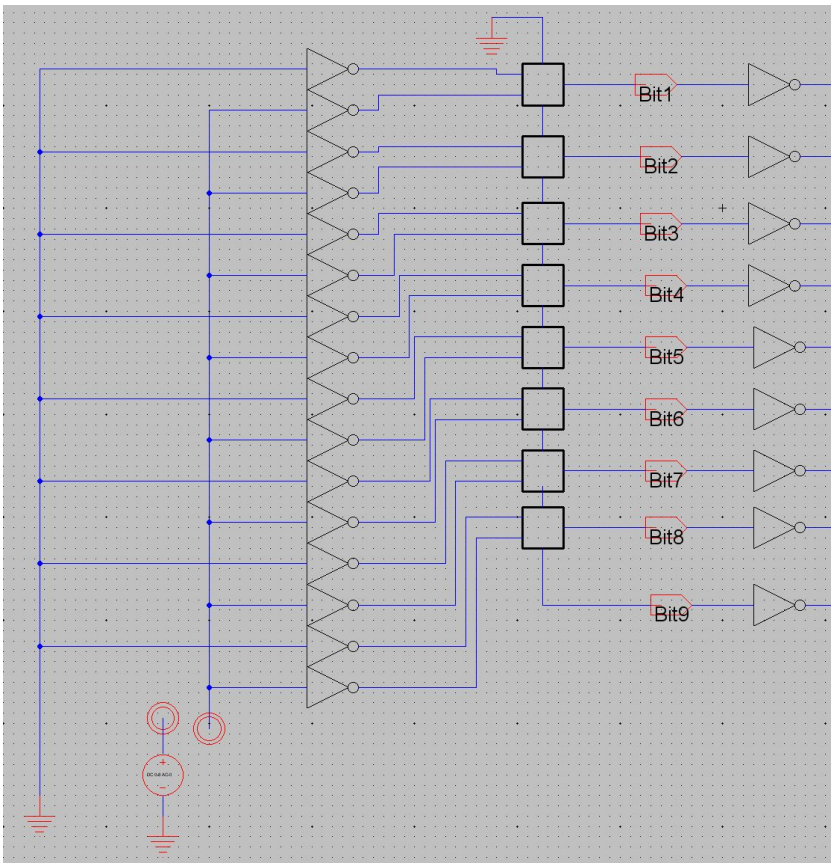


Figure 1.15: Circuit for In_1= 0, In_2= 255

Test Case In_1= 255, In_2= 255:

Ideal Expected Output: 510 (All outputs high except for Bit1)

Measured Output:

```
bit1 = 9.234655e-05
bit2 = 7.998718e-01
bit3 = 7.998718e-01
bit4 = 7.998718e-01
bit5 = 7.998718e-01
bit6 = 7.998718e-01
bit7 = 7.998718e-01
bit8 = 7.998718e-01
bit9 = 7.999485e-01
```

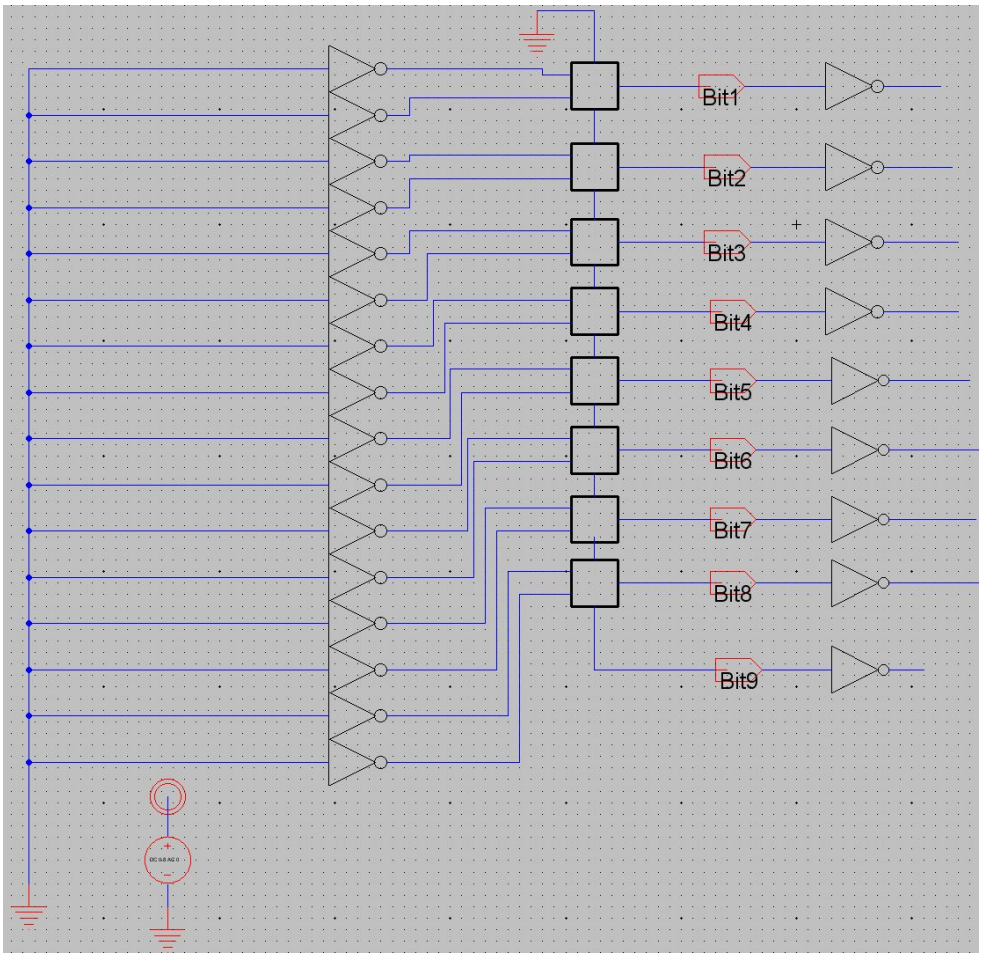


Figure 1.16: Circuit for In_1= 255, In_2= 0

As the input and output of the single adder can be observed visually through these graphs. The test that needs to be conducted with 8-bit adder is the full propagation of the carry all from from the first adder to the last.

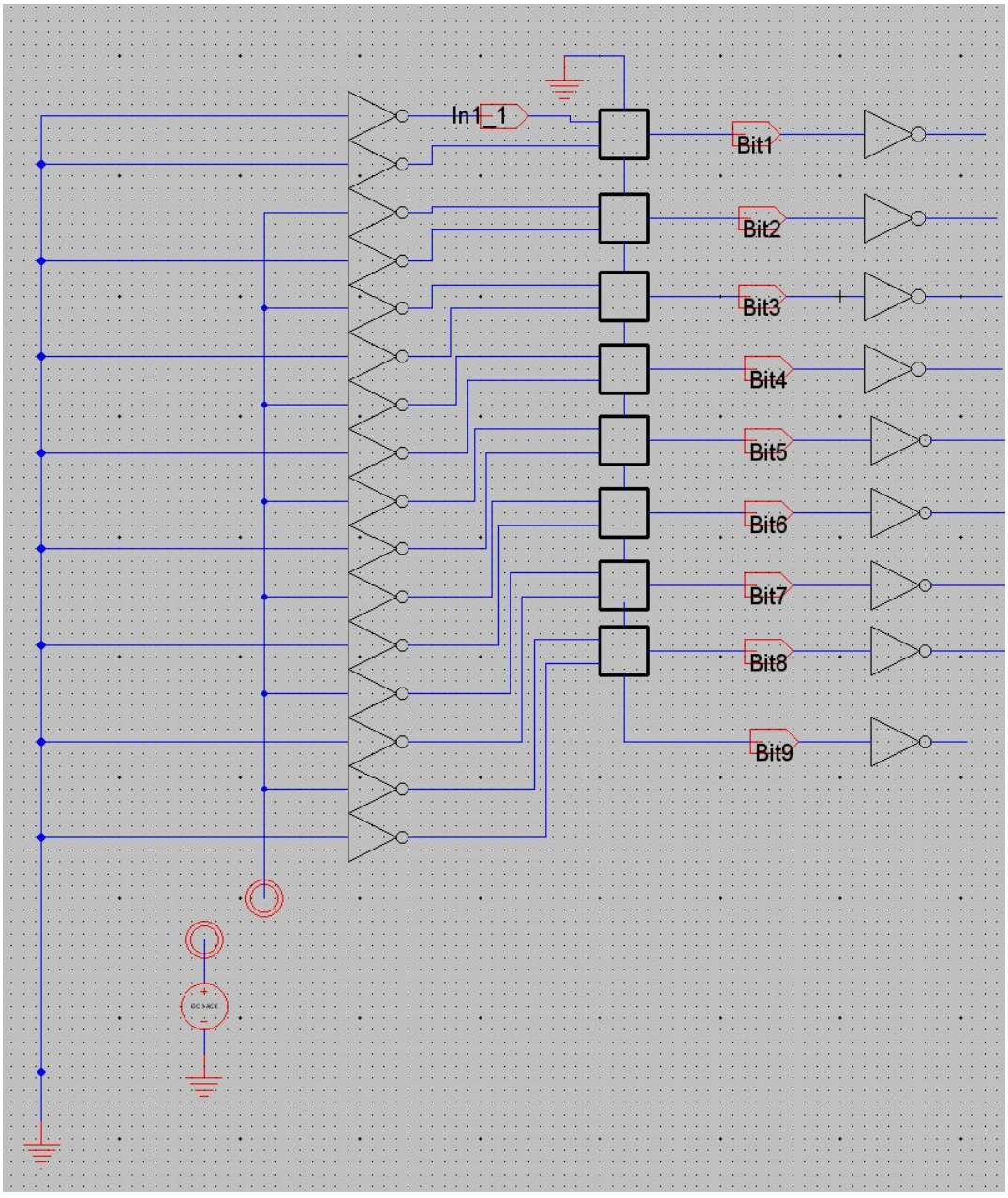


Figure 1.16.1: The schematic of the carry propagation test (11111111 + 00000001)

op Run the dc solver on ngspice

```
print Bit1 Bit2 Bit3 Bit4 Bit5 Bit6 Bit7 Bit8 Bit9
```

Print out all the outputs

Ideal Expected Output: 256 (only final carry is high)

Measured Output:

```
bit1 = 9.234655e-05
bit2 = 1.440410e-05
bit3 = 1.440409e-05
bit4 = 1.440409e-05
bit5 = 1.440409e-05
bit6 = 1.440409e-05
bit7 = 1.440409e-05
bit8 = 1.440409e-05
bit9 = 7.999485e-01
```

Test is upheld.

1.5. Test Case Justification For Evaluation Metrics

Delay:

The justification for the 11111111 + 00000001 case producing the maximum delay is discussed at the end of section 1.3. As the inputs are loaded by inverters the provided input to the inverters must be the opposite of the test case that is to be evaluated.

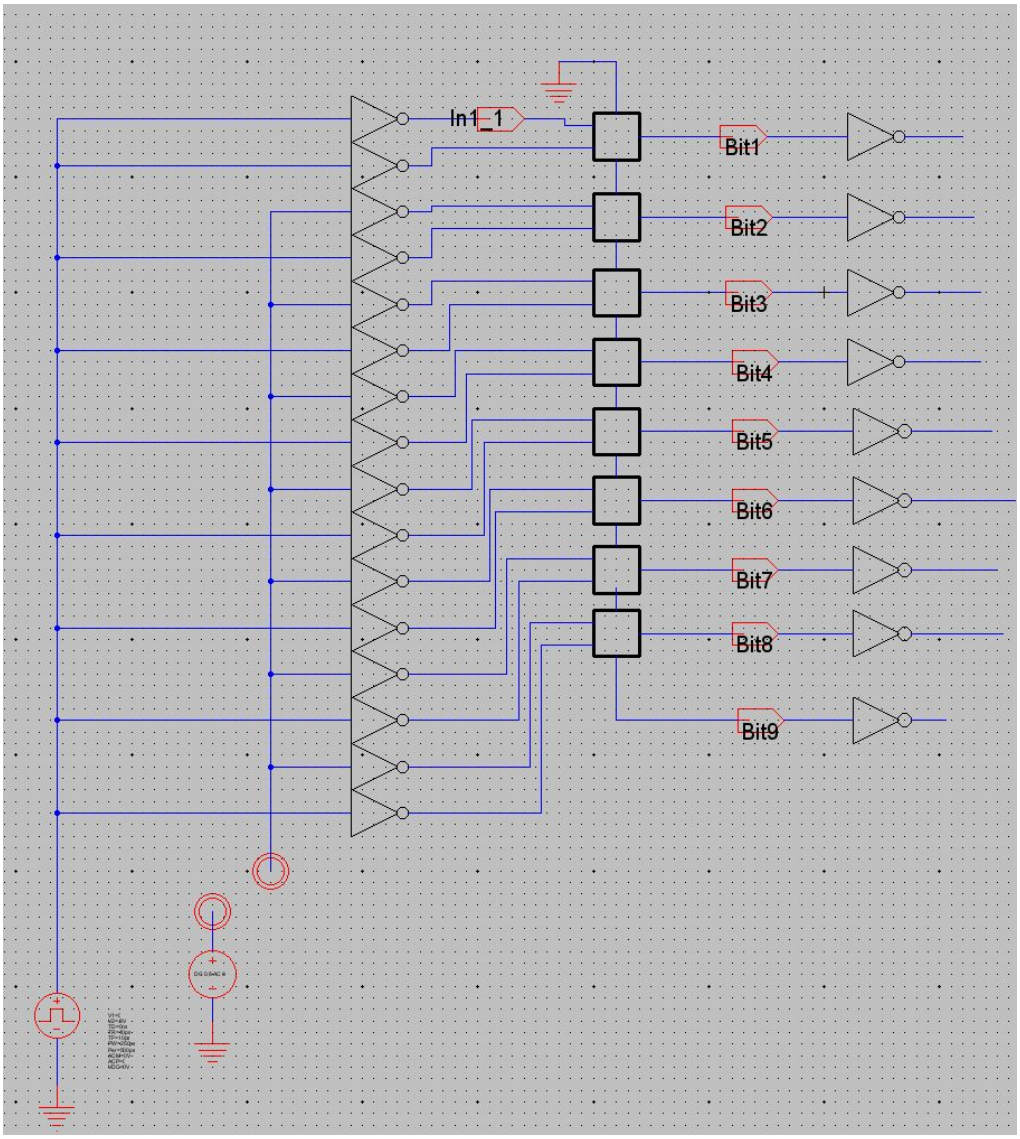


Figure 1.17: Circuit for maximum delay of 8-bit adder

Command to find the rise time between the first bit going high to the final carry out transitioning to high.

```
meas tran delay trig v(In1_1) val=0.4 rise=1 targ v(Bit9) val=0.4 rise=1
```

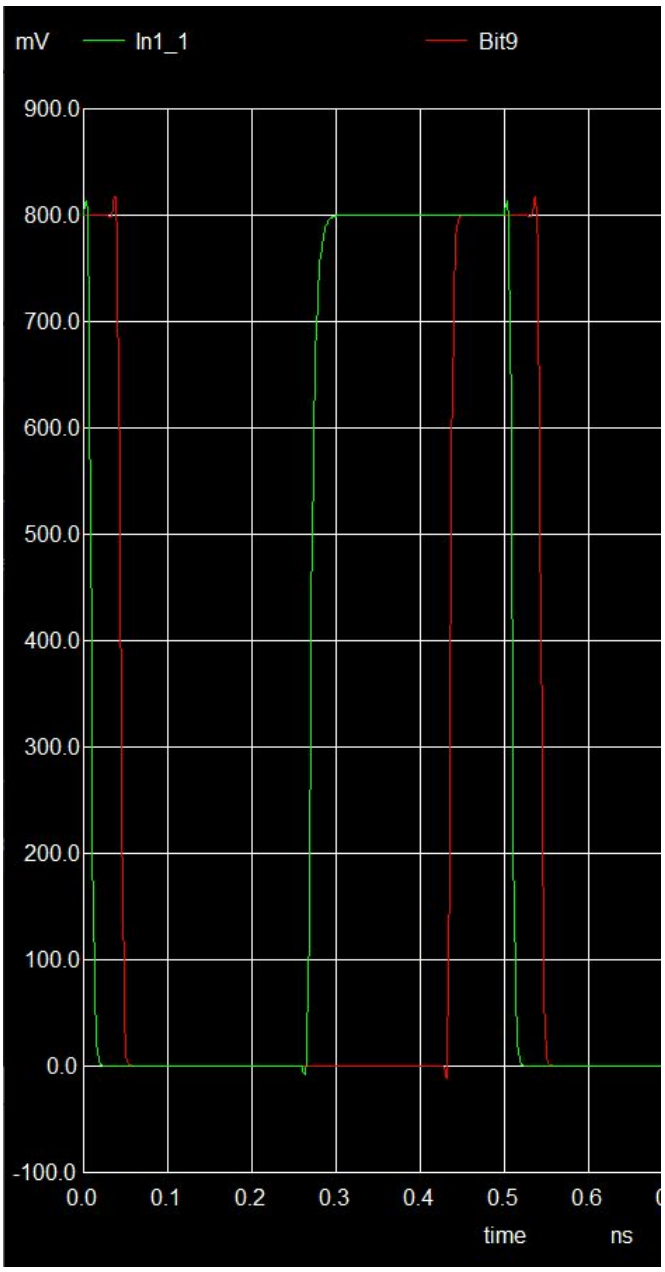


Figure 1.18: The rising value of In1_1 and Bit9 can be observed and the delay is very noticeable just through a single glance.

Max Switching Energy:

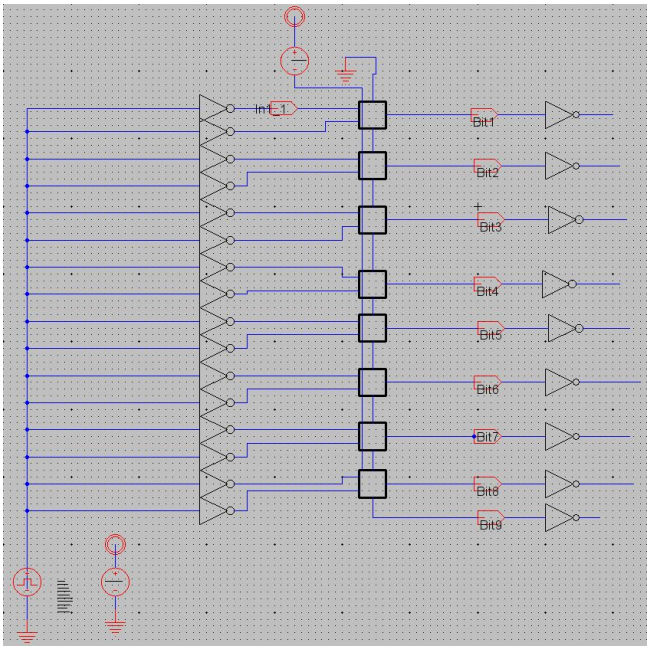


Figure 1.19: Circuit for maximum switching energy of 8-bit adder

The same adder as before, but for this section the power is provided individually to each adder through a power export such that we can look at the total power that is being consumed.

```
tran 1ps 1ns
```

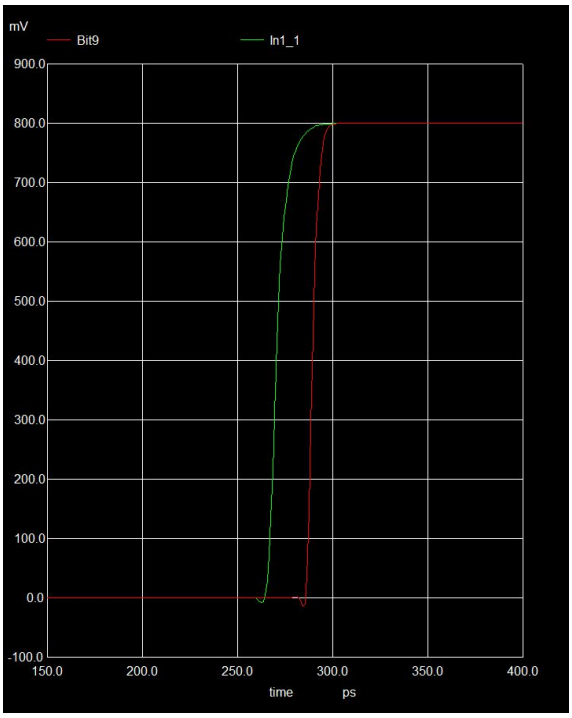


Figure 1.20: The slice of the graph where the bits are going from 0 to 1.

```
ngspice 4 -> meas tran yint integ I(vv_generi@1) from=260ps to=310ps
yint                =  1.74891e-15 from=  2.60000e-10 to=  3.10000e-10
```

Average Switching Energy:

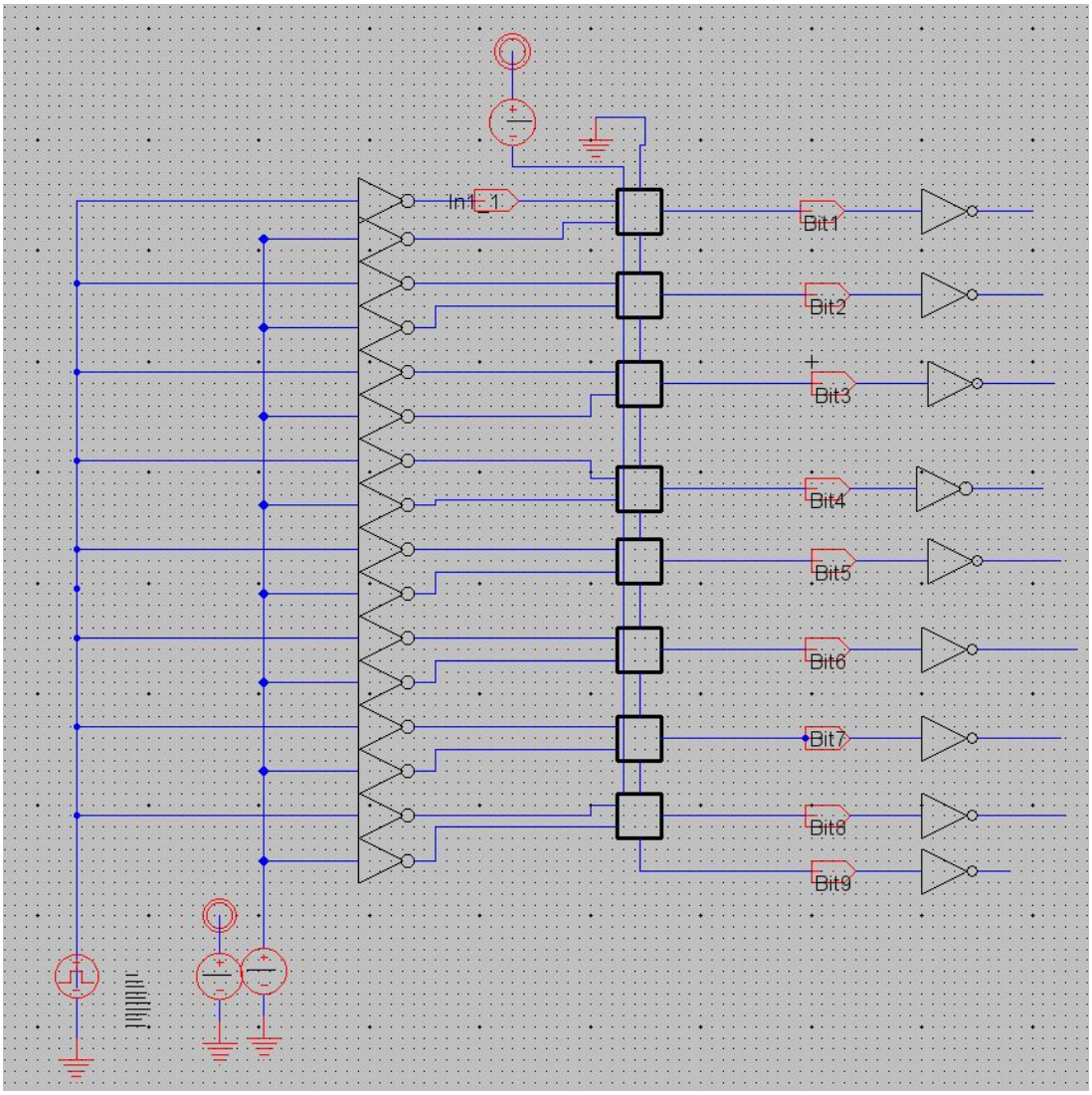


Figure 1.21: Circuit for average switching energy of 8-bit adder
All non-switching bits will have input to the adder as 0.

```
tran 1ps 1ns
meas tran yint integ I(vv_generi@1) from=260ps to=310ps
```

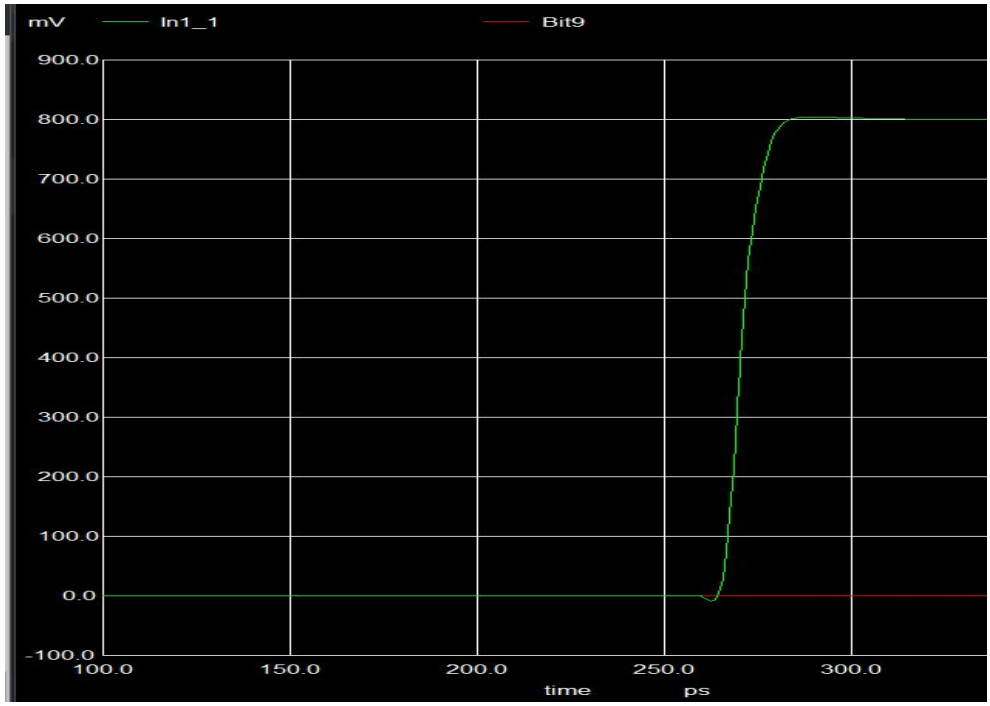


Figure 1.22: Graph of In1_1 switching while the final carry will stay 0.

Max Leakage Energy:

Single Adder Delay: $19.34R_{on}C_{load}$

RC from Project 1.3: 6.334e-13 seconds

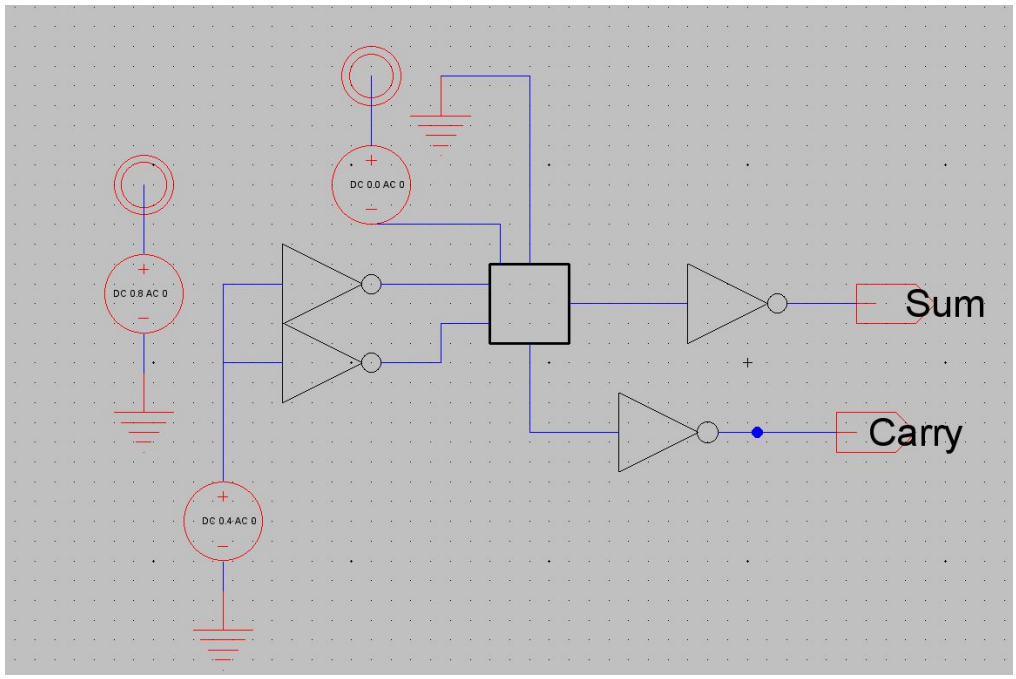


Figure 1.23: Circuit for maximum leakage energy

With the assumption that V_{th} = 300mv, the currents at each input can be evaluated. To find the maximum leakage current. Consider the values

above the threshold voltage, for which $1.8*10^{-4}(V_{gs}-V_{th})$ will be used to find the max current input.
The input voltages are set to .4 V this half of the V_{dd} which above the threshold voltage for both the pMOS and nMOS, so that both transistors will be on.

Vin	Vgsp	Inmos	Ipmos	Ipwr,gnd	MAX I
					0.000018
310	490	0.0000018	0.0000342	0.0000018	
320	480	0.0000036	0.0000324	0.0000036	
330	470	0.0000054	0.0000306	0.0000054	
340	460	0.0000072	0.0000288	0.0000072	
350	450	0.000009	0.000027	0.000009	
360	440	0.0000108	0.0000252	0.0000108	
370	430	0.0000126	0.0000234	0.0000126	
380	420	0.0000144	0.0000216	0.0000144	
390	410	0.0000162	0.0000198	0.0000162	
400	400	0.000018	0.000018	0.000018	
410	390	0.0000198	0.0000162	0.0000162	
420	380	0.0000216	0.0000144	0.0000144	
430	370	0.0000234	0.0000126	0.0000126	
440	360	0.0000252	0.0000108	0.0000108	
450	350	0.000027	0.000009	0.000009	
460	340	0.0000288	0.0000072	0.0000072	
470	330	0.0000306	0.0000054	0.0000054	
480	320	0.0000324	0.0000036	0.0000036	
490	310	0.0000342	0.0000018	0.0000018	

A chart which finds the maximum of the minimum currents from the I_{nmos} and I_{pmos} when Vin is greater than the voltage threshold for both transistors.

```
tran 1ps 1ns
meas tran yint integ I(vv_generi@1) from=00ps to=12ps
```

One-bit Leakage: 2.71985e-17J
This is the leakage energy for an adder, now as the outputs do not change over the period that the measurement is done, the leakage energy can be additive for all other adders in the 8-bit adder. Such that, 2.71985e-17J * 8 will calculate the leakage energy for the 8-bit adder.
8-bit Max Leakage: 2.2-16 J

Min Leakage Energy:

The case for the minimum leakage energy is when $V_{in} = 0$ or V_{dd}.

Vin	Vgsp	Inmos	Ipmos	Ipwr,gnd	MAX I
					1.66E-09
500	300	0.000036	3.00E-06	3.00E-06	
600	200	0.000054	2.46E-07	2.46E-07	
700	100	0.000072	2.02E-08	2.02E-08	
800	0	0.00009	1.66E-09	1.66E-09	

The case to be done is when the input into the adder is equal to 0V.

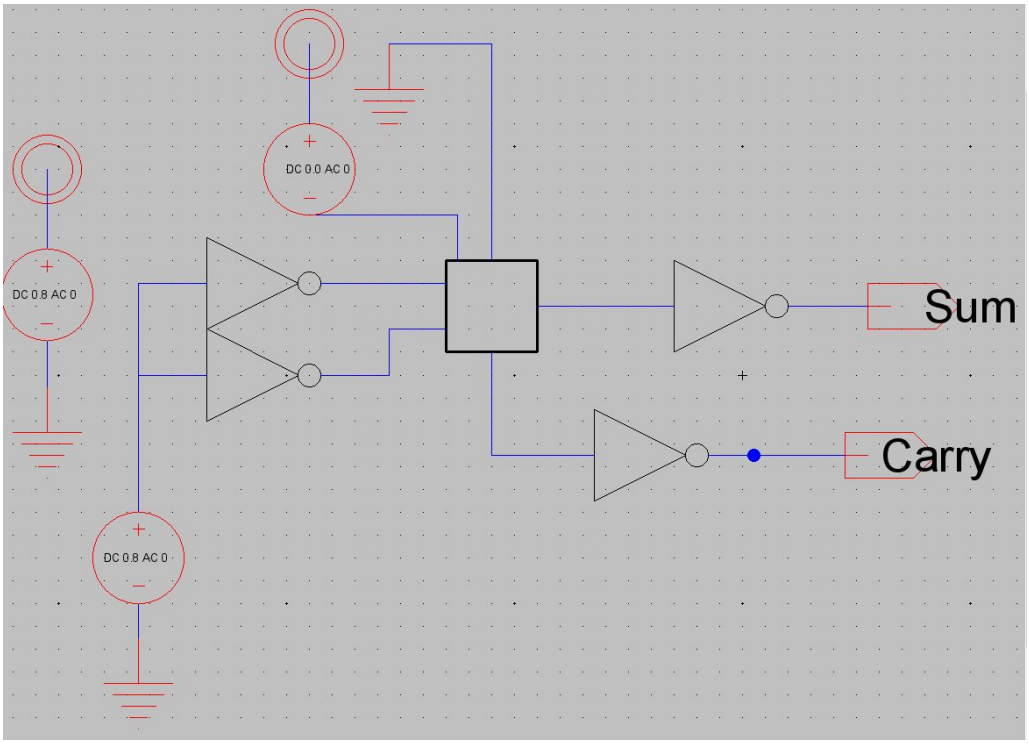


Figure 1.23: Circuit for minimum leakage energy

As the adder is loaded by an inverter the values that are input must be the complement of the values that are to be input

```
tran 1ps 1ns
meas tran yint integ I(vv_generi@1) from=00ps to=12ps
```

One-bit leakage: 3.05201e-19 J

For the same reasons described above, the value of the One-bit leakage can be extrapolated to calculate the 8-bit leakage.

8-bit Min Leakage: 3.05201e-19 J * 8 = 2.4e-18 J

Area:

The total area for the one-bit adder can be calculated from summing the area of 2 XOR gates, 2 AND gates, and an OR gate.

As all are minimum sized we can just count the number of transistors and multiply it by the area of a single transistor which is 22nm, as we defined the area to be the sum of the widths

Transistors in XOR: 12

Transistors in AND: 4

Transistors in OR: 4

Total Transistors in Adder: 2(12) + 2(4) + 2 = 34

Total Transistors in 8-bit Adder: 34 * 8 = 272

Total Area: $2.2*10^{-8}$ m^2 * 272 = $6.0*10^{-6}$ m^2

1.6. Evaluation Metrics

Delay	Max Switching Energy	Average Switching Energy	Max Leakage Energy	Min Leakage Energy	Area
1.7-10s	1.7e-15J	9.1e-16J	2.2-16 J	2.4e-18 J	6.0e-6m^2

The delay calculated from using tau found in question 3 of project 1 produces a value of 9.7999648e-11s which when compared to the value that was observed while testing the circuit there is about a 73% increase of delay from the equations. This tau estimate takes attempts to extrapolate a delay from very basic information which could have caused the very large discrepancy in the two terms.

2. Delay Optimized

2.1. Design

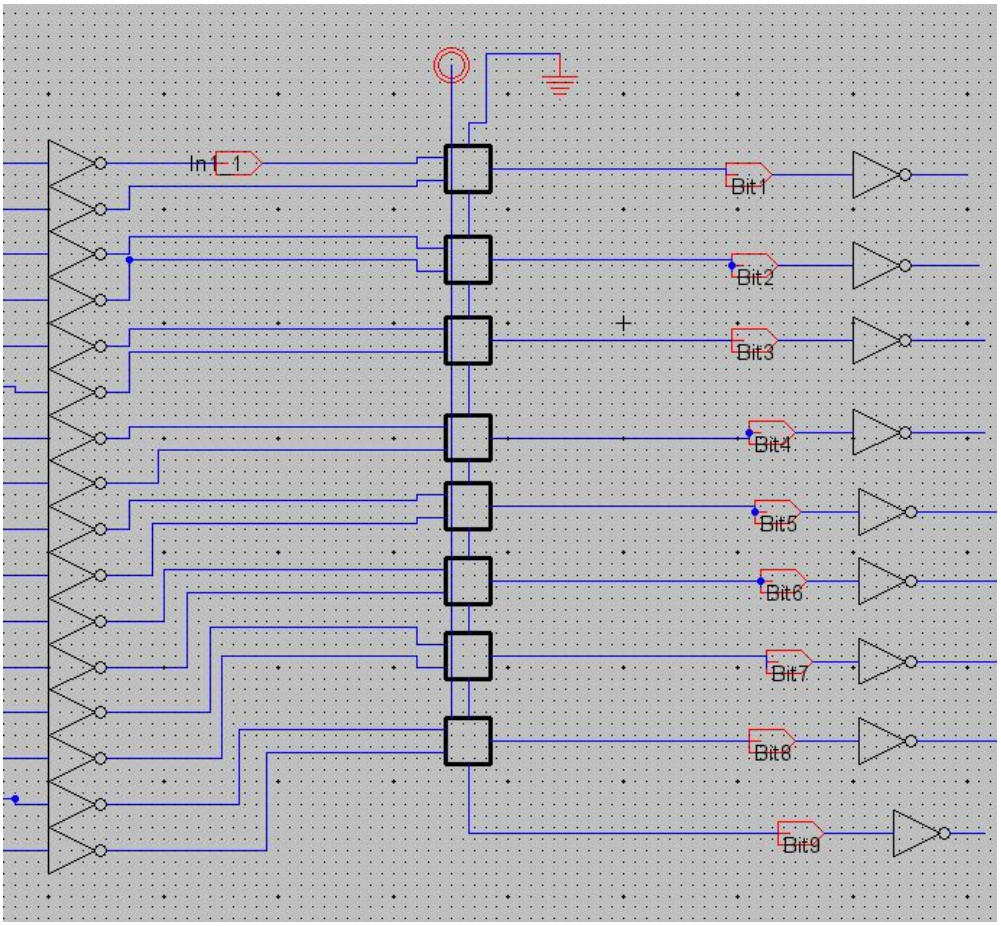


Figure 2.1: similar to Figure1.1 this is the final schematic of the 8 bit adder

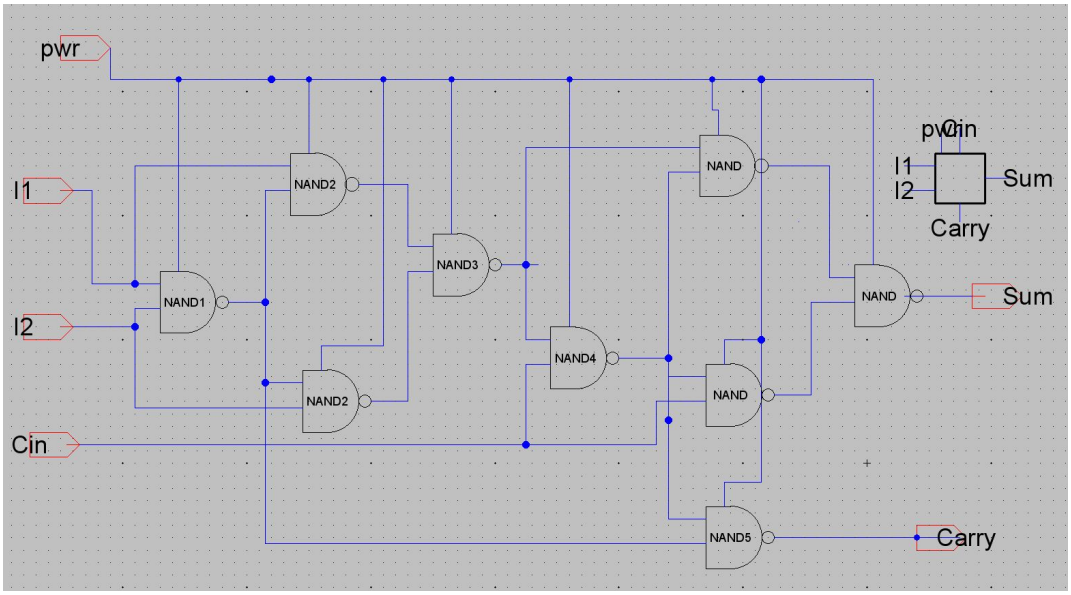


Figure 2.2: NAND topology for the fuller adder

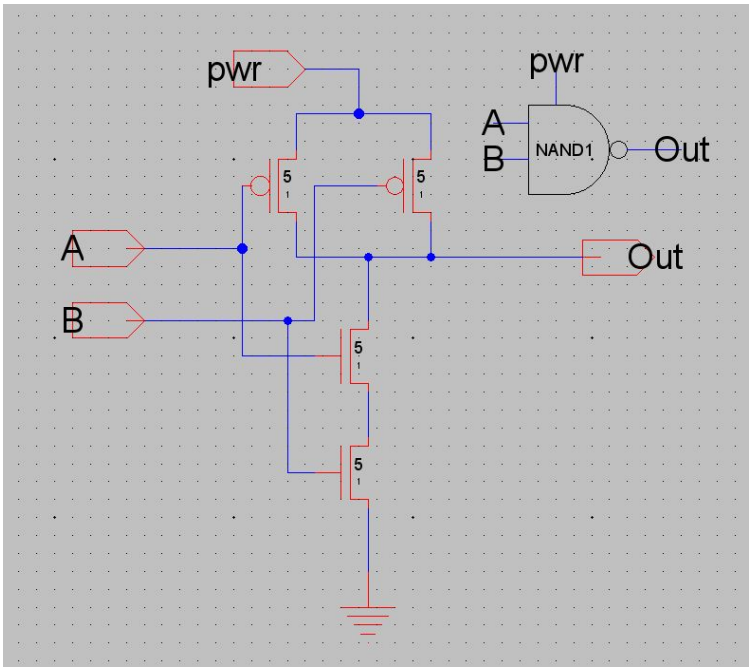


Figure 2.3: The NAND1 (1 is a descriptor to differentiate against the other sized NAND gates, similar for the rest) gate with different sized transistors W = 1

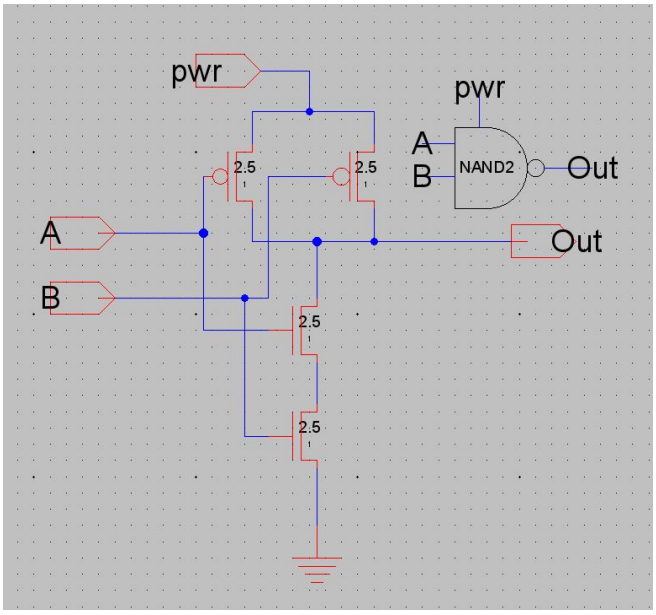


Figure 2.4: The NAND2 gate with different sized transistors W = 2.5

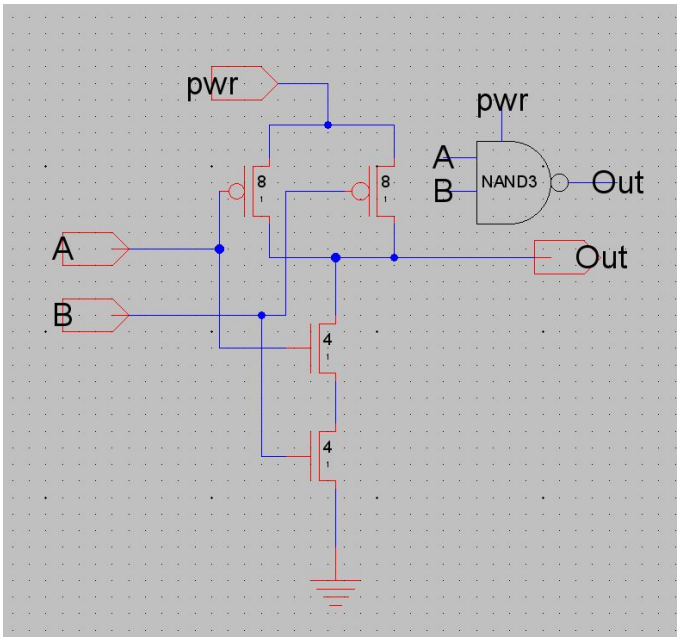


Figure 2.5: The NAND3 gate with different sized transistors W_p = 8 and W_n = 4.

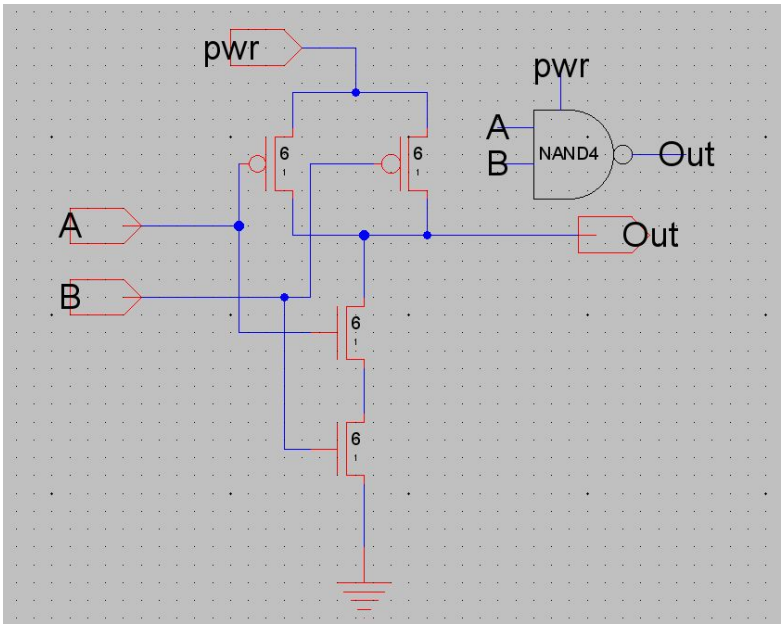


Figure 2.6: The NAND4 gate with different sized transistors W = 8.

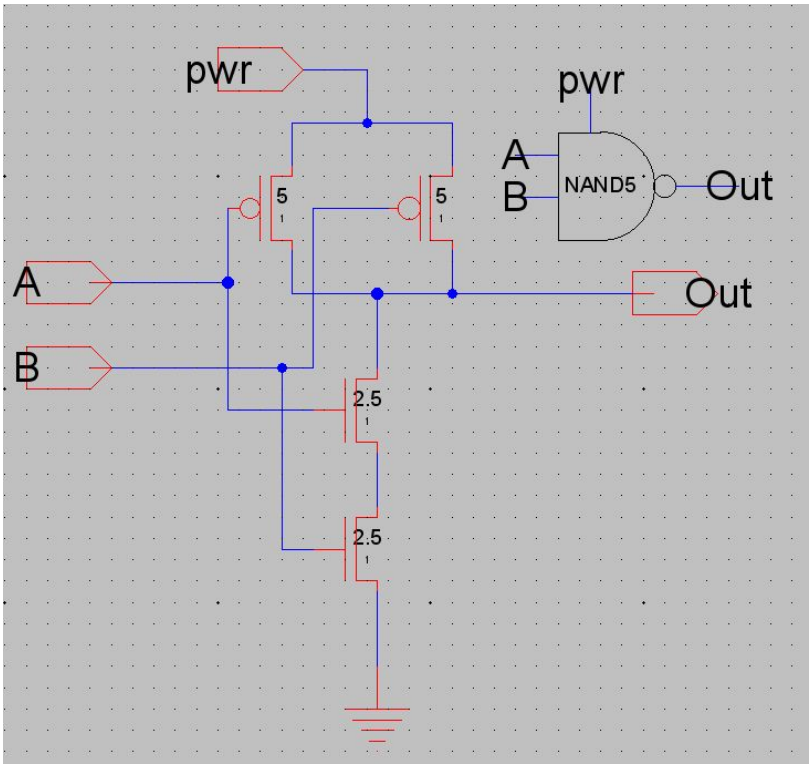


Figure 2.7: The NAND3 gate with different sized transistors W_p = 5 and W_n = 2.5.

2.2. Description of Logic and Operation

Using the concept of NAND universality there must be a way to create the full adder from a set of NAND gates.

Sum = A ⊕ B ⊕ Cin
Carry = AB + Cin (A ⊕ B)

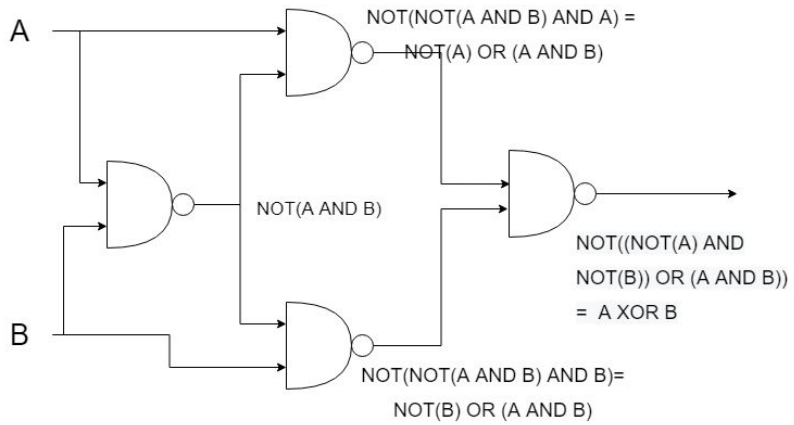


Figure 2.8: A 2input XOR gate implementation from NAND gates
With this 2input XOR, there can be a chain in order to make it able to evaluate the C_in input to evaluate the sum.

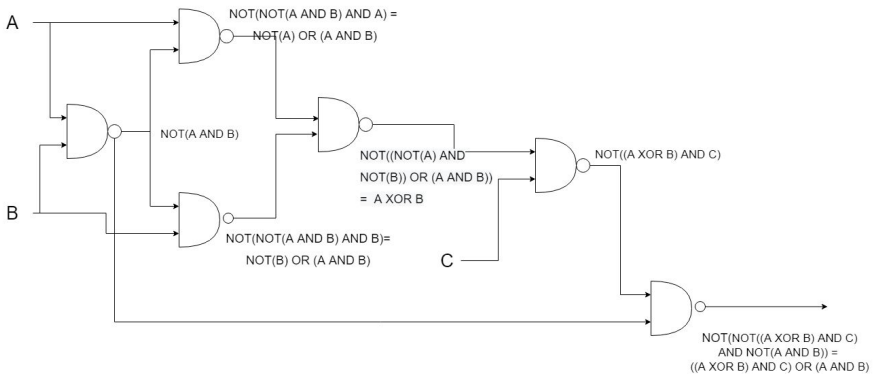


Figure 2.9: The carry of the adder implementation from NAND gates built on the XOR from above
By using the NOT(A AND B) and A XOR B values that are produced from the XOR the AB + Cin (A ⊕ B) can be easily made from the A XOR B and a nand gate to get NOT(Cin (A ⊕ B)) then passing NOT(A AND B) and NOT(Cin (A ⊕ B)) into a NAND gate(simplifying with demorgans) will produce AB + Cin (A ⊕ B)
The speed up on the carry arises from the ability to immediately switch the carry on a critical path (11111111 + 00000001) when the C_in arrives. The reason for this is because the NAND that takes in the C will turn 0 causing the carry to go to 1. The delay of these 2 NAND gates (at worst) are $2R*2C_{load} + 2R*2C_{load} + 2R*C_{load} = 10R*C_{load}$ compared to the carrying of the baseline at $14R*C_{load}$

2.3. Delay Evaluation Using τ

The individual components within the single adder must be optimized. The critical path of the carry is the most important, so the optimization will be on the path with greatest over impact on delay.

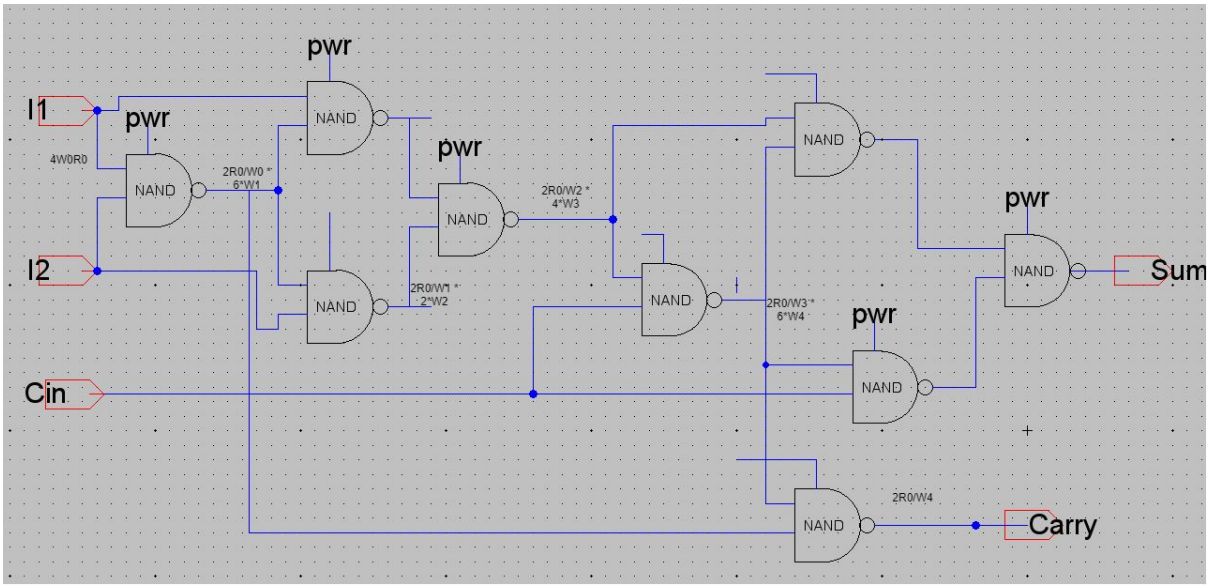


Figure 2.10: Annotated full adder with delays on each input

*Inputs from inverter

Input from inverter into NAND:

$4W0R0C_{load}$ (A single input will go into 2 transistors into the gate that will be looked at next and 2 into another NAND gate)

First NAND into Second NAND:

Worst case Ron = 2R0 / W0

Capacitance Load = $6W1C_{load}$ (2 transistors for each of the 3 NAND gates that this loads)

Delay = $2R0 / W0 * 6W1C_{load}$

Second NAND into Third NAND:

Worst case Ron = 2R0/W1

Capacitance Load = $2W2C_{load}$ (2 transistors for the single NAND gates that this loads)

Delay = $2R0 / W1 * 2W2C_{load}$

Third NAND into Fourth NAND:

Worst case Ron = 2R0 / W2

Capacitance Load = $4W3C_{load}$ (2 transistors for the 2 NAND gates that this loads)

Delay = $2R0 / W2 * 4W3C_{load}$

Fourth NAND into Fifth NAND:

Worst case Ron = 2R0/W3

Capacitance Load = $6W4C_{load}$ (2 transistors for the 3 NAND gates that this loads)

Delay = $2R0 / W3 * 6W4C_{load}$

Output:

2R0/W4

$4R0*W0C_{load}$
$+ 2R0/W0 * 6W1C_{load} + 2R0/W1 * 2W2C_{load}$
$+ 2R0/W2 * 4W3C_{load} + 2R0/W3 * 6W4C_{load}$
$+ 2R0/W4C_{load}$ = Delay of critical path

The partial derivatives are taken and must be minimized. With input and output loads of a minimum sized inverter

W0:

Find root of derivative in respect to W0: $4R0 - 12R0/W0^2 * W1$

W0 = sqrt(3W1)

W1:

Find root of derivative in respect to W1: $12R0/W0 - 4R0*W2/W1^2$

W1 = sqrt(W2*W0/3)

W2:

Find root of derivative in respect to W2: $4R0/W1 - 8R0*W3/W2^2$

W2 = sqrt(2W3*W1)

W3:

Find root of derivative in respect to W3: $8R0/W2 - 12W4 * R0/W3^2$

W3 = sqrt(3W4*W2/2)

W4:

Find root of derivative in respect to W4: $12R0/W3 - 4R0*W5/W4^2$

W4 = sqrt(W3/3)

As there are many possible solutions, set the value for W0 = 3.

Now solve for the rest using the derivatives.

$W1 = W0^2 / 3 = 3$

$W2 = W1^2 / W0 * 3 = 9$

$W3 = W2^2 / 2 / W1 = 13.5$

$W4 = W3^2 / W2 / 3 * 2 = 13.5$

These values will be used as a starting point for the optimization. As this was not the complete optimization the values for delay and charts are present within section 3.2. This section will only consider the final optimized adder.

After fine fine tuning values for width:

W0 = 5

W1 = 2.5

W2_p = 8; W2_n = 4
W3 = 6
W4_p = 5; W4_n = 2.5

Now the delay can be estimated using τ by replacing the width variable with the values that were found here.
Total delay after substitution for a single add = **46.0333 R_0C_{load}**
Thus for carry to be evaluated through all 8 bit slices, 368.26$R_{on}C_{load}$ (single delay * 8) is the total delay to the final carry. Once again the worst case delay will arise on the path of carry in propagating the carry out to the rest of the adders. The critical path lays on top of the path that follows Cin so we can evaluate the worst delay time with the following bit addition 11111111 + 00000001.

2.4. Validation of Logical Correctness

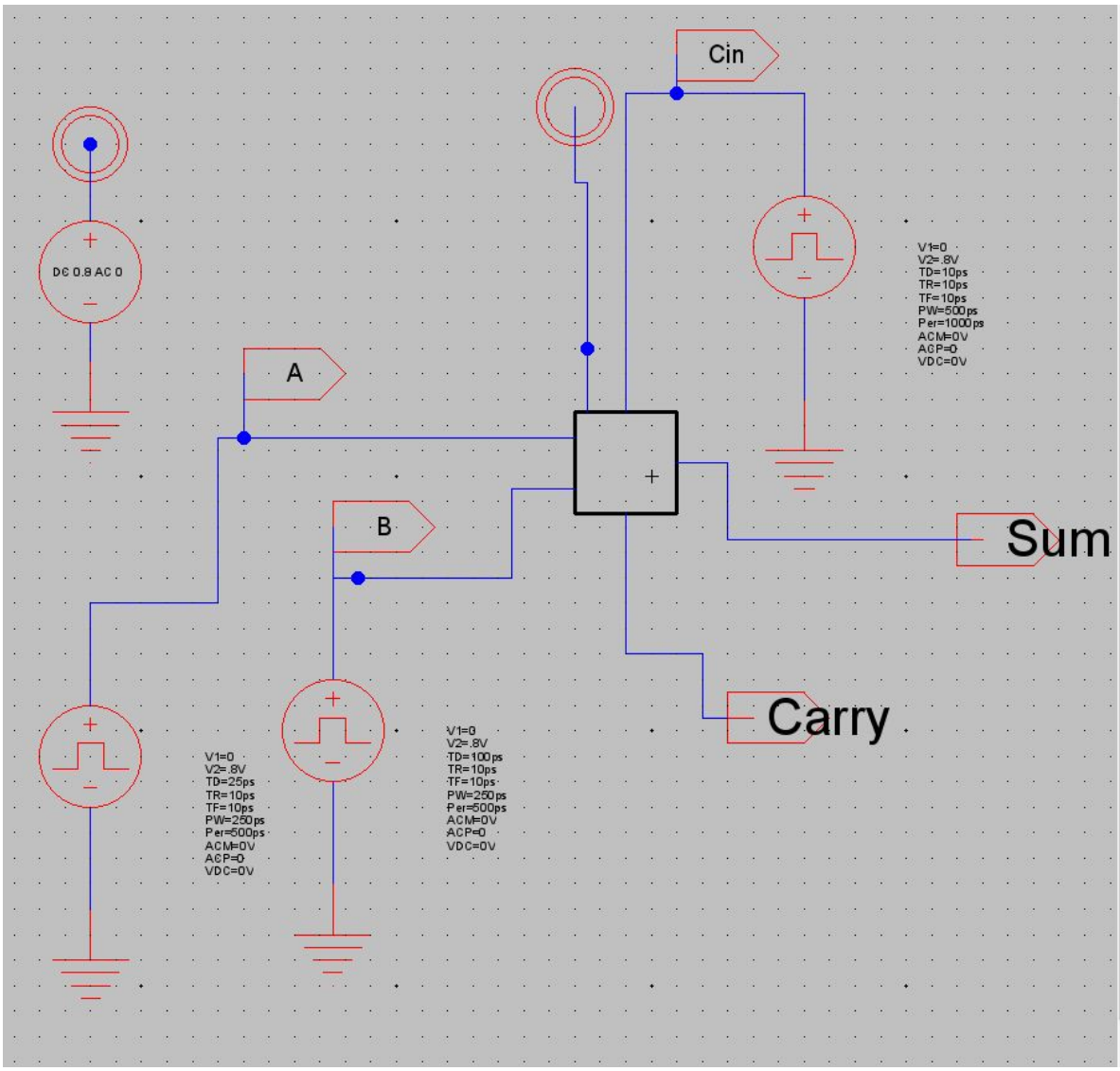


Figure 2.11: Using 3 voltage pulse generators at different pulse length all 8 inputs can tested in a single run for the 1-bit full adder

`tran 1ps 1ns; plot A B C carry` Run the transient simulation and plot the inputs and carry.

`plot A B C sum` plots the inputs and sum.

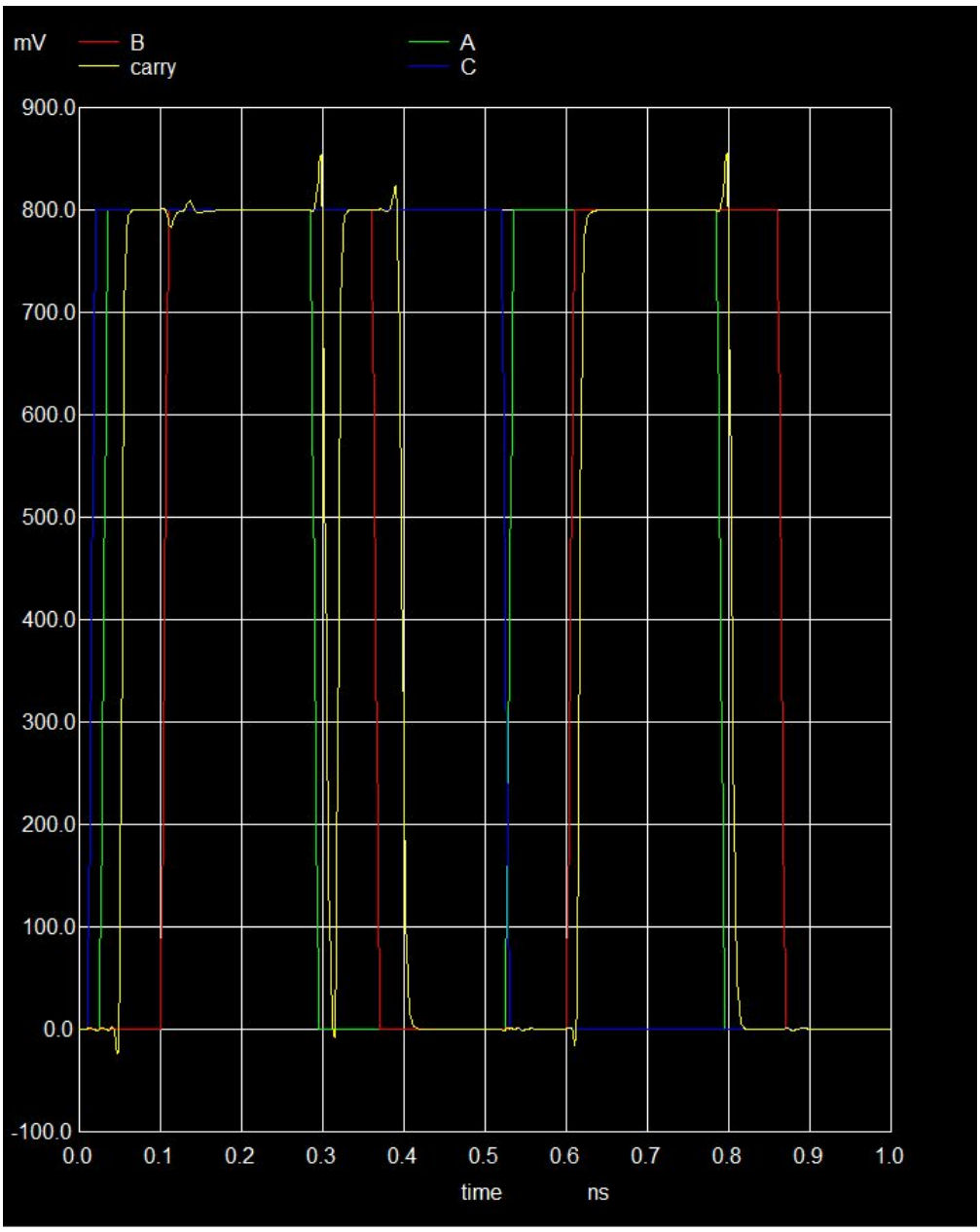


Figure 2.12: A plot of all inputs and the carry

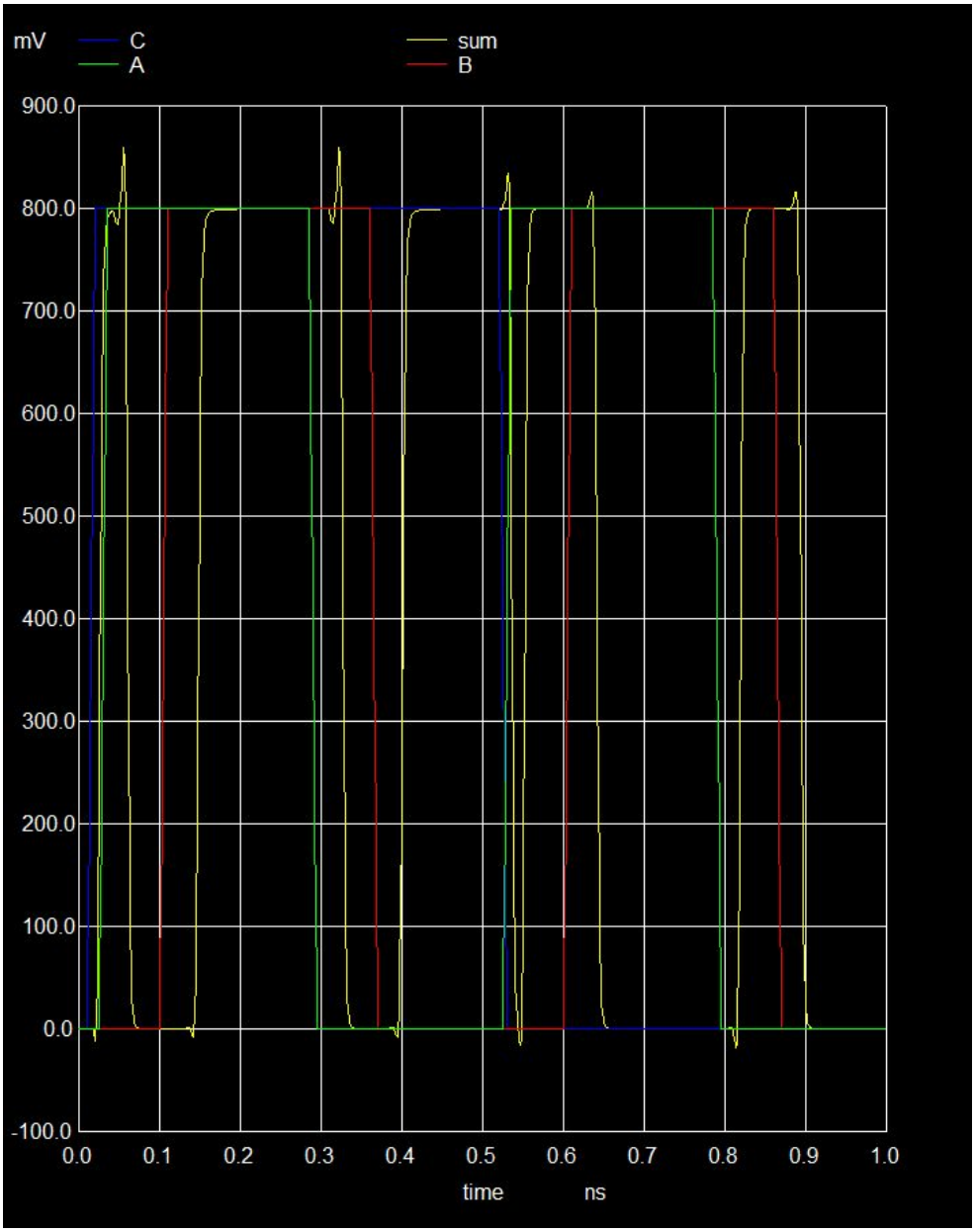


Figure 2.13: A plot of all inputs and the sum

As the input and output of the single adder can be observed visually through these graphs. The test that needs to be conducted with 8-bit adder is the full propagation of the carry all from from the first adder to the last.

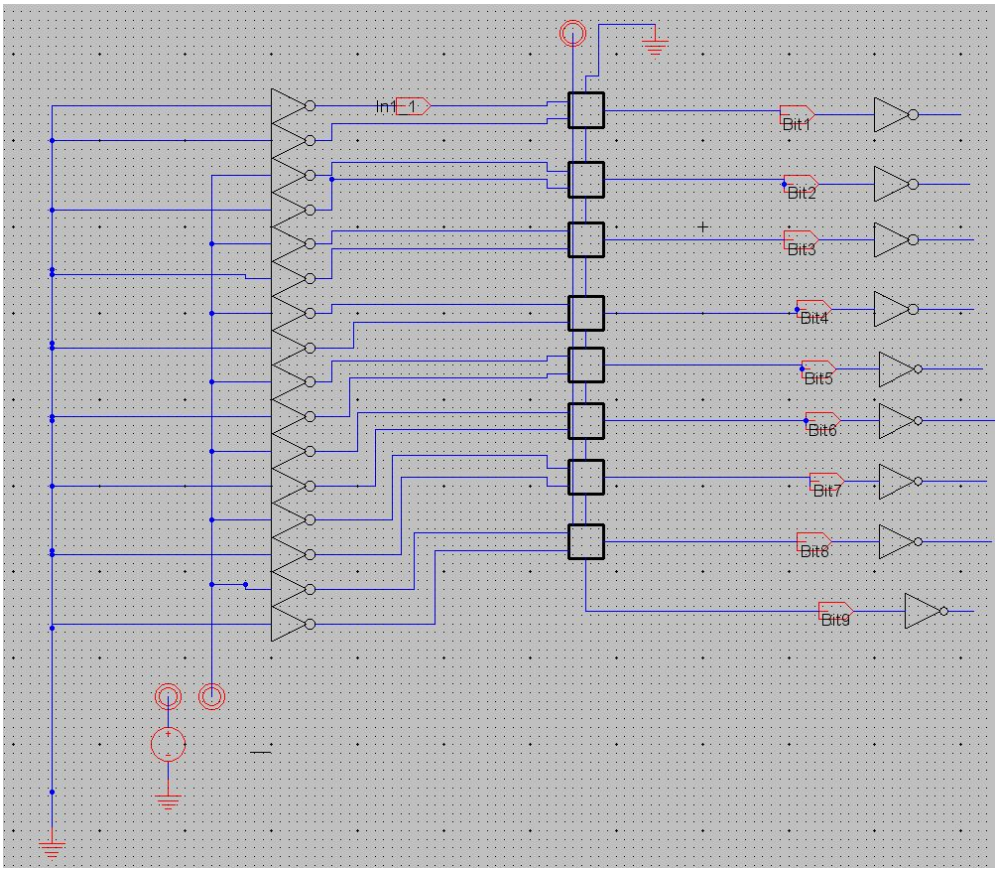


Figure 2.14: The schematic of the carry propagation test (11111111 + 00000001)

op Run the dc solver on ngspice

```
print Bit1 Bit2 Bit3 Bit4 Bit5 Bit6 Bit7 Bit8 Bit9
```

Print out all the outputs

Ideal Expected Output: 256 (only final carry is high)

Measured Output:

```
bit1 = 9.250336e-05
bit2 = 9.250322e-05
bit3 = 9.250322e-05
bit4 = 9.250322e-05
bit5 = 9.250322e-05
bit6 = 9.250322e-05
bit7 = 9.250322e-05
bit8 = 9.250322e-05
bit9 = 7.999752e-01
```

The test is upheld.

2.5. Test Case Justification

Delay:

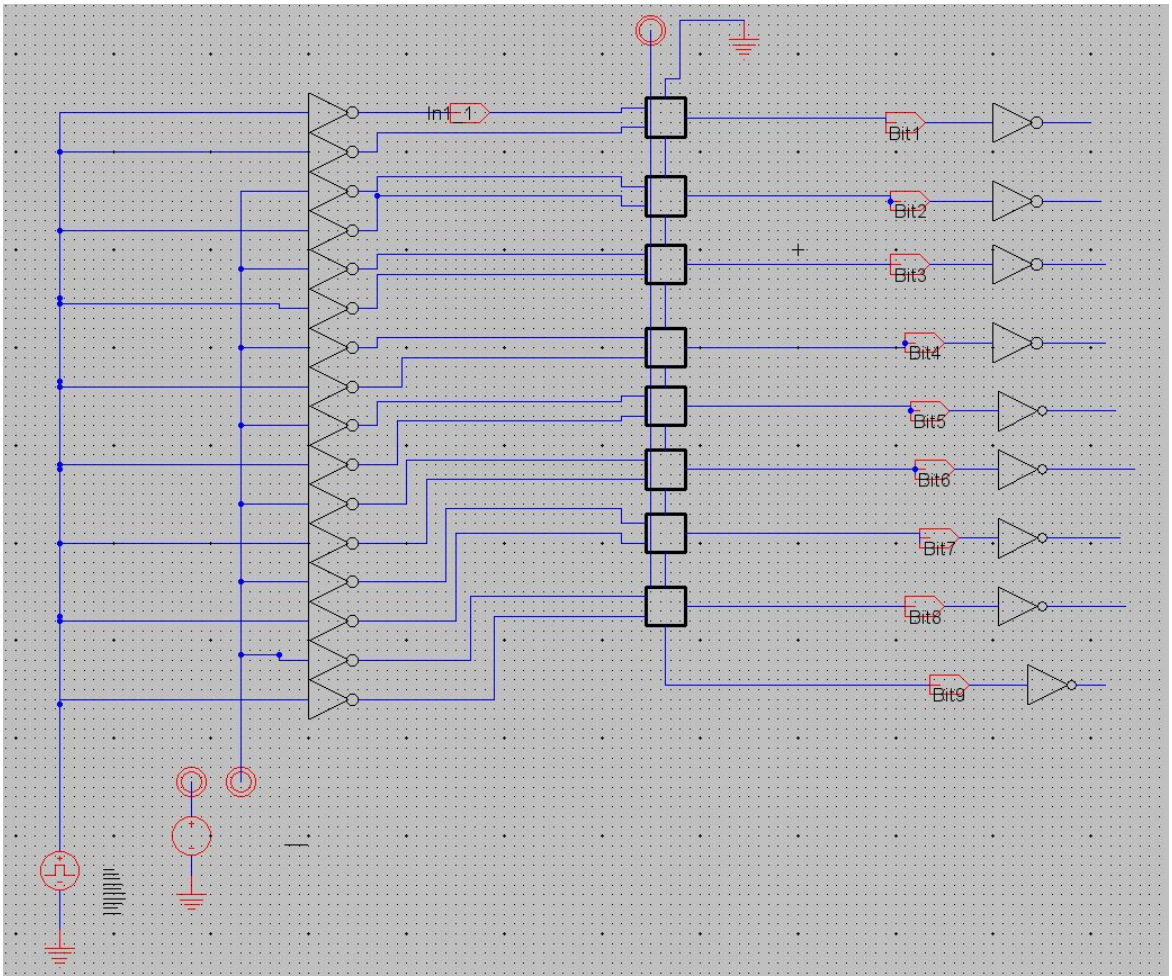


Figure 2.15: Schematic of the delay test case (11111111 + 00000001) for final carry delay

```
tran 1ps 1ns;plot In1_1 Bit9;meas tran delay trig v(In1_1) val=0.5 rise=1 targ v(Bit9) val=0.5 rise=1
```

Command to run the transient simulation, plot the input bit and last carry out, and measure the rise time

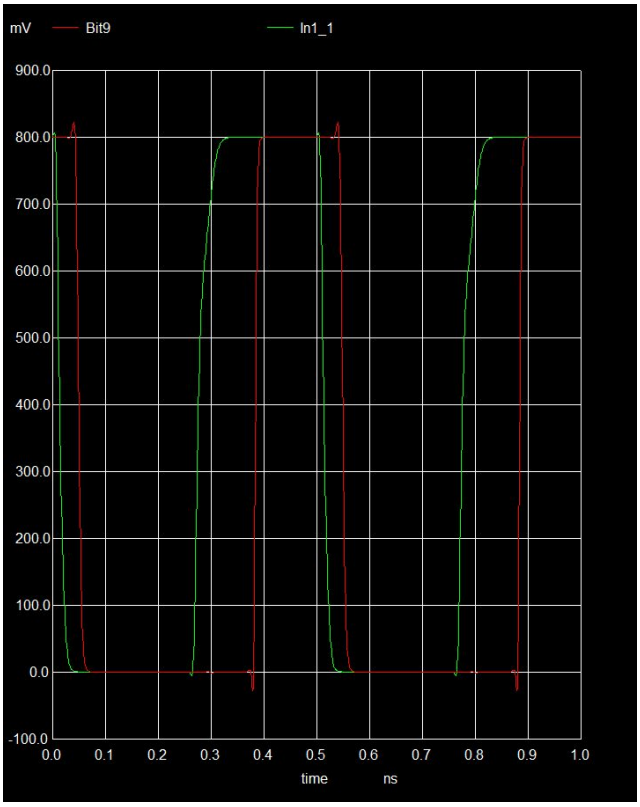


Figure 2.16: Plot of In1_1 and Bit9 visual striking compared to the baseline as the rise edges are noticeably closer.

```
delay                    =  1.061443e-10
```

Max Switching Energy:

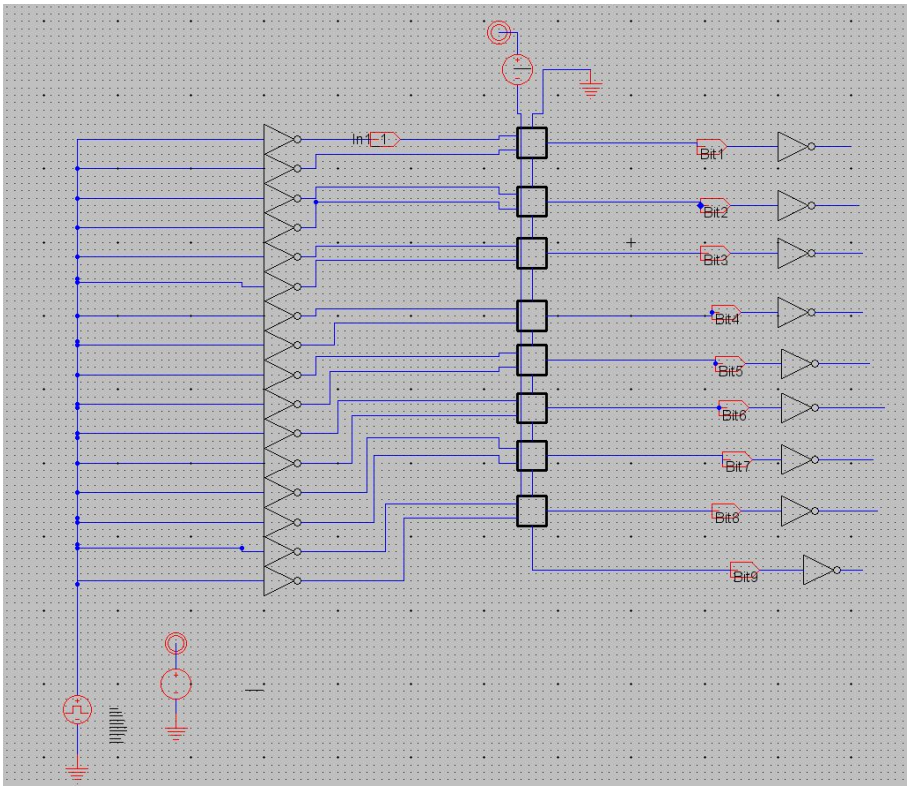


Figure 2.17: Circuit for maximum switching energy of 8-bit adder

The adder has not changed, but the inputs are now all tied to the voltage pulse generator. With this all inputs can switch to 1 to measure the maximum switching energy.

`tran 1ps 1ns` Run the transient simulation

plot In1_1 bit9 plot first bit input and last carry to know timing for energy measurement

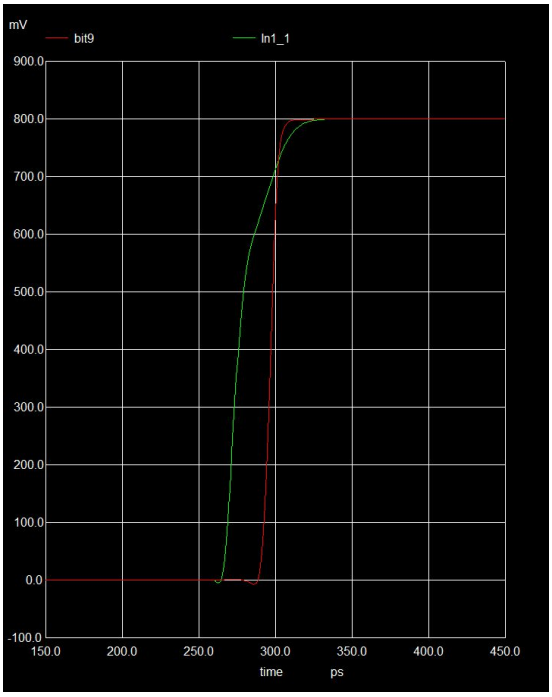


Figure 2.18: The slice of the graph where the bits are going from 0 to 1.

```
ngspice 6 -> meas tran yint integ I(vv_generi@1) from=250ps to=350ps
yint                =  8.17173e-15 from=  2.50000e-10 to=  3.50000e-10
```

Average Switching Energy:

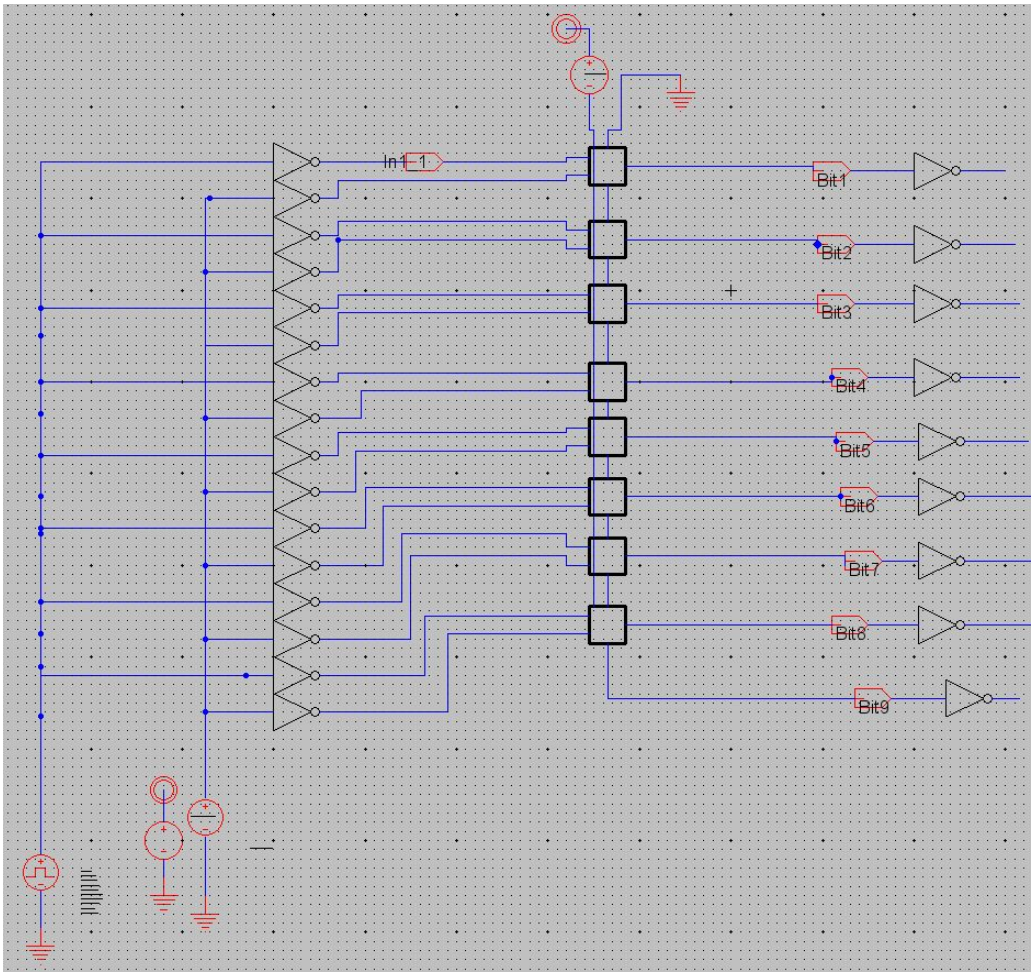


Figure 2.19: Circuit for average switching energy of 8-bit adder
All non-switching bits will have input to the adder as 0.

```
tran 1ps 1ns
```
Run the transient simulation

```
plot In1_1
```
plot first bit input to know timing for energy measurement

```
meas tran yint integ I(vv_generi@1) from=250ps to=350ps
yint                =  2.67686e-15 from=  2.50000e-10 to=  3.50000e-10
```

Leakage Energy Setup:

Single Adder Delay: 46.0333 R_0C_{load}

RC from Project 1.3: 6.334e-13 seconds

Time for single adder delay: **2.91575133e-11 s**

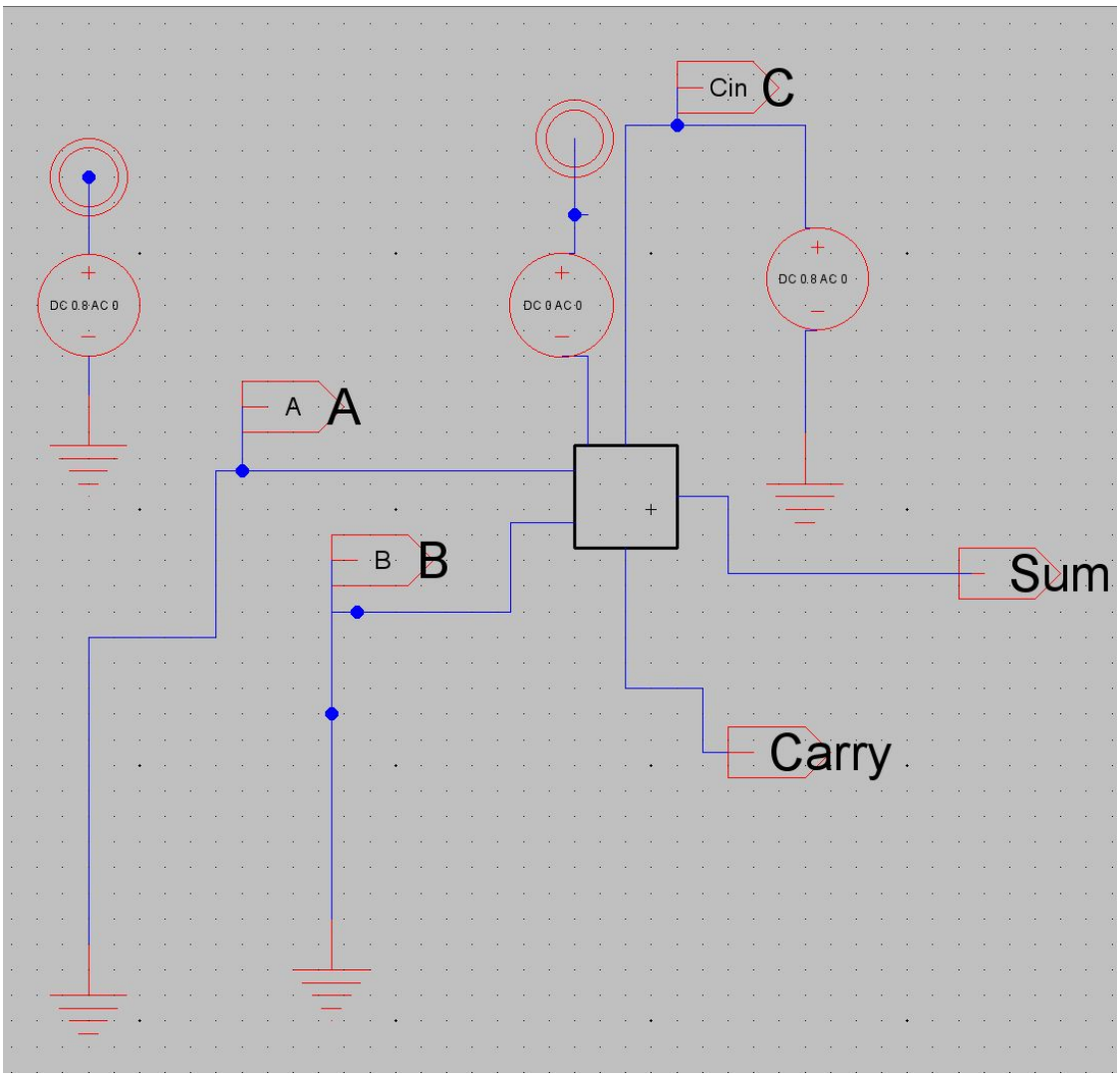


Figure 2.20: Example circuit of test case, all test cases were set by changing the voltages on each input and the simulation below was run,

```
tran 1ps 1ns;meas tran yint integ I(vv_generi@1) from=0ps to=29ps
```

Command to run the transient simulation and get leakage energy for single adder delay.

Case	Leakage (J)
A = 0 B = 0 C = 0	2.38595e-18
A = 0 B = 0 C = 1	2.81408e-18
A = 0 B = 1 C = 0	1.95793e-18

A = 0 B = 1 C = 1	2.17716e-18
A = 1 B = 0 C = 0	1.78072e-18
A = 1 B = 0 C = 1	2.00056e-18
A = 1 B = 1 C = 0	2.41893e-18
A = 1 B = 1 C = 1	2.84651e-18

Max Leakage Energy:

Case: In1 = 1, In2 = 1, C_in = 1

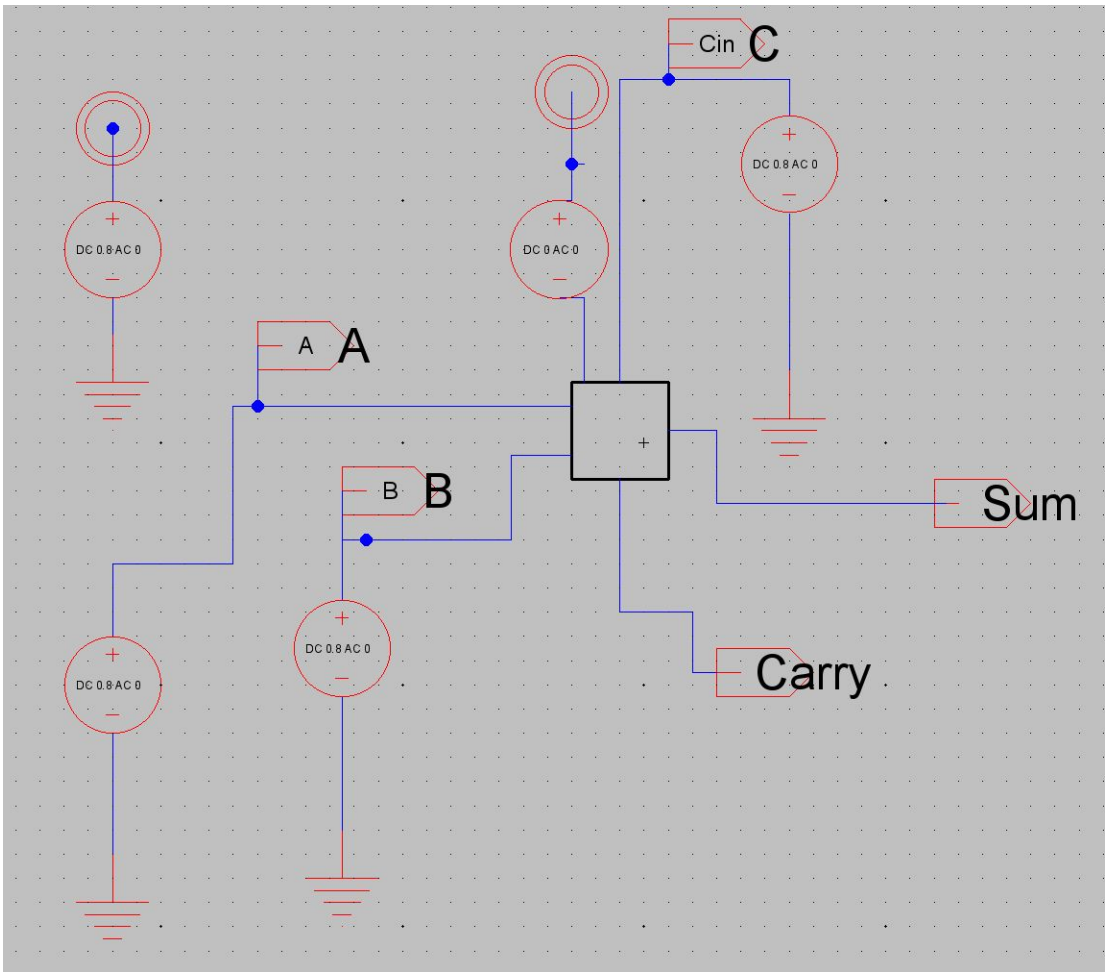


Figure 2.21: The schematic of the test case where In1(A) = 1, In2(B) = 1, C_in(C) = 1

```
tran 1ps 1ns;meas tran yint integ I(vv_generi@1) from=0ps to=29ps
```

Single Adder Maximum Leakage: 2.84651e-18 J

8-bit Adder Maximum Leakage: 2.84651e-18 J * 8 = 2.277208e-17 J

Min Leakage Energy:

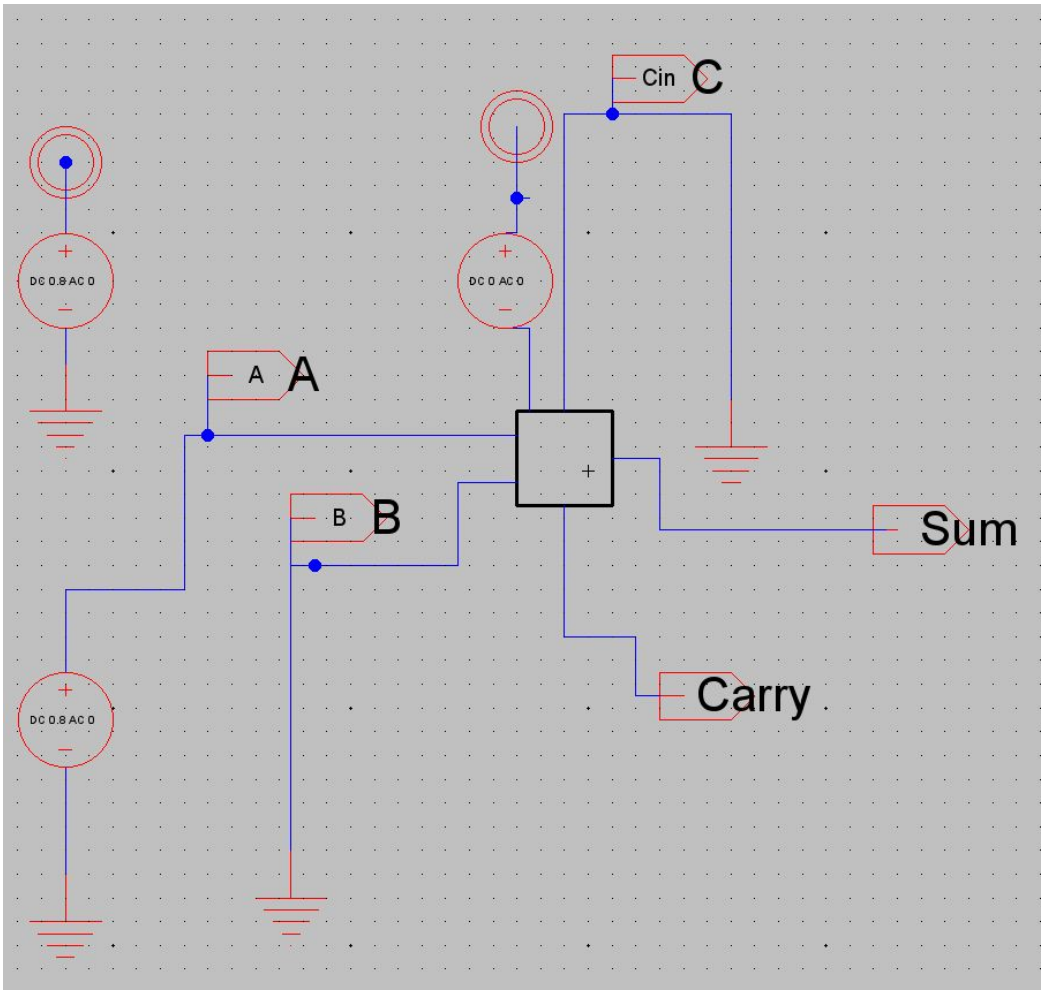


Figure 2.22: The schematic of the test case where In1(A) = 1, In2(B) = 0, C_in(C) = 0

```
tran 1ps 1ns;meas tran yint integ I(vv_generi@1) from=0ps to=29ps
```

Single Adder Maximum Leakage: 1.78072e-18 J

8-bit Adder Maximum Leakage: 1.78072e-18 J * 8 =1.424576e-17 J

Area:

The total area for the one-bit adder can be calculated from summing the area of 8 NAND gates.

The area was defined as the sum of the widths of the transistors.

The first sized NAND: 4 * 5 = 20

The second sized NAND: (2.5 * 4) * 2 = 20

The third sized NAND: 8*2 + 4 *2 = 24

The fourth sized NAND: 6 * 4 = 24

The fifth sized NAND: 5 * 2 + 2.5 * 2 = 15
Unchanged NANDs: 3 * 1 * 4 = 12
Total Area = 20 + 20 + 24 + 24 + 15 + 12 = 115

2.6. Design Metrics

Delay	Max Switching Energy	Average Switching Energy	Max Leakage Energy	Min Leakage Energy	Area
1.1e-10s	8.2e-15J	2.7e-15J	2.3-17 J	1.4e-17 J	2.53e-6

The delay that was calculated in respect to the delay that was measured vastly differs.However, the following steps of the optimization for the transistor width produced values that established a starting point that was able to beat the baseline speed. The reduction in maximum delay is 31.25% total decrease in delay. The change in the gate topology also decreased the maximum switching energy, but increased the average case. The same outcome occurs for the leakage energy, the maximum decreases, but the minimum case increases. The area compared to the baseline is significantly less.

3. Design Decision-Making

3.1. Possible Design-Options

Varying the transistor sizings to optimize for delay
Changing the structure of the gates from the standard cmos to pass based.
This can be done for the xors that are used extensively.
Use a different gate layout that will use fewer transistors, if possible.
Increase Vdd to decrease delay at the exchange of energy usage
Apply ratioed logic to decrease the overall gate capacitance
Decrease V_{th}

3.2. Alternate Attempts/Variations

There was an attempt to use the ratioed logic in combination with the pass logic to build the xor gate. However, this kept slowing down the timing of the single gate on the fall of the carry, though there was a significant speed up on the rise. I was unable to find a way to augment the rest of the circuit to make up for this delay. The next idea was to go completely back to the baseline and optimize the sizing from there.

```
tran 1ps 1ns;plot net@18 carry;meas tran delay trig v(net@18) val=0.5 fall=1 targ v(carry) val=0.5 fall=1
```

Old fall time :

```
g v(net@18) val=
 =  3.453422e-11
```

New Fall time:

```
}
 -2.582157e-10 t
```

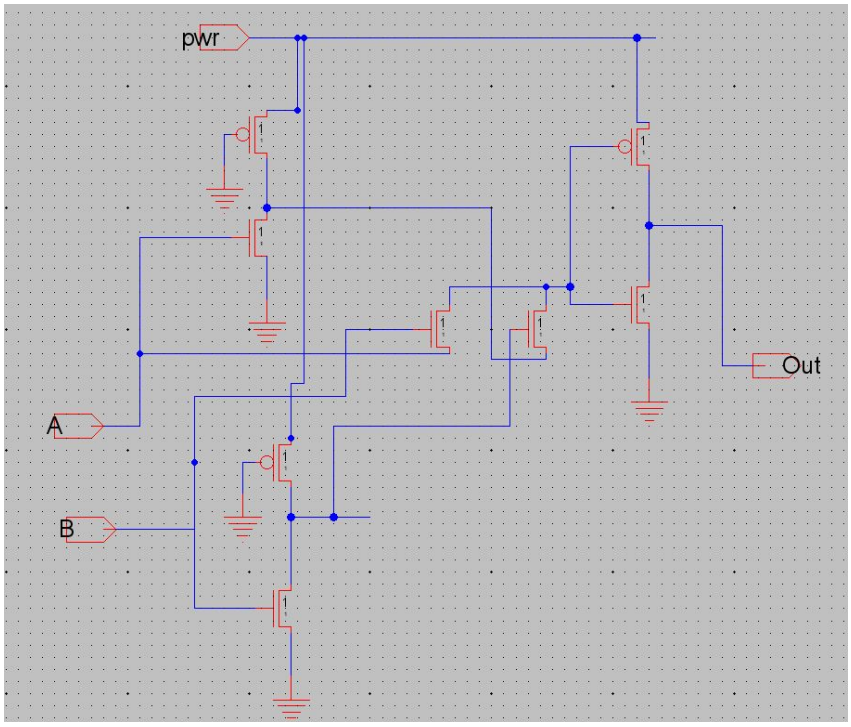


Figure 3.1: An attempt as decreasing the delay of the XOR gate using pass and ratioed logic.

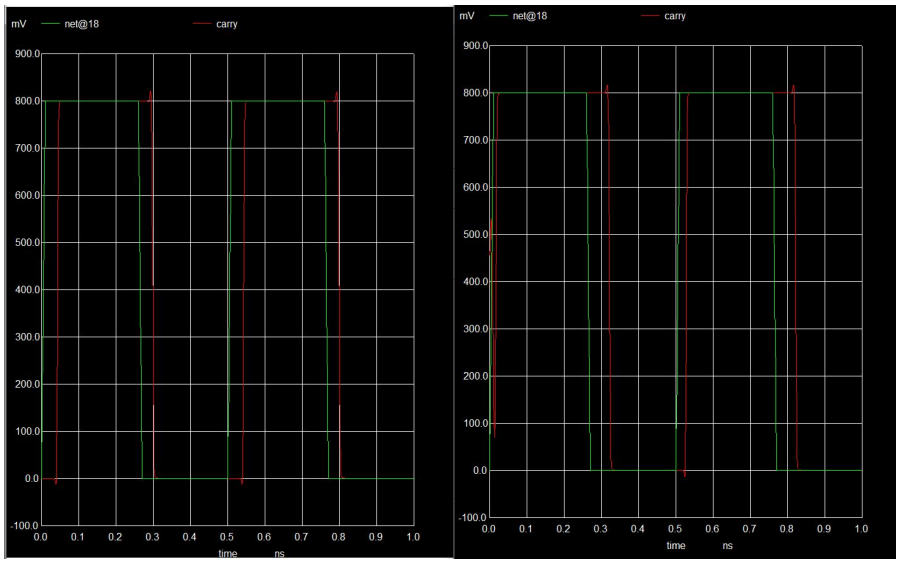


Figure 3.2: graphs comparing the attempt showing that there was increase in delay on the falling edge which is not the desired outcome (left: baseline, right; pass/ratioed)

Attempt 2: Baseline optimization

Changes of Design:

Increasing Vdd to 1.0V

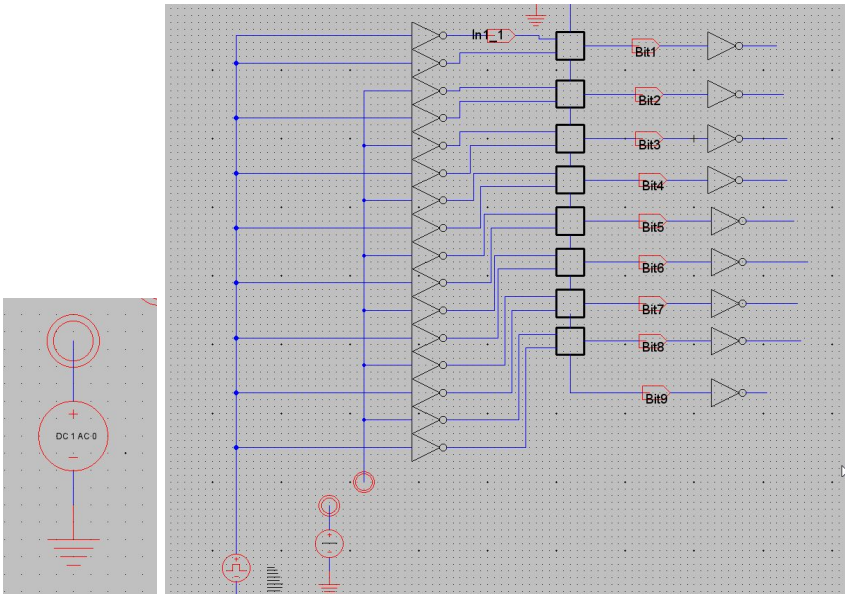


Figure 3.3 The 8-bit adder with vdd set to 1V

```
tran 1ps 1ns;plot In1_1 Bit9;meas tran delay trig v(In1_1) val=0.5 rise=1 targ v(Bit9) val=0.5 rise=1
```

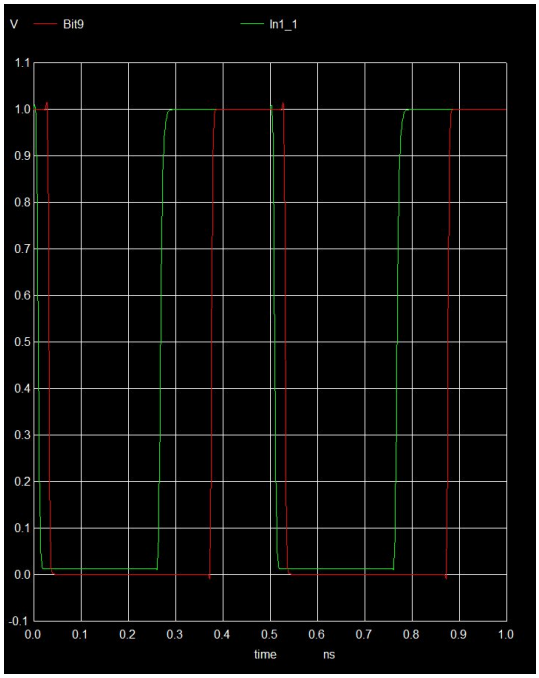


Figure 3.4 The plot of the input and output with Vdd=1V

```
delay                =  1.075539e-10
```

A decrease in time of delay by 37% from the baseline at .8V.

There really is not any insight to be gained from increasing the Vdd other than to observe the increased energy usage. As a direct correlation can be seen by increasing the Vdd, it is most likely a better idea to keep Vdd the same to see the true effects of topology on the delay when compared to the baseline.

Sizing of transistors to optimize delay for carry:

The individual components within the single adder must be optimized. The critical path of the carry is the most important, so the optimization will be on the worst path that was found in the baseline.

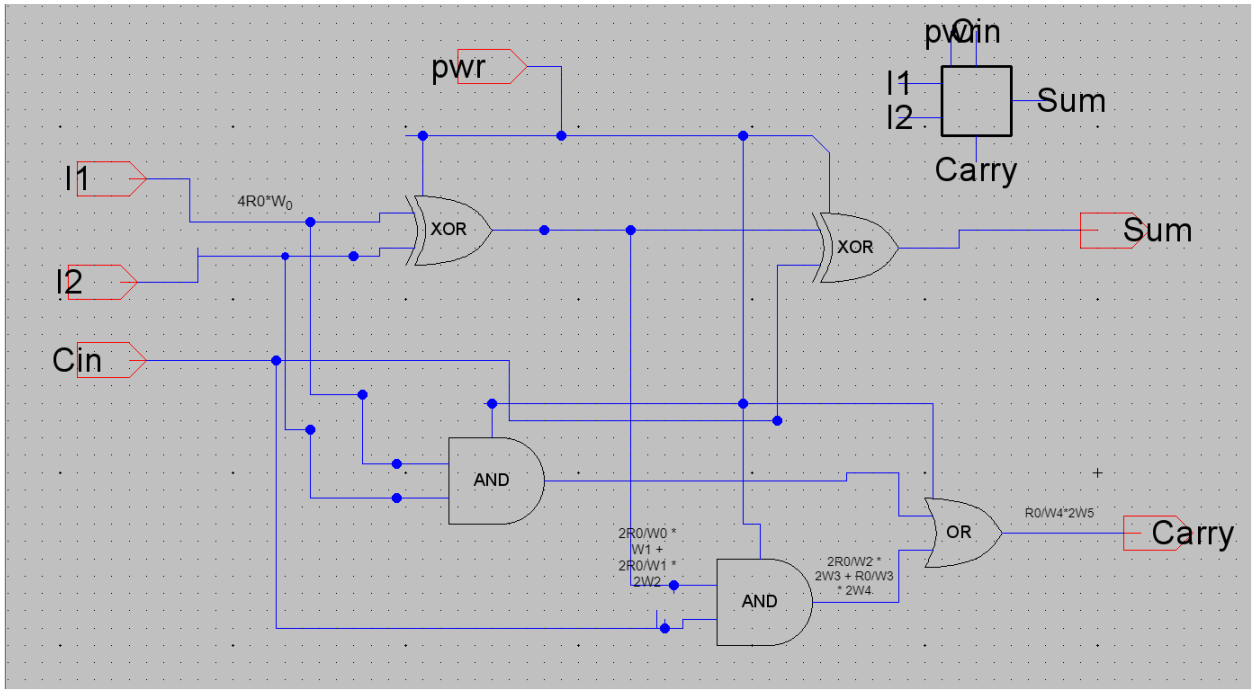


Figure 3.5: The critical path of the baseline implementation with delays annotated.

4R0*W0
+ 2R0/W0 * W1 + 2R0/W1 * 2W2
+ 2R0/W2 * 2W3 + R0/W3 * 2W4
+ R0/W4*2W5 = Delay of critical path

The partial derivatives are taken and must be minimized. With input and output loads of a minimum sized inverter

W0:

Find root of derivative in respect to W0: $4R0 - 2R0/W0^2 * W1$

$W0 = sqrt(W1/2)$

W1:

Find root of derivative in respect to W1: $2R0/W0 - 4R0*W2/W1^2$

$W1 = sqrt(2*W2*W0)$

W2:

Find root of derivative in respect to W2: $4R0/W1 - 4R0*W3/W2^2$
W2 = sqrt(W3*W1)

W3:

Find root of derivative in respect to W3: $4R0/W2 - 2W4 * R0/W3^2$
W3 = sqrt(W4*W2/2)

W4:

Find root of derivative in respect to W4: $2R0/W3 - 2R0*W5/W4^2$
W4 = sqrt(W3*W5)

As there are many possible solutions, set the value for W0 = 1.
Now solve for the rest using the derivatives.

W0 = 1
$W1 = W0^2 * 2 = 2$
$W2 = W1^2 / 2 / W0 = 2$
$W3 = W2^2 / W1 = 2$
$W4 = W3^2 * 2 / W2 = 4$
$W5 = W4^2 * W3 = 8$

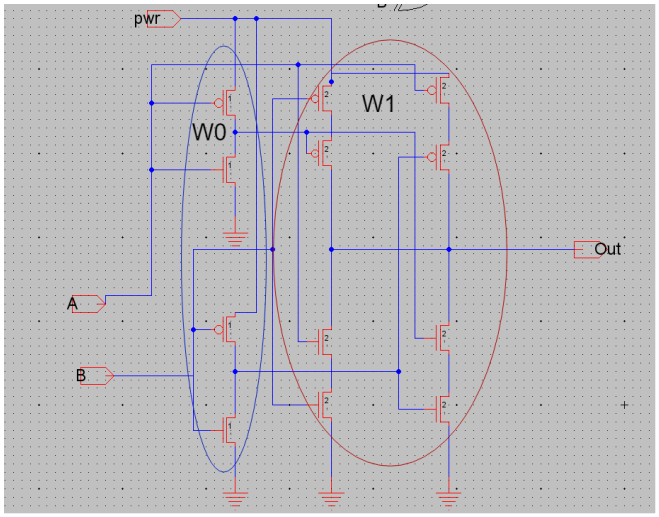


Figure 3.6: Within the XOR showing which transistors widths are represent by W0 and W1

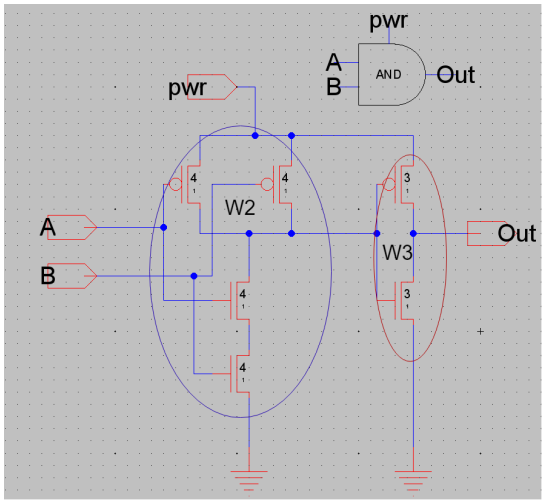


Figure 3.7: Within the AND showing which transistors widths are represent by W2 and W3

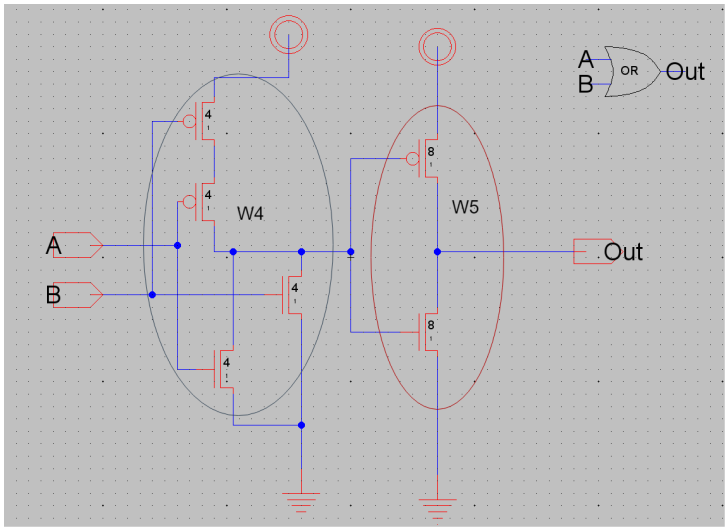


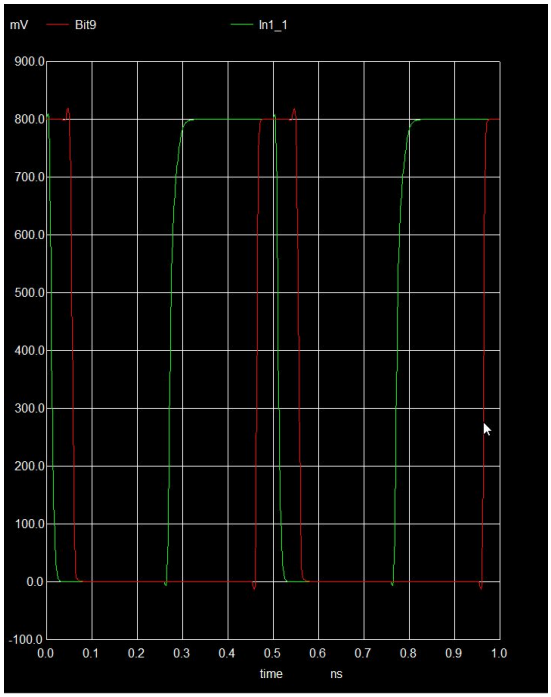
Figure 3.8: Within the OR showing which transistors widths are represent by W2 and W3

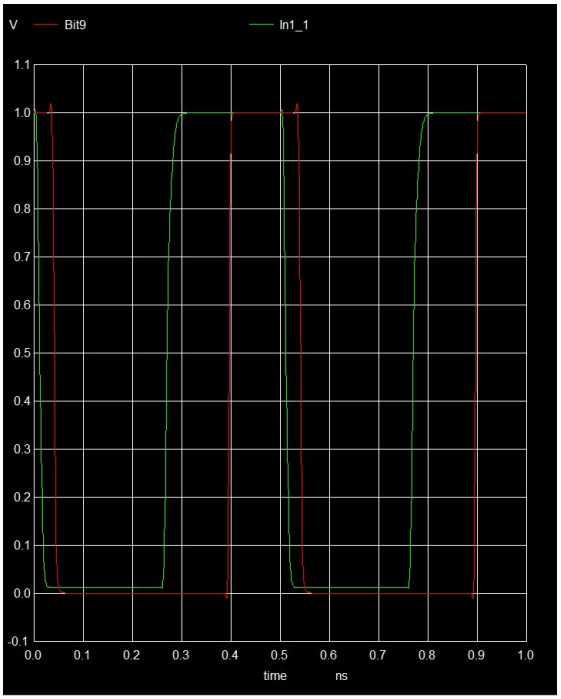
Values on Optimization:

```
tran 1ps 1ns;plot In1_1 Bit9;meas tran delay trig v(In1_1) val=0.5 rise=1 targ v(Bit9) val=0.5 rise=1
```

@Vdd = .8V

@Vdd = 1V






```
delay                  =  1.902546e-10
```

```
delay                  =  1.264570e-10
```

This delay is greater than the baseline for both voltages thus there must be some sort of fine tuning that must be done to improve the delay.

After Fine Tuning:

With the same descriptors for the widths as before

W0 = 1

W1 = 2

W2_p = 2; W2_n = 4

W3 = 3

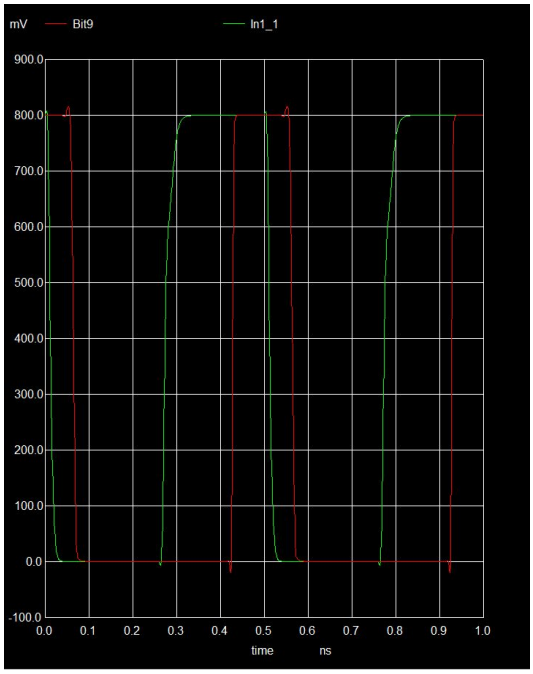
W4_p = 3; W4_n = 6;

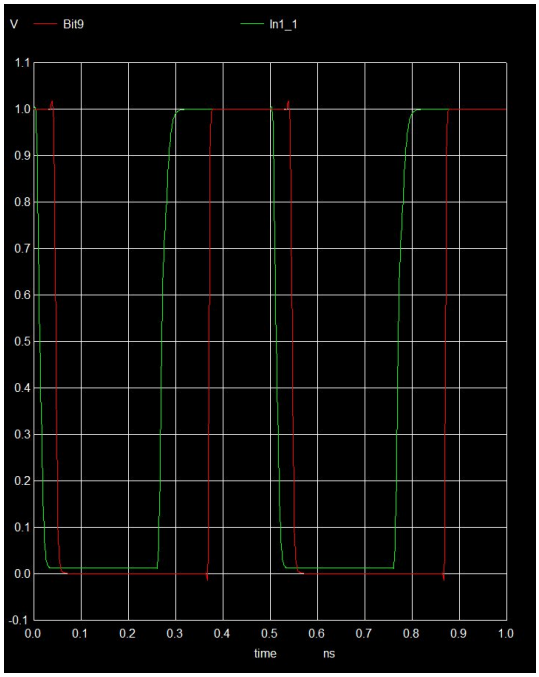
W5 = 6

```
tran 1ps 1ns;plot ln1_1 Bit9;meas tran delay trig v(ln1_1) val=0.5 rise=1 targ v(Bit9) val=0.5 rise=1
```

@Vdd = .8V

@Vdd = 1.0V






```
delay                =  1.527563e-10
```

```
delay                =  1.002613e-10
```

The delays for the critical path is $1.5*10^{-10}$ s and $1.0*10^{-10}$s for the Vdd=.8V case and the Vdd=.1V case, respectively. This is a 12% reduction in delay time at .8V and 6.5% at 1V.

NAND Topology:

NAND Baseline:

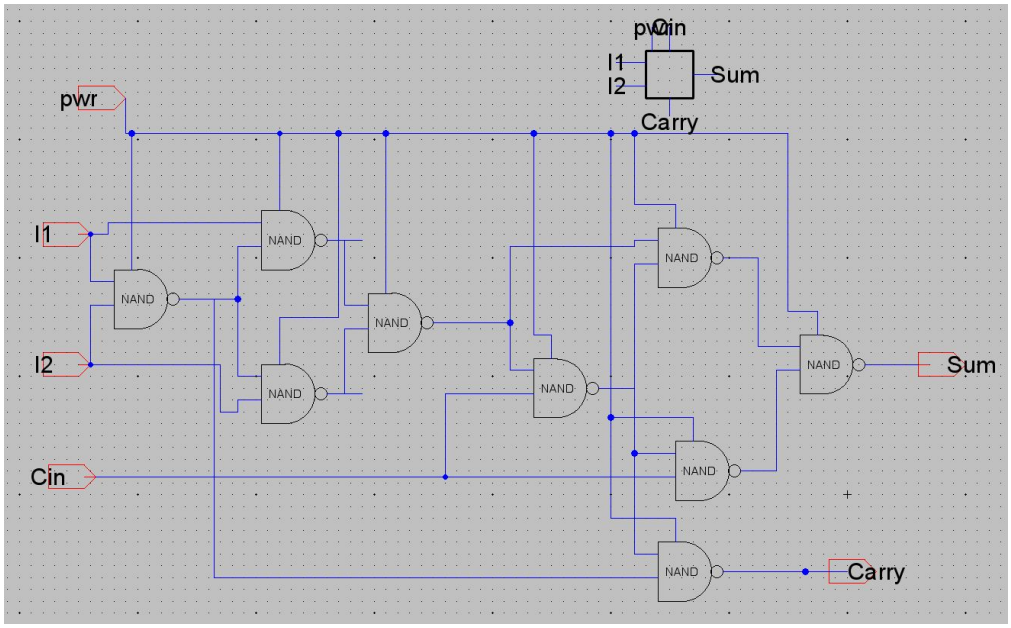


Figure 3.9: Schematic for NAND implementation of the full adder

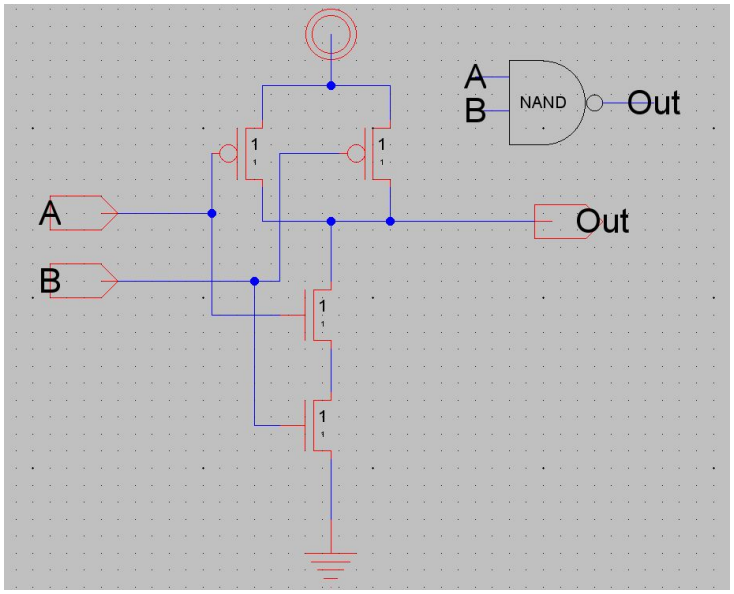


Figure 3.10: Schematic for NAND gate

Delay:

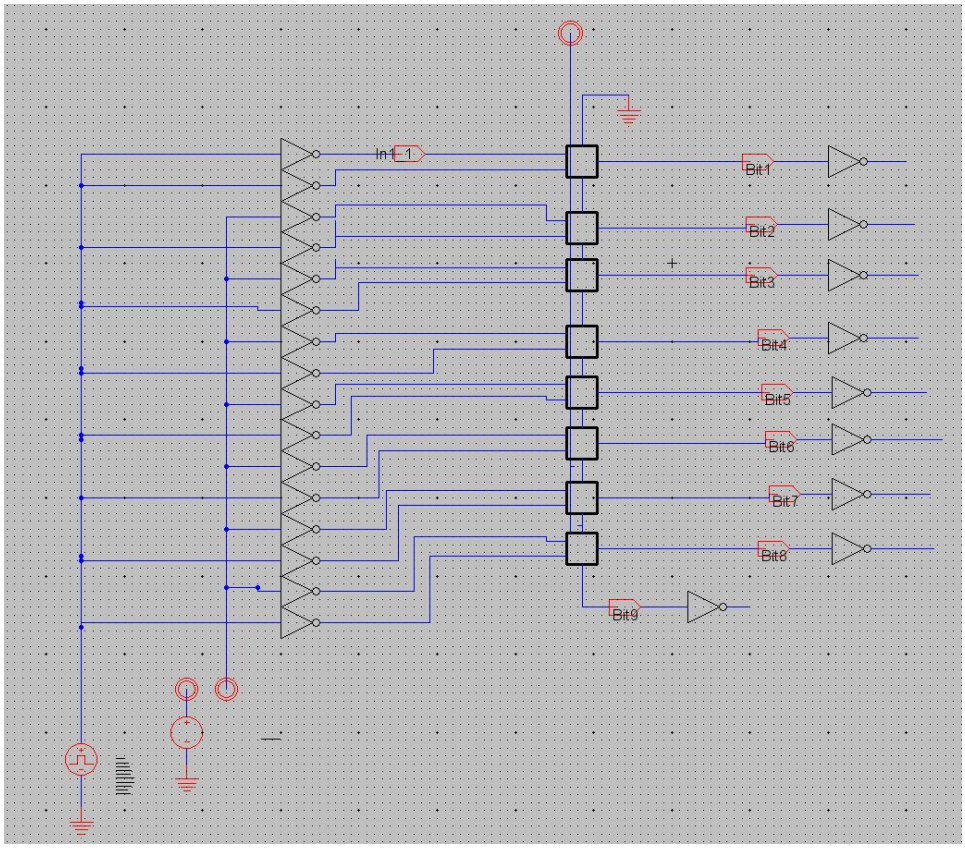


Figure 3.11: Schematic for 8-bit adder

```
tran 1ps 1ns;plot In1_1 Bit9;meas tran delay trig v(In1_1) val=0.5 rise=1 targ v(Bit9) val=0.5 rise=1
```

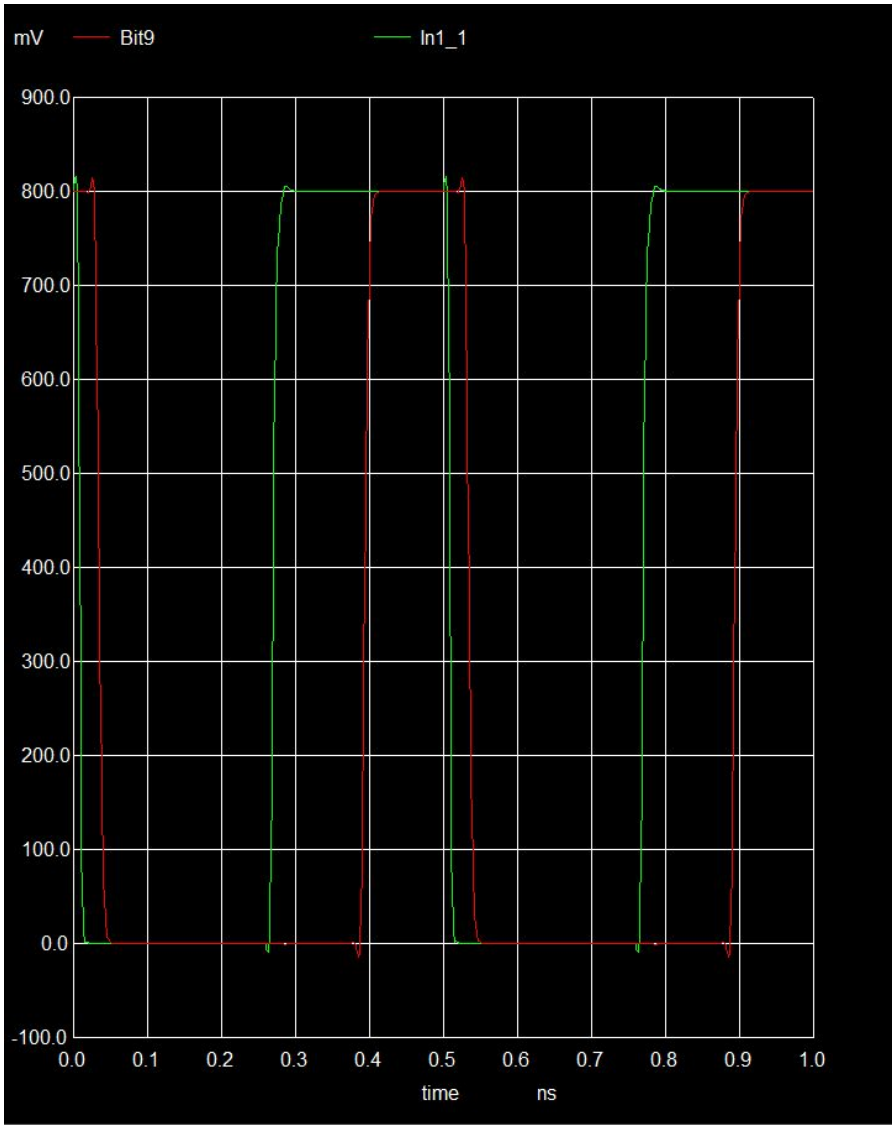


Figure 3.12: Plot of In1_1 and Bit9 showing a decrease in the delay

```
delay                =  1.247132e-10
```

Critical Path Delay:

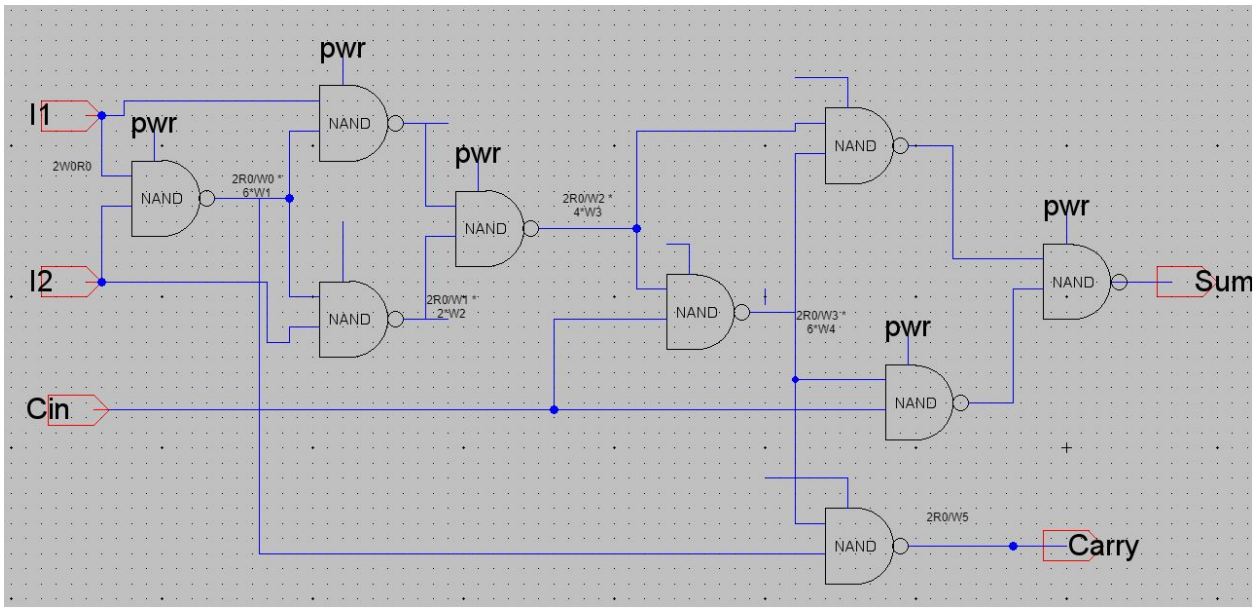


Figure 3.11: Critical path of NAND implementation annotated with delays.

4R0*W0
+ 2R0/W0 * 6W1 + 2R0/W1 * 2W2
+ 2R0/W2 * 4W3 + 2R0/W3 * 6W4
+ 2R0/W4 = Delay of critical path

The partial derivatives are taken and must be minimized. With input and output loads of a minimum sized inverter

W0:

Find root of derivative in respect to W0: $4R0 - 12R0/W0^2 * W1$

W0 = sqrt(3W1)

W1:

Find root of derivative in respect to W1: $12R0/W0 - 4R0*W2/W1^2$

W1 = sqrt(W2*W0/3)

W2:

Find root of derivative in respect to W2: $4R0/W1 - 8R0*W3/W2^2$

W2 = sqrt(2W3*W1)

W3:

Find root of derivative in respect to W3: $8R0/W2 - 12W4 * R0/W3^2$

W3 = sqrt(3W4*W2/2)

W4:

Find root of derivative in respect to W4: $12R0/W3 - 4R0*W5/W4^2$

W4 = sqrt(W3/3)

As there are many possible solutions, set the value for W0 = 3.
Now solve for the rest using the derivatives.
$W1 = W0^2 / 3 = 3$
$W2 = W1^2 / W0 * 3 = 9$
$W3 = W2^2 / 2 / W1 = 13.5$
$W4 = W3^2 / W2 / 3 * 2 = 13.5$
These values will be used as a starting point for the optimization.

```
tran 1ps 1ns;plot In1_1 Bit9;meas tran delay trig v(In1_1) val=0.5 rise=1 targ v(Bit9) val=0.5 rise=1
```

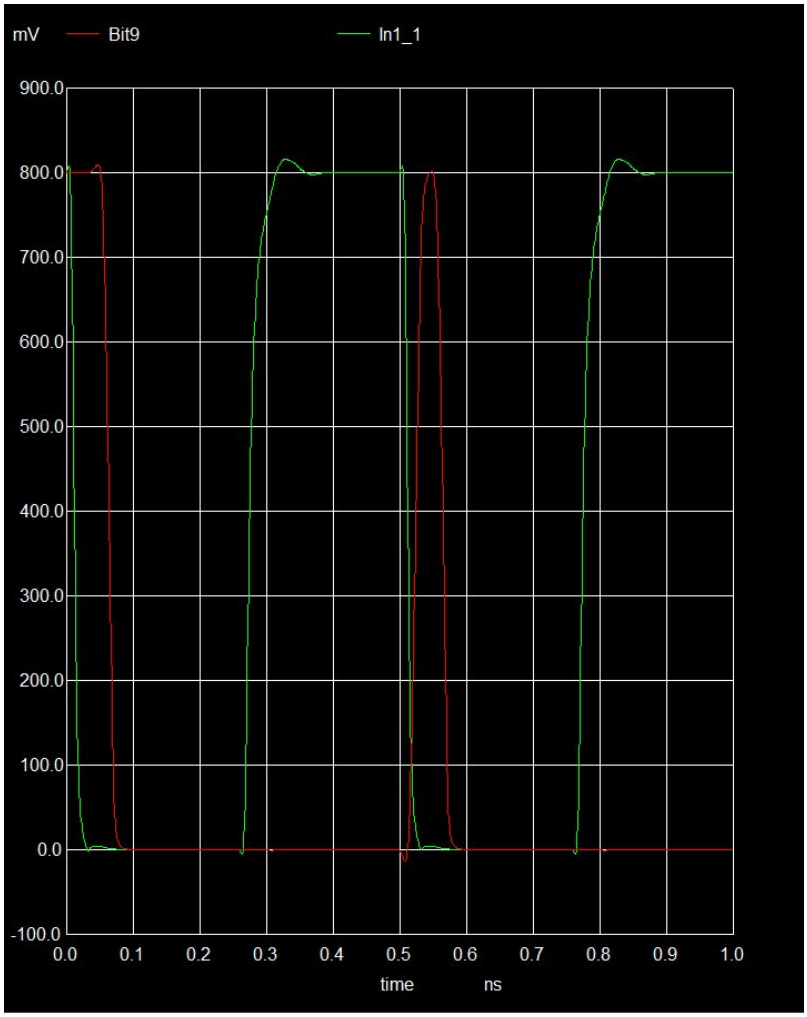


Figure 3.12: Plot of In1_1 and Bit9 showing a increase in the delay

```
delay                =  2.493355e-10
```

As the delay has almost doubled there is a lot of tuning to do to get to a better delay.

3.3. Impact of alternate attempts

The alternate versions started with attempting to decrease the delay by using the pass and ratioed logic. As I did not fully grasp the concept, I did not continue the

path because it was hard to create improvements while knowing what the changes were exactly doing. The next improvement was increased Vdd, though this allowed for major improvements, this change would seem to overshadow the improvements that were more fine tuned. Then the delay optimized widths of the baseline were attempted and gains were made upto around 10%, these were reasonable gains to be made, but other topologies were to be tried. That is where the NAND gate topology is implemented. This form at the minimum sizing had significantly less delay than the baseline. Then a delay optimized widths was taken, this increased the delay, but after fine tuning the final design was completed.

I, Abel Chacko, certify that I have complied with the University of Pennsylvania's Code of Academic Integrity in completing this project.