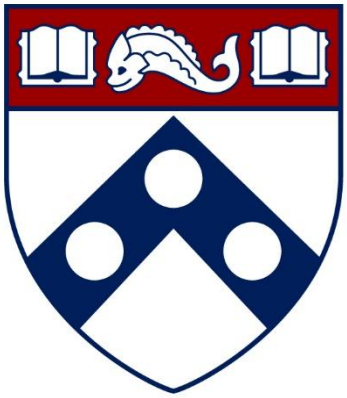


University Of Pennsylvania

School of Engineering and Applied Science

Department of Electrical and Systems Engineering

ESE 3700 Spring 2025

Circuit-Level Modeling, Design, And Optimization For Digital Systems

---

Project 1: Logic Optimization

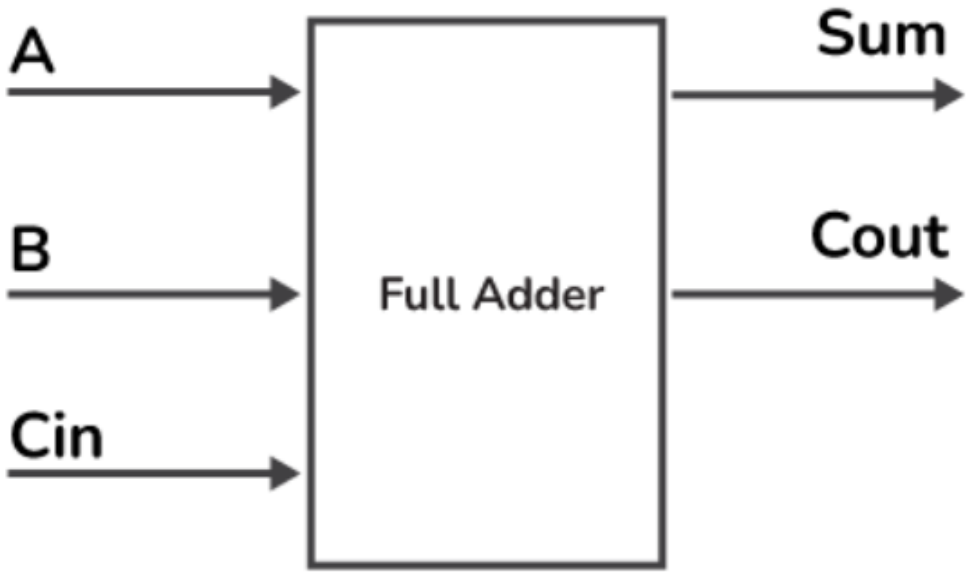
Mohamed Elsheshtawy 17363793

Contents

1- Logic and gates

1.1- Full adder circuit logic

1.1.1- Inputs and outputs



We have three inputs. Two normal inputs and one carry input which is the result of a previous stage. We have two outputs which are Sum and Cout. In order to build this circuit we need to create a truth table for and a logic expression for each output.

1.1.2- Sum and Cout Truth table

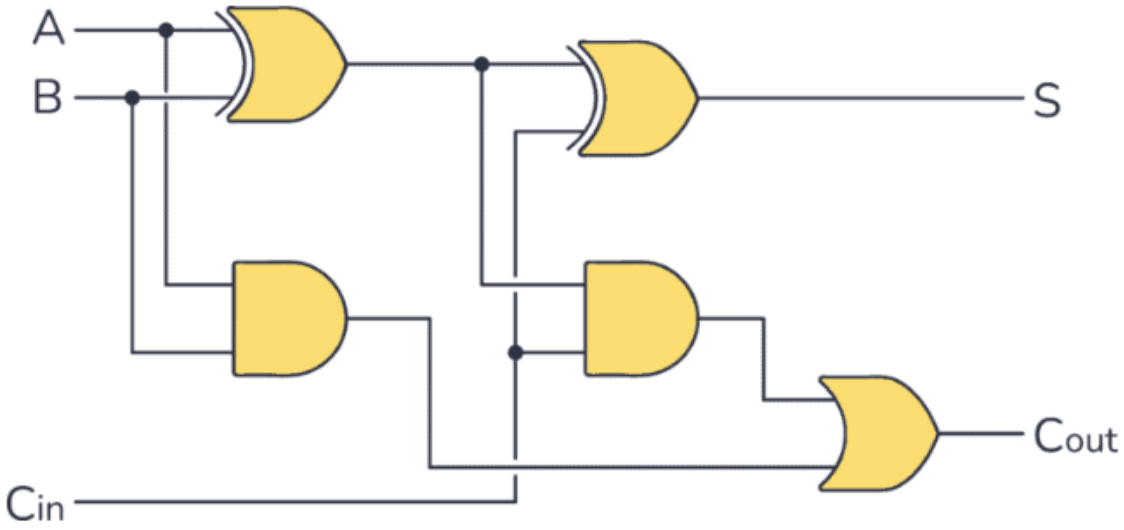
A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

After doing Kmap for each output the result was the following:

Sum = A^B^Cin Cout = AB + Cin(A^B) (using a Kmap trick)

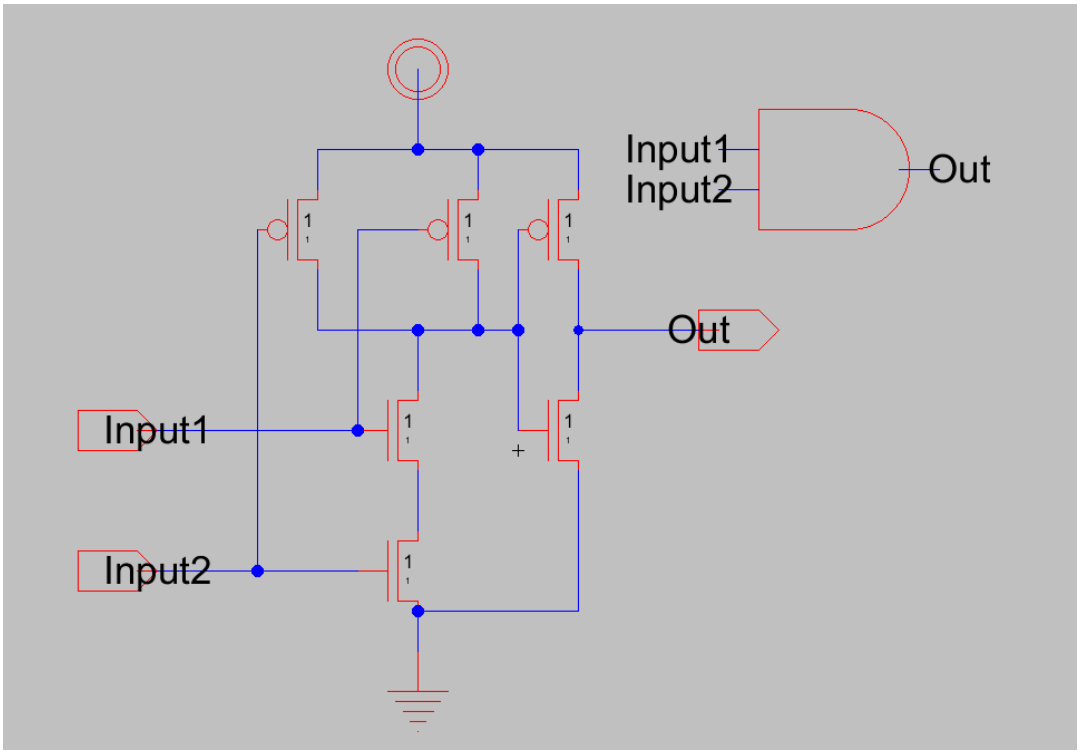
Now,as we can see we need three types of gates: 1- AND gate 2- XOR gate 3- OR gate

1.1.3- Circuit topolgy

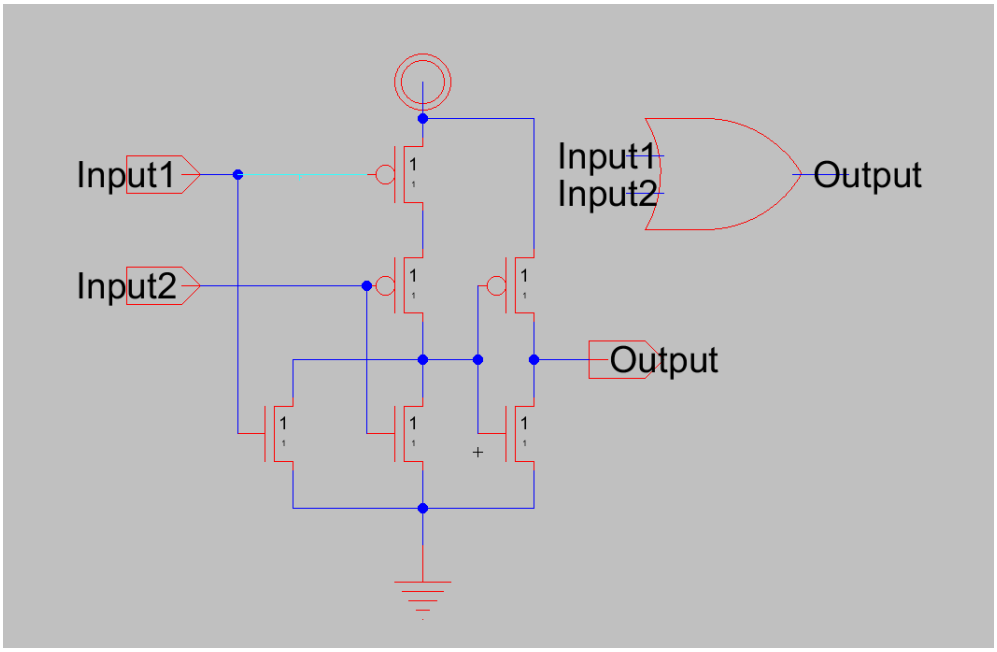


1.2- Adder and Gates Design on Electric

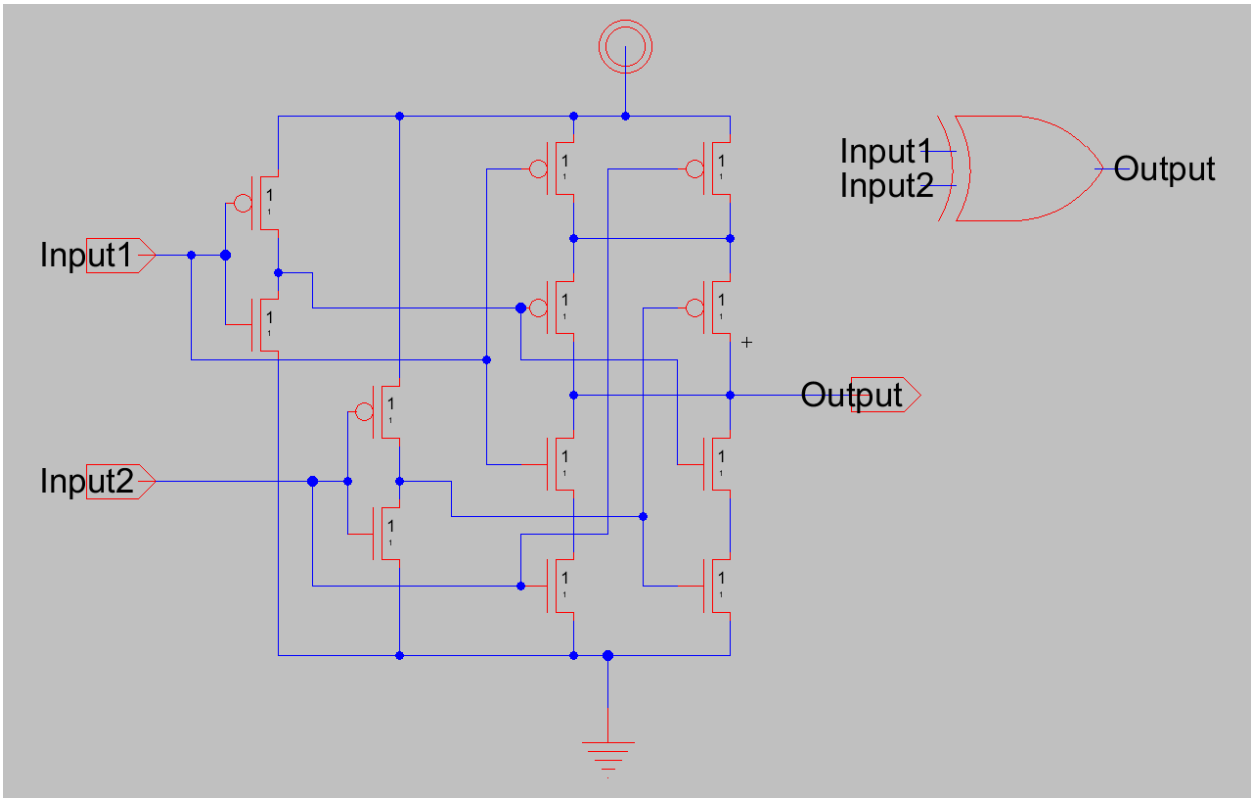
1.2.1- AND gate:



1.2.2- OR gate:

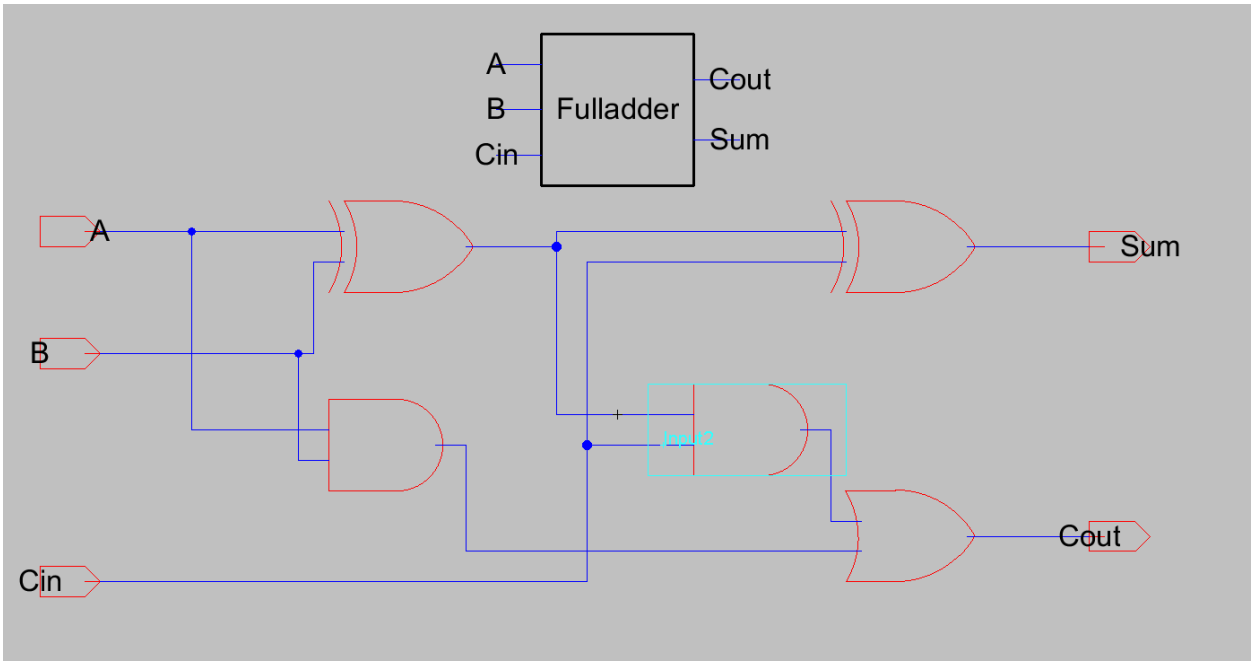


1.2.3- XOR gate:



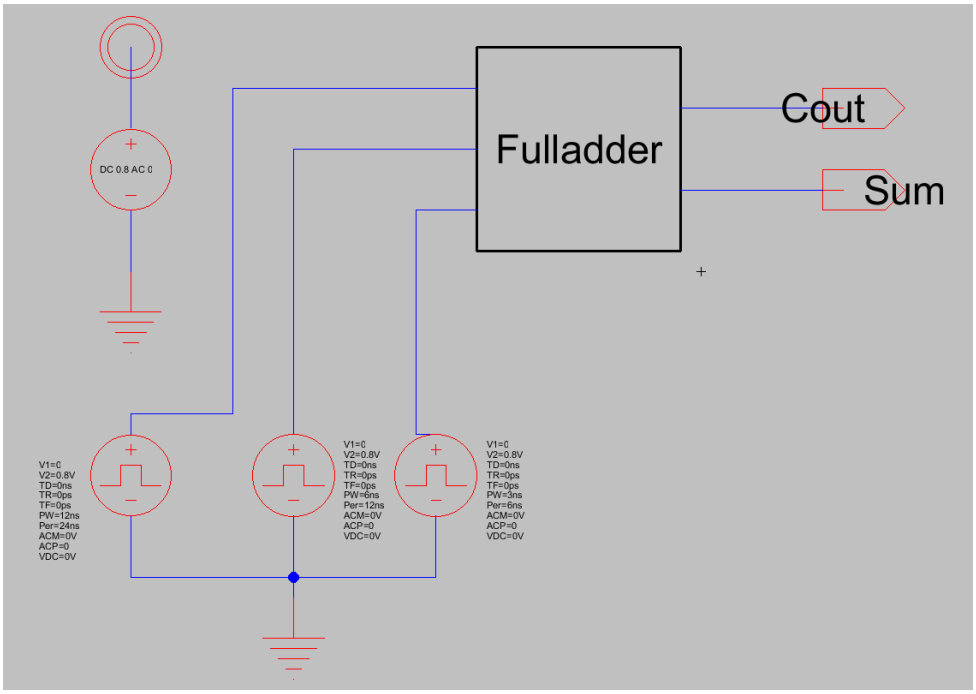
Now, we have all the gates we need to build a full adder and then cascade it to build an 8-bit adder.

1.2.4- Full adder:



1.2.5- Testing:

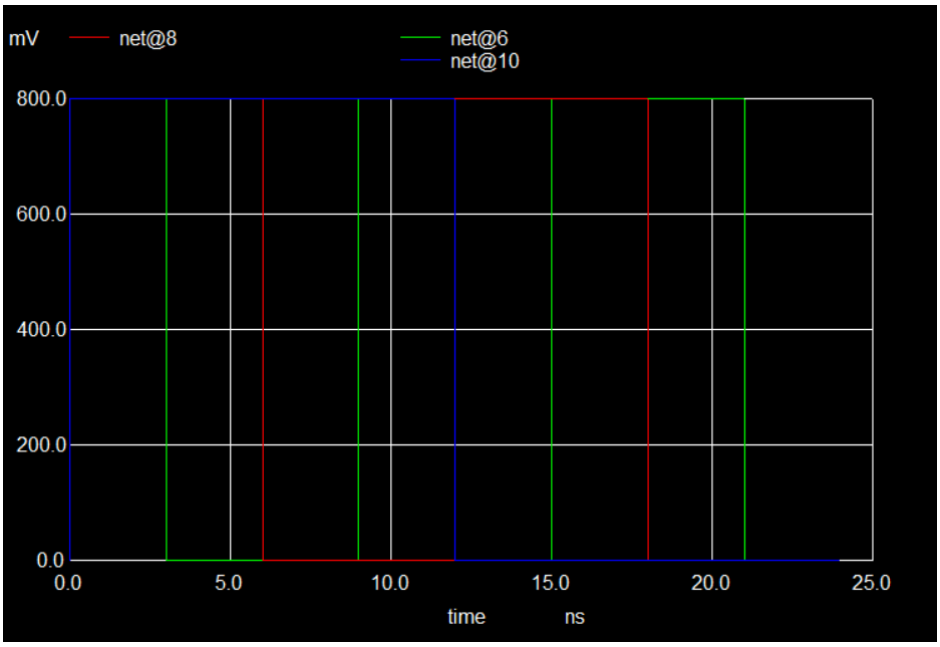
In order to test it in one shot I will add three V_pulses such that first one has a period A and second one has period 2A and third one has period 4A. This way we will cover all 8 cases as the voltage sources will make all the combination due to the period difference.



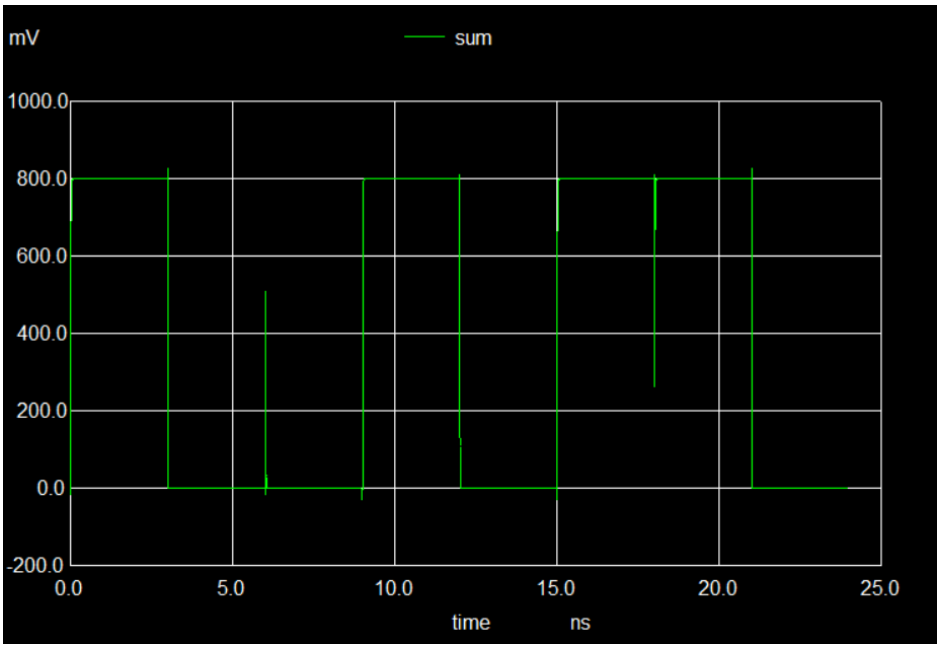
I will simulate it on spice using the following codes:

FATest.spi
tran 1ps 24ns
plot net@6 net@8 net@10
plot Sum
plot Cout

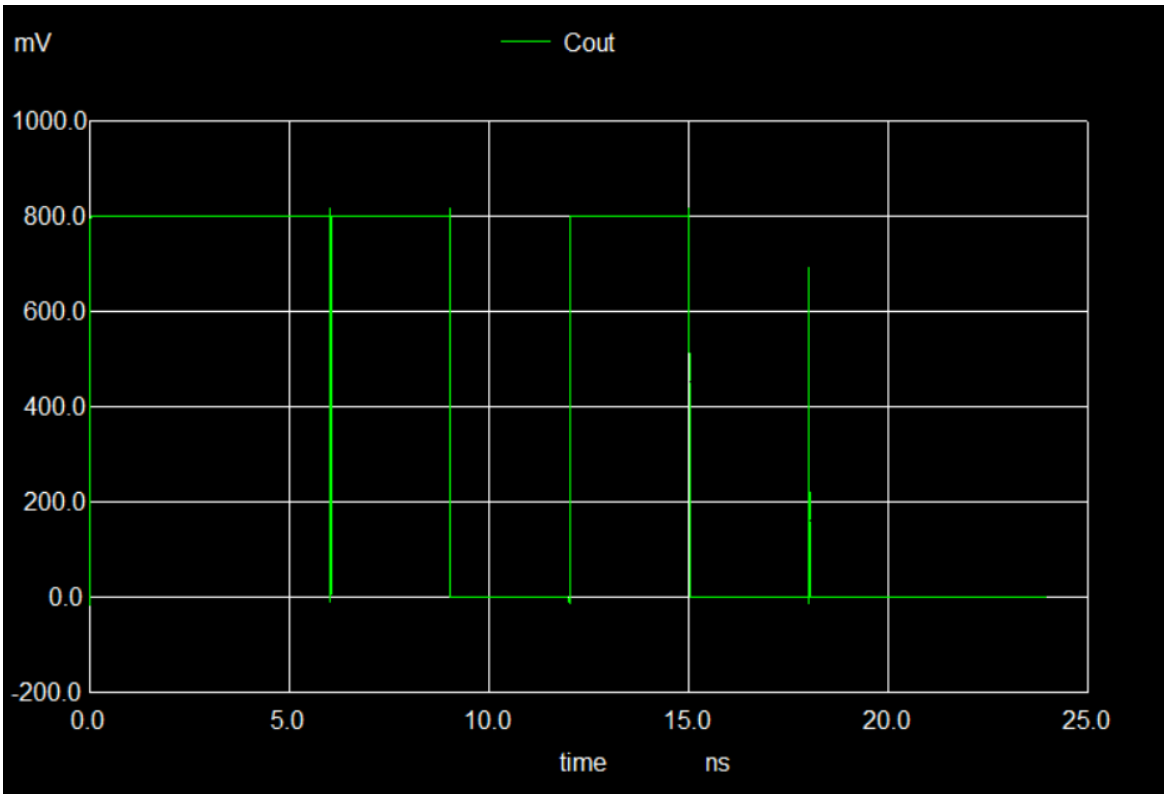
Inputs Curve: (starts from 111 and goes to 000)



Sum Curve: (those random spikes are due to transitions)



Cout Curve: (same case for dips and spikes due to transitions)



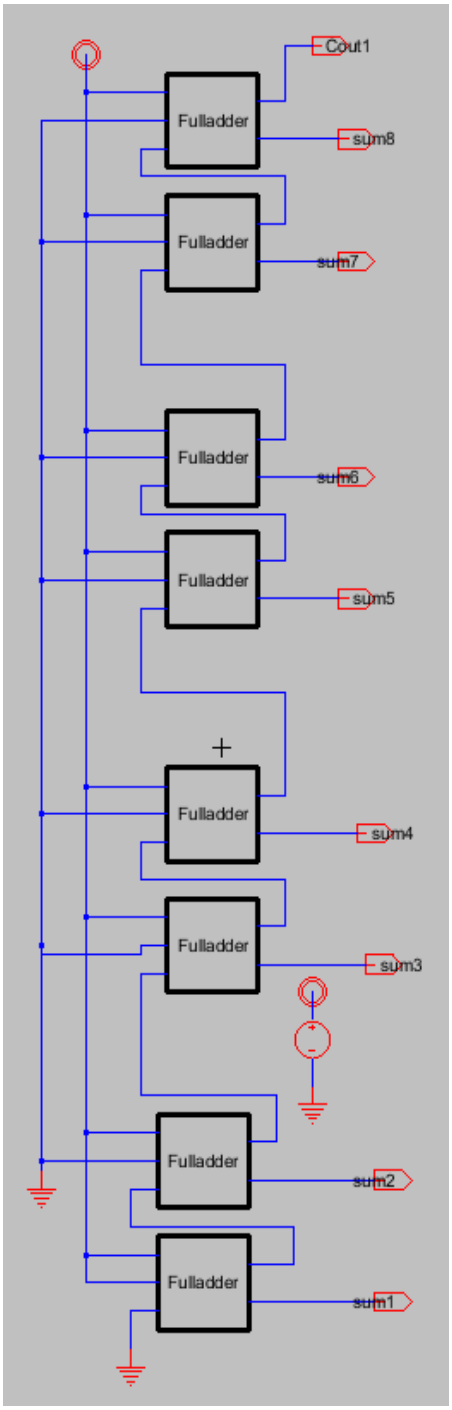
2- Unoptimized Design:

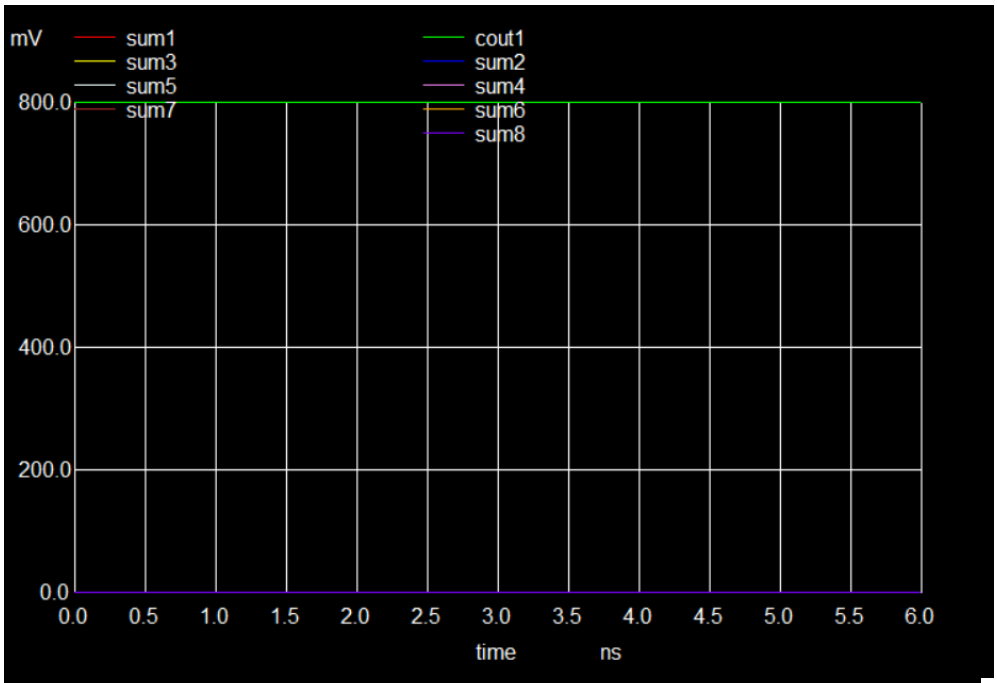
2.1- Testing:

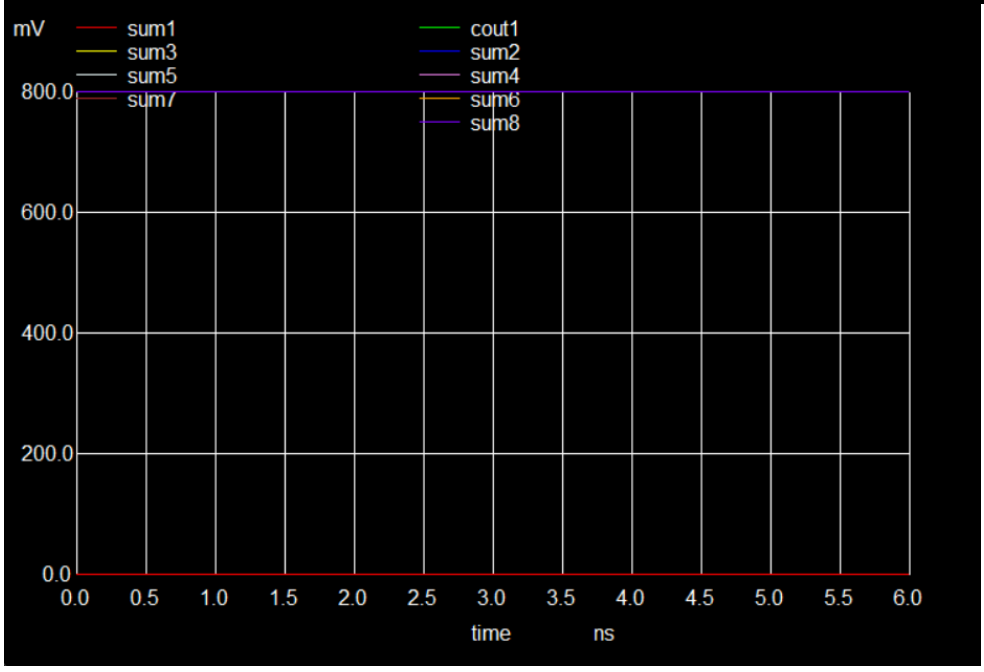
We need to test the functionality of the circuit so lets try those test cases.

1) 11111111 + 00000001 (this will test if Couts are cascaded correctly)
2) 11111111 + 11111111 (this will test cascade and also addition)

Our circuit is on the left and the first graph on the right is first case and second graph is 2nd case.



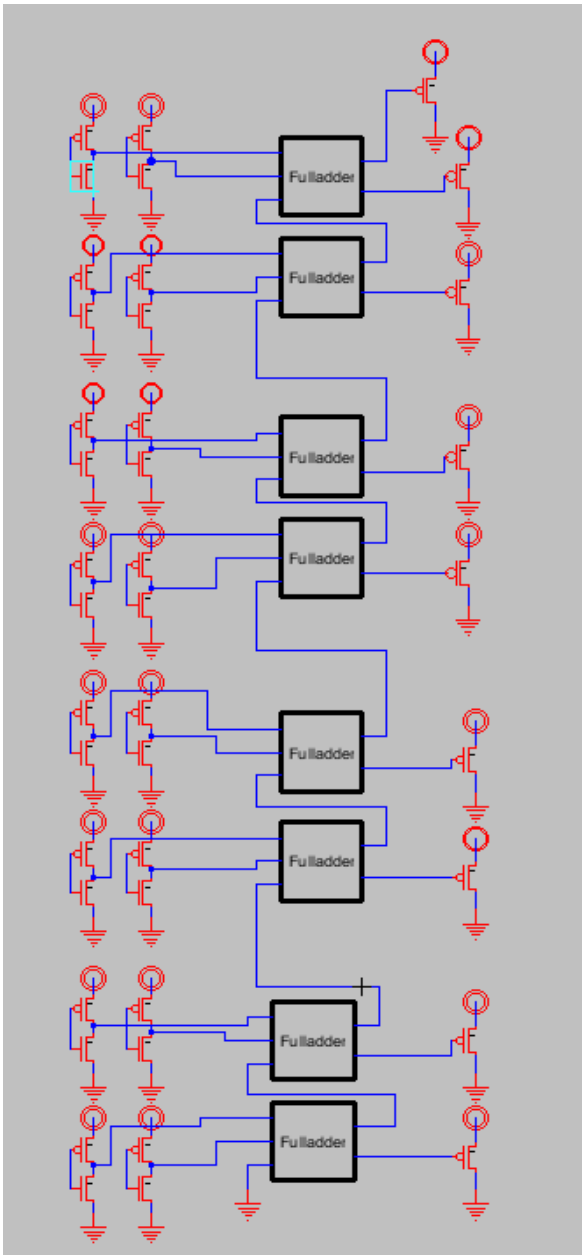


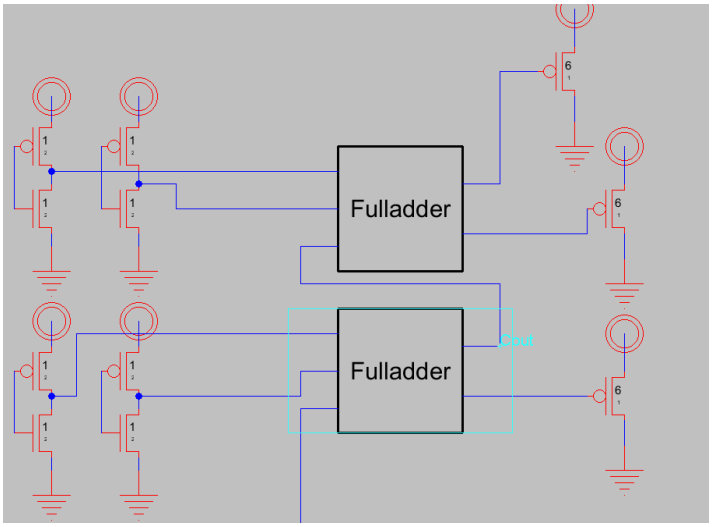


As we can see the results are totally right.

2.2- Delay:

Now we need to drive the input with the same drive of the output and load the output. The drive should be equal to the sum node output which was driven by XOR so we need to calculate the drive of XOR. The highest resistance is 2R, so we should drive the input with an equivalent W/L = 0.5. Regarding the output load it should be the same as A input or B input. The output load should be 6Co.





2.2.1- Test Case:

This is the circuit that we will measure the delay for.
The worst delay is the input A or B of the first full adder to the output Cout of the last adder. And a good input that would drive a Cout to 1 is (11111111)+(00000001) such that the second number switch from all zeros to this. This is the worst delay we can have. But first I will calculate the tau expression for it then I will do the simulation.
We will calculate the delay for 1st full adder then 2nd full adder multiply it by 7 and add the output path of the 8th adder.

2.2.2- Theoretical Calculations:

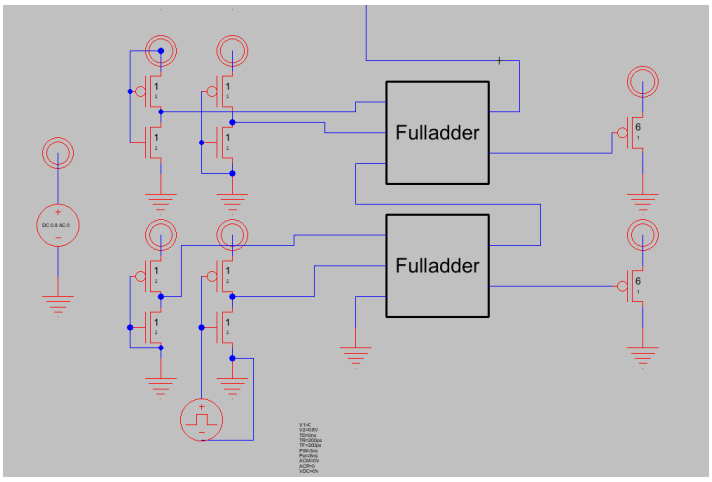
First full adder:
From inverter to XOR and AND: Ro*6Co= 6RoCo.
From XOR to NAND and NAND to Inverter: 2Ro*2Co + 2Ro*2Co = 8RoCo
From AND to NOR and NOR to Inverter: Ro*2Co +2Ro*2Co = 6RoCo

Second full adder:
From Inverter of the OR to input Cin (one NAND and XOR): Ro*6Co = 6RoCo
From NAND to its inverter: 2Ro*2Co = 4RoCo
From AND to NOR and NOR to Inverter: Ro*2Co +2Ro*2Co = 6RoCo

If we add them we get last Cout = (6+8+6)RoCo + (16*7)RoCo = 132RoCo
Now last Cout is loaded with 6Co then Total Delay is = 138RoCo

2.2.3- Simulation

Lets now measure the simulation. I will measure the net of the output of the inverter driven by Vpulse and Cout of last full adder



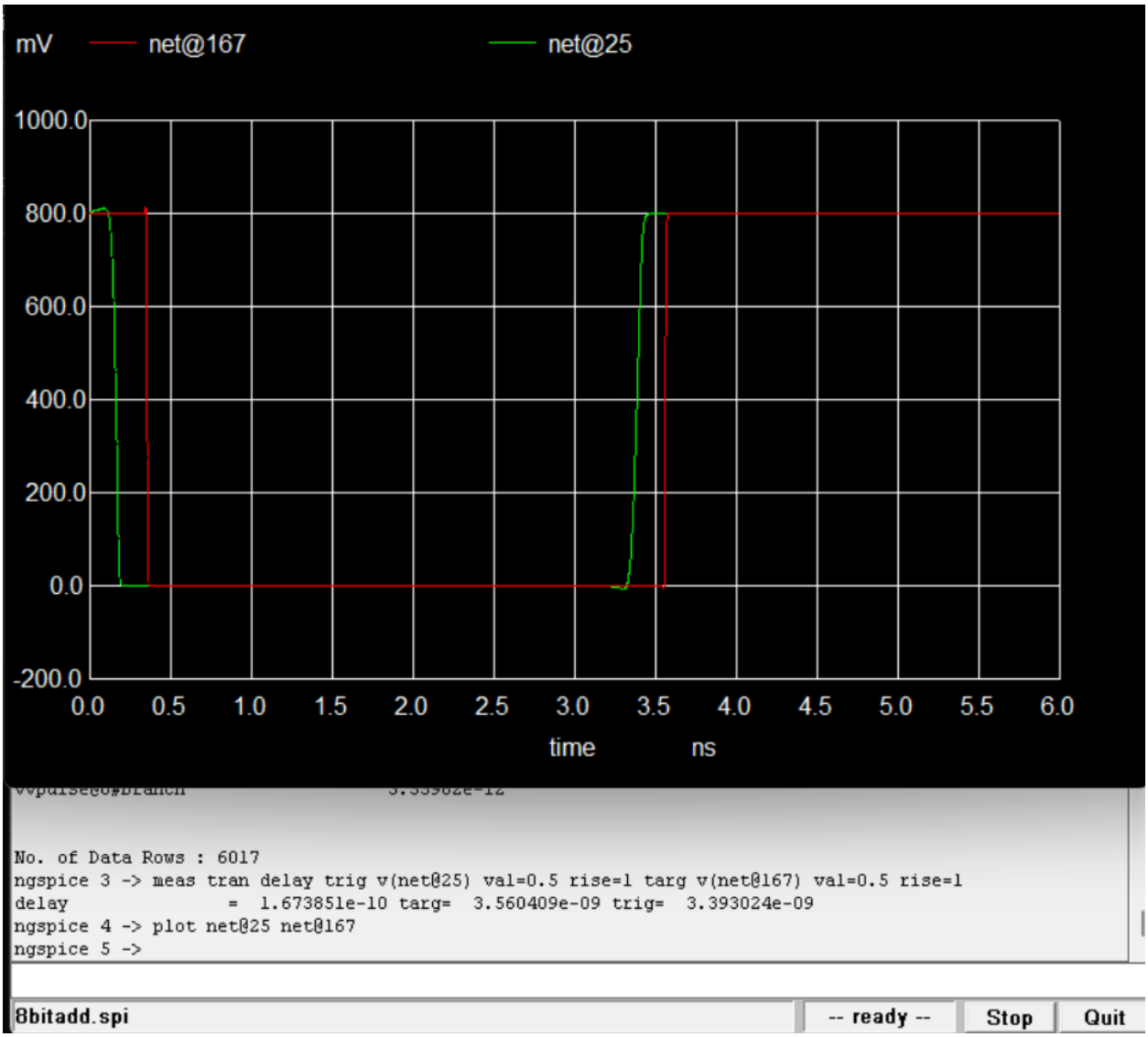
I will use the following Spice code:

8bitadd.spi

tran 1ps 6ns

meas tran delay trig v(net@25) val=0.5 rise=1 targ v(net@167) val=0.5 rise=1

plot net@25 net@167

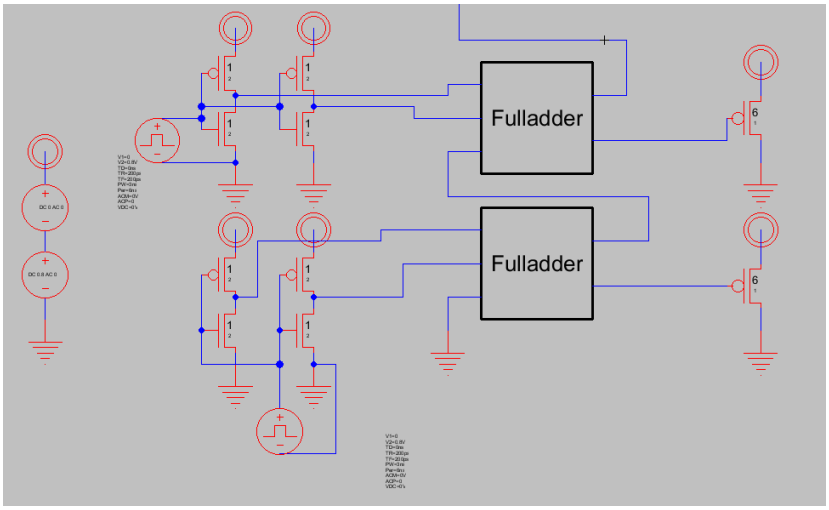


The Delay = 167 ps

2.3- Energy:

We will calculate the active energy for the maximum energy which is the switching case from 0 to 1 for all of them.

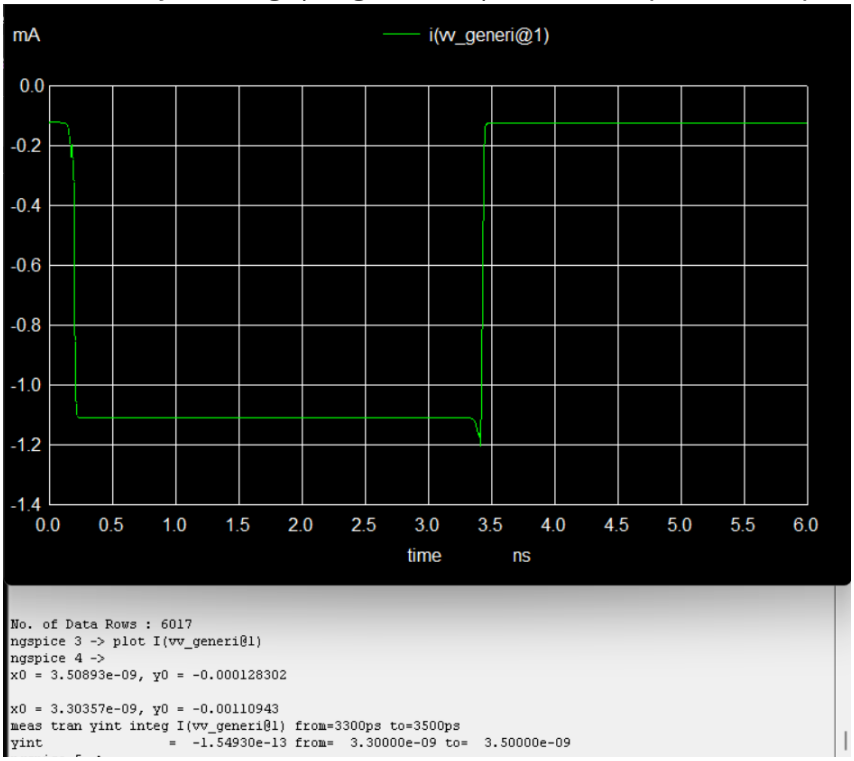
2.3.1- Circuit



2.3.2- Maximum Energy:

The Code I used to measure this circuit:

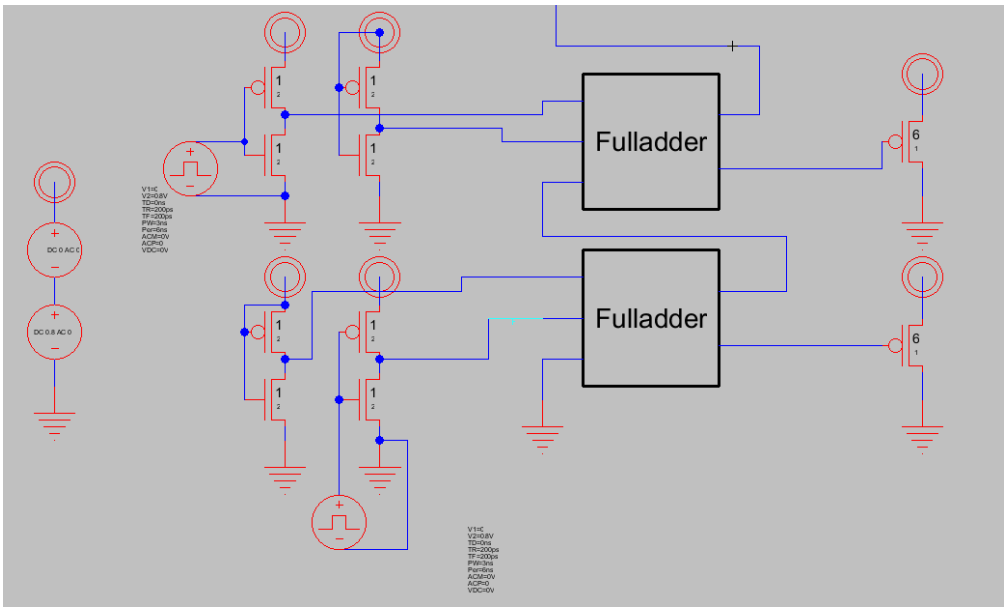
-8bitadd.spi -tran 1ps 6ns -plot I(vv_generi@1)
-meas tran yint integ I(vv_generi@1) from=3300ps to=3500ps



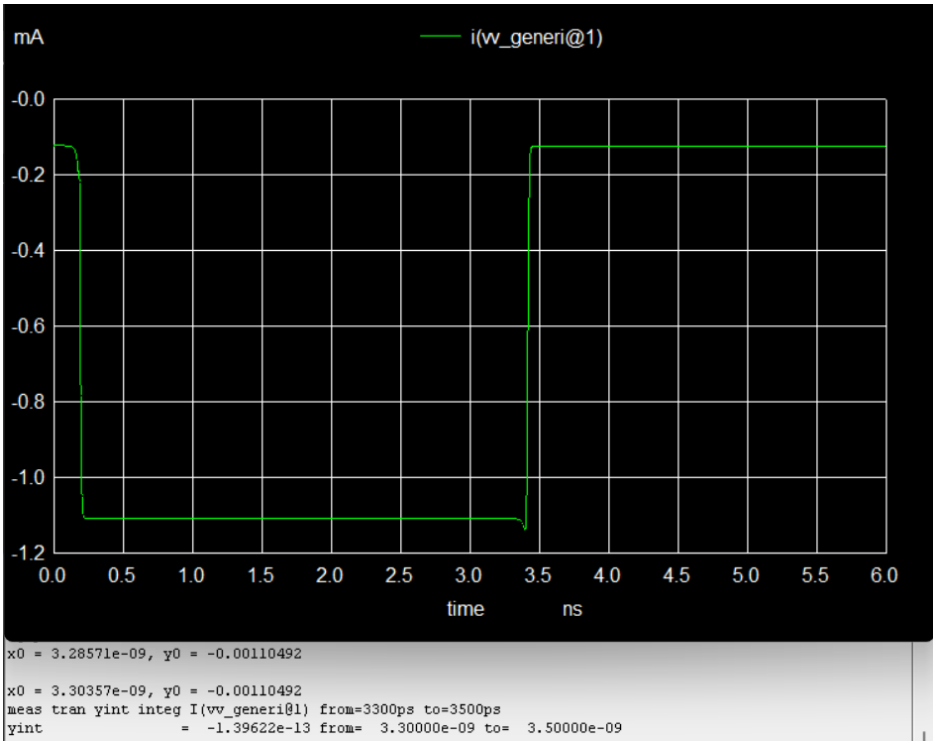
I took integration from 3300 to 3500 because this is the part where switching happened.
Energy = 1.55*10^(-13)*0.8 = 1.24*10^(-13)J

2.3.3- Average Energy:

Half of the inputs from 0 to 1



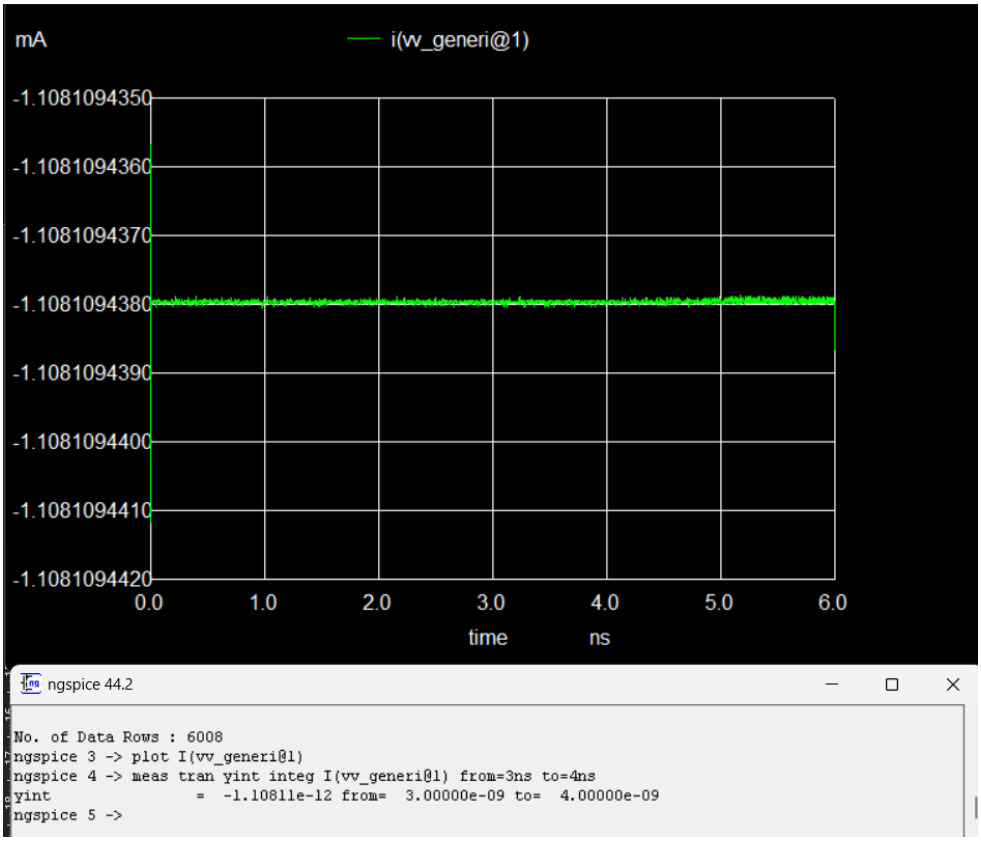
-8bitadd.spi -tran 1ps 6ns -plot I(vv_generi@1)
meas tran yint integ I(vv_generi@1) from=3300ps to=3500ps



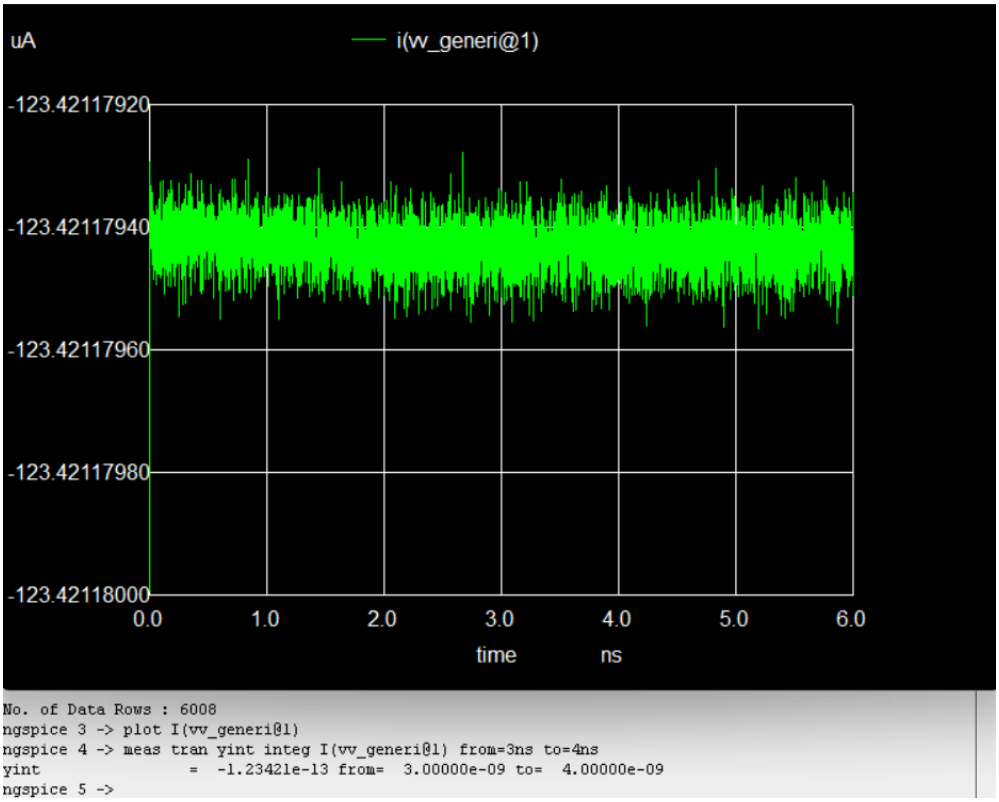
The Energy =
1.4*10^(-13)*0.8 =
1.12*10^(-13)J

2.4- Leakage Energy:

The highest leakage energy is at inputs all zeros. This is because that all the outputs will be 0s and this will result in all the PMOSs leaking current. When everything is outputting zeros this means that NMOS is on and PMOS always leak more current when closed. The lowest leakage would be the inverse of this (all ones).



Maximum leakage energy in nanosecond = 1.1 * 10^(-12)*0.8 = 0.88pJ



Minimum leakage energy in nanosecond = 0.123 * 10^(-12)J*0.8 = 0.1pJ

2.5- Area:

The area calculation for the full adder will be as follow. Since I am cascading 8 full adders then I will just calculate the area for one adder and multiply it by 8.
In the full adder there is two XORs and two ANDs and one OR.
XOR size: 12
AND size: 6
OR size: 6
Size of full adder: 2*12 + 2*6 +6 = 42
Full size of 8bit adder = 42*8 = 336 (we can multiply this by 22nm to get the size and multiply it with the length 22nm and get the full size in nanometers but this wasn't said in project doc)

Spec	Value
Maximum Delay	167ps
Maximum active energy (during switching)	0.12 pJ
average active energy (during switching)	0.11pJ
Maximum leakage energy in a nanosecond	0.88 pJ
Minimum leakage energy in a nanosecond	0.1pJ
Area	336

3- Optimized Design

3.1- Design

3.1.1- Logic

Sizing the transistors only won't get us so far. As a result, I will do another design. If we are adding two numbers the logic is as following:

Sum = A^B^Cin Cout = AB + Cin(A^B) (using a Kmap trick)

The biggest problem in the ripple carry design is that the carry has to be computed in each stage then propagate to the other with this induction relation:

$$C_i = AB + (A^B)C_{i-1}$$

If we call AB -> G and call A^B -> P we can produce an equation for any C.

Let our base case be C_0 = Cin.

i =1:

$C_1 = G_1 + P_1C_0 = G_1+P_1C_{in}$

i =2:

$C_2 = G_2 + P_2C_1 = G_2+P_2(G_1 + P_1C_{in}) = G_2+P_2G_1 + P_2P_1C_{in}$

i =3:

$C_3 = G_3 + P_3C_2 = G_3+P_3(G_2 + P_2G_1+ P_2P_1C_{in}) = G_3+P_3G_2 + P_3P_2G_1+ P_3P_2P_1C_{in}$

i =4:

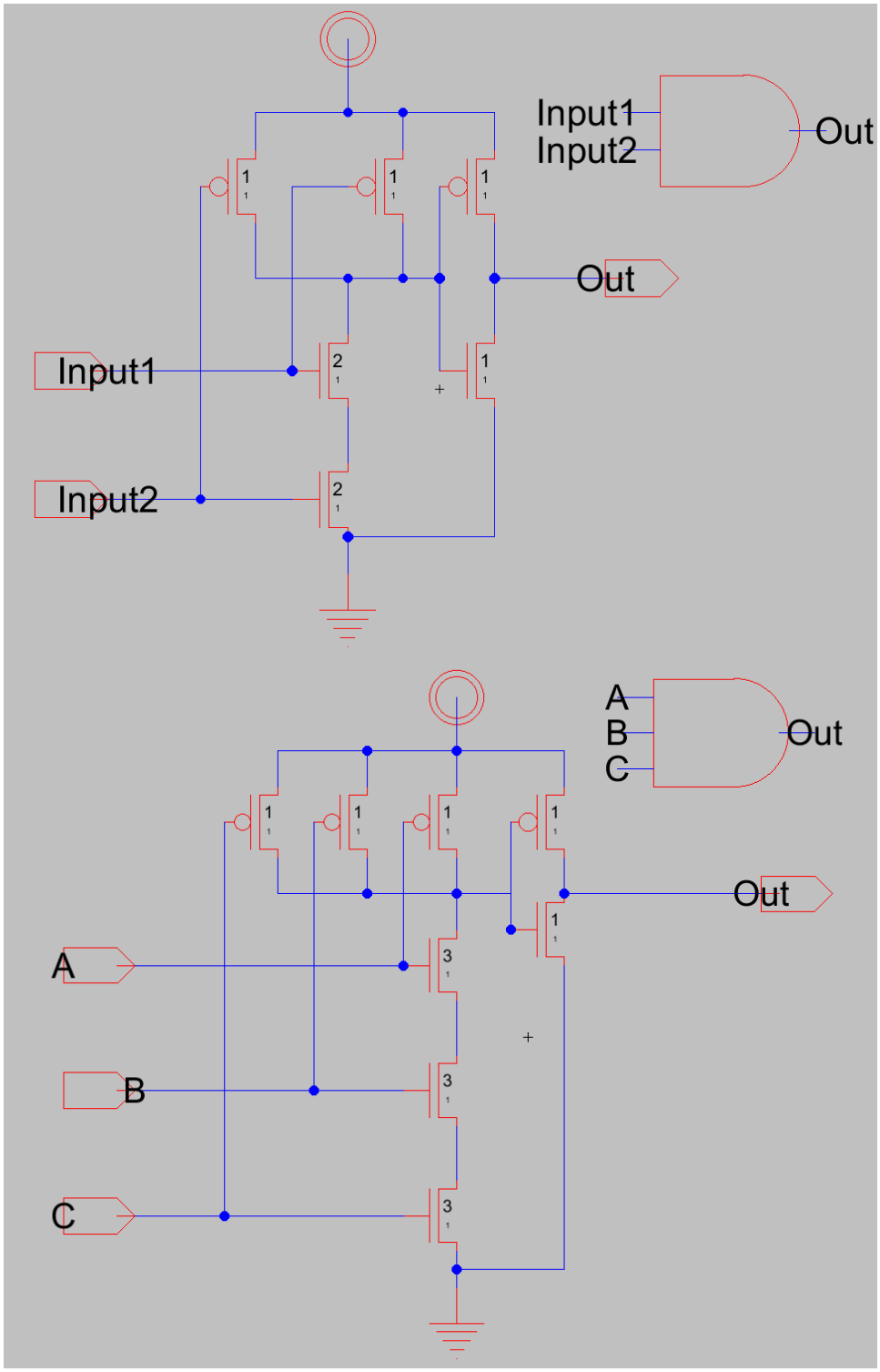
$C_4 = G_4 + P_4C_3 = G_4+P_4(G_3+P_3G_2 + P_3P_2G_1+ P_3P_2P_1C_{in}) = G_4+P_4G_3+ P_4P_3G_2 + P_4P_3P_2G_1+ P_4P_3P_2P_1C_{in})$

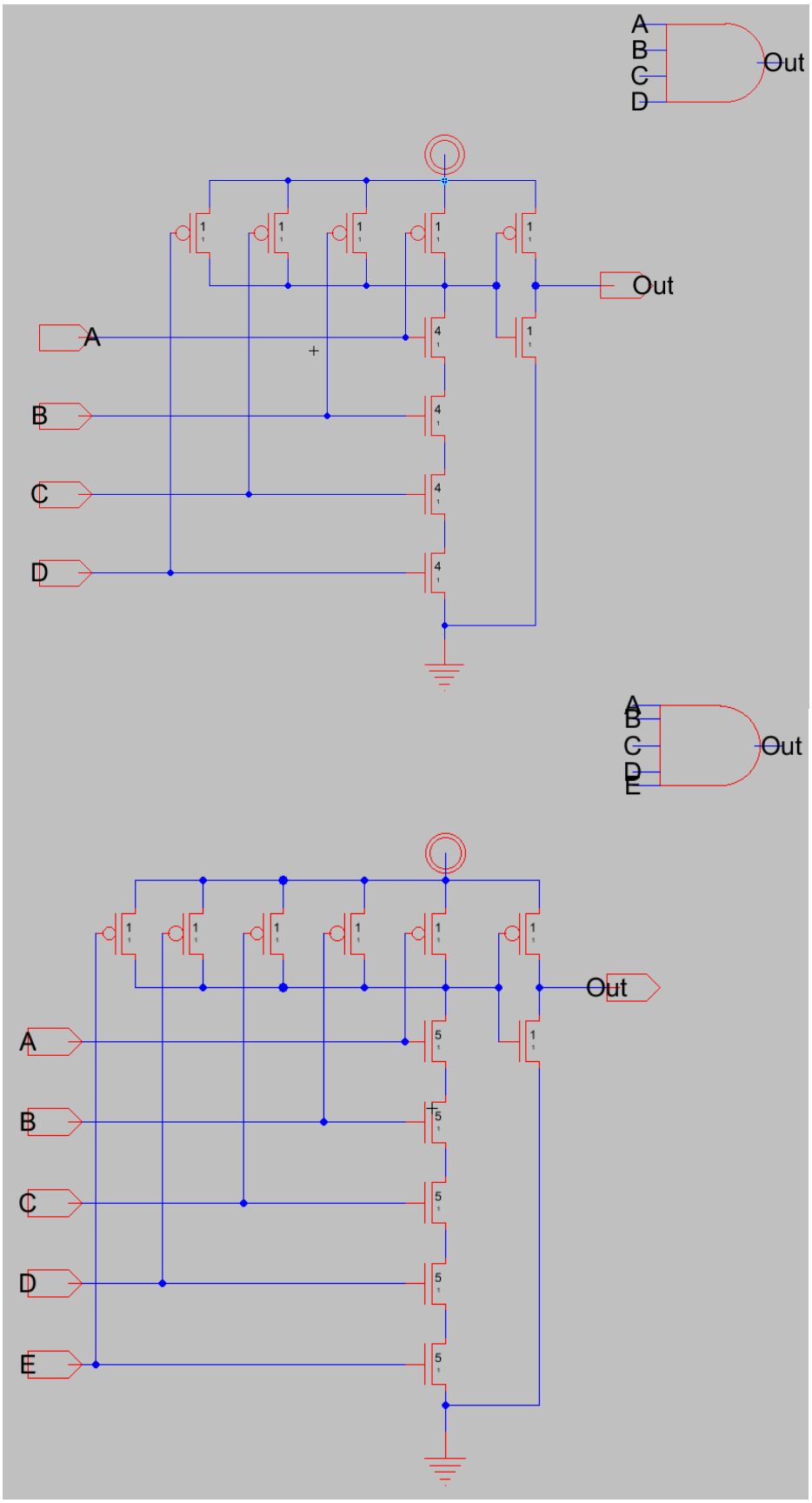
As we can see we created an expression for our carry without the need of the previous carry which will enable us to compute all outputs in parallel, which will reduce the delay, but we will have to create 5 input OR and AND gates. Also, we could have made the same, but for an 8 bit however this would have had a bigger delay because of the 10 input AND.

Now lets start implementing:

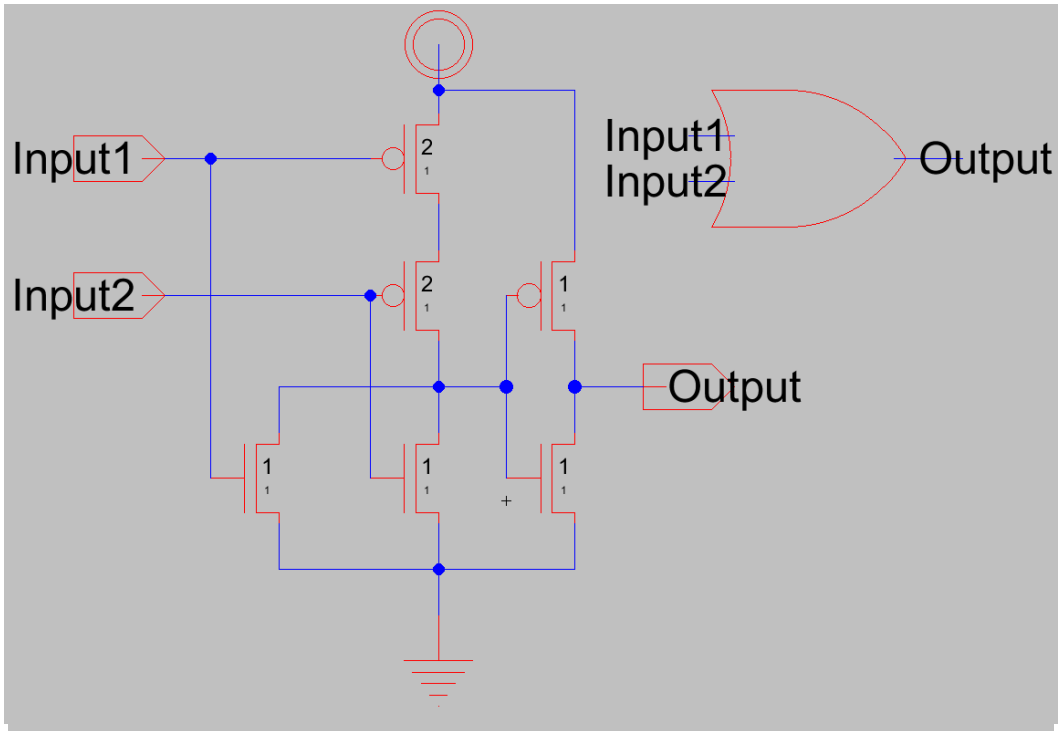
We had already did XOR, AND,OR for 2 inputs, so we need the other gates and we need to size all of them so that they have R = Ro.

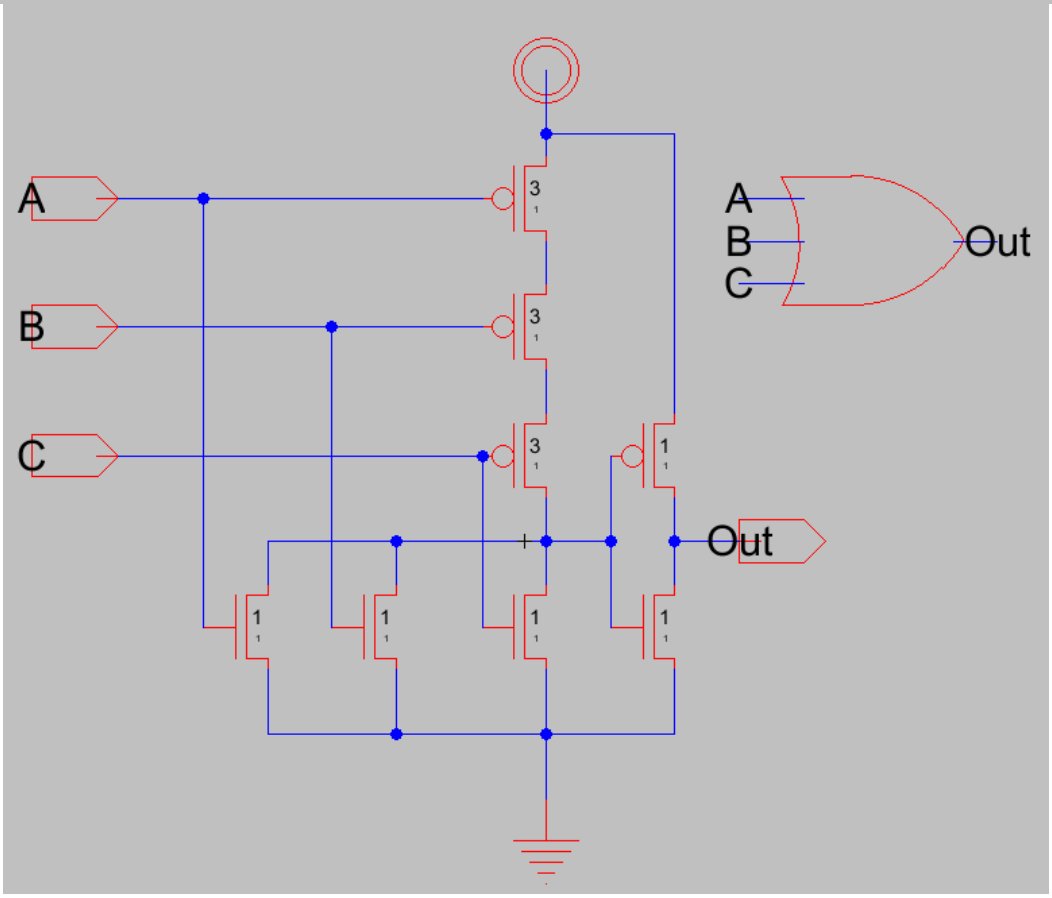
3.1.2- AND gates

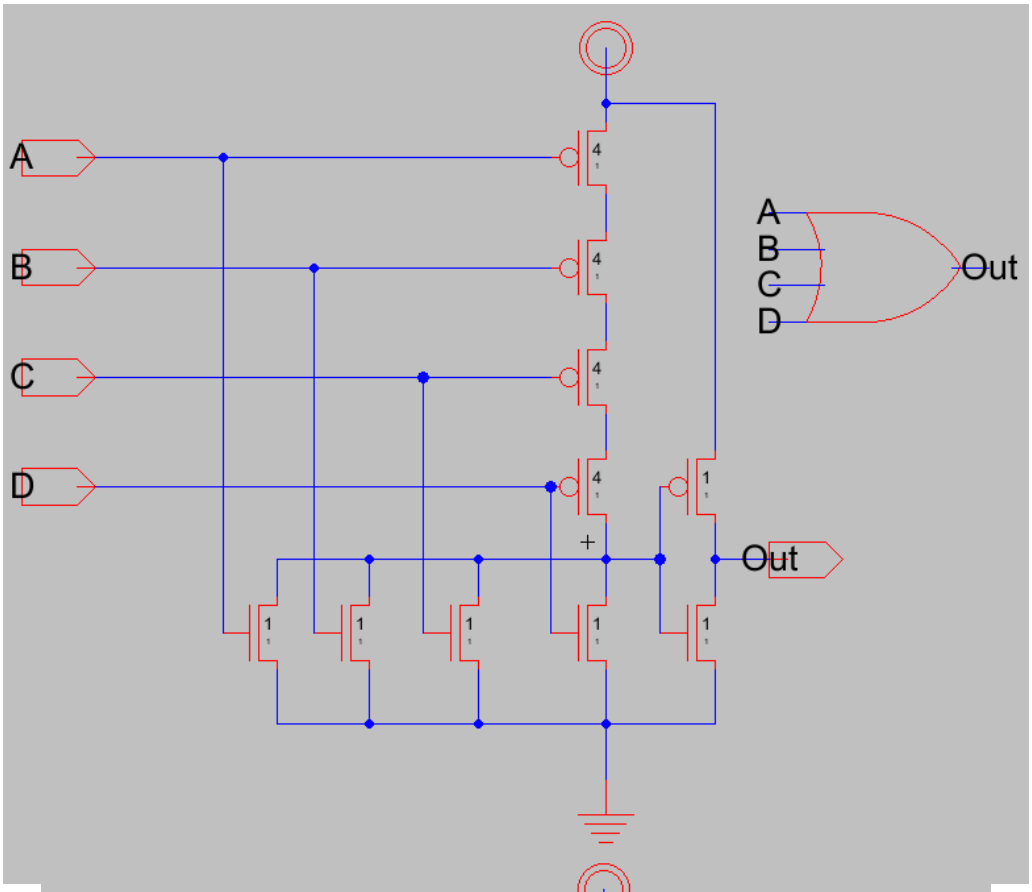


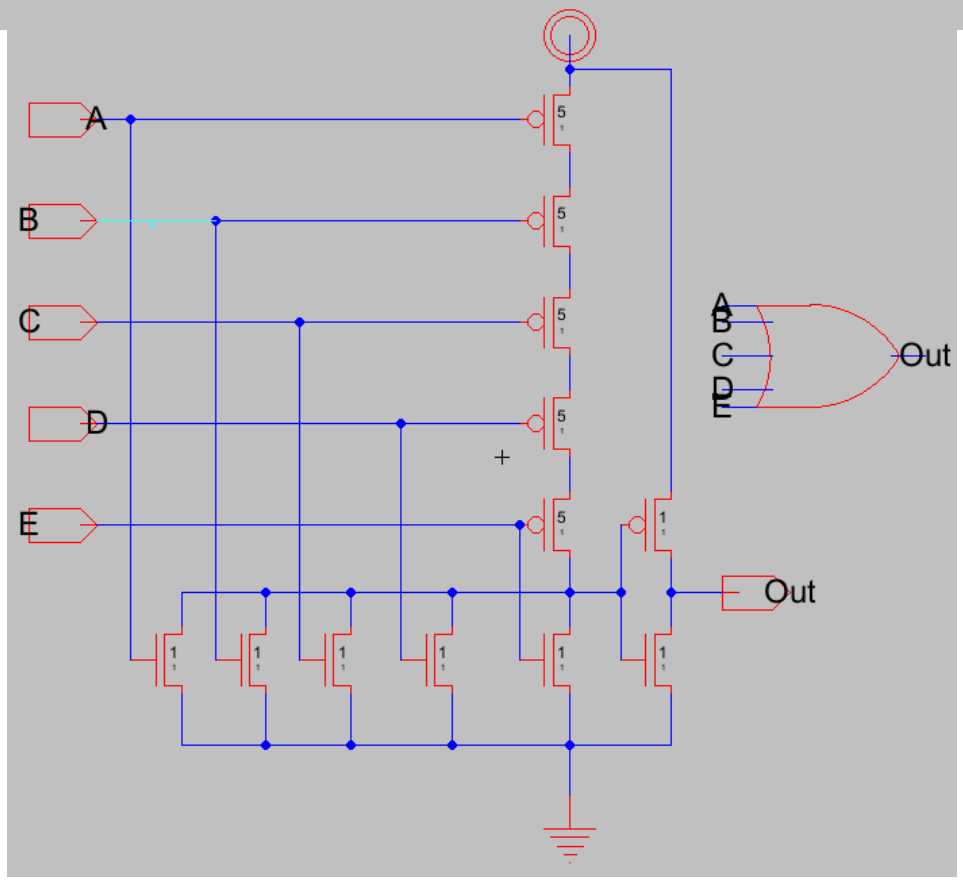

A
B
C
D
Out
Out
A
B
C
D
Out
A
B
C
D
E
Out
A
B
C
D
E

3.1.3- OR gates

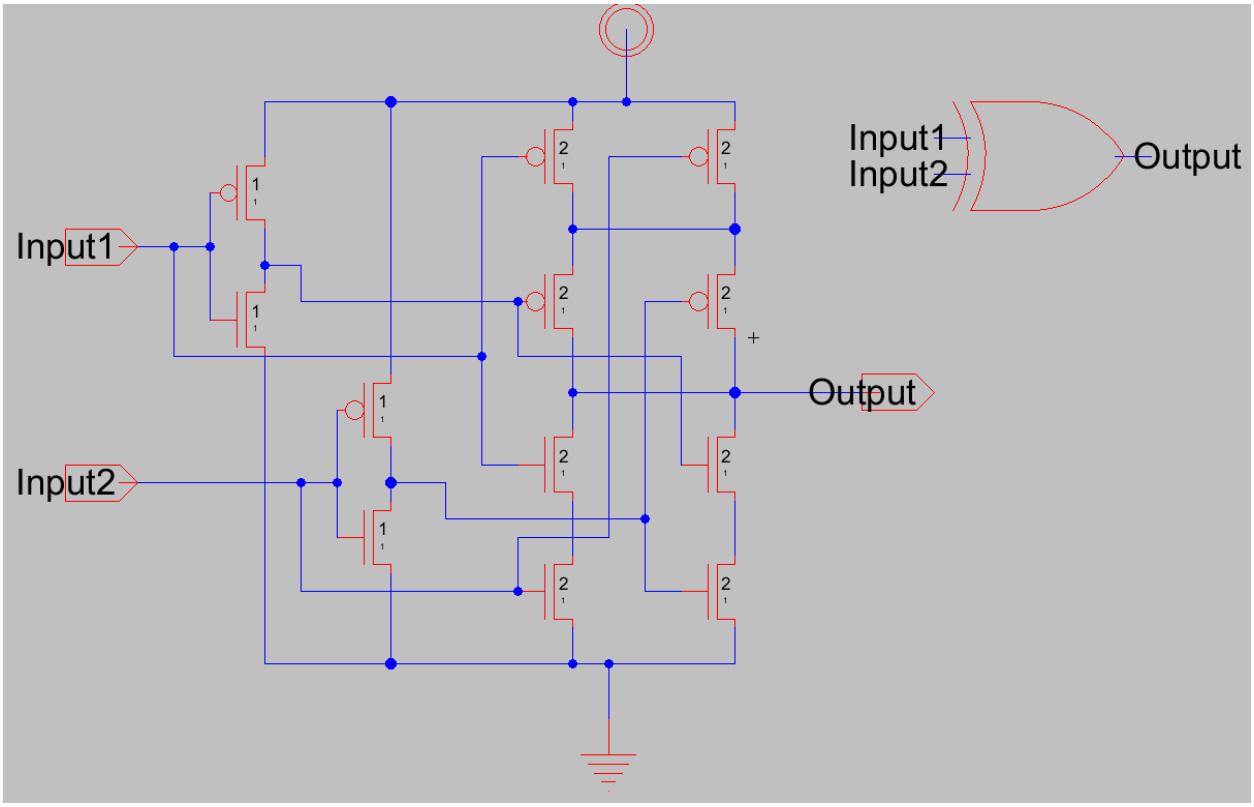





A
B
C
D
Out
A
B
C
D
Out
4
1


A
B
C
D
E
Out
A
B
C
D
E
Out
5
1

3.1.4- XOR gate:

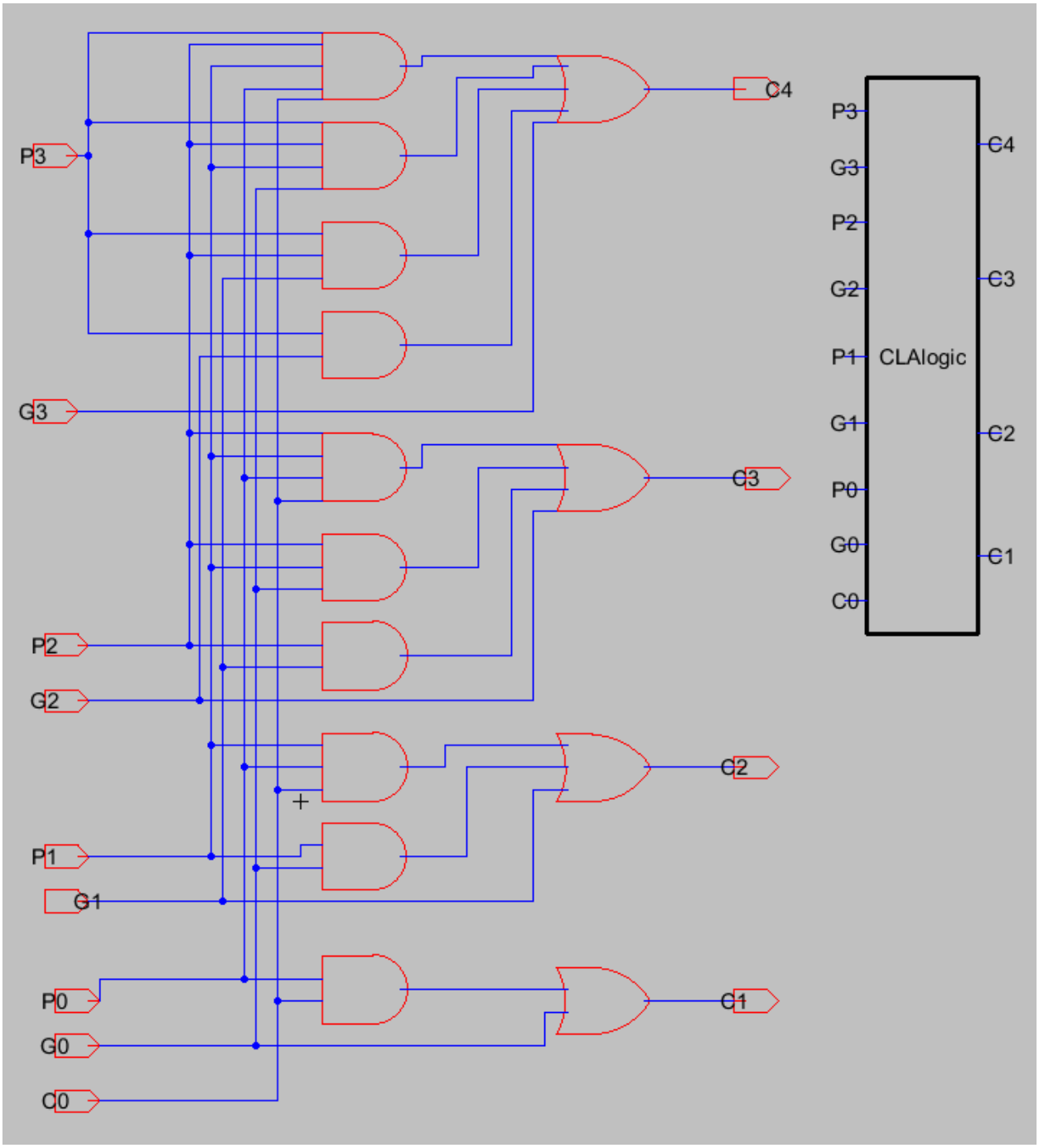


3.1.5- Carry look ahead logic

This logic is called CLA(carry look ahead). Now we need to implement the circuit block that takes P and G and Cin as inputs and give us all C1,C2 etc...

Honestly implementing it was exhausting, but the results are worth it.

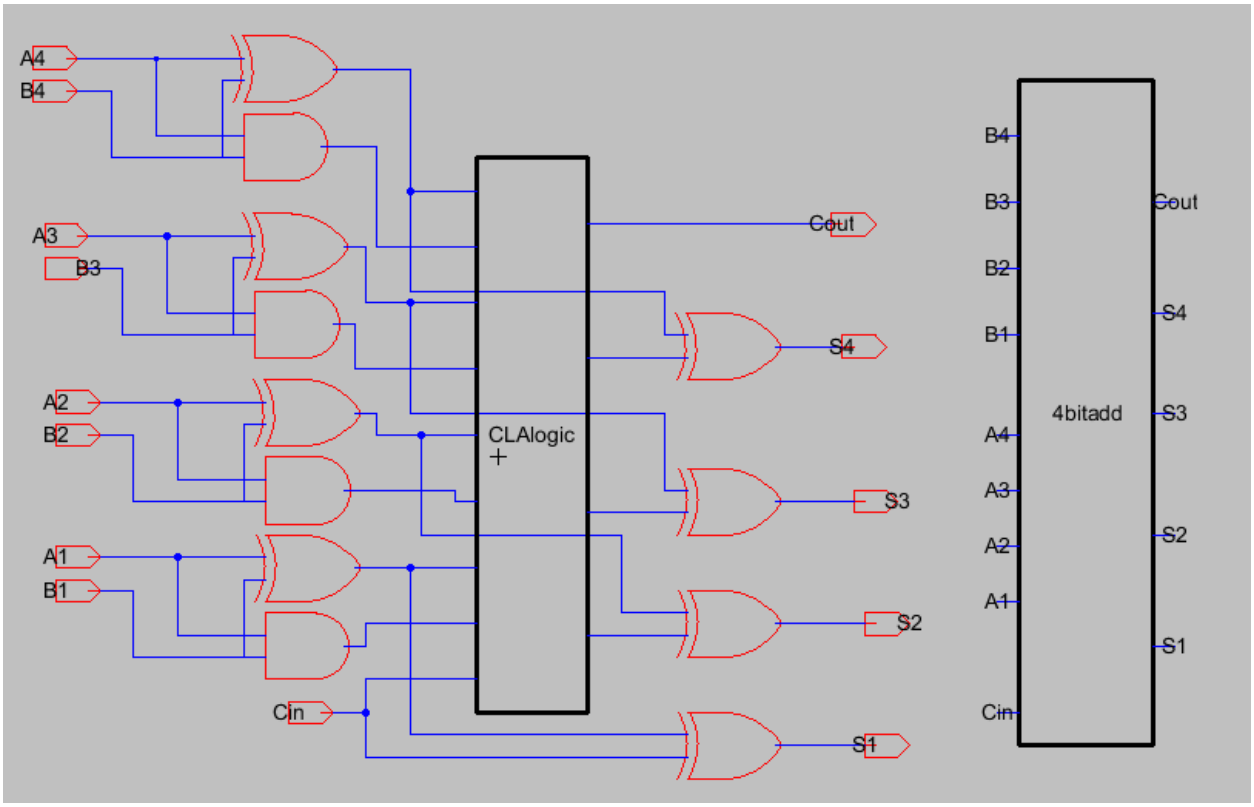
I named C1 -> S1 since I will XOR it with P1 and get S1 and I did the same with all the others except last carry because this will be fed into another circuit block or used as an output.



3.1.6- Note:

I tested all the gates one by one after making them and honestly it is useless to add it because the tests was trivial and their design is simple so I will test the 4 bit block and the 8bit block.

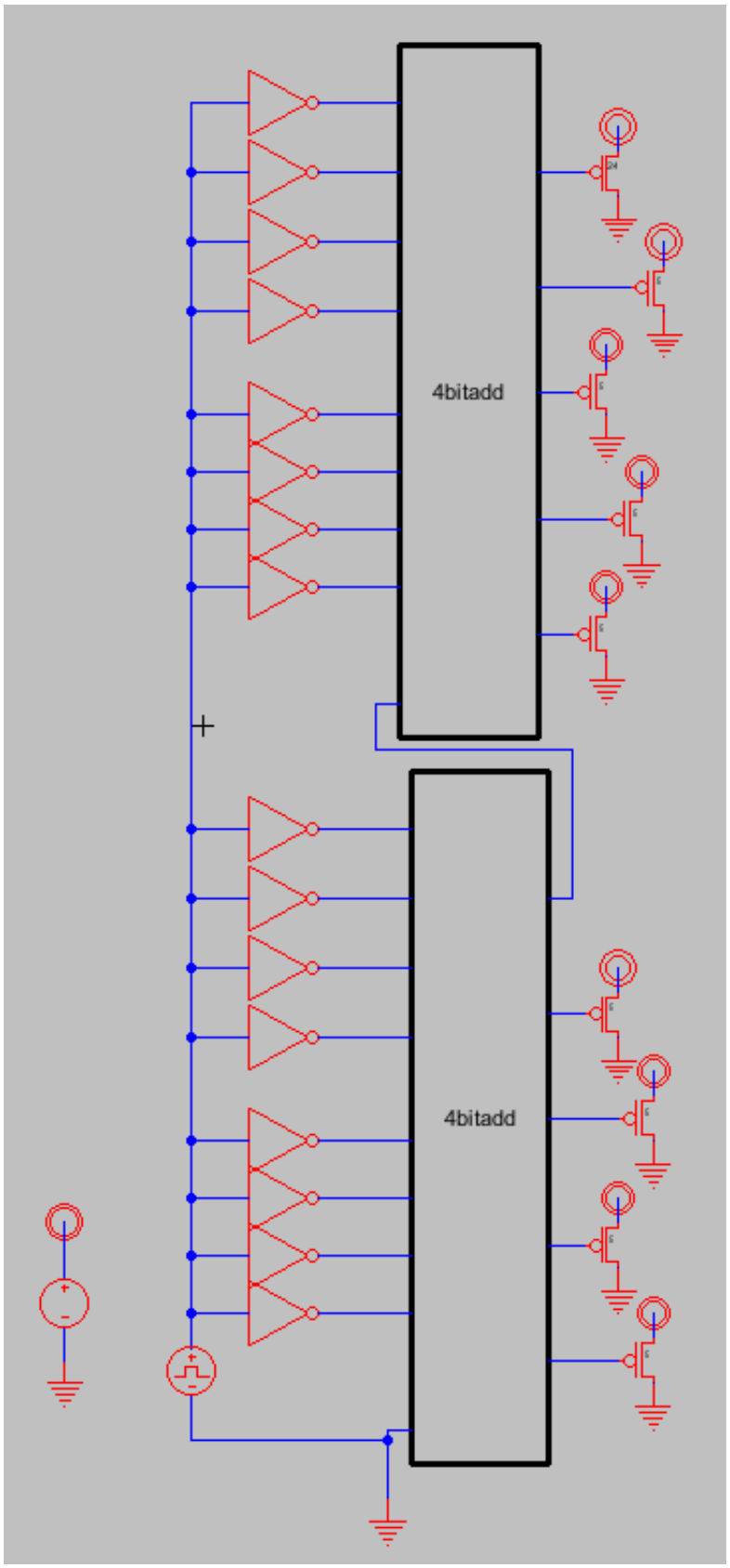
3.1.7- 4-bit adder

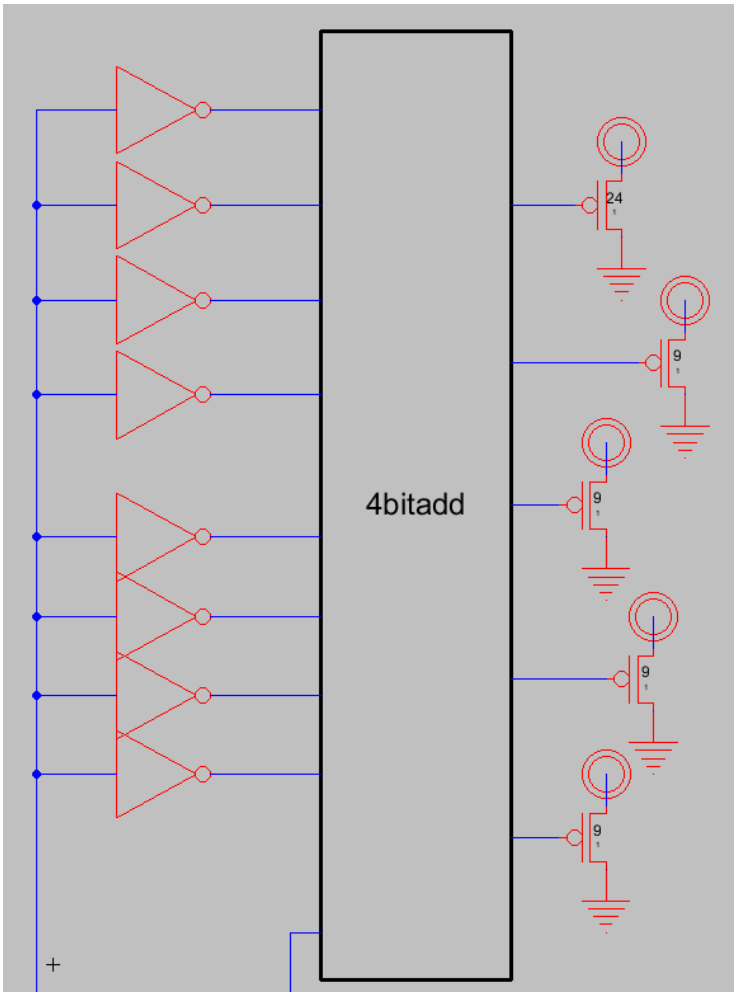


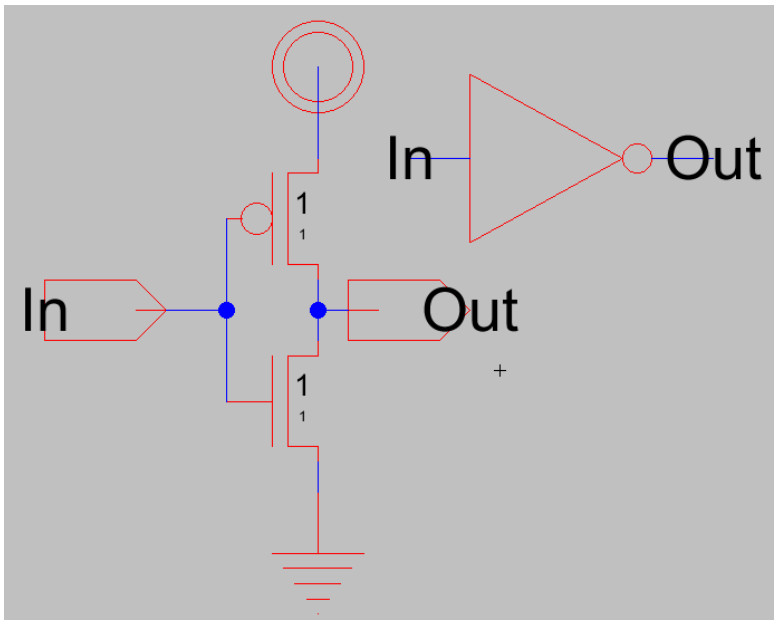
Now this is our design for the 4 bit block, which can be cascaded to create any other arbitrary addition. We could have made an 8 bit block directly but as I said we would have needed AND gates of sizes from 6 to 10 and same for OR and also the design will be very complicated.

3.1.8- 8 Bit adder

I cascaded two of those 4bit adders to create an 8 bit adder. Also I have drove the inputs with an equivalent resistance of Ro because the XOR which drives S1 or S2 etc... has resistance of Ro. Also I have loaded the outputs with 9Co since the outputs will be usually connected to XOR and AND if cascaded as trees (multiplication, etc...) The only output that was loaded differently was Cout since this will be usually fed into another 4 bit block which is (5 bit, 4 bit, 3 bit ,2 bit AND) and XOR which have equivalent capacitance of 24Co.

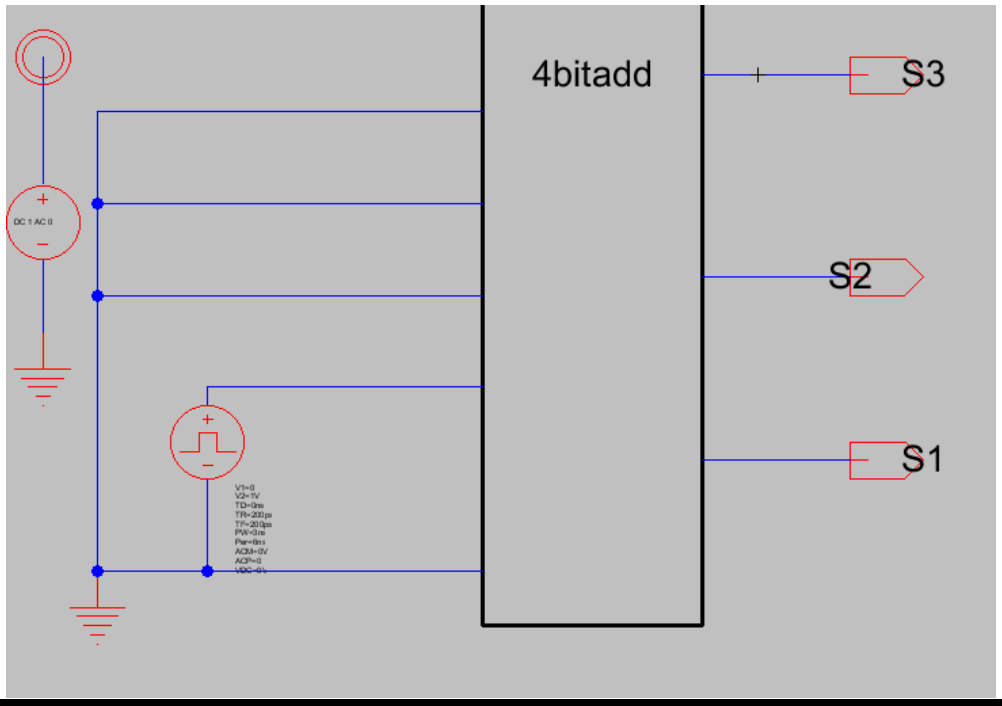

4bitadd
4bitadd

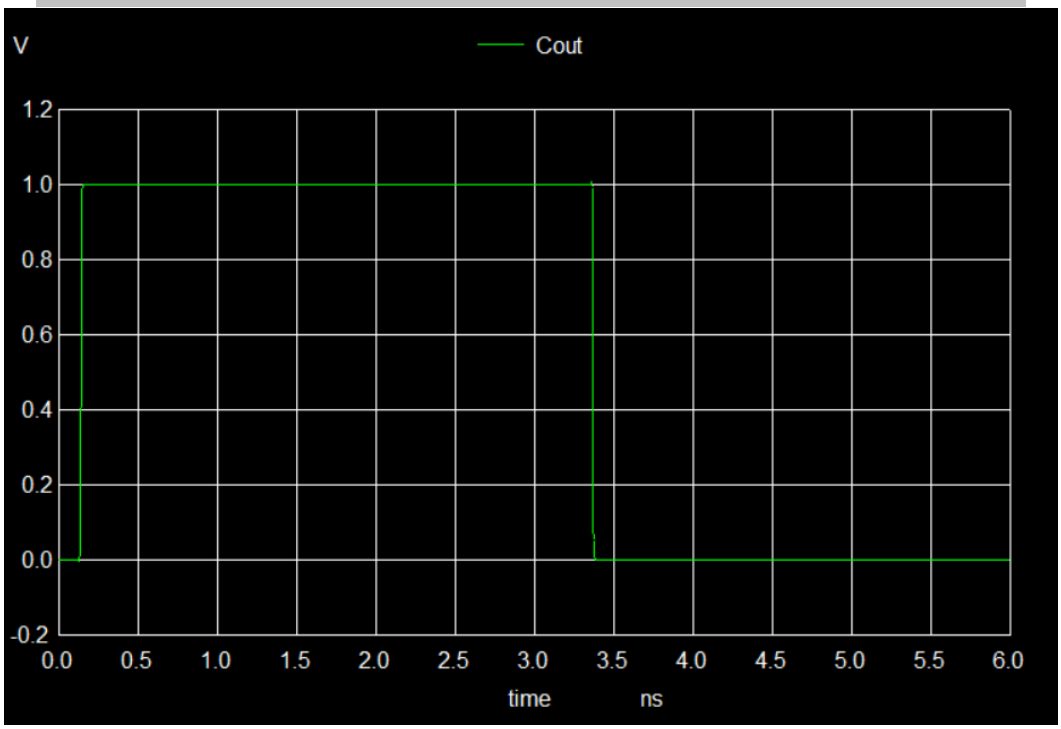

4bitadd

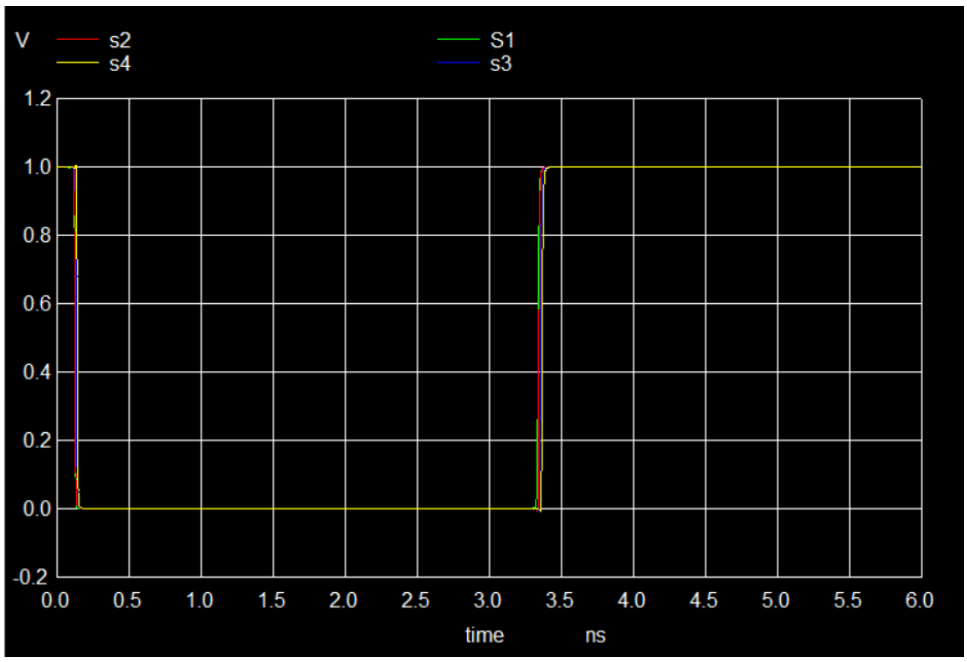

In
Out
In
Out

3.2- Testing:

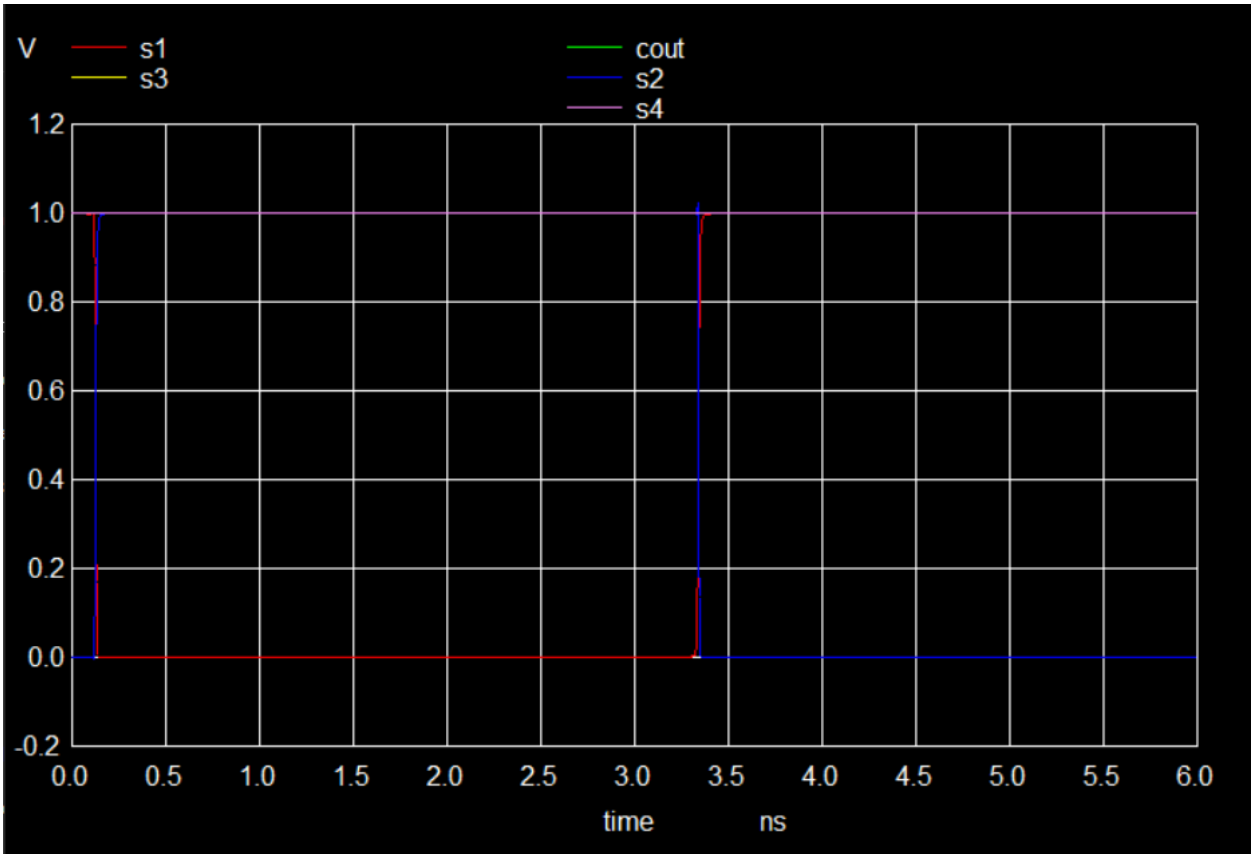
We need to test the functionality of the circuit so lets test the 4 bit block and if it is working the 8 bit cascade is trivial, so we don't need to test it.
I will input 1111 + 0000 and 1111 + 0001 using a V pulse. This will allow us to see the change in the 4 outputs and also the carry output if all of them switched right then the logic is write.







I will try another test which is 1111+1110 and 1111 + 1111 to be sure everything is working.



As we can see when Vpulse = 1 S1 =0 and when Vpulse =0 S2= 1. Working great!!

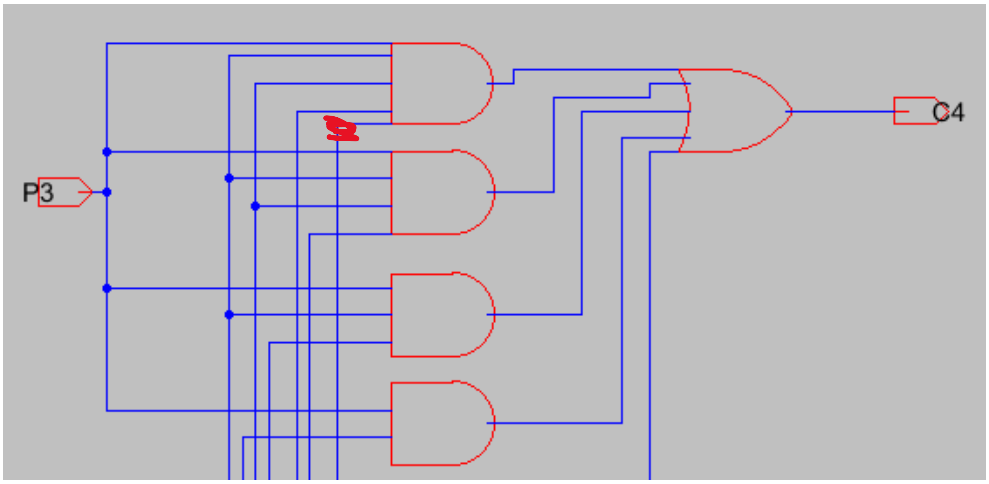
3.3- Delay:

I made the voltage 1Vdd, since higher Vdd means faster circuit.

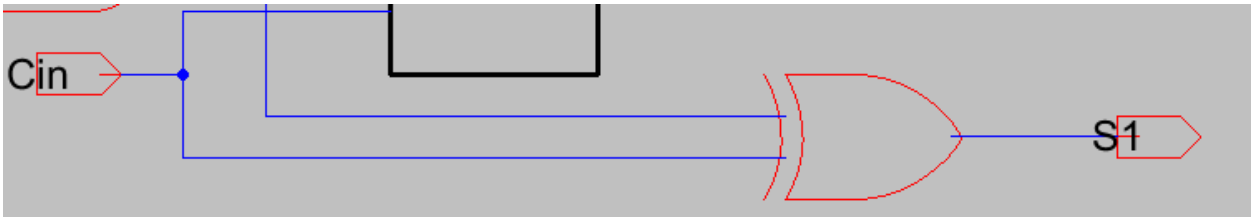
The worst delay is the Cin of the first adder to the output Cout of the last adder. And a good input that would drive a Cout to 1 is (11111111)+(0000000)+1(carry). This is the worst is since this will test the switch of Cout which has the longest path especially that only one AND gate in the CLA logic block will be driving the input to its OR which mean highest resistance. This is the worst delay we can have, but first I will calculate the tau expression for it then I will do the simulation.

3.3.1- Theoretical Delay:

Since Co is connected to the highest load this will be our starting point. The marked wire is Cin. Then fed into OR then go through the 2nd 4bit into the same path again. So we will calculate the path for a 4 bit adder then multiply it by two.



As we can see we feed an XOR first then the CLA block.



Delay = Ro(6Co(XOR) + 3Co(2bit AND) + 4Co(3bit AND) +5Co(4bit AND) +6Co(5bit AND)) + Ro(5bit AND)*(6Co(5bit OR)) + Ro(OR)* (6Co(XOR) + 3Co(2bit AND) + 4Co(3bit AND) +5Co(4bit AND) +6Co(5bit AND)) = 24RoCo + 6RoCo + 24RoCo = 54RoCo

Multiply this by two:

Total Delay = 108RoCo

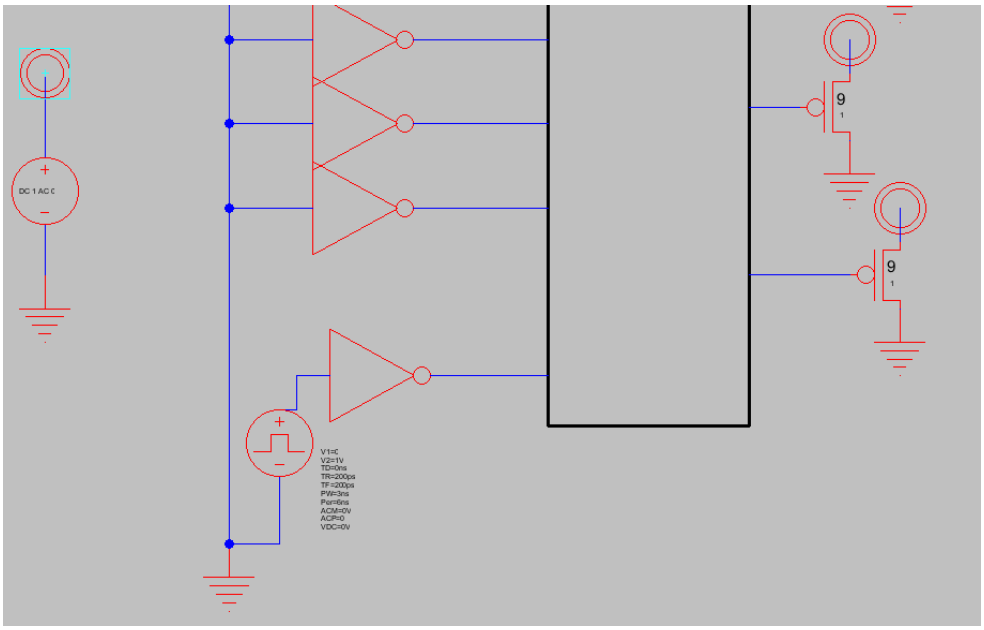
3.3.2- Simulation:

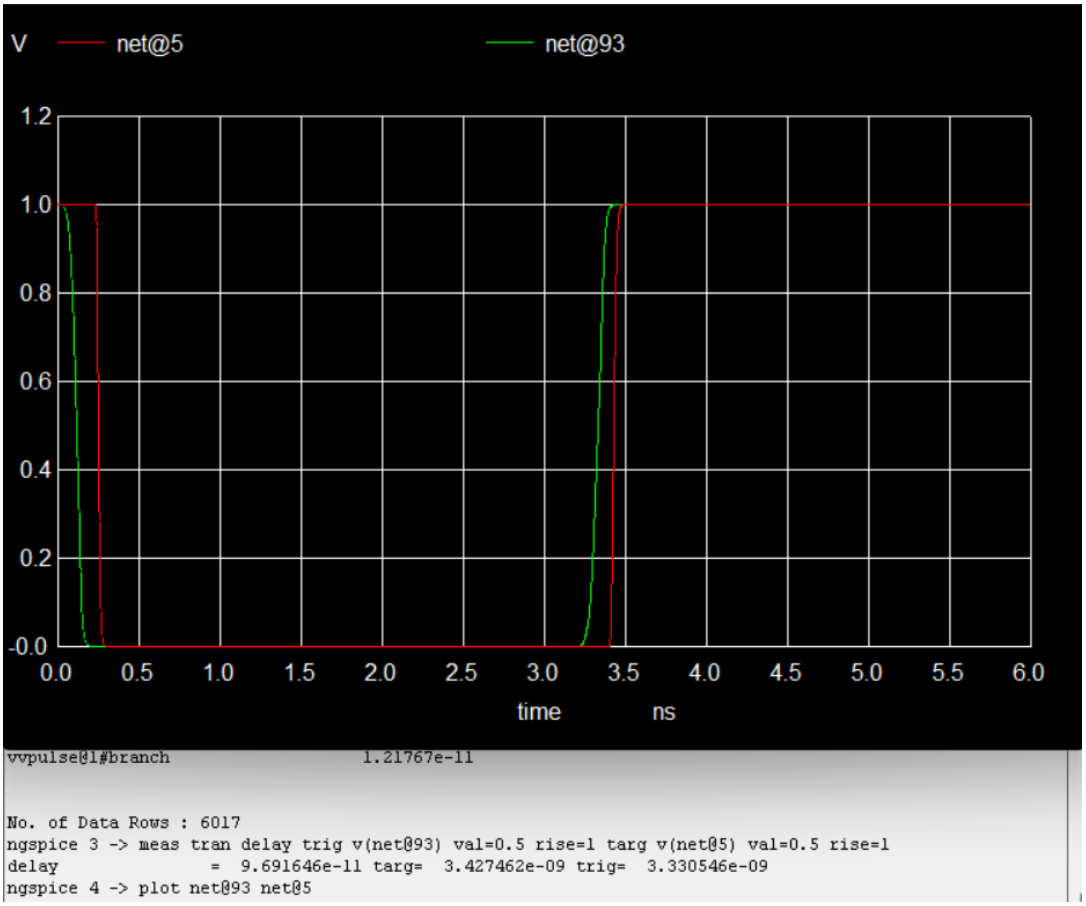
I have already shown the delay circuit in the design part 3.1.8, so no need to add it again. I will just feed in the inputs I have specified in 3.3.1.

I will use the following Spice code:

8bitadd.spi
tran 1ps 6ns
meas tran delay trig v(net@93) val=0.5 rise=1 targ v(net@5) val=0.5 rise=1
plot net@93 net@5

1st test: (11111111)+(0000000)+1(carry)





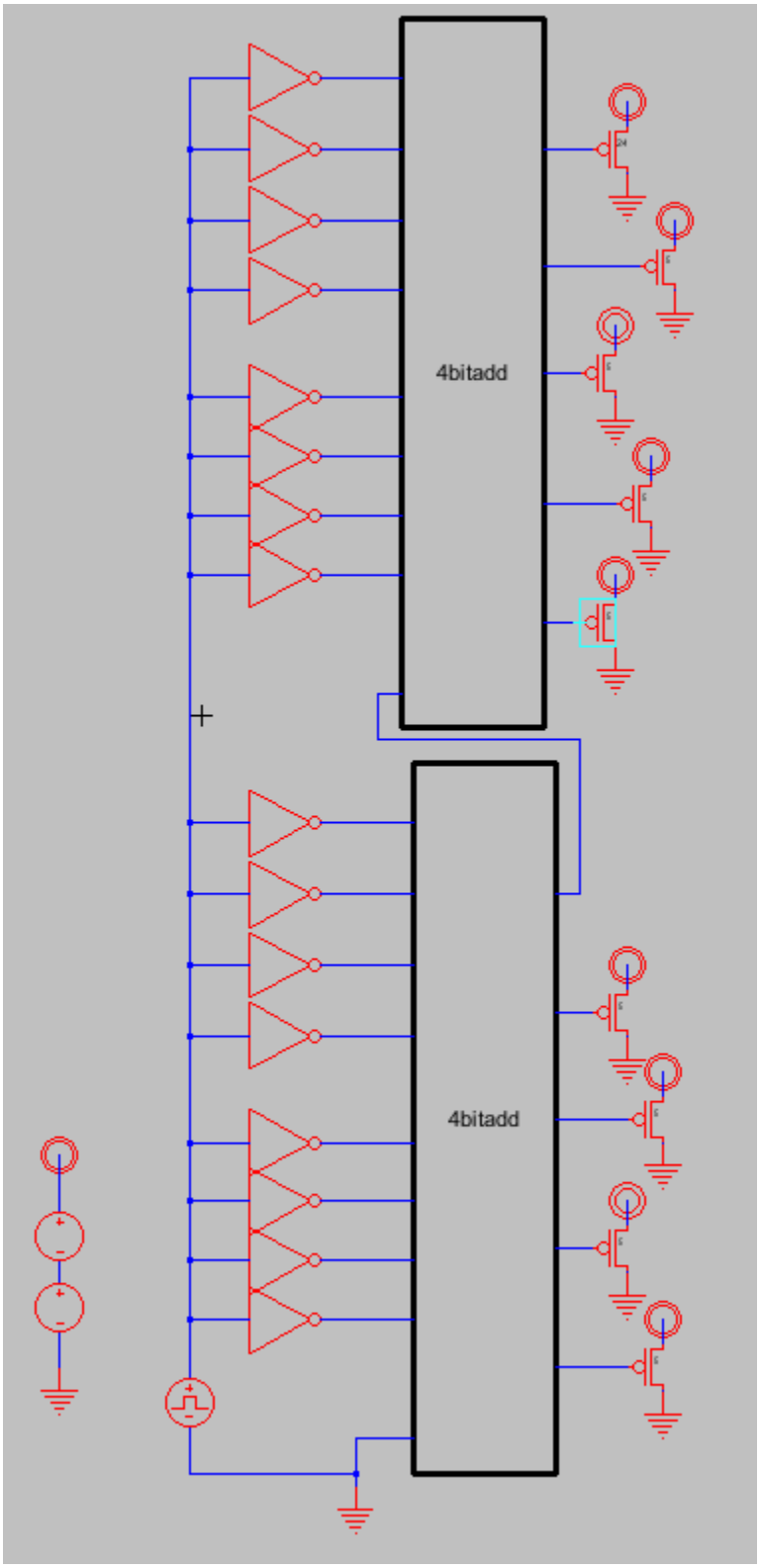
The Delay is 97ps.

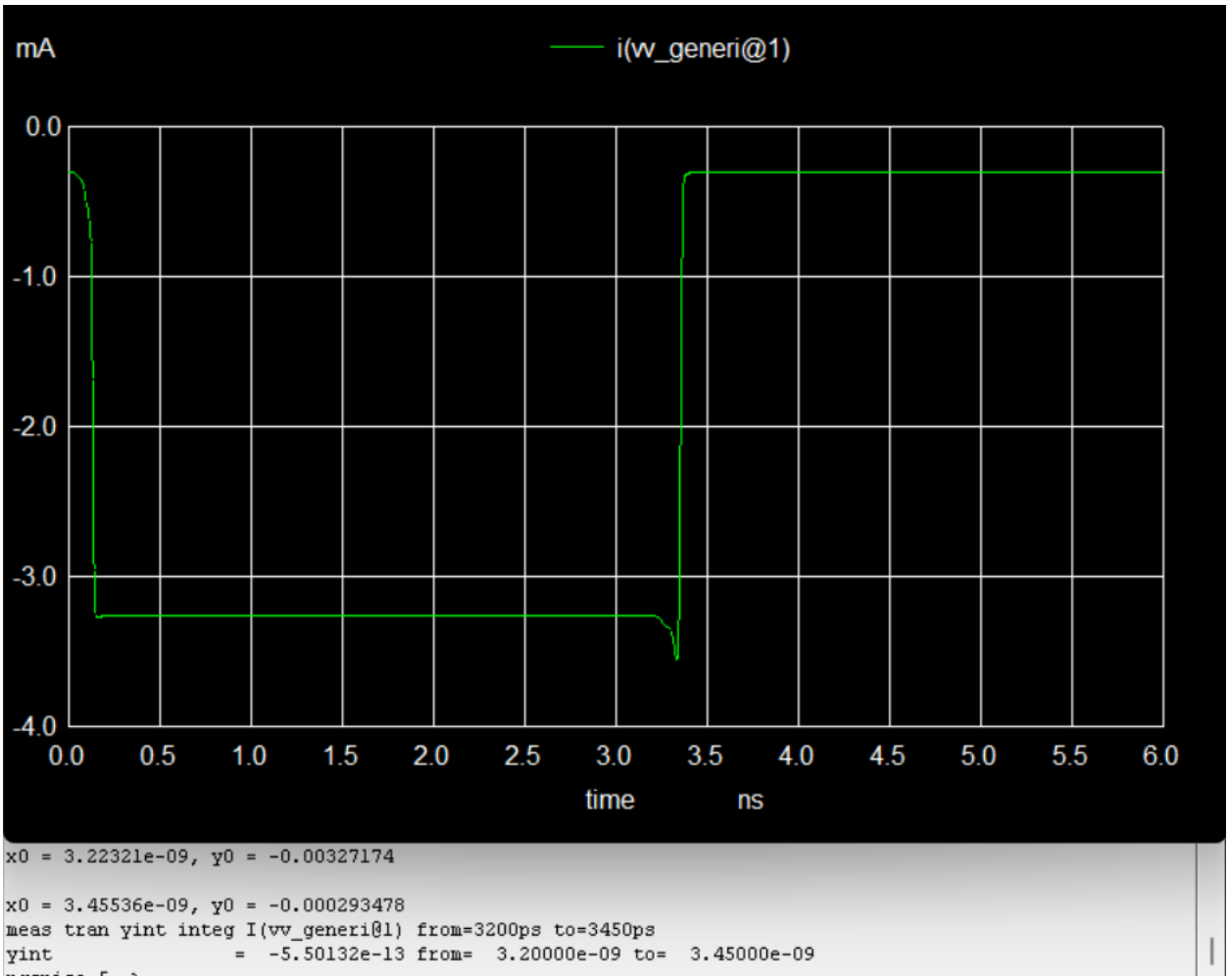
3.4- Active Energy:

We will calculate the active energy for the maximum energy which is the switching case from 0 to 1 for all of them.

The Code I used to measure this circuit:
8bitadd.spi
tran 1ps 6ns
plot I(vv_generi@1)
then find the switching time and plug it into the next line:
meas tran yint integ I(vv_generi@1) from=t1 to=t2

3.4.1- Maximum Energy:





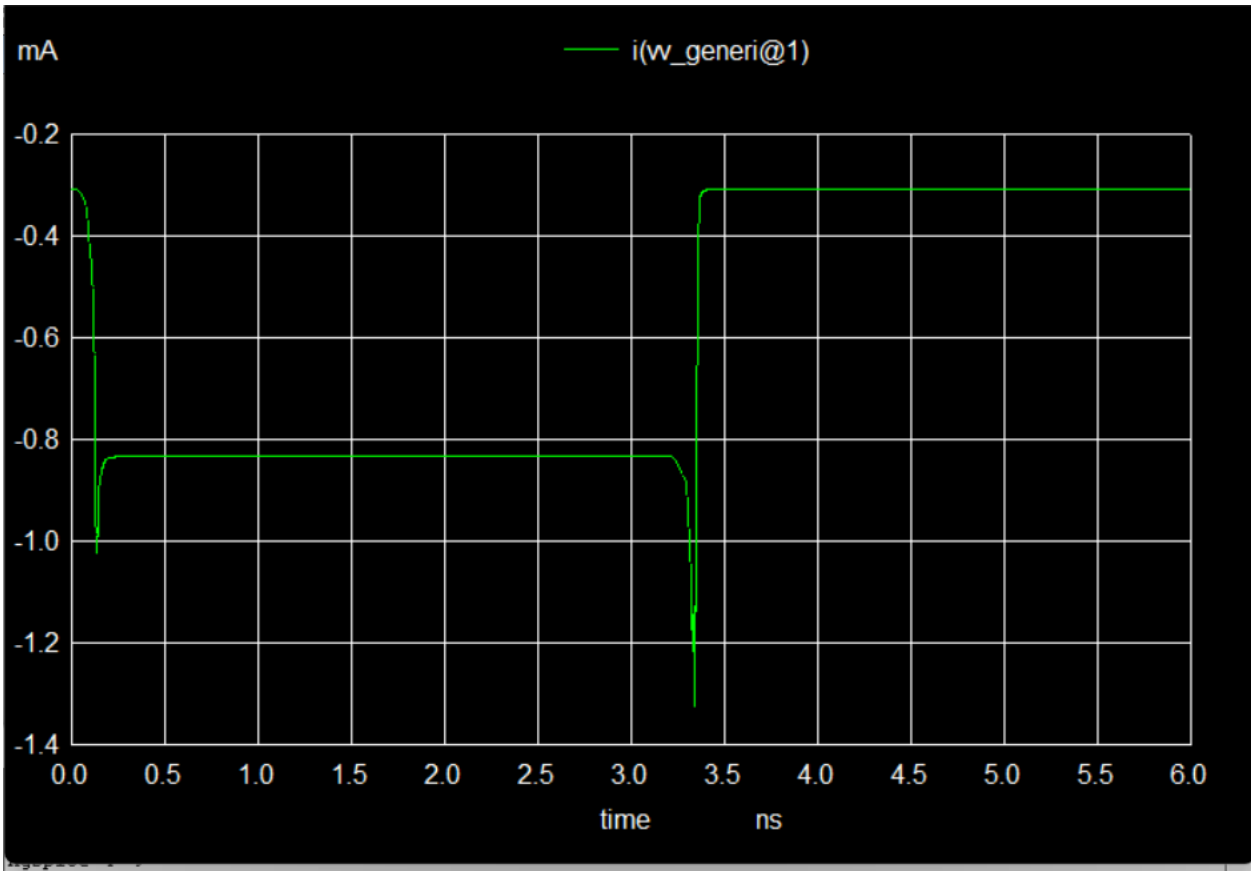

```
x0 = 3.22321e-09, y0 = -0.00327174

x0 = 3.45536e-09, y0 = -0.000293478
meas tran yint integ I(vv_generi@1) from=3200ps to=3450ps
yint               =  -5.50132e-13 from=  3.20000e-09 to=  3.45000e-09
```

I took integration from 3200 to 3450 because this is the part where switching happened.
Energy = 5.5*10^(-13)J

3.4.2- Average Energy:

half the inputs from 0 to 1:



I took integration from 3200 to 3450 because this is the part where switching happened.

Energy = 1.7*10^(-13)J

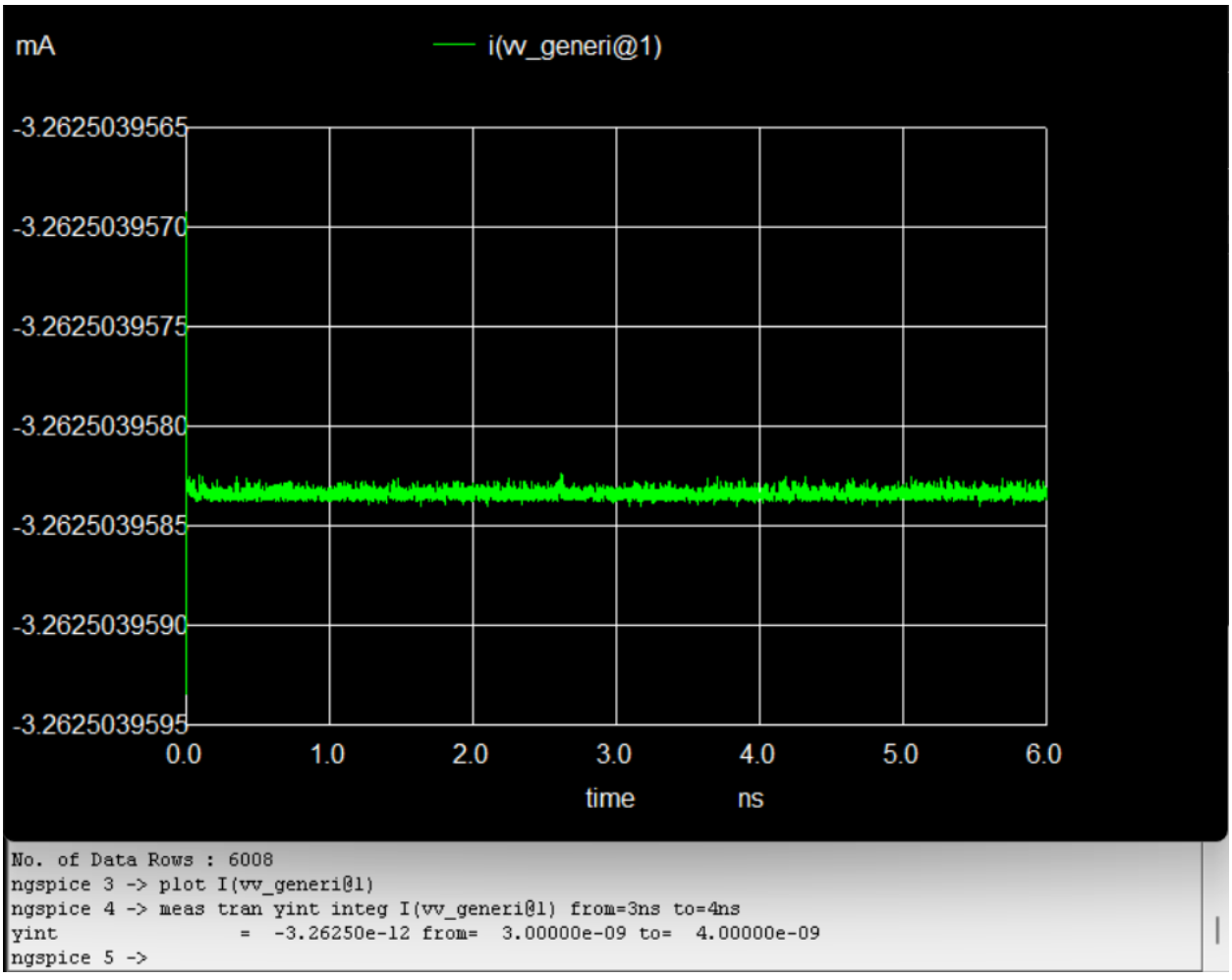
The energy = ampere since Vdd = 1 and Energy is IV.

3.5- Leakage Energy:

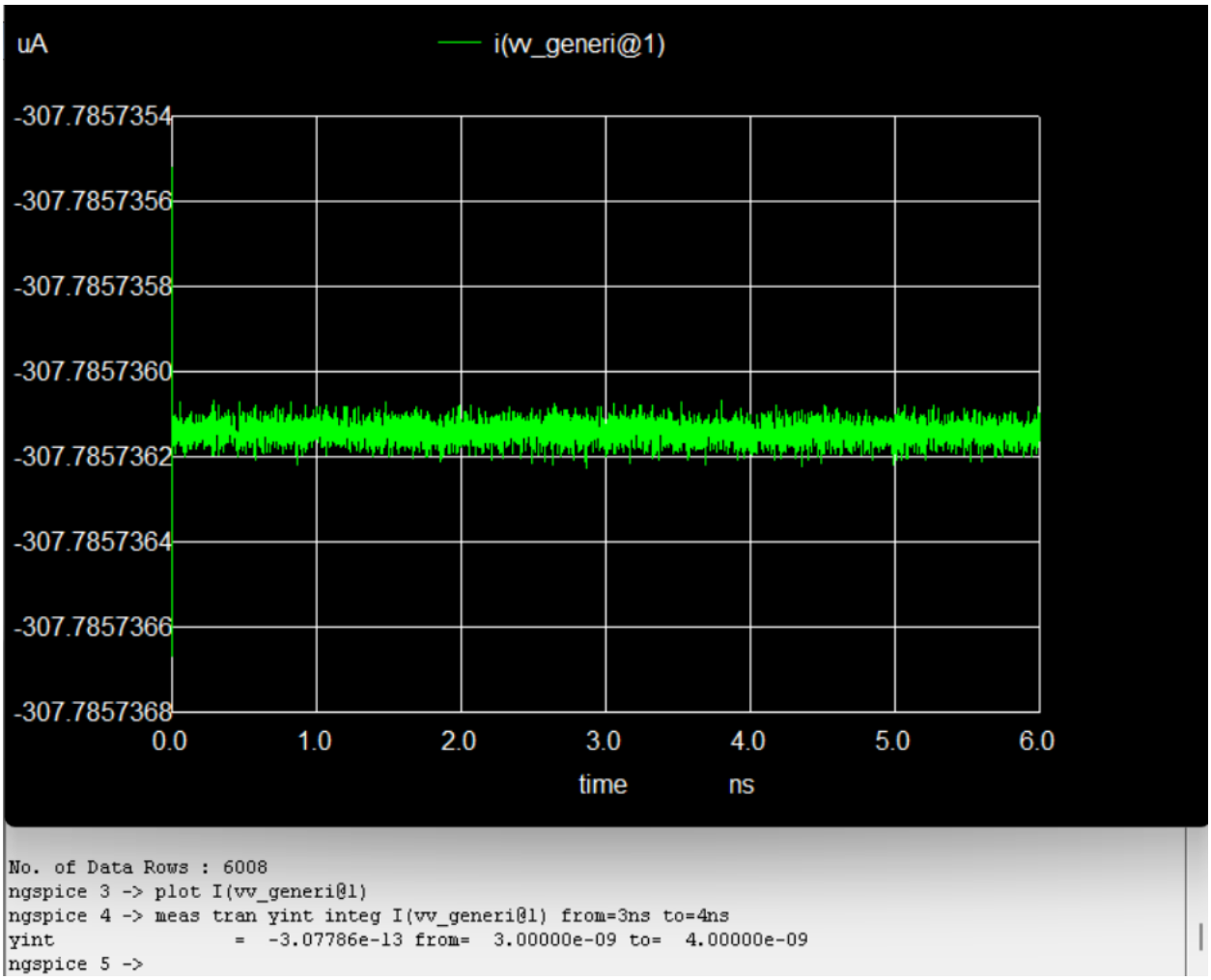
The highest leakage energy is at inputs all zeros. This is because that all the outputs will be 0s and this will result in all the PMOSs leaking current. When everything is outputting zeros this means that NMOS is on and PMOS always leak more current when closed. The lowest leakage would be the inverse of this (all ones).

The Code I used to measure this circuit:

8bitadd.spi
tran 1ps 6ns
plot I(vv_generi@1)
meas tran yint integ I(vv_generi@1) from=3ns to=4ns
I will integrate over 1ns to find the leakage energy per nanosecond.



Maximum leakage energy in nanosecond = 3.3 * 10^(-12)J



Minimum leakage energy in nanosecond = 3 * 10^(-13)J

3.6- Area:

We will calculate the area of one 4 bit block and multiply it by two.
Lets count the gates first:

2 bit AND: 9gate Area: 9*8 = 72
3 bit AND: 3 Area: 3*14 = 42
4 bit AND: 2 Area: 2*22 = 44
5 bit AND: 1 Area: 1*32 = 32
2 bit OR: 1 Area: 8
3 bit OR: 1 Area: 14
4 bit OR: 1 Area: 22
5 bit OR: 1 Area: 32
XOR: 8 Area: 8*20= 160

Size of 4 bit adder: 160+ 32+22+14+8+32+44+42+72 = 426
Size of 8 bit adder: 426*2 = 852
Full size of 8bit adder = 48*8 = 384 (we can multiply this by 22nm to get the size and multiply it with the length 22nm and get the full size in nanometers but this wasn't said in project doc)

Spec	Value
Maximum Delay	97ps
Maximum active energy (during switching)	0.55 pJ
average active energy (during switching)	0.17pJ
Maximum leakage energy in a nanosecond	3.3 pJ
Minimum leakage energy in a nanosecond	0.3pJ
Area	852

4- issues and design-options:

4.1- Issues:

1- This design uses a lot of area and doesn't give a good trade off when it comes to speed. Certainly, this design is faster than ripple carry, but it consumes more energy, take more area, and have higher leakage current. This design would be best when we scale for more than 8 bits.

2- The need for 5 bit gates etc... Those custom gates aren't optimal for VLSI design sense we will have to design with some fundamental gates which will result in a more complicated circuit

4.2- Design options:

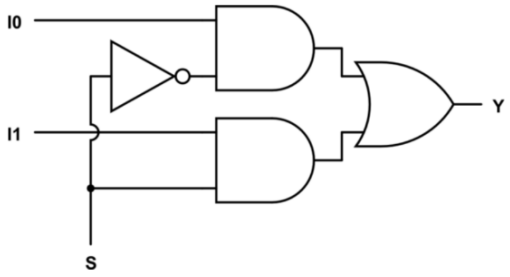
1) One major alternative would have been to implement all of the 8 bit adder as a huge CLA block. It would have been so fast, but the problem is I will have to create 6,7,8,9,10 input AND gates and also wiring all of this would have been a pain and honestly I don't think it will provide more than a couple of picosecond of speed.
2) Carry Select Adder (CSEL)
   - Concept: Precomputes the sum with carry-in = 0 and carry-in = 1 for each block, then selects the correct result once the carry is known.
   - Benefit: faster than RCA, as it reduces waiting time.
   - Tradeoff: Requires extra area for duplicated adder units and also is bad for small adders if the adder is more than 8-bit it will give a better delay else it won't and it consumes more energy too
   - I have tried it and it gave me a bad delay compared to CLA
3) Carry Skip Adder (CSA)

   - Concept: Groups bits into blocks and allows the carry to skip over blocks when possible.
   - Benefit: Reduces worst-case delay compared to RCA.
   - Tradeoff: Requires additional multiplexers and gates and also might give worse delay for low number of bits 8 and lower so the ripple carry was the best

4) For decreasing energy we could have decreased Vdd but the project focus was on delay.

5- alternates and variations to arrive at the final design:

1) I raised the Vdd from 0.8 to 1V as this will increase the speed of charging capacitors making the delay less
2) I choose didn't change the path of A and B inputs at all as they don't contribute to the delay this much because the longest path was the carry path.
3) After I knew that I should change the carry path I thought about changing the dependence on the carry and I choose the CLA design
4) I then choose different width for transistors as needed in order to decrease the tau delay.
5) I tried adding a mux to create a CSEL but I found out that the carry will have to go through a not the and then or gate which was useless if I am implementing an 8-bit adder.



6- area and delay relation to size:

6.1- Area:

For the area my design is used for 4 bits, but actually it can be extended to arbitrary number of bits by implementing the recursion equation or by creating a smaller blocks of 2 bit or 3 bits. So scalling this will be different for 1 bit it will be a normal full adder for two bits I will just count in the first two outputs and what is needed to drive them in the CLA logic. For 8 and 4 it will be the same as calculated before.

Bits	Area
1	44
2	80
4	426
8	852

6.2- Delay:

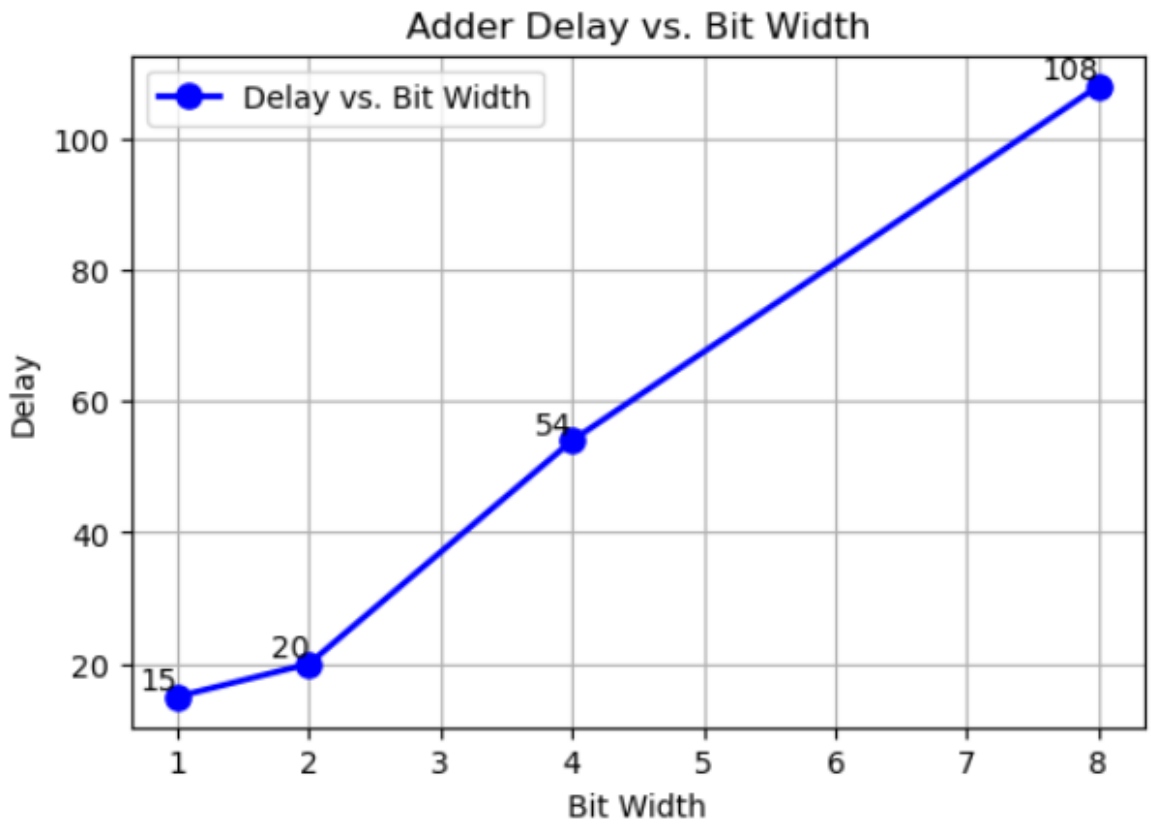
The delay of my circuit will continue increasing linearly also it can be reduced if we increased the size of the CLA logic block. It is linear if we are scalling it with multiples of

4bit but if we are scalling from 1 to 8 it is different. I will calculate the worst delay for each case outside then plot a graph of it.

For 1 bit: 15RoCo

For 2 bit: 20RoCo

And we have already calculated the rest before.(4 bit = 54 – 8bit = 108RoCo)



I guess in ML and AI they use low precision for energy efficiency and also for speed since smaller ones are faster they can trade accuracy to speed and perform addition on the most significant bits only.

7- Certification:

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