ILLINOIS INSTITUTE OF TECHNOLOGY

Advanced VLSI Design

Full-Adder Configuration -D3L and SPD3L Implementation.

Adalberto Claudio Quiros
A20294552

Index

Abstraction

The aim of this project is to study and implement a 4-bit Full Adder using two different techniques D3L and SP-D3L, also performing the Full Adder with two different logic functions in order to compare the results in terms of delay and power consumption.

For this purpose Cadence Virtuoso will be used to design the schematic of the Full Adder and the tests of them, creating the netlist than will be analyzed with Hspice and Nanosim.

Introduction

The steps that have been taken in this project are first to study the implementation and it's results, build an schematic on Cadense Cirtuoso, creating a test case and a netlist of it.

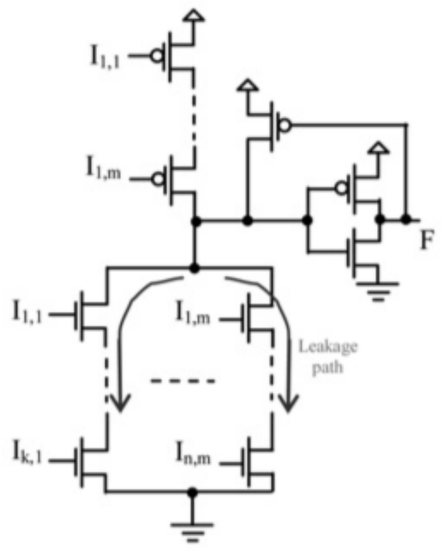
Then use Hspice and Nanosim to measure it's power consumption and parameters, and to finish measure the delay with Cscope.

Background

The D3L technique is based in the domino logic of dynamic gates, but in this case the aim of this one is to implement the gate without using the clock signal. This makes a difference in terms of the clock distribution on a circuit, because with this implementation won't be needed.

There is on requisite for this to work and this is that the inputs must be 0 in the first term; this is because there must be a pre-charge phase before using the gate. To perform this all inputs will be qualified with the clock signal, simply using an and gate with the input and the clock signal, this way when there a pre-charge phase we make sure that the D3L gate inputs are 0, making the pull-up network work opening a path between the voltage source and the output, and the contrary case evaluation the inputs in the gate as a regular one.

To perform this technique, the logic function F will be placed as the pull-down network since we are going to invert the signal later. Then for each of the branches of the pull-down network, one signal of each branch must be an input of the pull-up network; this way we make sure that in the evaluation phase the path between source voltage and the output is broken. Therefore if the F = 0, there won't be a path to ground and if F = 1 there will be.

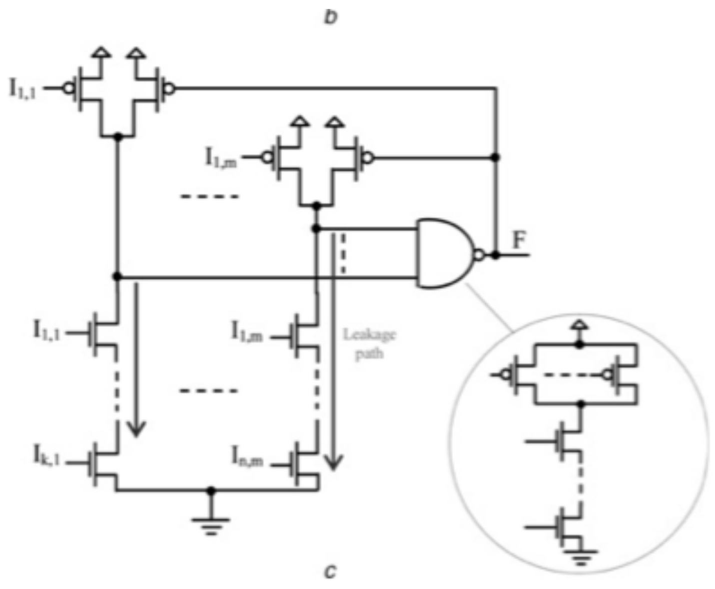


For sizing the gate, the requirements of the keeper will the set the base. Since with the minimum width possible is 90 nm and the keeper size must be $W_n/2$, therefore the equivalent width for the pull-down network must be 180 nm and for the pull-up network 360 nm.

Using this technique have different issues, unlike domino logic where the inputs signals are evaluated with the clock signals, with D3L the propagation takes through the pull-down networks of the D3L gates, and this have being taking into account is the frequency of the clock signal. Another issue is the power consumption the leakage currents through the pull-down and pull-up network may discharge the equivalent capacitances in the gate, to prevent this a keeper is used, but the use of this keeper increment the power consumption, also the dynamic logic of charging and discharging the capacitances make an impact on the power consumption.

This keeper also make an impact on the delay, slowing the gate and the speed degradation is proportional on the width of this, due to this the width chosen of the keeper will be the minimum possible, 90 nm.

To improve the speed of the previous implementation, we will perform the SPD3L technique. This one consists on splitting the branches of the pull-down network, the branch of the complement of F and the use a Nand gate with the output of each branch, this way we perform an Or. Doing this we evaluate each branch separately, which makes the gate faster because the equivalent capacitances are spited and it take less time to charge or discharge them.



This split between the branches make the difference in this implementation, not only the speed is incremented, therefore the delay is reduced, that also the power consumption is reduced. Although the static power consumption is increment the overall power consumption of the circuit still be lower than in the case of the D3L.

In terms of sizing the keeper as we did on the D3L technique we will make the transistor as possible but this time the equivalent pull-down network width must be $W_n/4$, then it will be 360 nm and the pull-up 720 nm.

Architectural Exploration of Adders

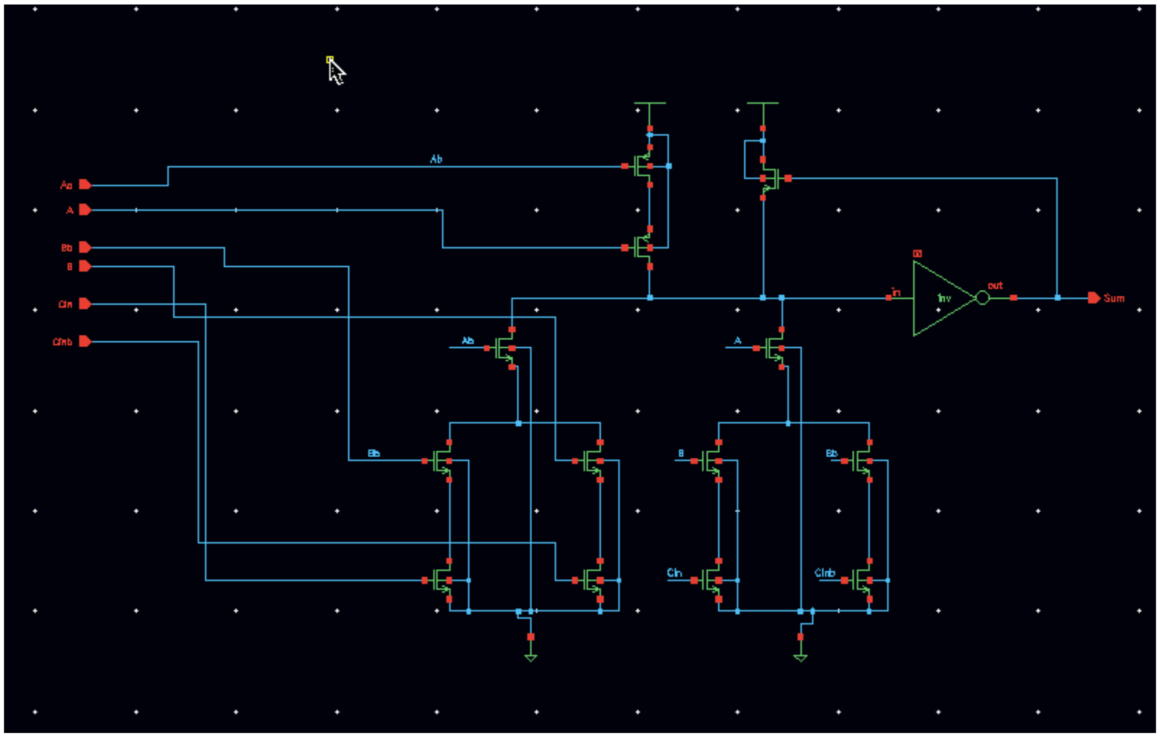
There are two possible method of implemented the full adder functions, Sum and Cout in this project. The two of then will be discuss and explain in terms of sizing in this part of the project.

Method 1

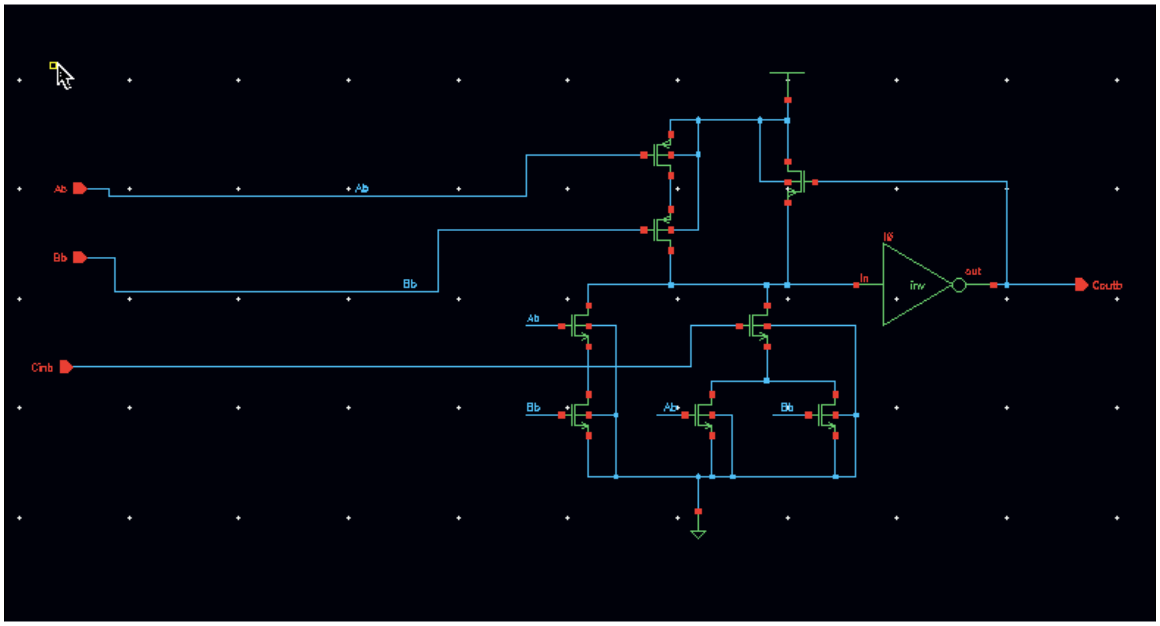
$$Sum = A \oplus B \oplus Cin$$
$$Cout = AB + Cin(A + B)$$

D3L Implementation

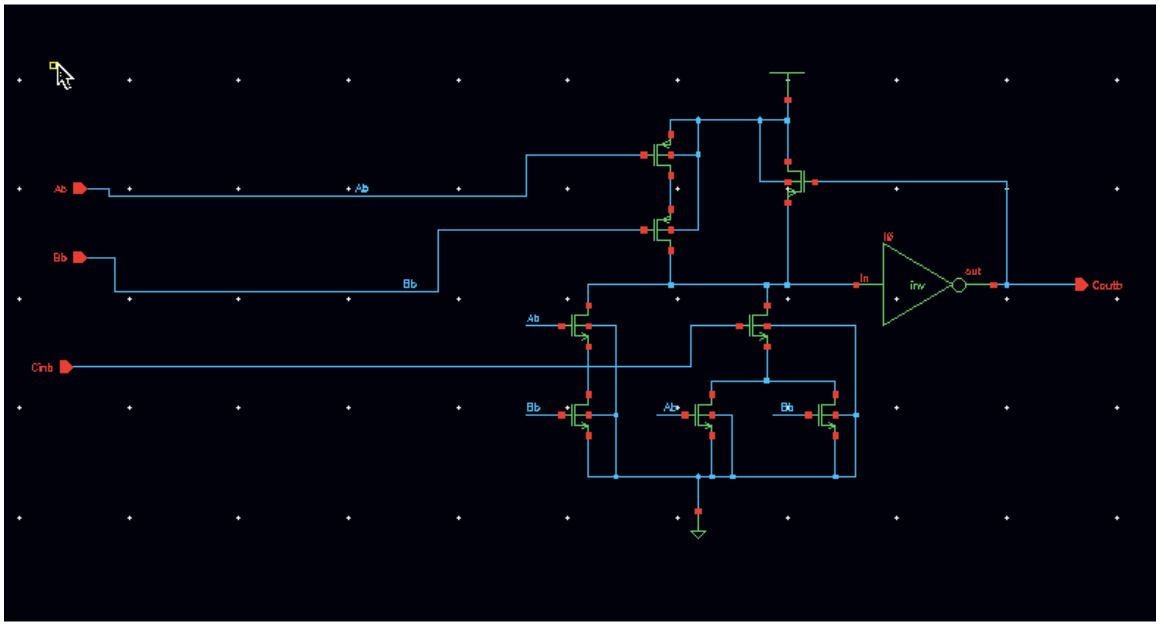
For the Sum implementation, the sizes of the transistors are the following. The pull-down network transistors have a width of 540 nm (3x180nm) to make the equivalent width 180nm, and the pull-up network 720 nm (2x360nm) for the same reason. The keeper width is 90nm as it was said in the previous section.



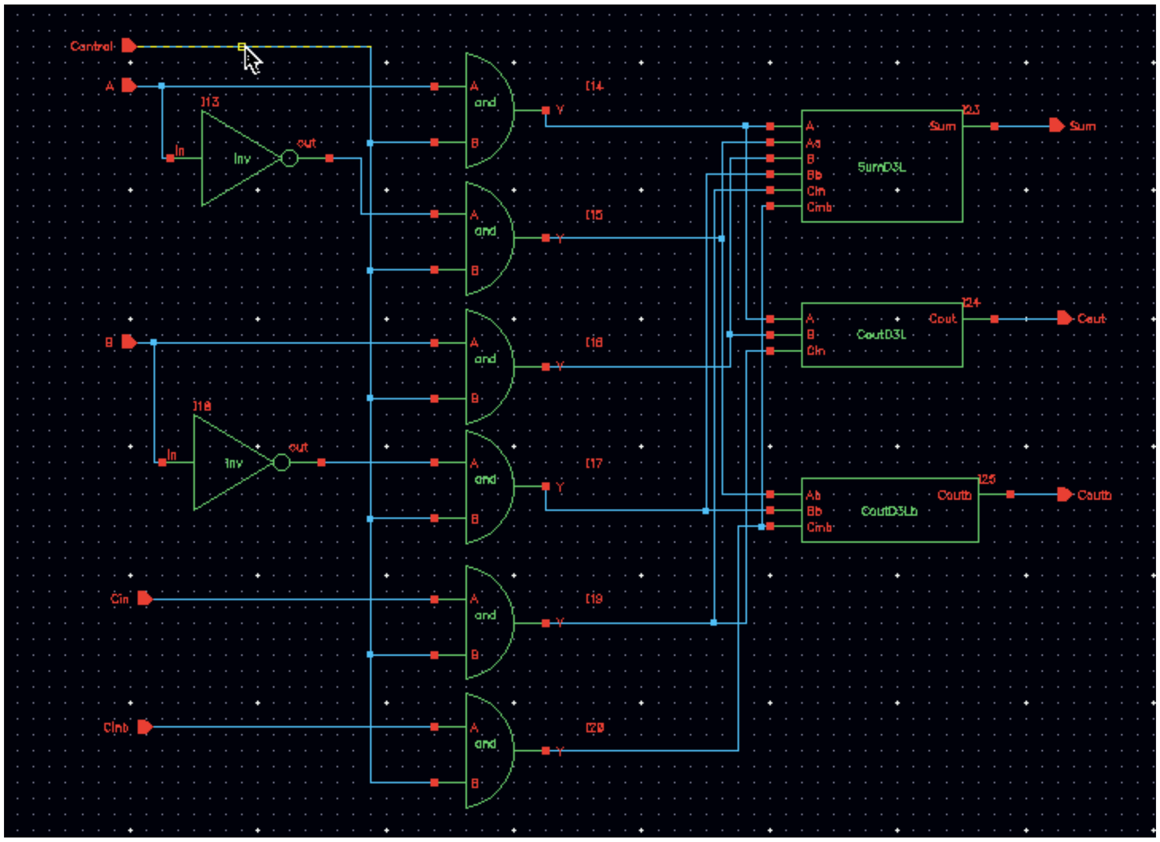
For the Cout, the nMOS transistors width will be 360nm and the pMOS 720nm.



And to finish the $\overline{Cout}$, the need of implementing this function and not inverting the Cout is because all of our input must be monotonically increasing. The sizing is the same as Cout.

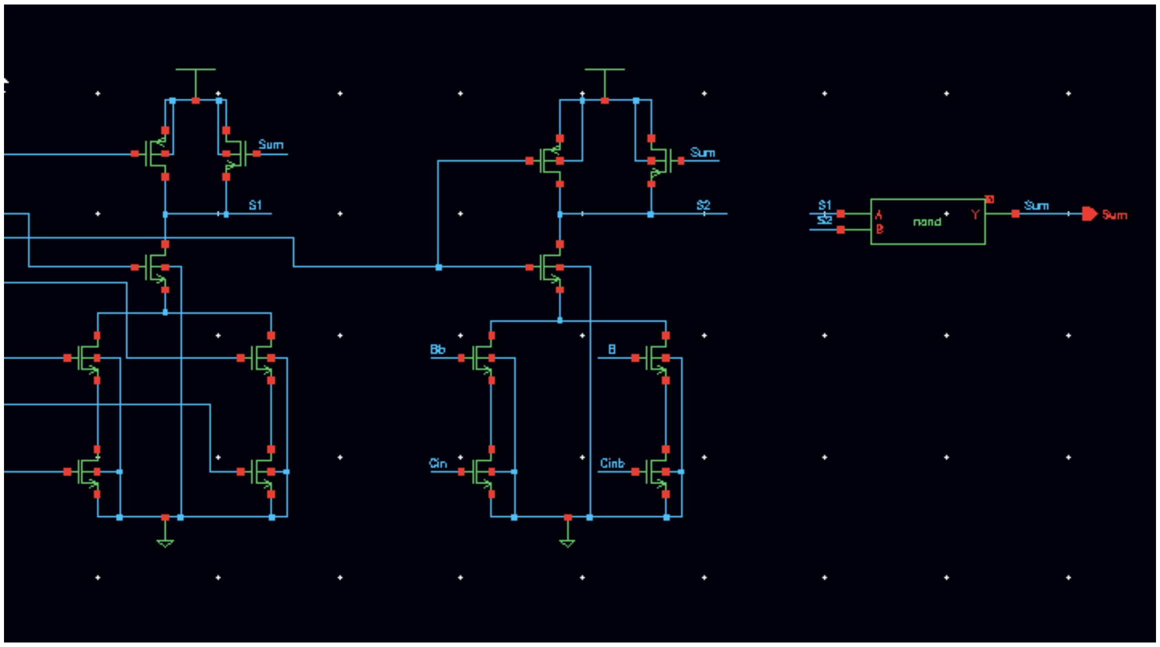


And the Full Adder Schematic.

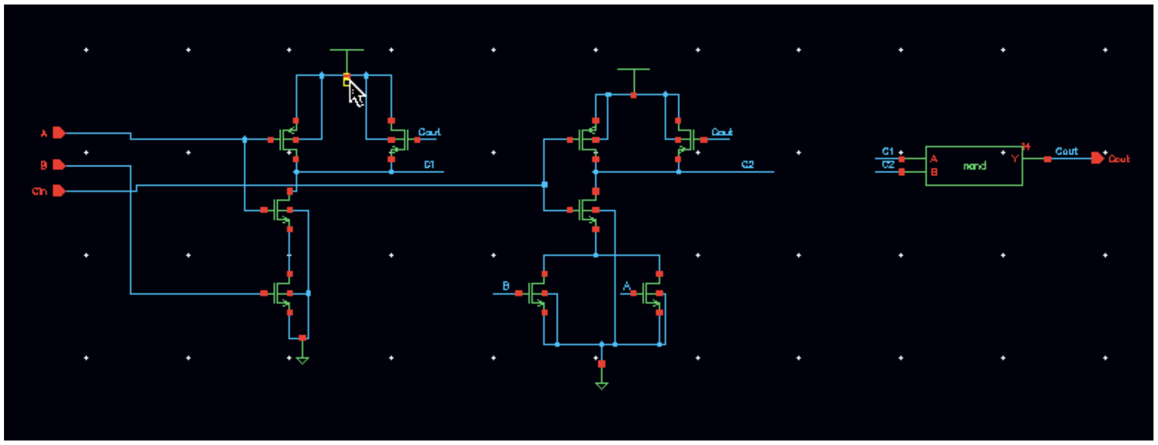


SPD3L Implementation

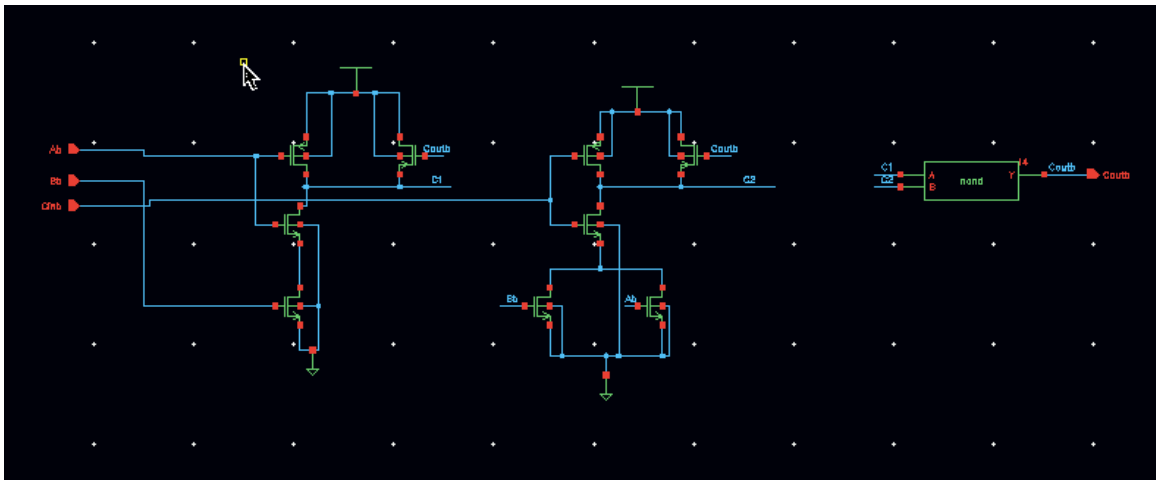
The Sum function in SPD3L has the nMOS transistors of 1080 nm (3*360nm) and pMOS 720nm and the keeper 90nm.



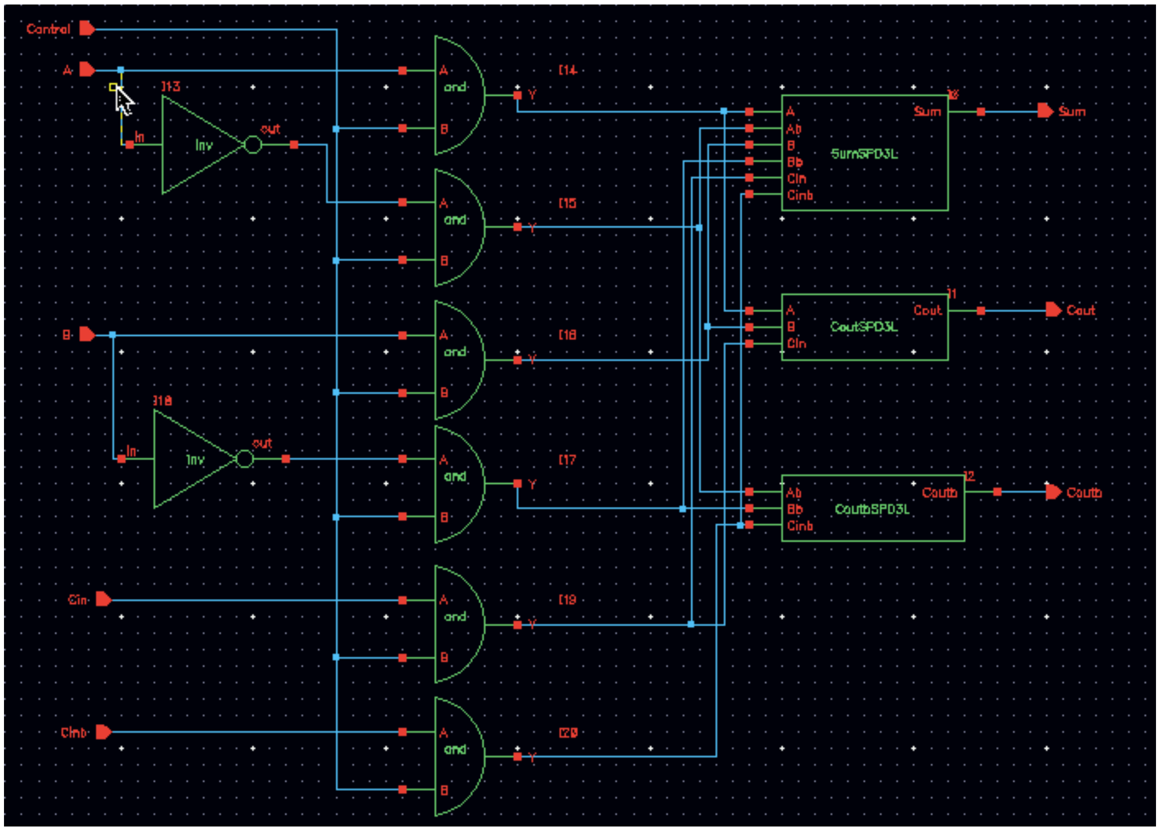
For the Cout the nMOS sizing 720nm and for the pMOS the same, also the keeper has 90nm.



For the same reason the $\overline{Cout}$ has been implemented, with the sizing as the Cout.



Full Adder Schematic.

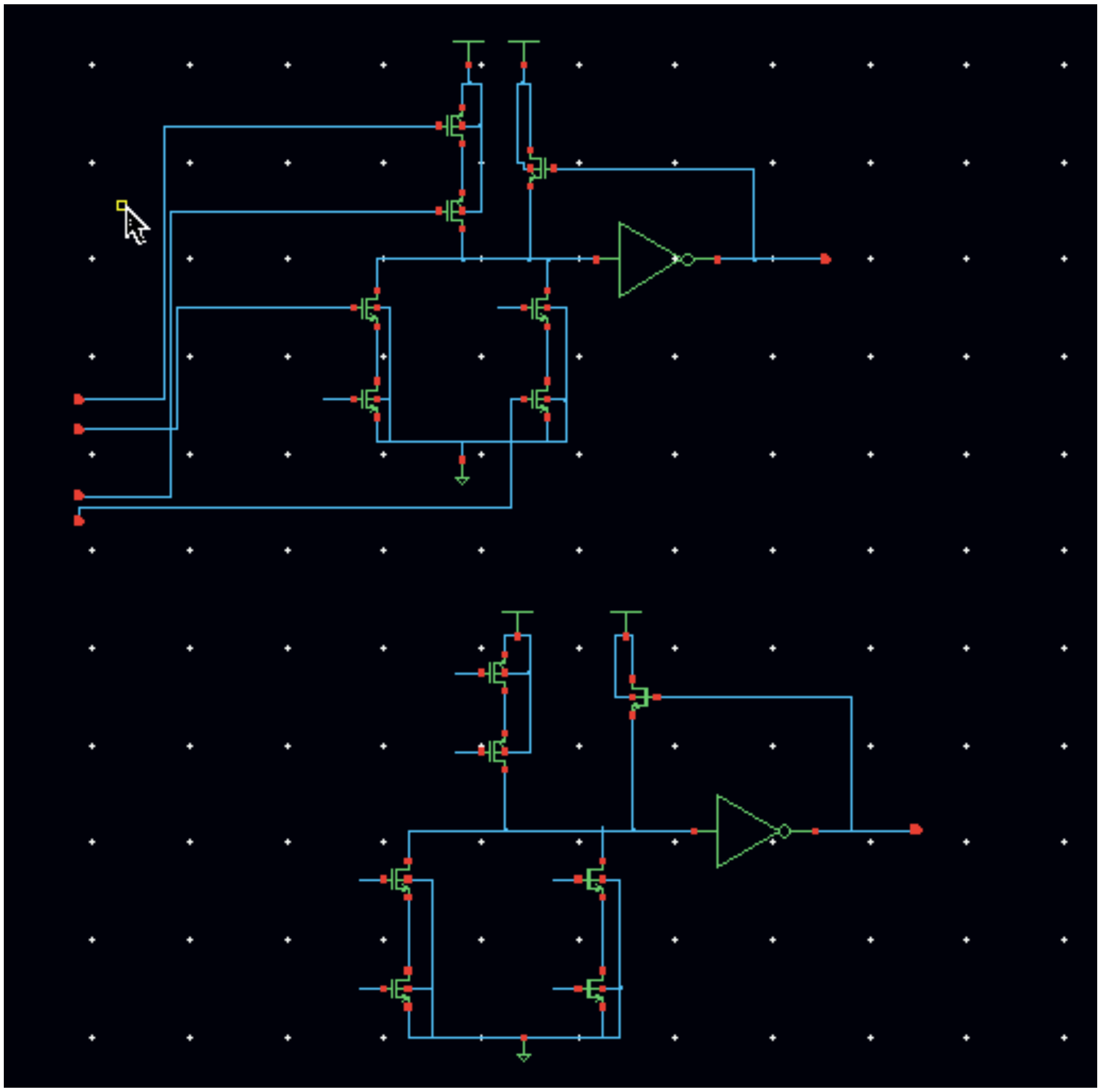


Method2

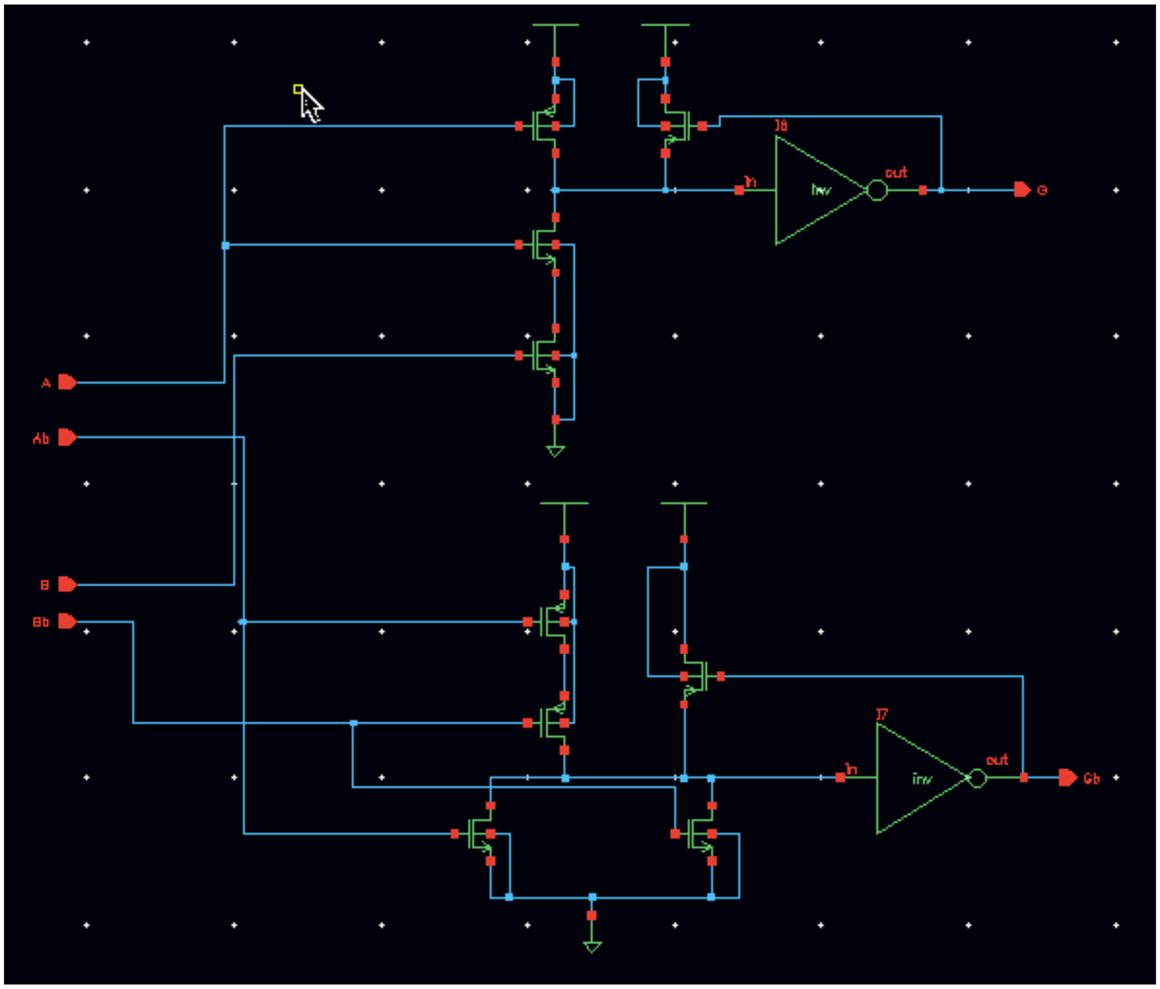
$$Sum = A \oplus B$$
$$G = AB$$
$$Sum = P \oplus Cin$$
$$Cout = G + PCin$$

<u>*D3L Implementation*</u>

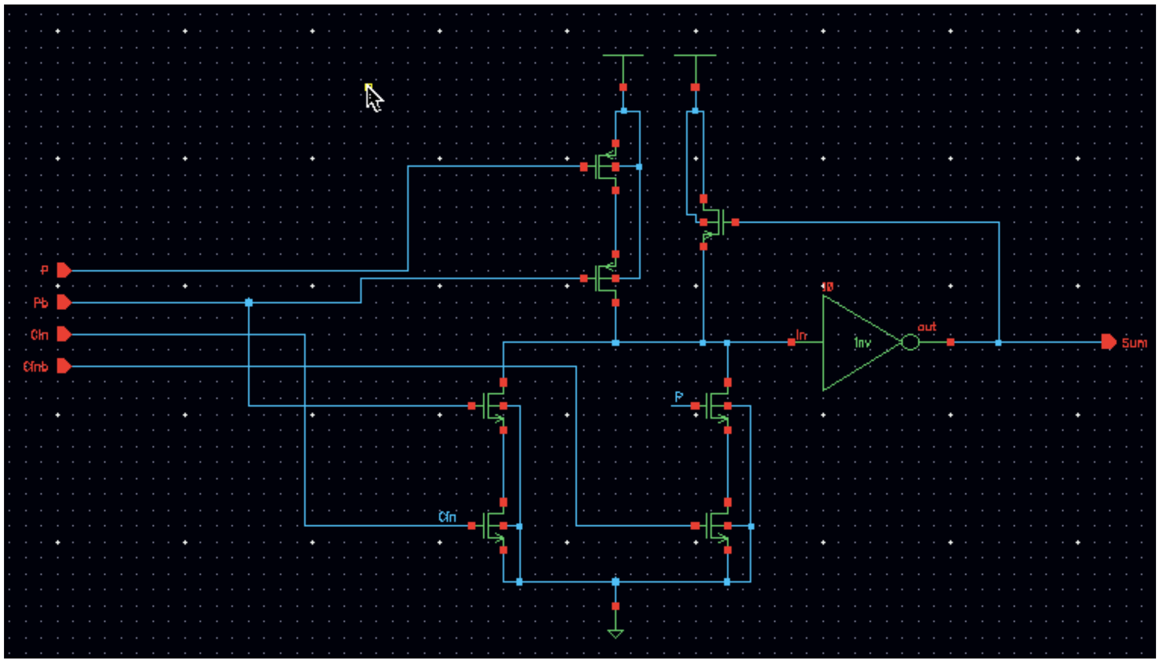
The P and $\bar{P}$ have the same sizing with nMOS transistors 720nm (2x360nm) and pMOS 720nm, with the keeper of 90nm.



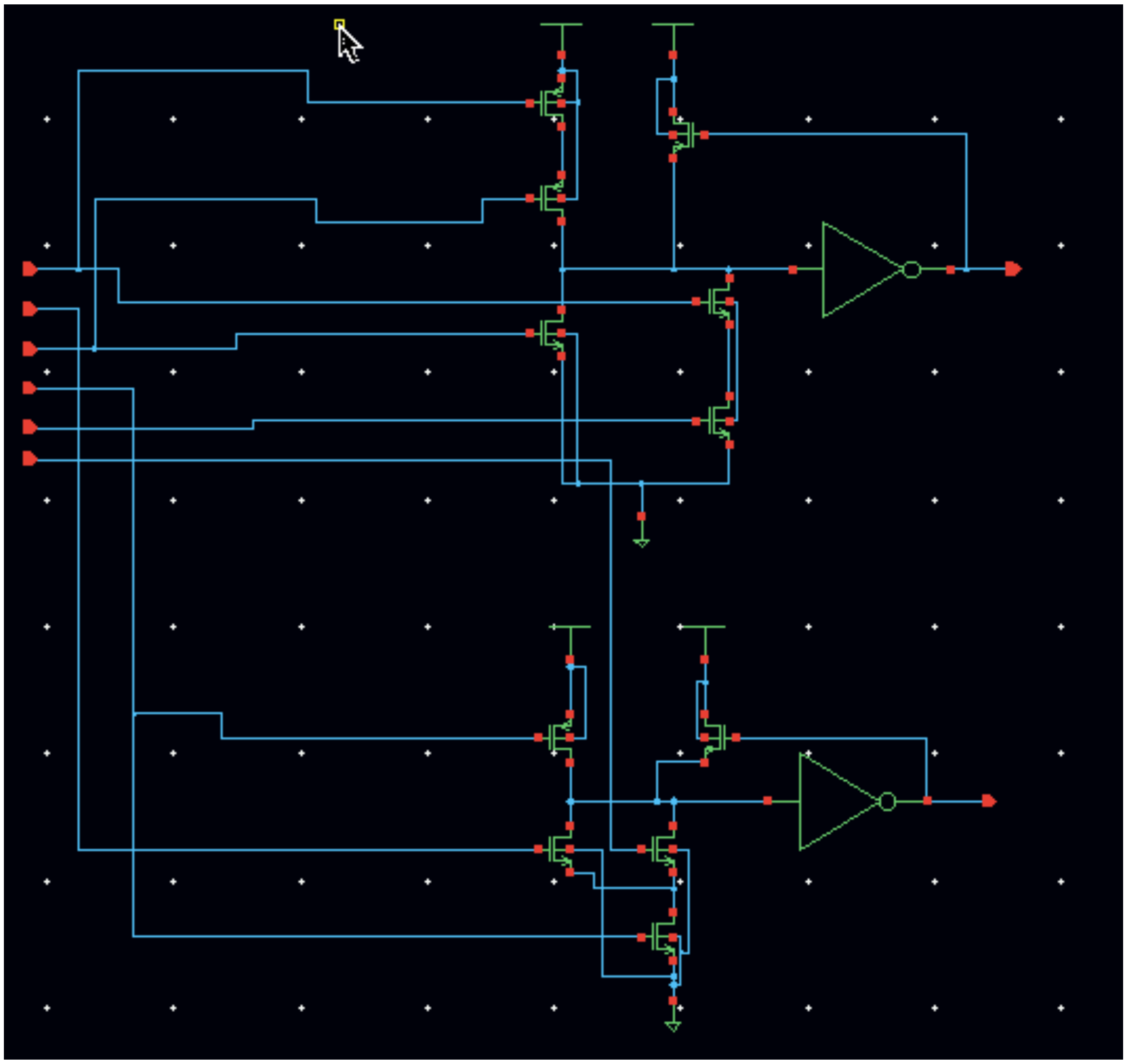
And G with nMOS and pMOS 720nm, $\bar{G}$ with nMOS 360nm and pMOS 720nm.



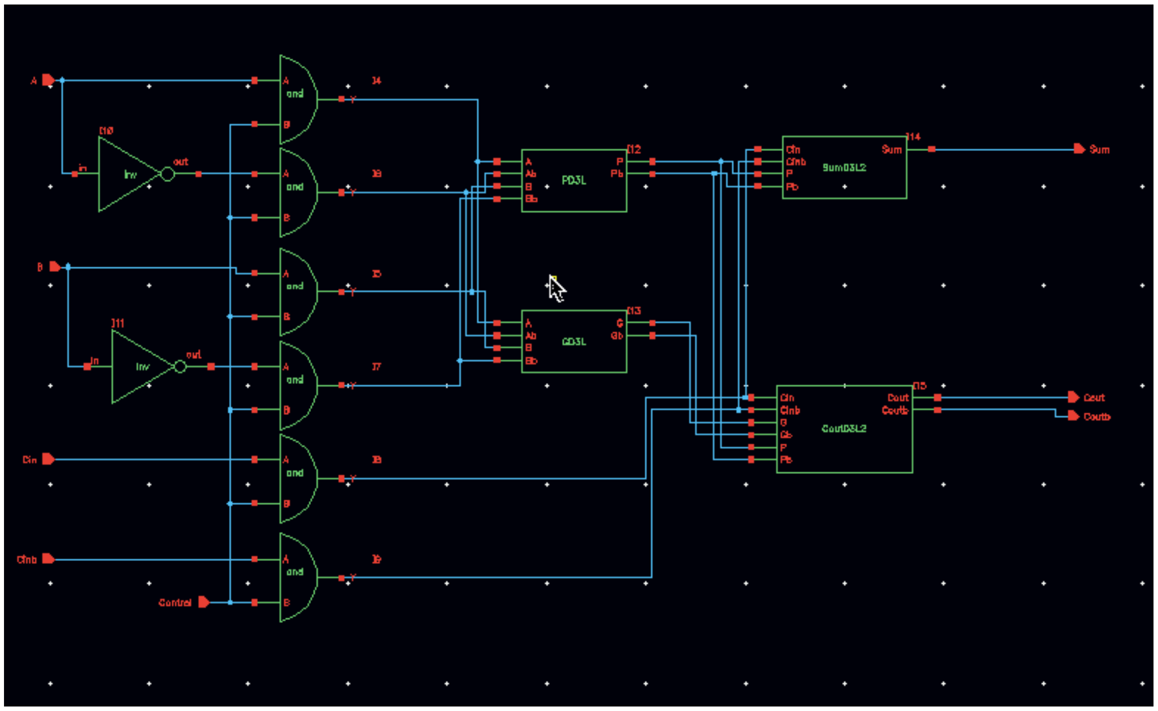
Sum with nMOS and pMOS 720nm.



Then the Cout and $\overline{Cout}$ with nMOS 180, 360 nm of and pMOS of 720 nm.



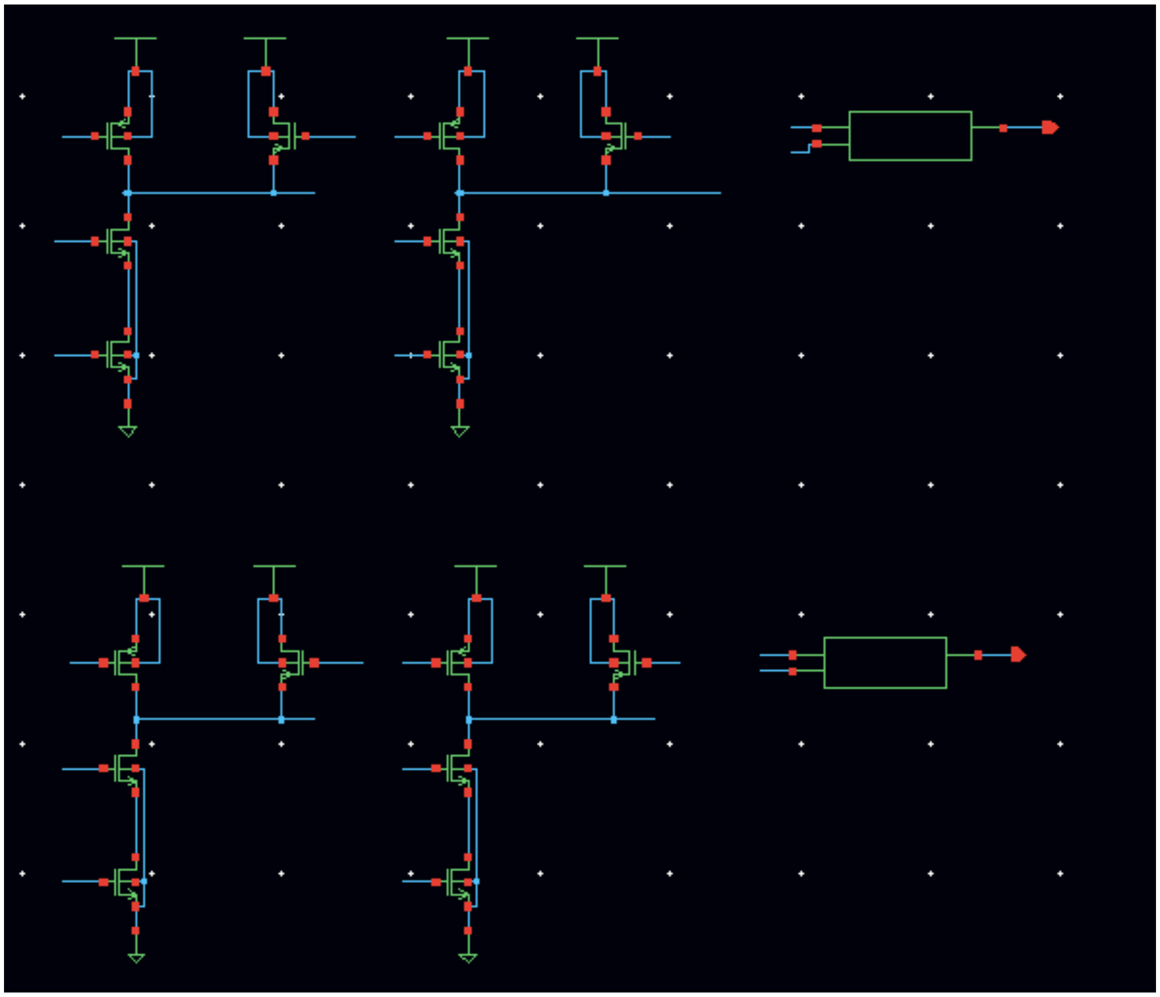
Full Adder Schematic.



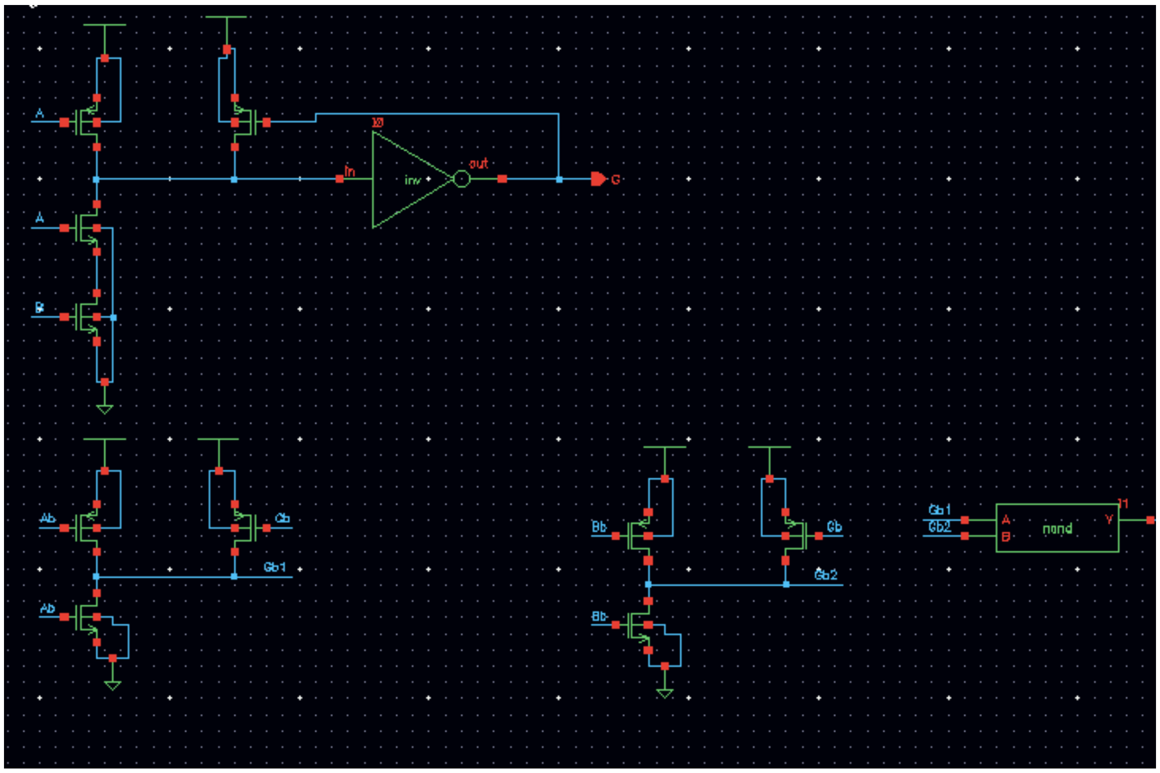
SPD3L Implementation

Again for this method we need to implement the functions P and G.

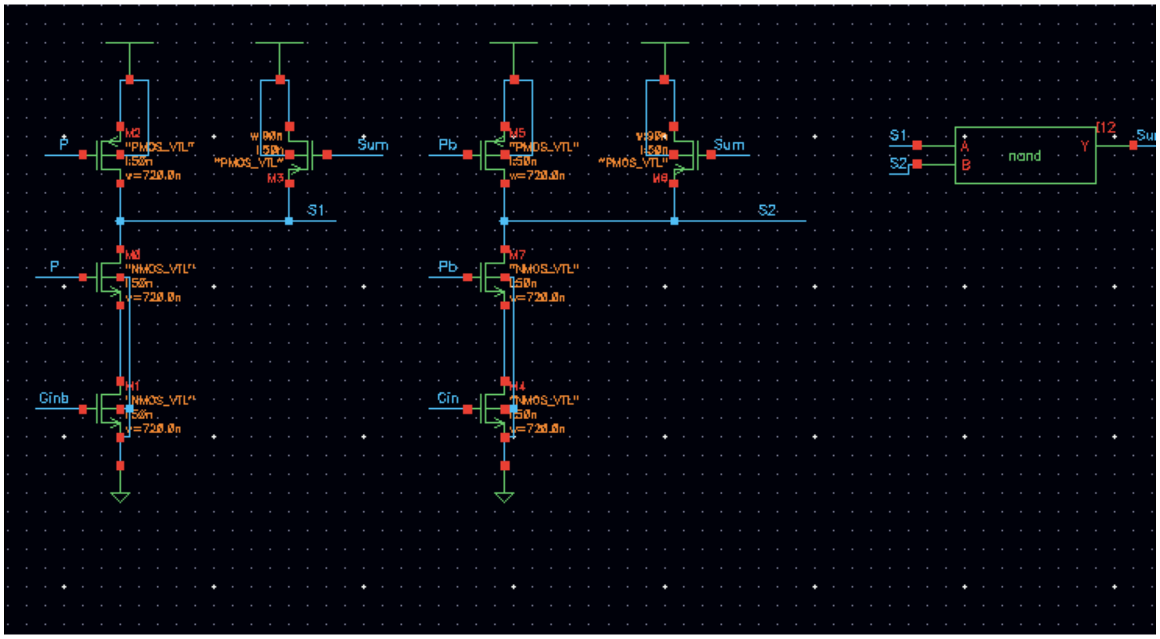
For P and $\bar{P}$ the sizes of the nMOS are 720nm and the pMOS the same, for the keeper also 90 nm.



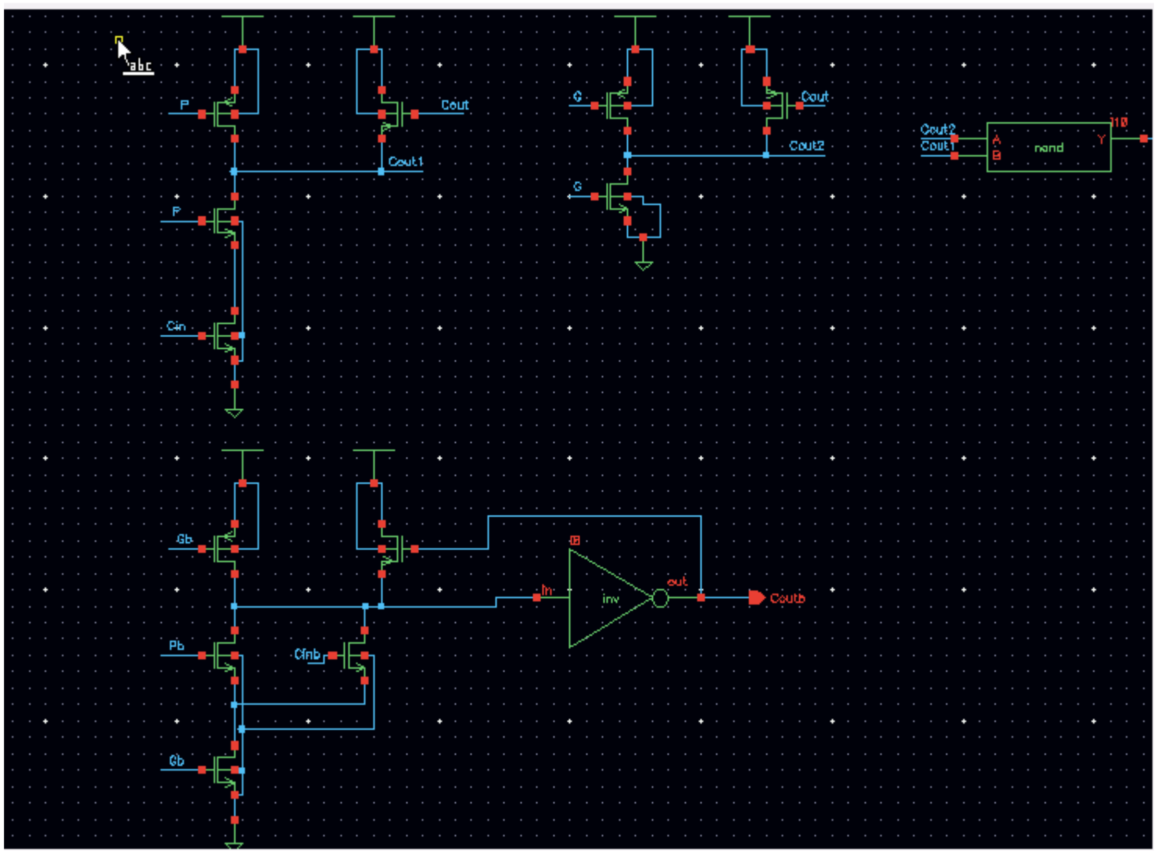
For G nMOS and pMOS are 720nm and for $\bar{G}$ nMOS are 360 nm and pMOS 720 nm.



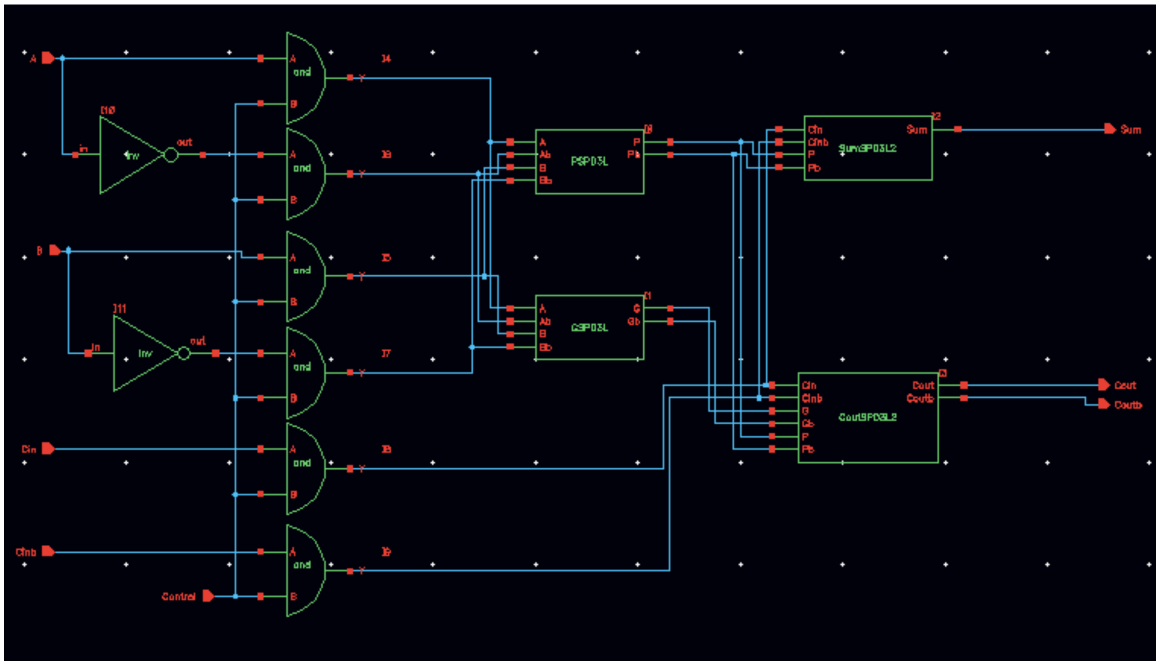
Sum with nMOS and pMOS both of 720nm.



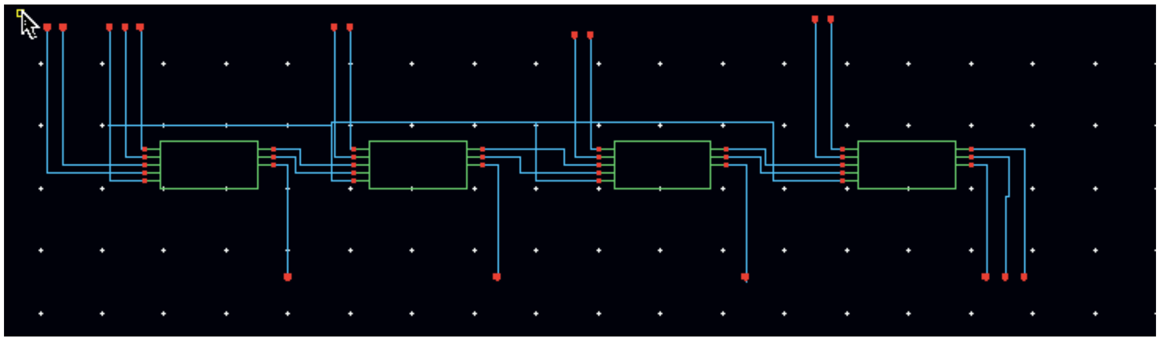
And to finish Cout with 720nm and 360 nm of nMOS, 720nm pMOS. $\overline{Cout}$ with both 720nm.



Full Adder Schematics.



Then for all type of Full Adders, a 4 bit Full Adder have been built in order to test the design.



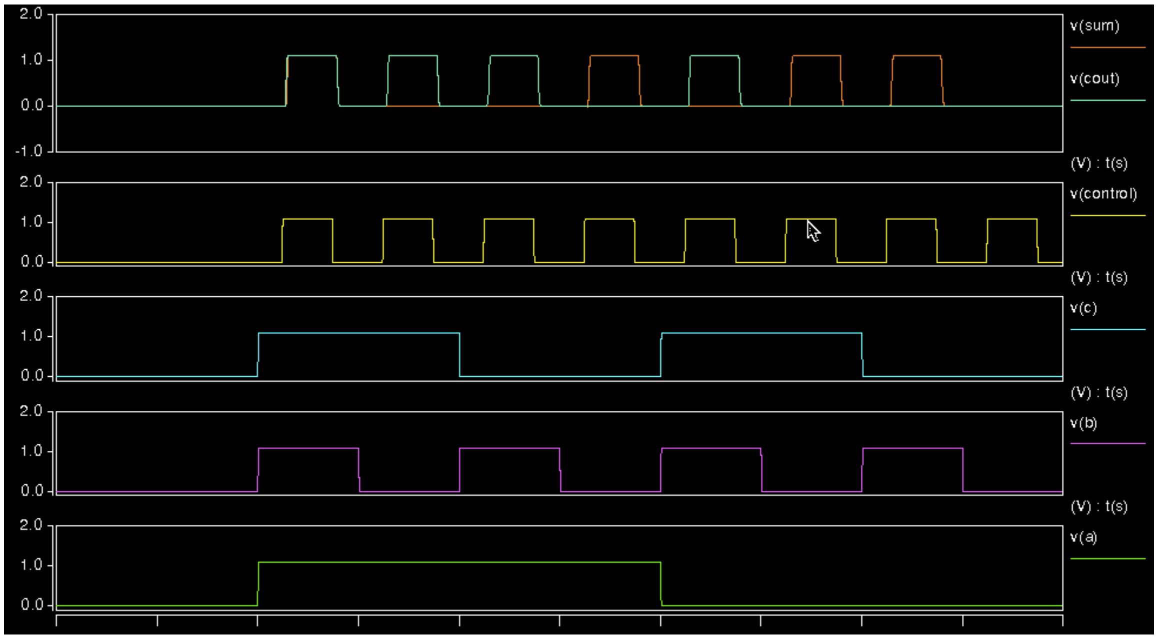
Functional Validation and Verification

To verificate the functionality of the 4 bit Full Adder, first the 1 bit Full Adder have been tested. Once the functionality have been checked the verification of the 4 bit is the next step, and then the worst case scenario of A=”1111”, B=“1111” and Cin = ‘1’ is used to make the delay measures and also the power estimation.

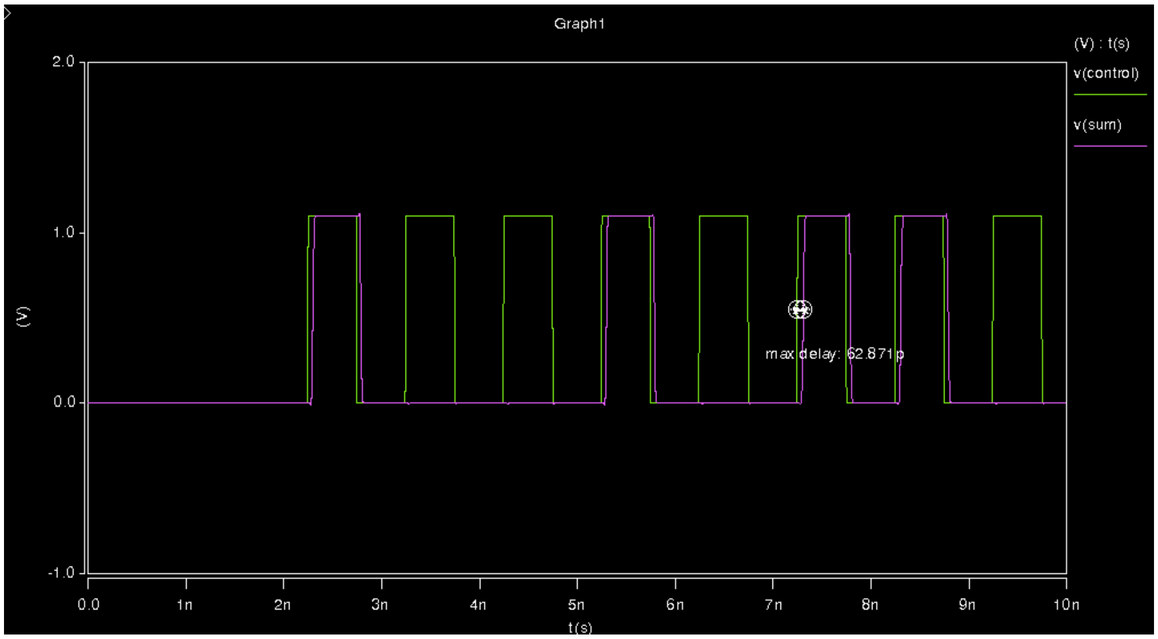
Method 1

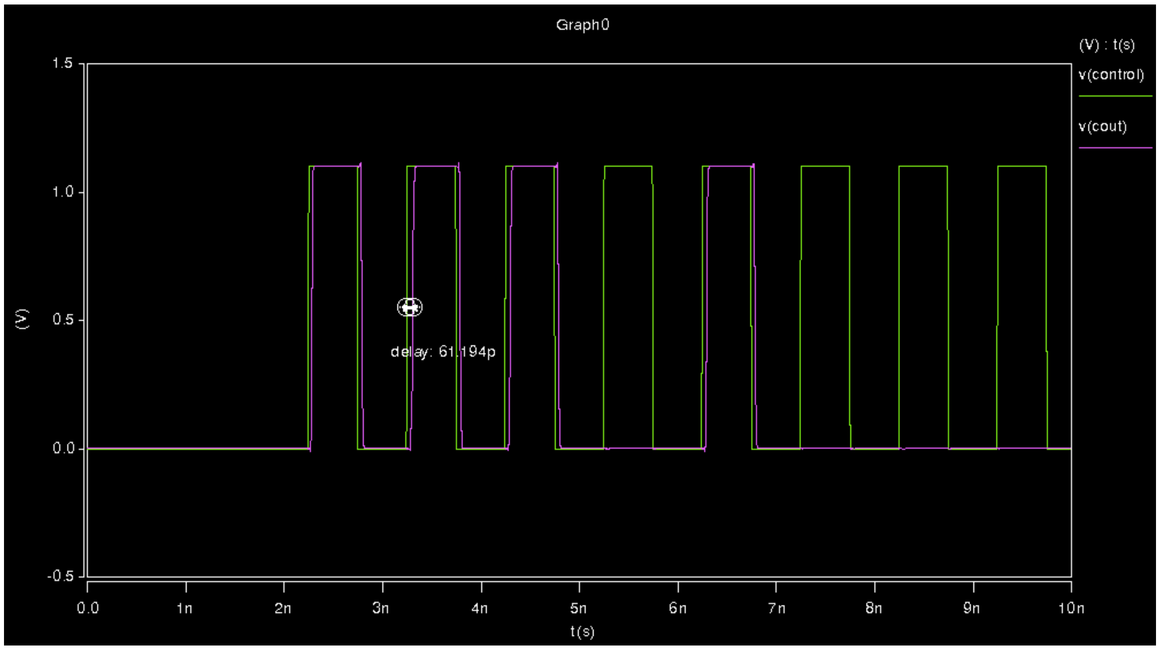
$$Sum = A \oplus B \oplus Cin$$
$$Cout = AB + Cin(A + B)$$

D3L Implementation



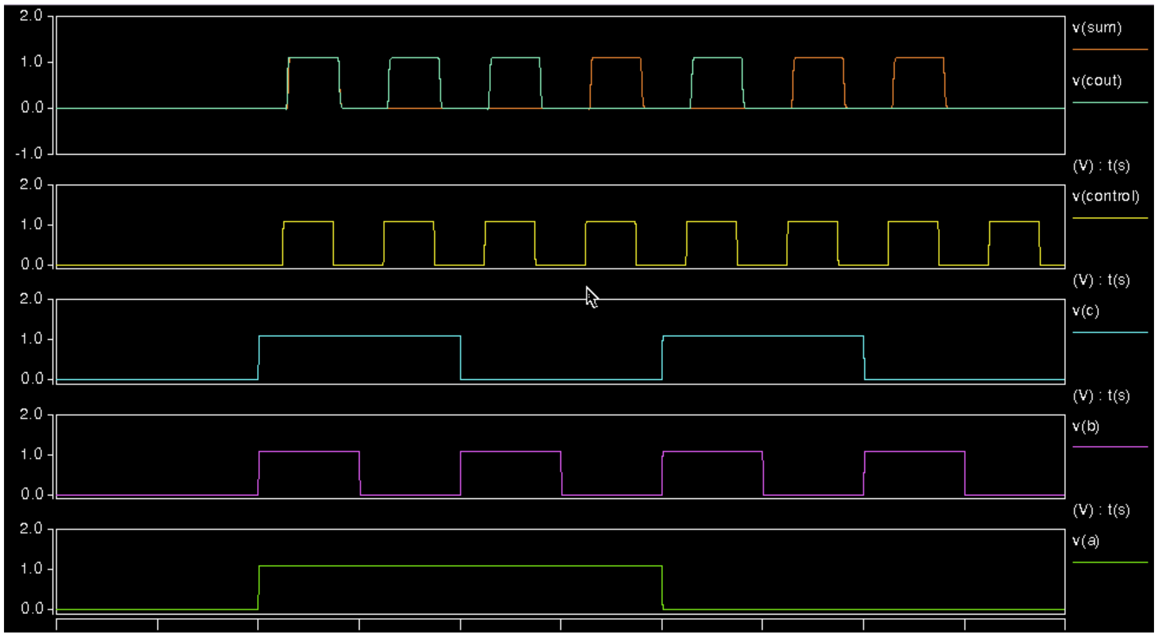
As we can see in this plot all possible combinations have been tested. Also measures in terms of delay have been made.



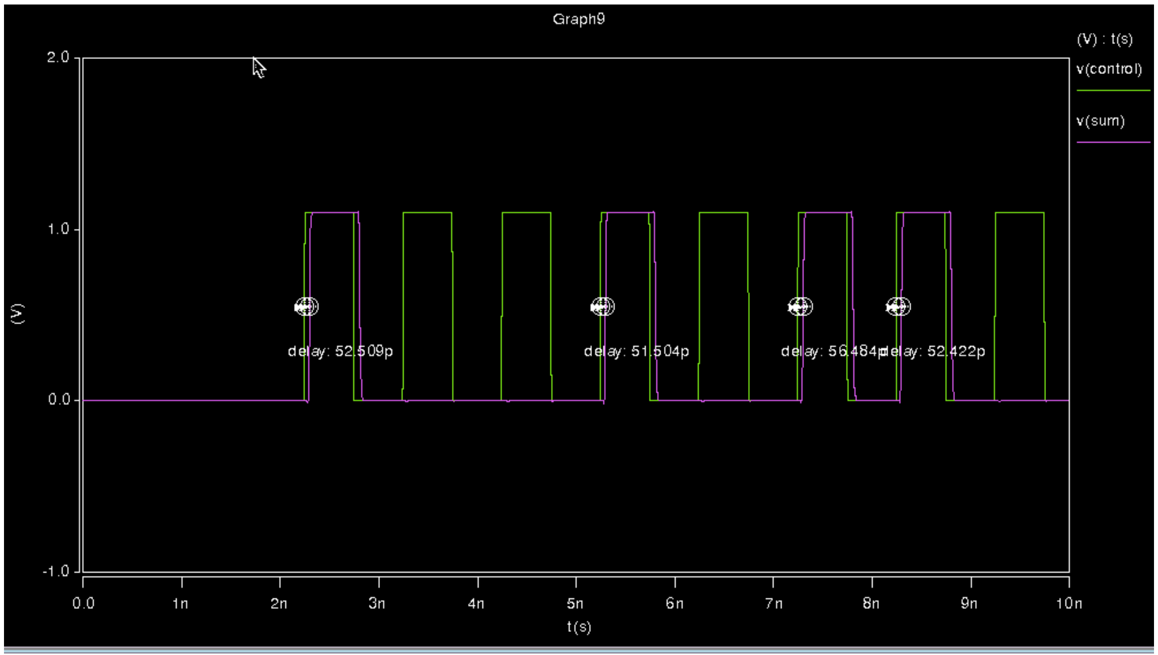


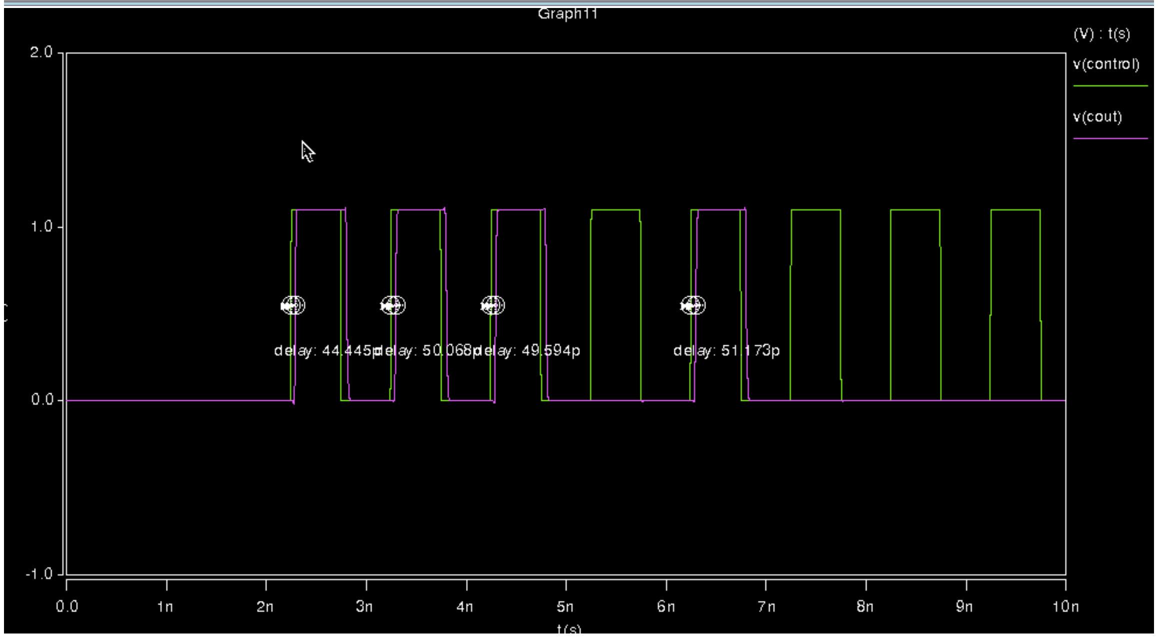
Test.

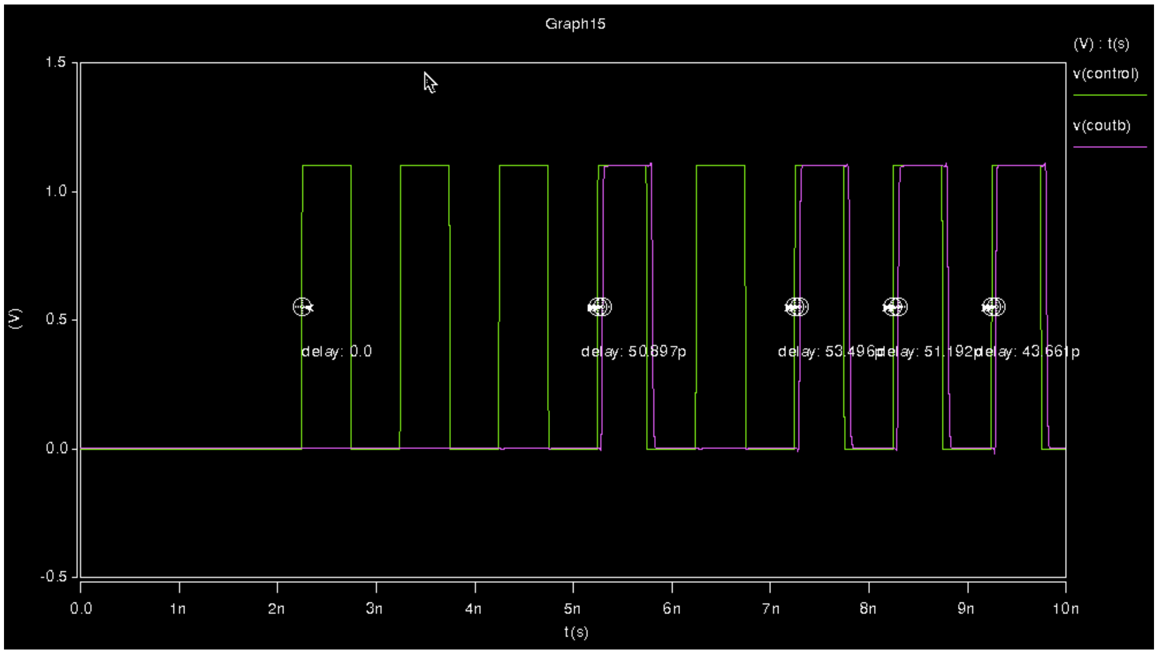
SPD3L Implementation



Delay.


Graph9
(V) : t(s)
v(control)
v(sum)
(V)
2.0
1.0
0.0
-1.0
delay: 52.509p
delay: 51.504p
delay: 56.484p
delay: 52.422p
0.0
1n
2n
3n
4n
5n
6n
7n
8n
9n
10n
t(s)


Graph11
(V) : t(s)
v(control)
v(cout)
2.0
1.0
0.0
-1.0
delay: 44.445p
delay: 50.068p
delay: 49.594p
delay: 51.173p
0.0
1n
2n
3n
4n
5n
6n
7n
8n
9n
10n
t(s)

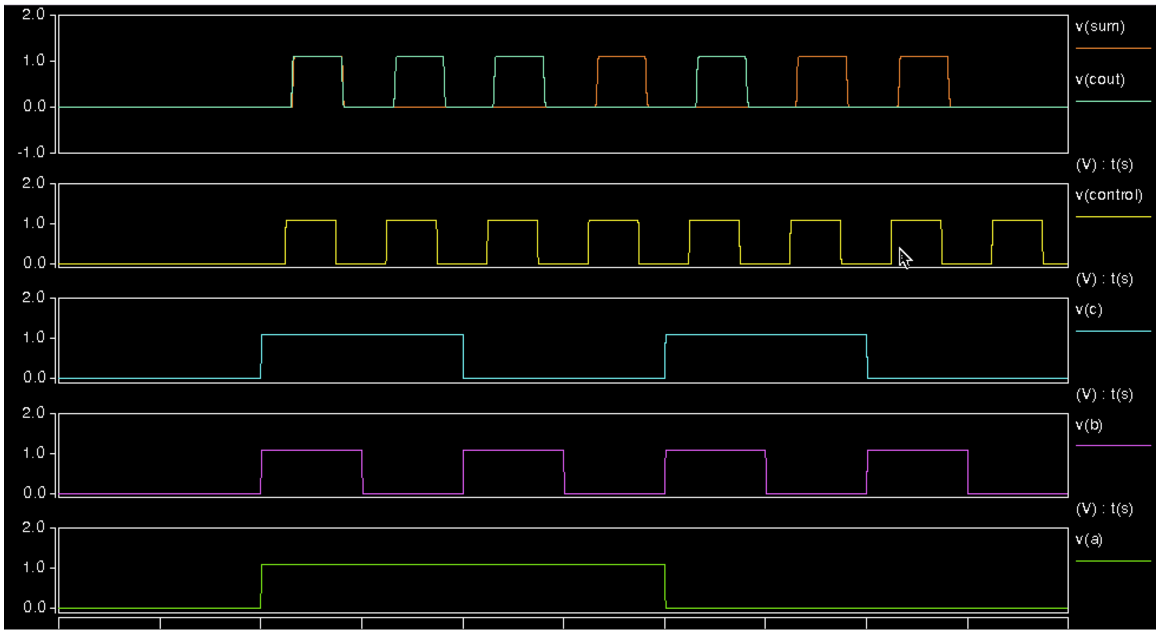


Method2

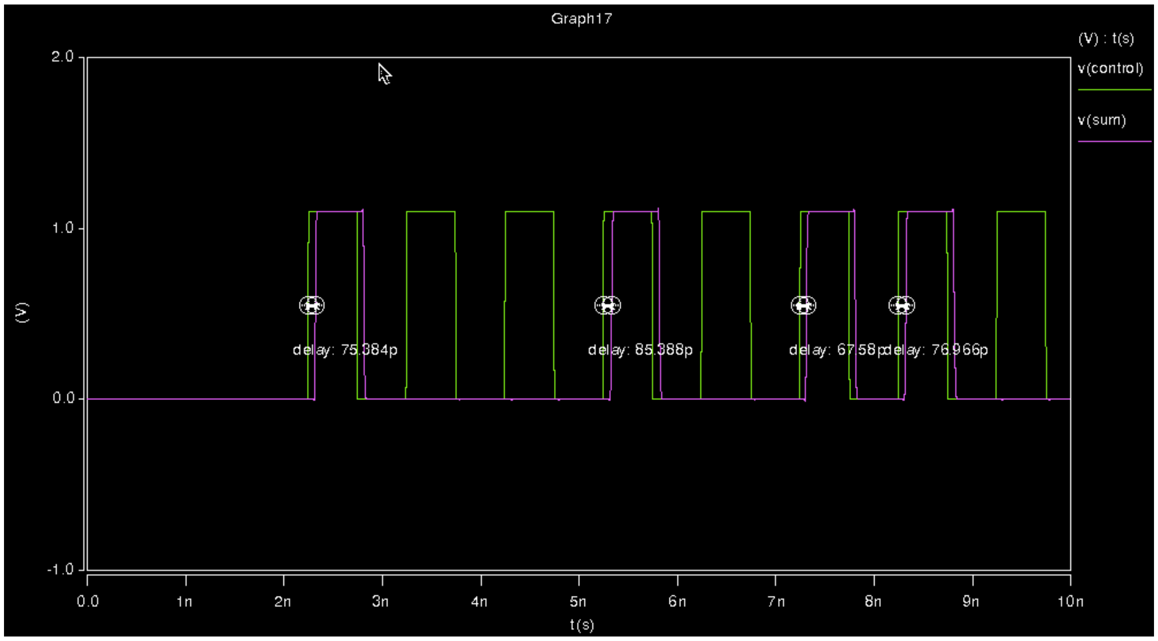
$$Sum = A \oplus B$$
$$G = AB$$
$$Sum = P \oplus Cin$$
$$Cout = G + PCin$$

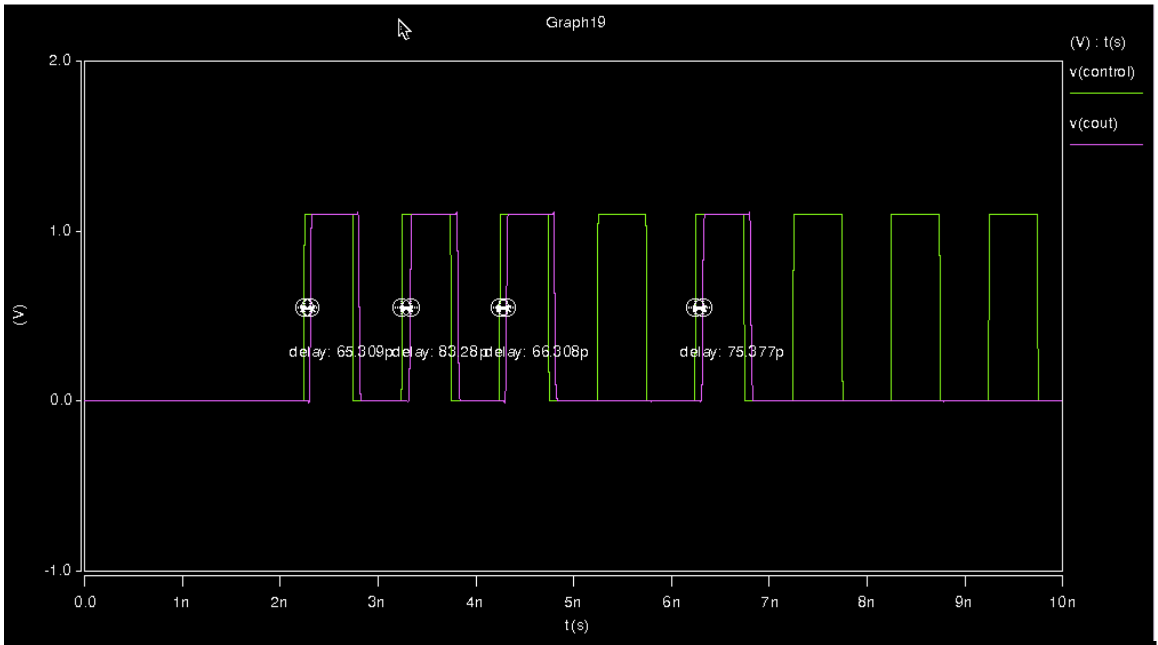
<u>*D3L Implementation*</u>

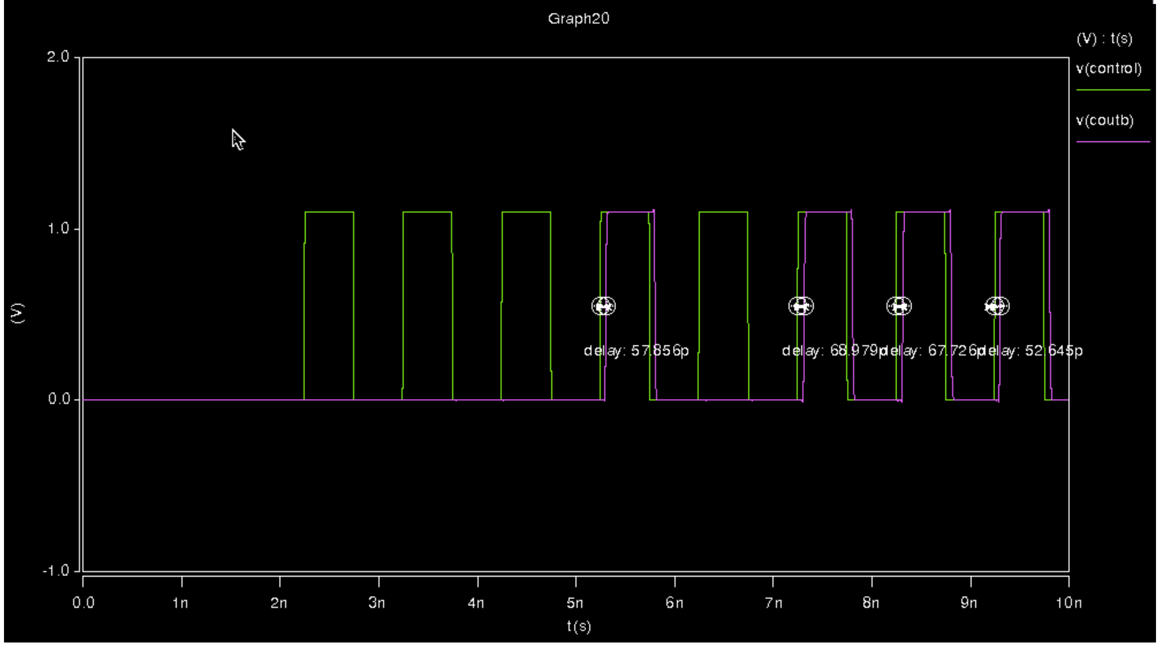
Test.



Delay.

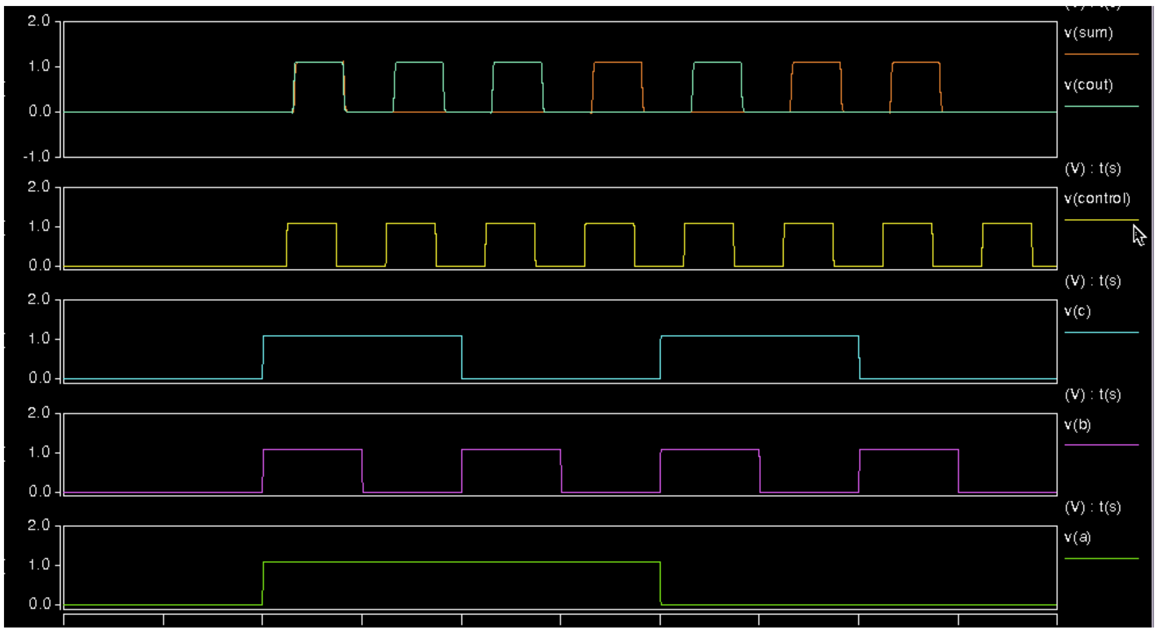

Graph17
(V) : t(s)
v(control)
v(sum)
2.0
1.0
0.0
-1.0
(V)
delay: 75.384p
delay: 85.388p
delay: 67.58p
delay: 76.966p
0.0
1n
2n
3n
4n
5n
6n
7n
8n
9n
10n
t(s)



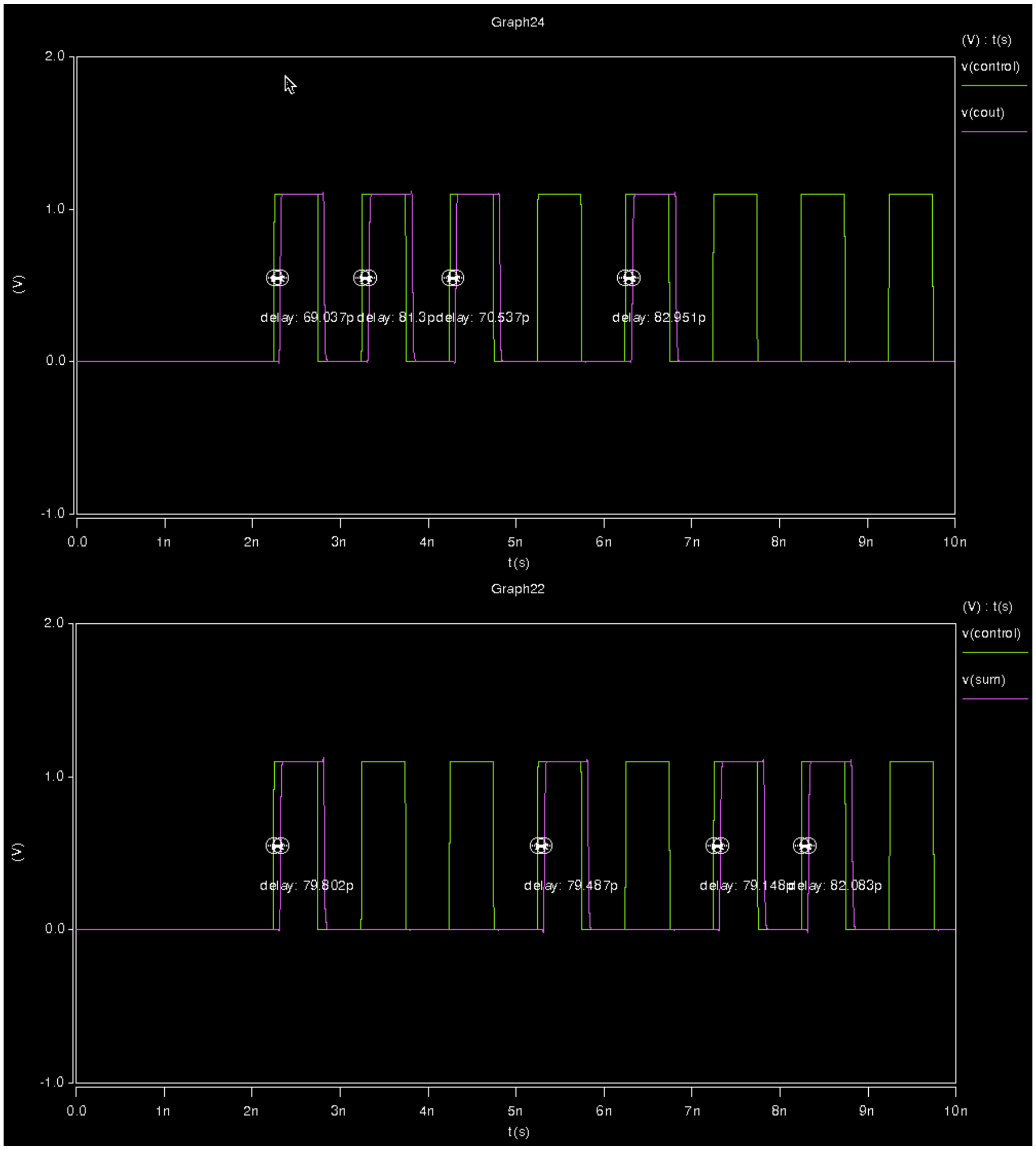


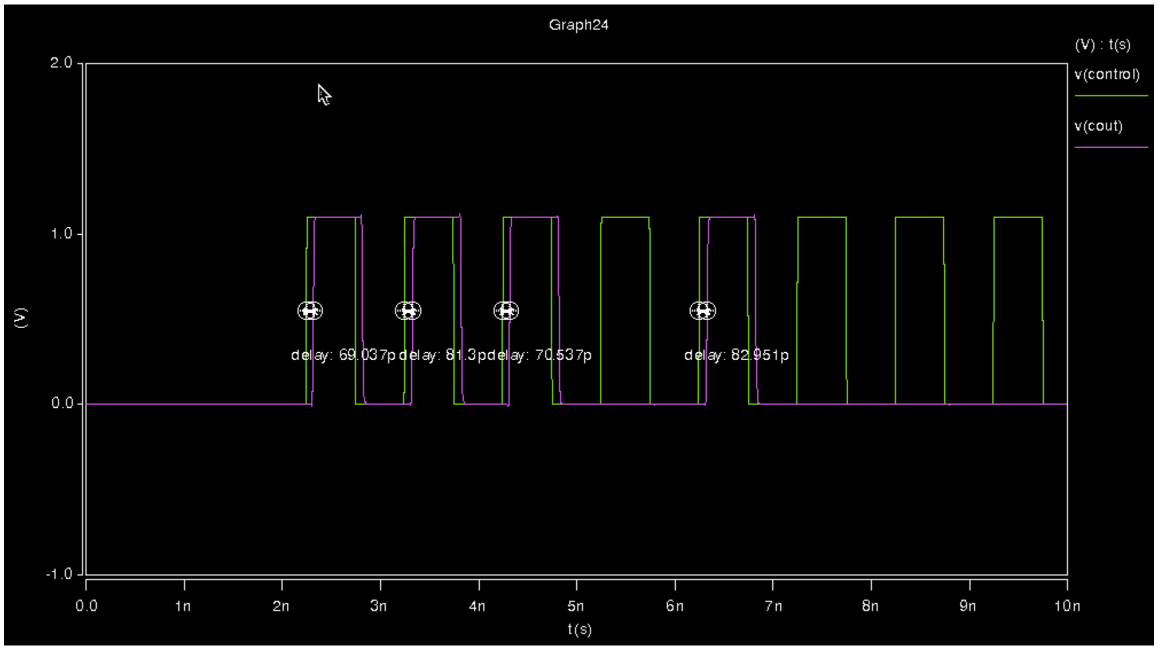
SPD3L Implementation

Test.



Delay.


Graph24
(V) : t(s)
v(control)
v(cout)
2.0
1.0
0.0
-1.0
(V)
delay: 69.037p
delay: 81.3p
delay: 70.537p
delay: 82.951p
0.0
1n
2n
3n
4n
5n
6n
7n
8n
9n
10n
t(s)
Graph22
(V) : t(s)
v(control)
v(sum)
2.0
1.0
0.0
-1.0
(V)
delay: 79.802p
delay: 79.487p
delay: 79.148p
delay: 82.083p
0.0
1n
2n
3n
4n
5n
6n
7n
8n
9n
10n
t(s)

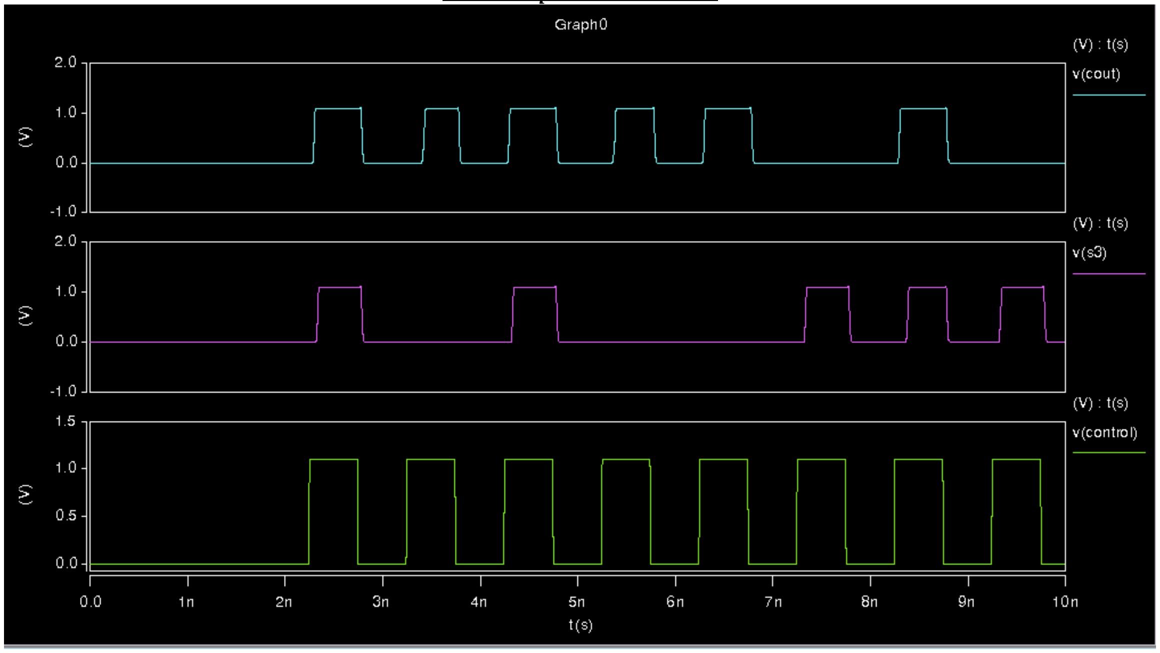


Now the test for the 4 bit Full Adder, once their functionality are tested, the measures are taken. In these images I will plot only the cases for the delay, showing all the test images will make the report even bigger.

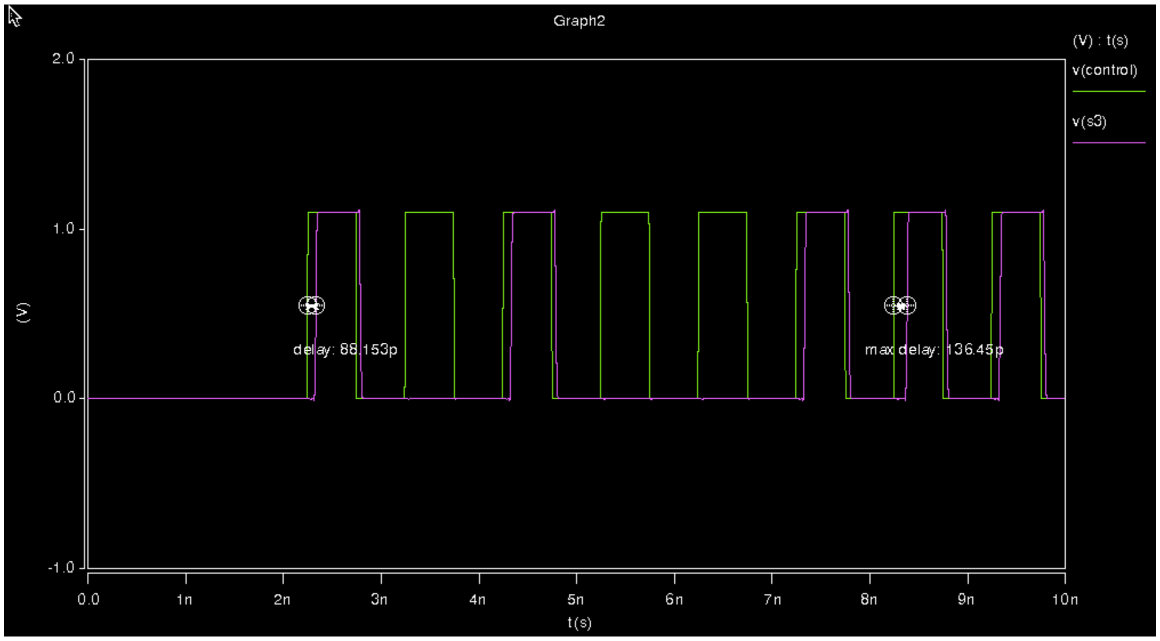
Method 1

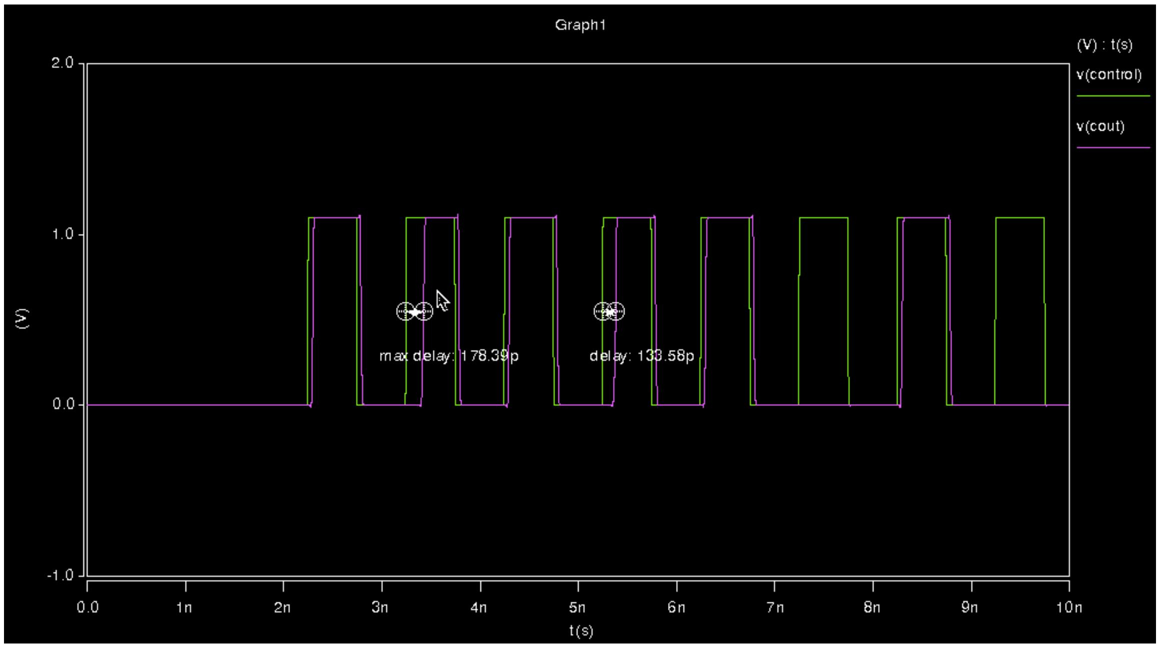
$$Sum = A \oplus B \oplus Cin$$
$$Cout = AB + Cin(A + B)$$

<u>D3L Implementation</u>

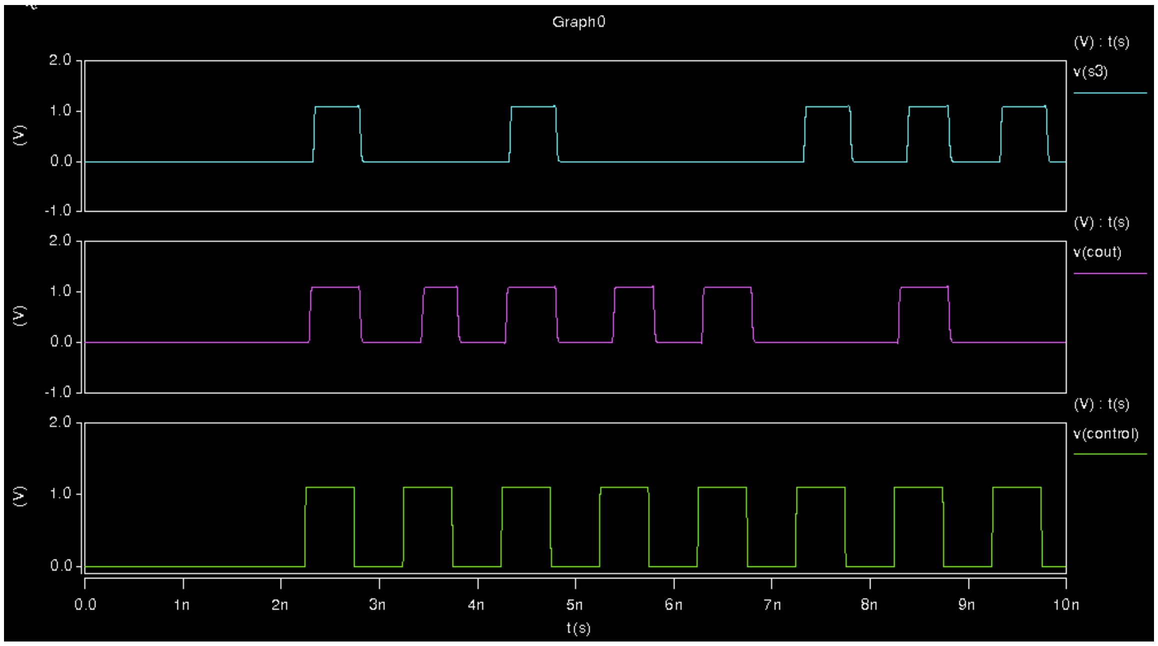


The delay have been measure for the 2 worst signals in terms of the delay, S3 and Cout. Although in theory the worst case scenario is the delay for S3.

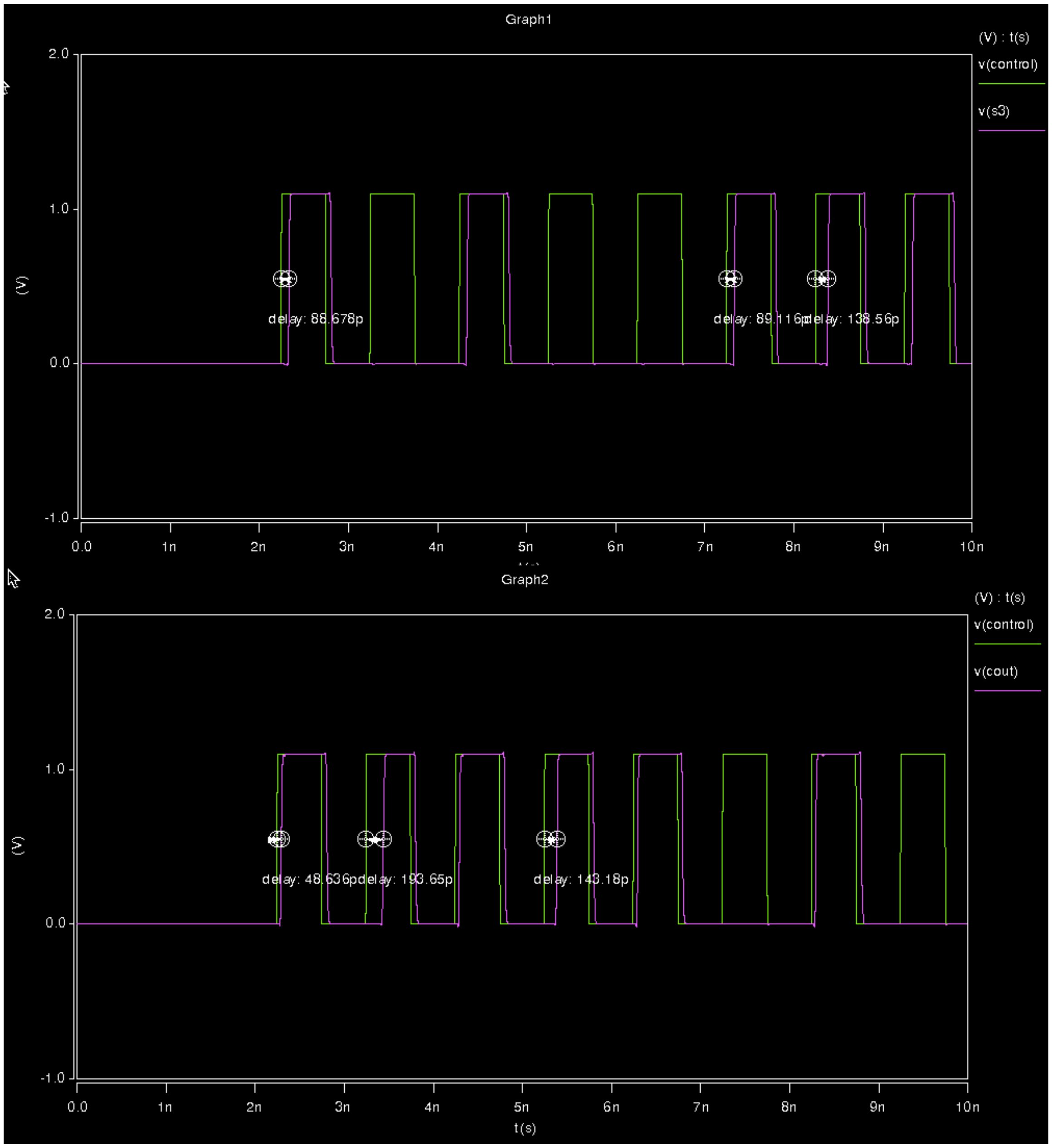




<u>SPD3L Implementation</u>


Graph0
(V) : t(s)
v(s3)
(V) : t(s)
v(cout)
(V) : t(s)
v(control)
(V)
t(s)
0.0
1n
2n
3n
4n
5n
6n
7n
8n
9n
10n
2.0
1.0
0.0
-1.0

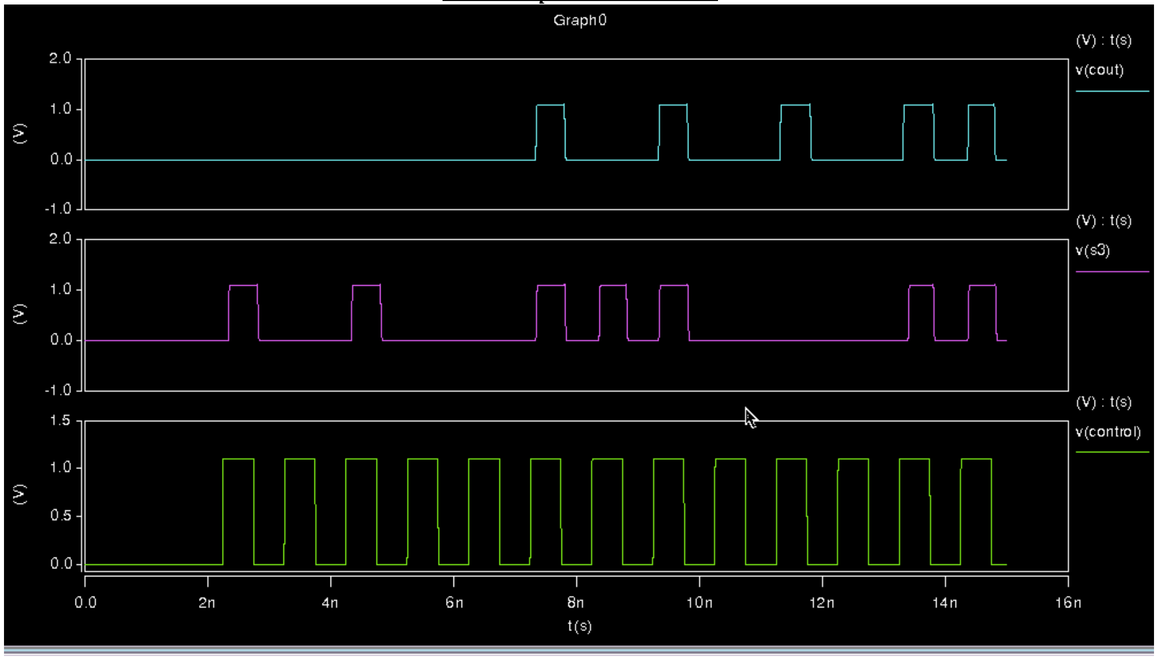
Delay


Graph1
(V) : t(s)
v(control)
v(s3)
2.0
1.0
0.0
-1.0
(V)
delay: 88.678p
delay: 89.116p
delay: 138.56p
0.0
1n
2n
3n
4n
5n
6n
7n
8n
9n
10n
Graph2
(V) : t(s)
v(control)
v(cout)
delay: 48.636p
delay: 193.65p
delay: 143.18p
t(s)

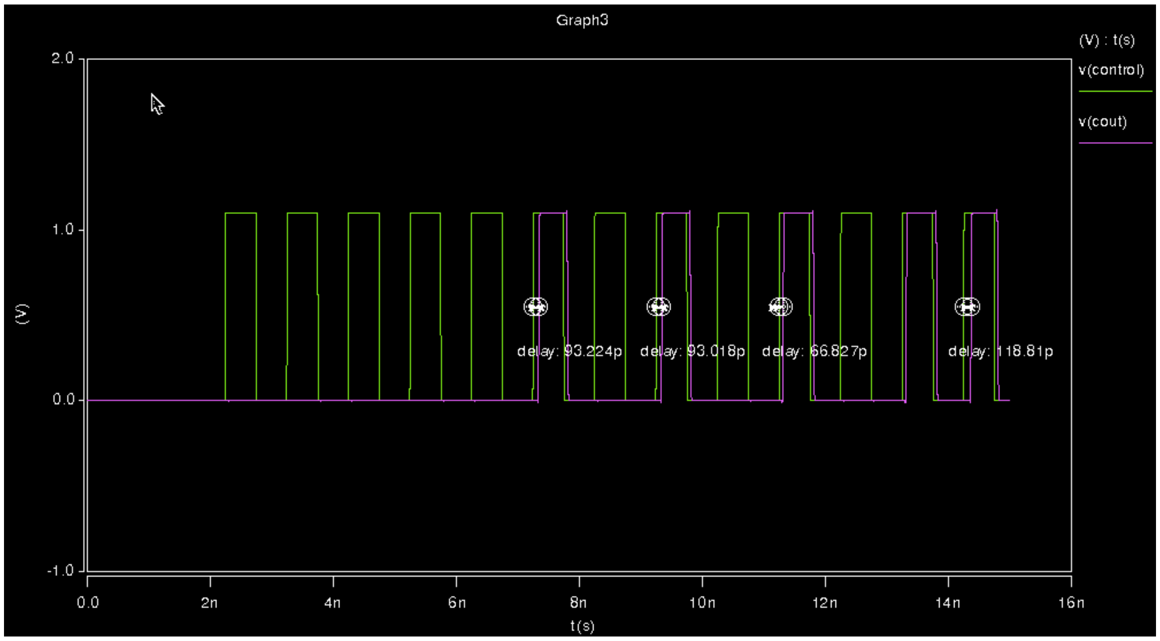
Method2

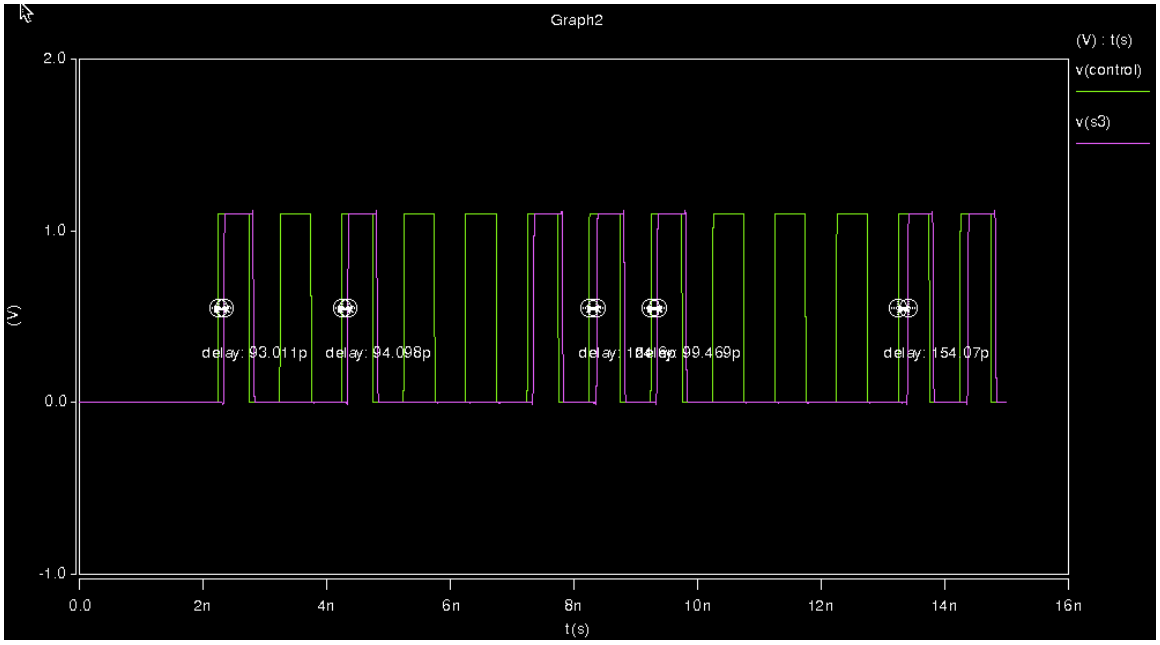
$$Sum = A \oplus B$$
$$G = AB$$
$$Sum = P \oplus Cin$$
$$Cout = G + PCin$$

D3L Implementation

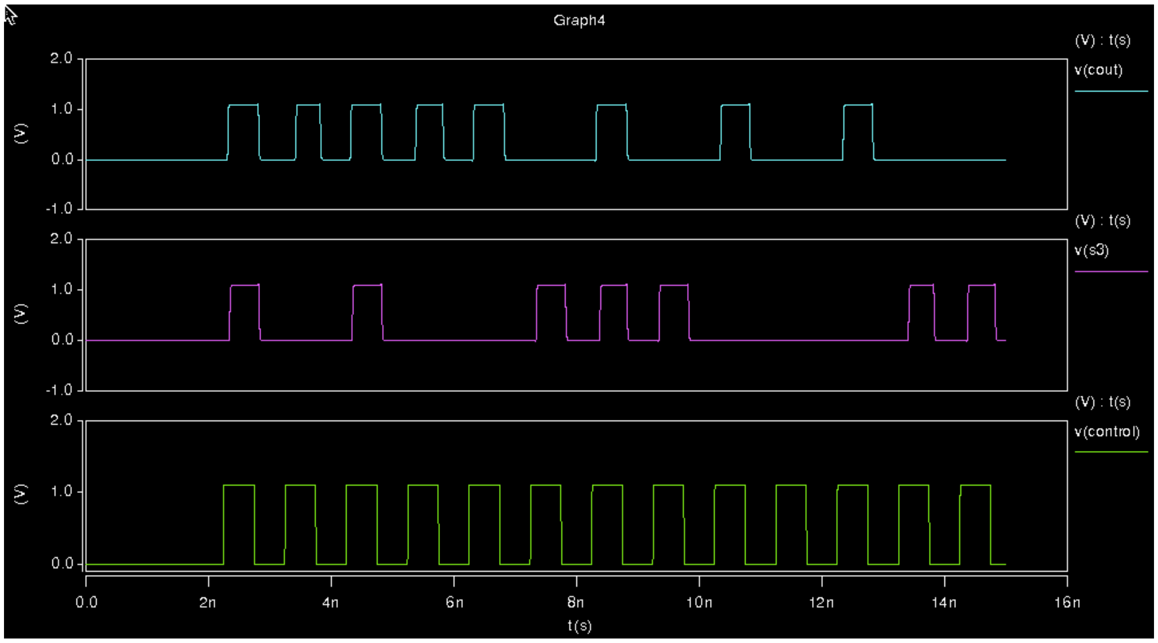


Delay

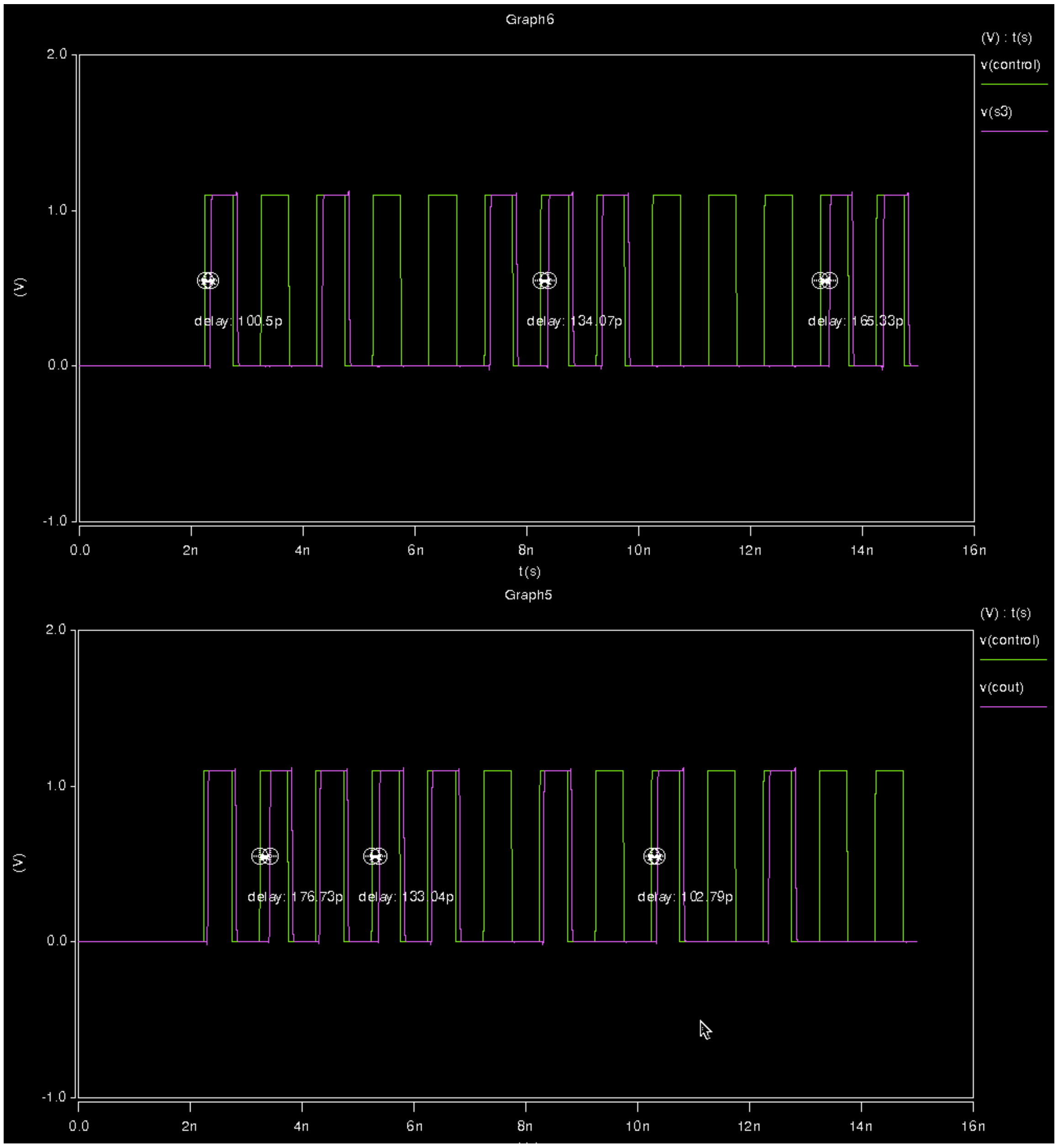




<u>SPD3L Implementation</u>



Delay



Results

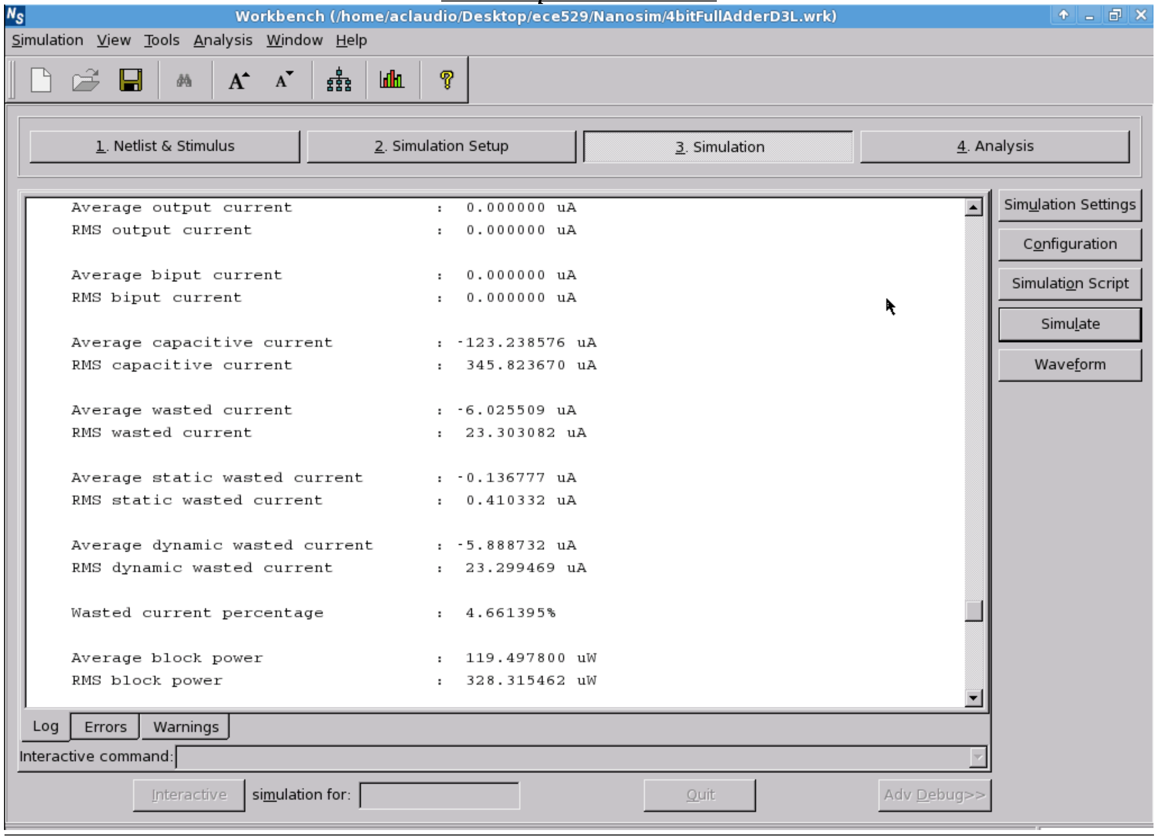
In order to measure the power consumption with Hspice, the following code is introduced to make take this data.

```
.PRINT TRAN POWER
.MEASURE TRAN avgpwr AVG POWER FROM 0 TO 100E-12
.MEASURE TRAN maxpwr MAX POWER FROM 0 TO 100E-12
```

And the following screenshot corresponds to the Nanosim simulation.

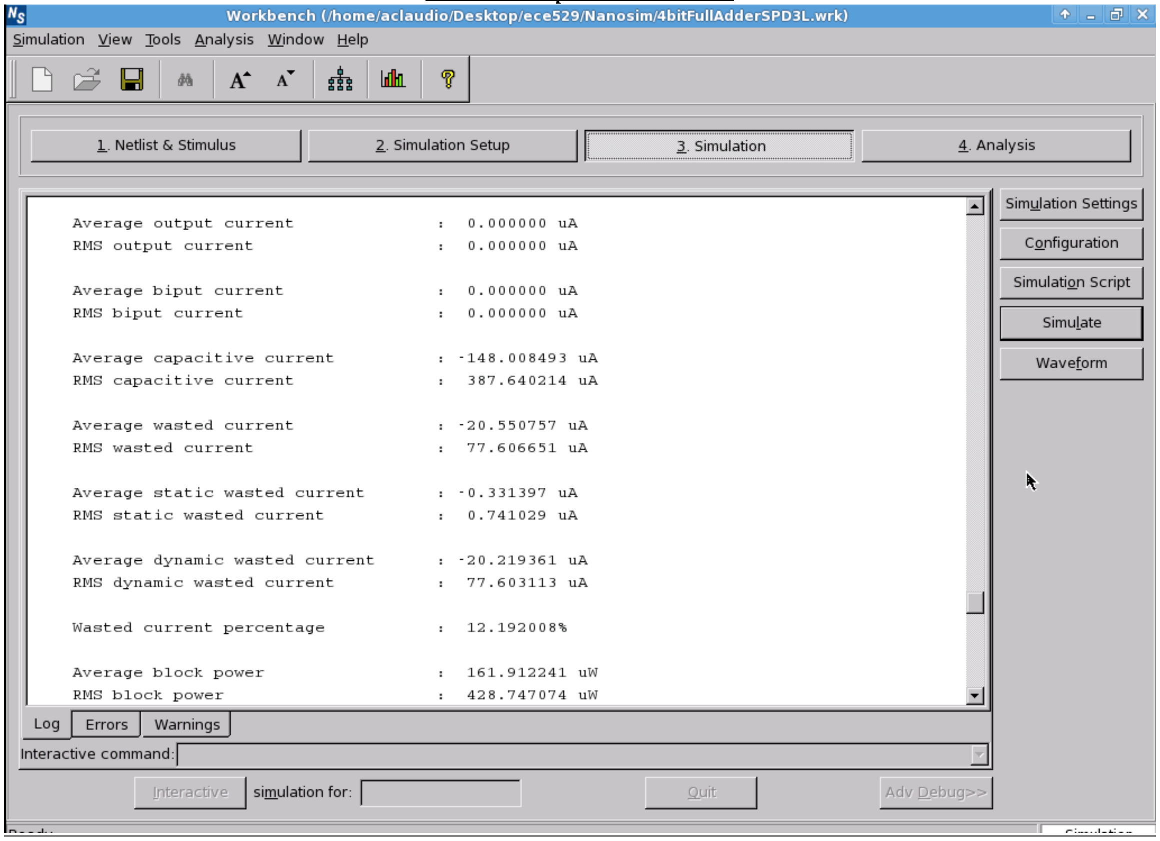
Method 1

<u>*D3L Implementation*</u>



```
    Average output current          :  0.000000 uA
    RMS output current              :  0.000000 uA

    Average biput current           :  0.000000 uA
    RMS biput current               :  0.000000 uA

    Average capacitive current      : -123.238576 uA
    RMS capacitive current          :  345.823670 uA

    Average wasted current          : -6.025509 uA
    RMS wasted current              :  23.303082 uA

    Average static wasted current   : -0.136777 uA
    RMS static wasted current       :  0.410332 uA

    Average dynamic wasted current  : -5.888732 uA
    RMS dynamic wasted current      :  23.299469 uA

    Wasted current percentage       :  4.661395%

    Average block power             :  119.497800 uW
    RMS block power                 :  328.315462 uW
```

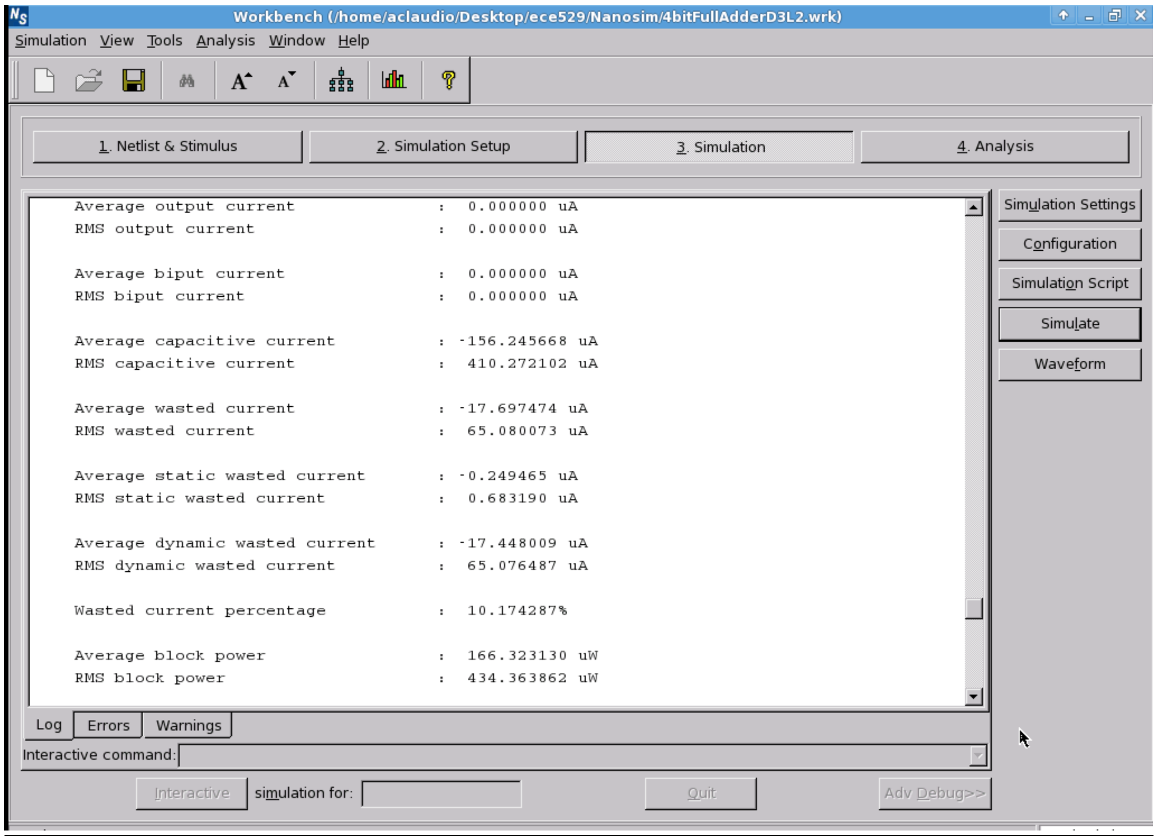
SPD3L Implementation



```
    Average output current           :  0.000000 uA
    RMS output current               :  0.000000 uA

    Average biput current            :  0.000000 uA
    RMS biput current                :  0.000000 uA

    Average capacitive current       : -148.008493 uA
    RMS capacitive current           :  387.640214 uA

    Average wasted current           : -20.550757 uA
    RMS wasted current               :  77.606651 uA

    Average static wasted current    : -0.331397 uA
    RMS static wasted current        :  0.741029 uA

    Average dynamic wasted current   : -20.219361 uA
    RMS dynamic wasted current       :  77.603113 uA

    Wasted current percentage        :  12.192008%

    Average block power              :  161.912241 uW
    RMS block power                  :  428.747074 uW
```

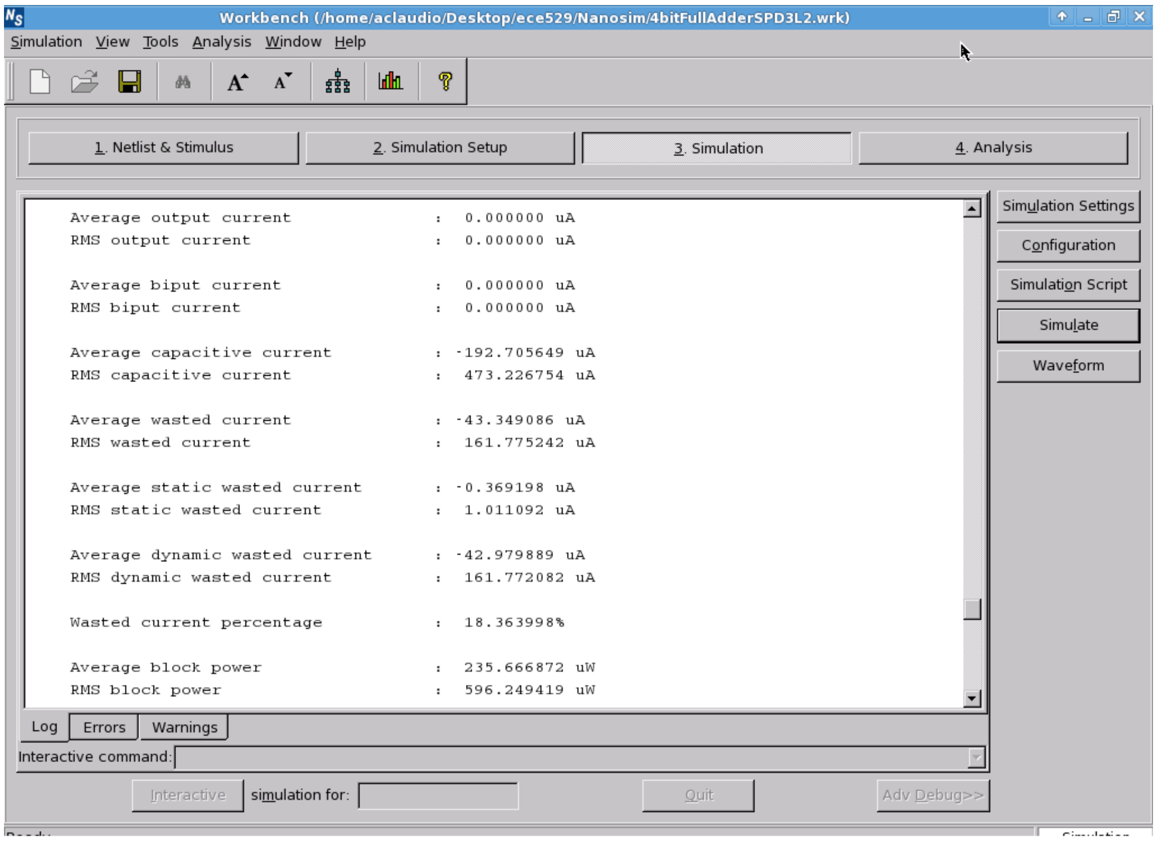
Method 2

D3L Implementation



```
    Average output current           :  0.000000 uA
    RMS output current               :  0.000000 uA

    Average biput current            :  0.000000 uA
    RMS biput current                :  0.000000 uA

    Average capacitive current       : -156.245668 uA
    RMS capacitive current           :  410.272102 uA

    Average wasted current           : -17.697474 uA
    RMS wasted current               :  65.080073 uA

    Average static wasted current    : -0.249465 uA
    RMS static wasted current        :  0.683190 uA

    Average dynamic wasted current   : -17.448009 uA
    RMS dynamic wasted current       :  65.076487 uA

    Wasted current percentage        :  10.174287%

    Average block power              :  166.323130 uW
    RMS block power                  :  434.363862 uW
```

SPD3L Implementation



```
    Average output current              :  0.000000 uA
    RMS output current                  :  0.000000 uA

    Average biput current               :  0.000000 uA
    RMS biput current                   :  0.000000 uA

    Average capacitive current          : -192.705649 uA
    RMS capacitive current              :  473.226754 uA

    Average wasted current              : -43.349086 uA
    RMS wasted current                  :  161.775242 uA

    Average static wasted current       : -0.369198 uA
    RMS static wasted current           :  1.011092 uA

    Average dynamic wasted current      : -42.979889 uA
    RMS dynamic wasted current          :  161.772082 uA

    Wasted current percentage           :  18.363998%

    Average block power                 :  235.666872 uW
    RMS block power                     :  596.249419 uW
```

Method 1

	D3L	SPD3L
Delay S3 (ps)	136.45	136.56
Delay Cout (ps)	193.65	178.39
Average Power Consumption (uW)	1.737	2.366
Max Power Consumption (uW)	1.737	2.366

Method 2

	D3L	SPD3L
Delay S3 (ps)	186.6	168.33
Delay Cout (ps)	118.81	176.73
Average Power Consumption	2.854	3.951
Max Power Consumption	2.854	3.951

Conclusions

First thing to be mentioned is that some of the data returned does not match with theory explain in the background and this be due to the implementation of the gates inv and nand.

Leaving this aside, most of the data, the mayor of it explains the behavior of both the implementations, being the SPD3L faster than the D3L, also we can see that the average static power consumption of the SPD3L is greater than the one of the SPD3L. Also it can be noticed that the second method of implement the full adder is slower, this is due to the way is implemented; in this one the performance of two gates makes the delay greater and also the power consumption.

Although the implementation of the SPD3L includes a increase in terms in area, it's worth because of its speed and power consumption. But in terms of D3L, SPD3L and domino logic power consumption is always greater than the static CMOS implementation. The main aim of the two techniques are to make the clock distribution simpler, qualifying the inputs there is no need of the clock distribution, and between the two, SPD3L is the most optimal selection.

References

[1]Lower split-path data-driven dynamic logic – F.Frustaci, M.Lanuzza, P.Zicari, S. Perri, P.Corsonello

[2]A 16-Bit Barrel-Shifter Implemented in Data-Driven Dynamic Logic (D3L) – Ramin Rafati, Sied Mehdi Fakhraie, Kenneth Carless Smith.