EE5311: Digital IC Design

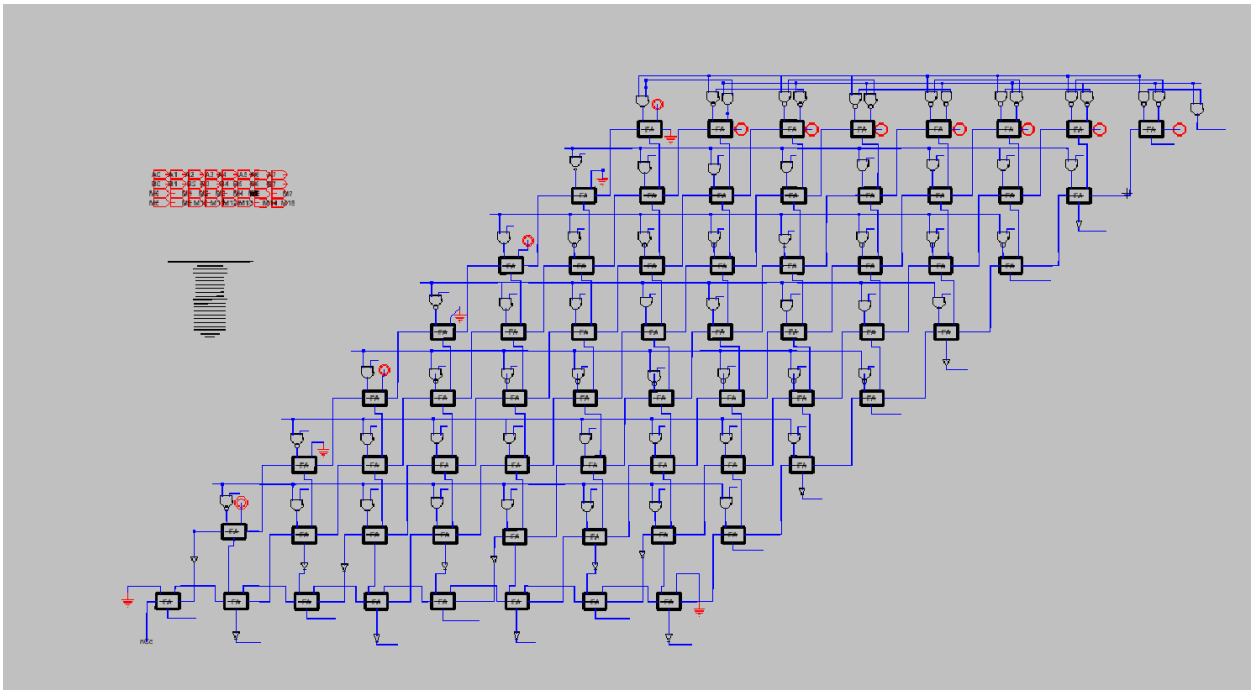
Design of 8-bit signed Carry-Save Multiplier

Group 7

- Rishi Nanda V (EE21B111)
- Gaurika Bindal (EP21B013)

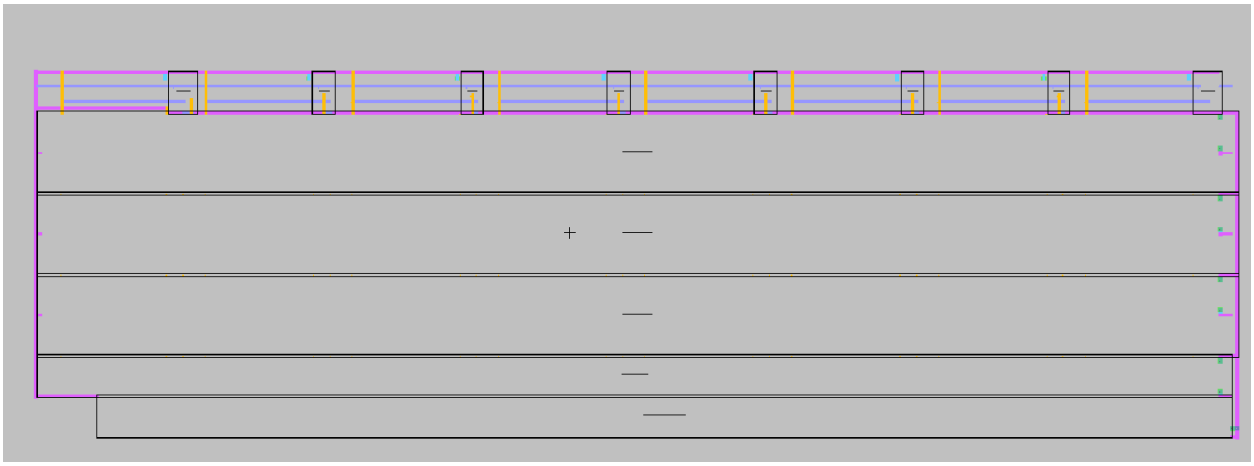
Note: The results and tables presented in this document are available in this spreadsheet.

Schematic of Combinational CSM



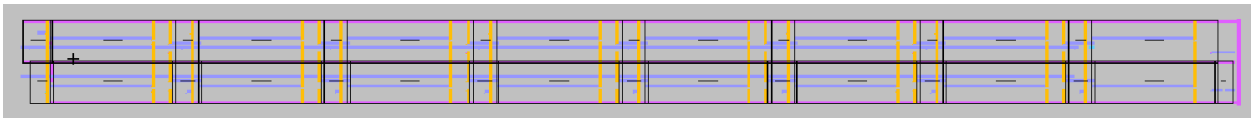
By default, the Full Adders are inverted. Because each alternate row produced inverted bits, the corresponding partial products input in these rows had to be inverted before being fed into the complete adder. All arriving bits require no further inversion before the final Vector Merge state. However, because a Ripple Carry Adder is utilized, each alternate full adder needs an inverted input bit to output the right sum bit. Including NAND gates supplemented the presence of inverting full adders, making it easier to construct an optimal multiplier.

Layout

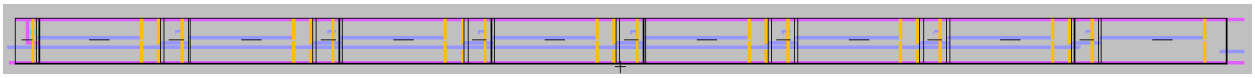


3 sub-layouts were used to make this layout.

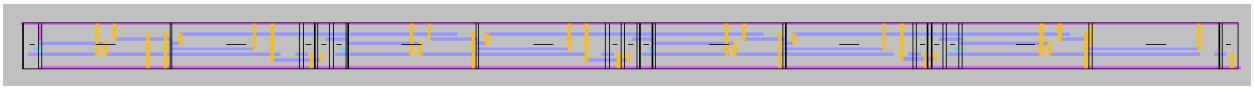
Layout of 1 odd 1 even row



Layout of the last odd row



Layout of the vector merge



This process makes it easier for us to change the vector merge layout easily later if we plan to reduce delay. Any other further optimizations become similarly easy due to the modularity of the layout.

DRC LVS Clean Messages

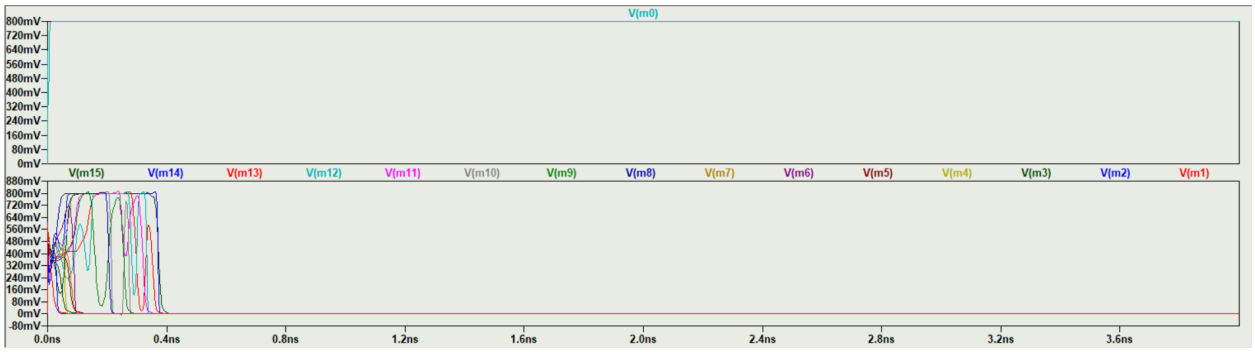
```
==================================15==================================
Running DRC with area bit on, extension bit on, Mosis bit
Checking again hierarchy .... (0.018 secs)
Found 103 networks
0 errors and 0 warnings found (took 0.104 secs)
==================================16==================================
Hierarchical NCC every cell in the design: cell 'CSMfinal:multiplier{sch}'  cell 'CSMfinal:multiplier{lay}'
Comparing: CSMfinal:AND2{sch} with: CSMfinal:AND2{lay}
  exports match, topologies match, sizes match in 0.275 seconds.
Comparing: CSMfinal:FA{sch} with: CSMfinal:FA{lay}
  exports match, topologies match, sizes match in 0.018 seconds.
Comparing: CSMfinal:NAND2{sch} with: CSMfinal:NAND2{lay}
  exports match, topologies match, sizes match in 0.005 seconds.
Comparing: CSMfinal:inv{sch} with: CSMfinal:inv_lay{lay}
  exports match, topologies match, sizes match in 0.004 seconds.
Comparing: CSMfinal:multiplier{sch} with: CSMfinal:multiplier{lay}
  exports match, topologies match, sizes match in 0.056 seconds.
Summary for all cells: exports match, topologies match, sizes match
NCC command completed in: 0.442 seconds.
```

Area

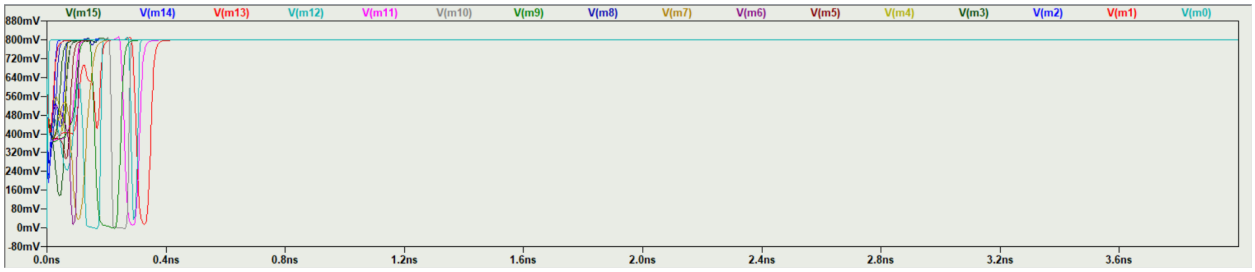
Dimensions:
x=1782.5**λ**
y=545**λ**
Area=1782.5**λ***545**λ**=116.7556μm2

Functionality

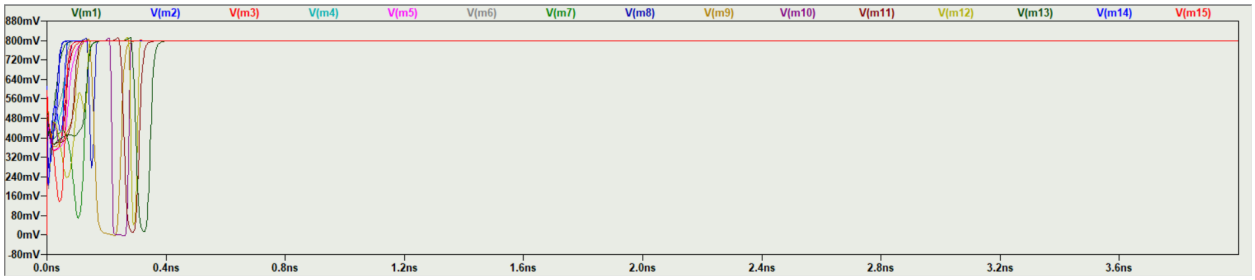
A=b’00000001 B=b’’00000001



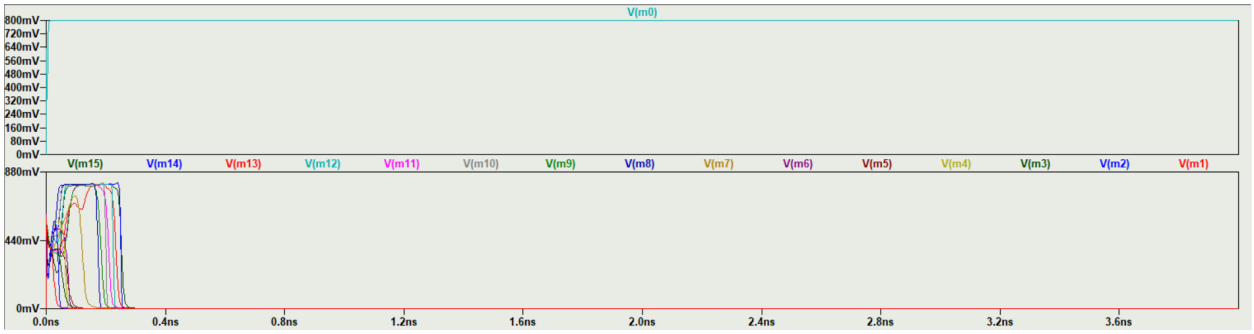
A=b’00000001 B=b’11111111



A=b’11111111 B=b’00000001



A=b’11111111 B=b’11111111

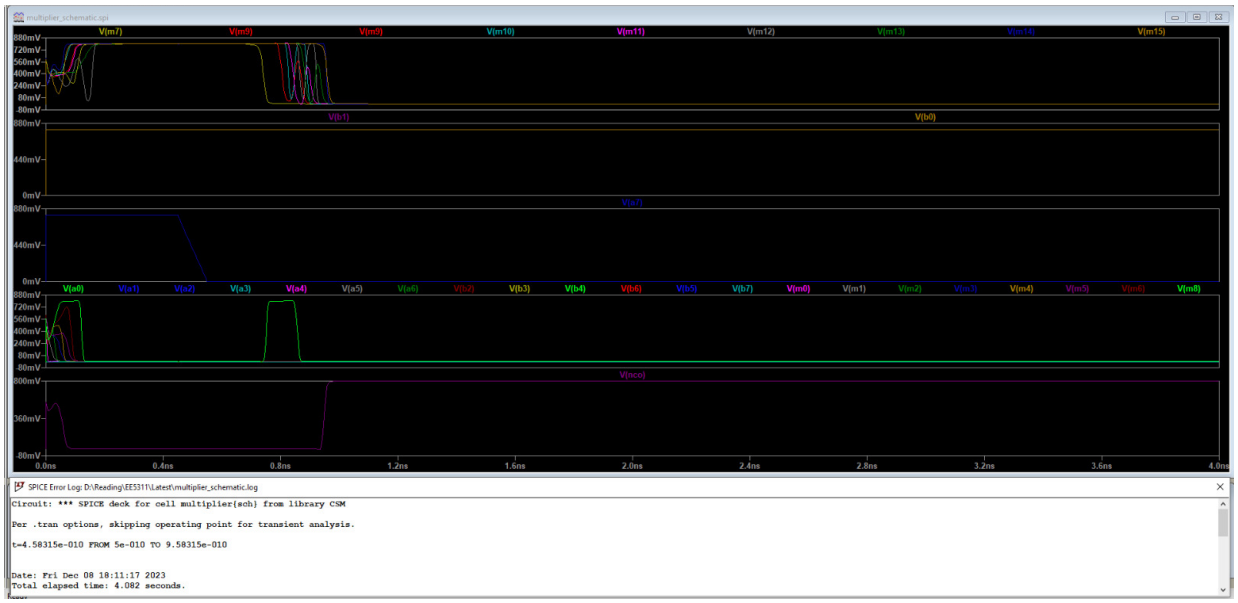


Evaluating worst case delay

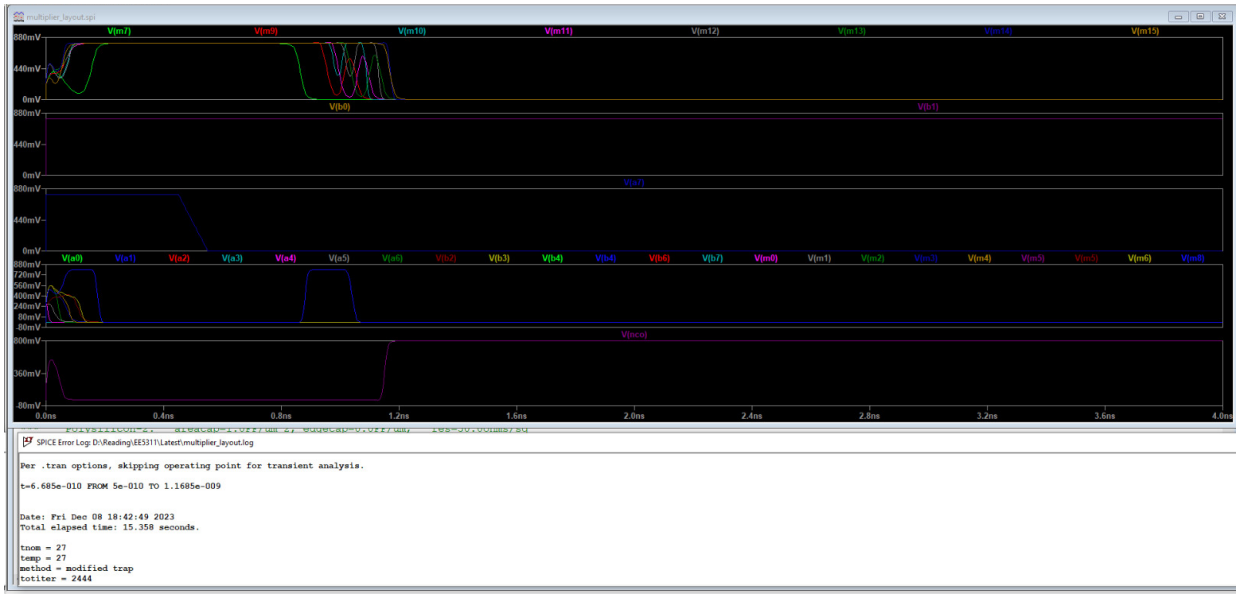
Code:

```
.include "C:\Users\pihoo\Desktop\DIC\22nm_HP.pm"
.param vdd=0.8
v1 vdd gnd DC {vdd}
v10 B0 gnd DC {vdd}
v11 B1 gnd DC {vdd}
v12 B2 gnd DC 0
v13 B3 gnd DC 0
v14 B4 gnd DC 0
v15 B5 gnd DC 0
v16 B6 gnd DC 0
v17 B7 gnd DC 0
v2 A0 gnd DC 0
v3 A1 gnd DC 0
v4 A2 gnd DC 0
v5 A3 gnd DC 0
v6 A4 gnd DC 0
v7 A5 gnd DC 0
v8 A6 gnd DC 0
v9 A7 gnd PWL(0 {vdd} 450p {vdd} 550p 0 1n 0)
.meas tran t
+trig V(a7)=0.4 cross=last
+targ V(m15)=0.4 cross=LAST
.tran 4n uic
.END
```

Based on Schematic



Based on Layout

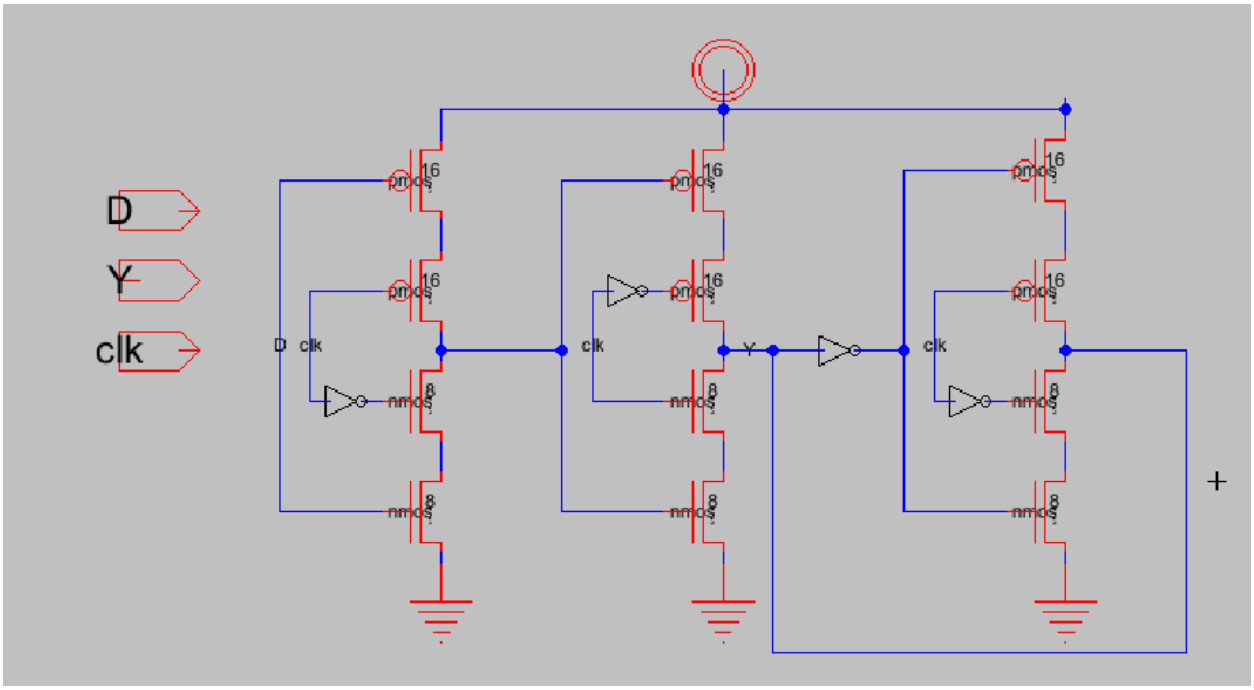


Schematic	4.58315E-10
Layout RC Extracted	6.685e-10

There is a 1.46x increase in delay upon using the RC extracted delay compared to the schematic Delay.

Pipelining

The first thing needed for this is a flip-flop. We made a dynamic C2MOS D flip-flop for this purpose. The right-most clocked inverter is used to ensure that Y does not deviate during the clock low state.



Flip-Flop Characterization

Rise Delay Code:

```
.include "C:\Users\pihoo\Desktop\DIC\22nm_HP.pm"
.param vdd=0.8
.STEP param tskew 350p 400p 0.1p
v1 vdd gnd DC {vdd}
v2 c gnd PULSE(0 {vdd} 400p 100p 100p 400p 1n 3)
v3 in gnd PWL(0 {vdd} 350p {vdd} 450p 0 {tskew+1n} 0 {1.1n+tskew}
{vdd})
.meas tran tdc
+ trig v(input) = ({vdd}/2) cross=2
+ targ v(clk) = ({vdd}/2) cross=3
.meas tran tcq
+ trig v(clk) = ({vdd}/2) cross=3
+ targ v(out) = ({vdd}/2) cross=2
.meas tran tdq
+ trig v(input) = ({vdd}/2) cross=2
+ targ v(out) = ({vdd}/2) cross=2
.tran 2n
.END
```

Fall Delay Code:

```
.include "C:\Users\pihoo\Desktop\DIC\22nm_HP.pm"
.param vdd=0.8
.STEP param tskew 350p 400p 0.1p
v1 vdd gnd DC {vdd}
v2 c gnd PULSE(0 {vdd} 400p 100p 100p 400p 1n 3)
v3 in gnd PWL(0 {vdd} {tskew} {vdd} {100p+tskew} 0 1n 0)
.meas tran tdc
+ trig v(input) = ({vdd}/2) cross=2
+ targ v(clk) = ({vdd}/2) cross=3
.meas tran tcq
+ trig v(clk) = ({vdd}/2) cross=3
+ targ v(out) = ({vdd}/2) cross=2
.meas tran tdq
+ trig v(input) = ({vdd}/2) cross=2
+ targ v(out) = ({vdd}/2) cross=2
.tran 2n
.END
```

The number of crosses is dependent on the stabilization of the inputs and outputs. So, that had to be observed and edited to get the correct delay values. These values were calculated for 1x, 2x and 3x-sized flip-flops. The results are tabulated below:

T_setup	1x	2x	3x
FALL	1.70E-11	1.90E-11	2.08E-11
RISE	8.76E-12	1.04E-11	1.14E-11

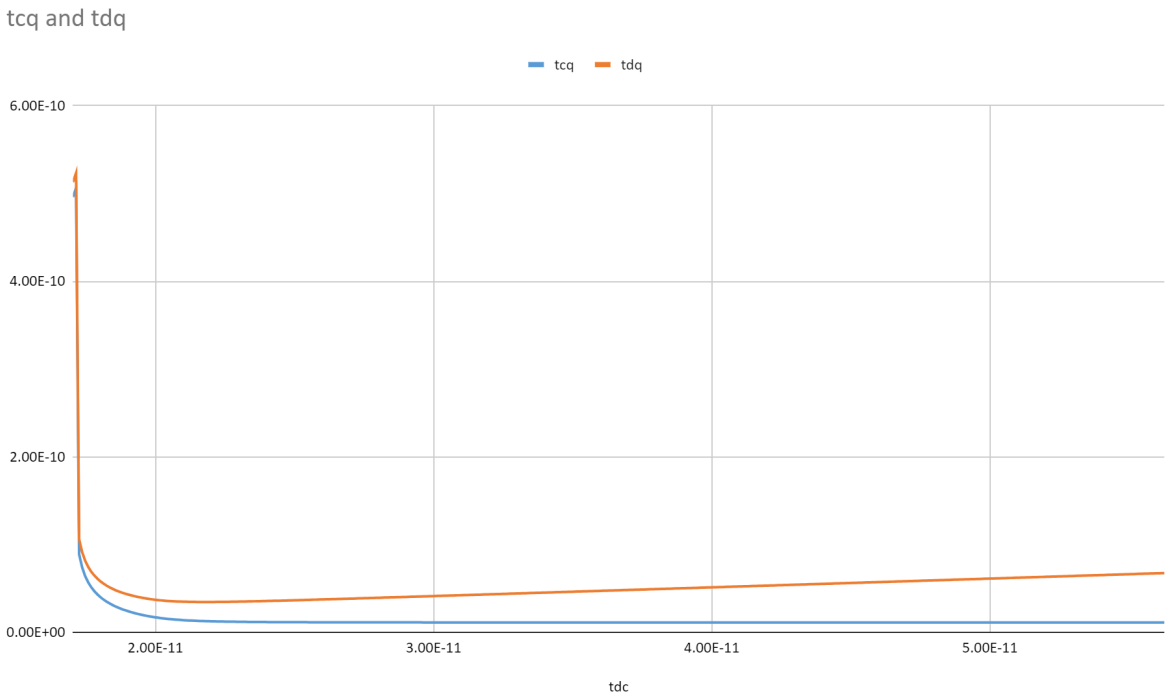
T_pcq	1x	2x	3x
FALL	1.13E-11	1.10E-11	1.15E-11
RISE	1.41E-11	1.67E-11	1.67E-11

T_dq	1x	2x	3x
FALL	3.46E-11	3.63E-11	3.89E-11
RISE	2.94E-11	3.46E-11	3.94E-11

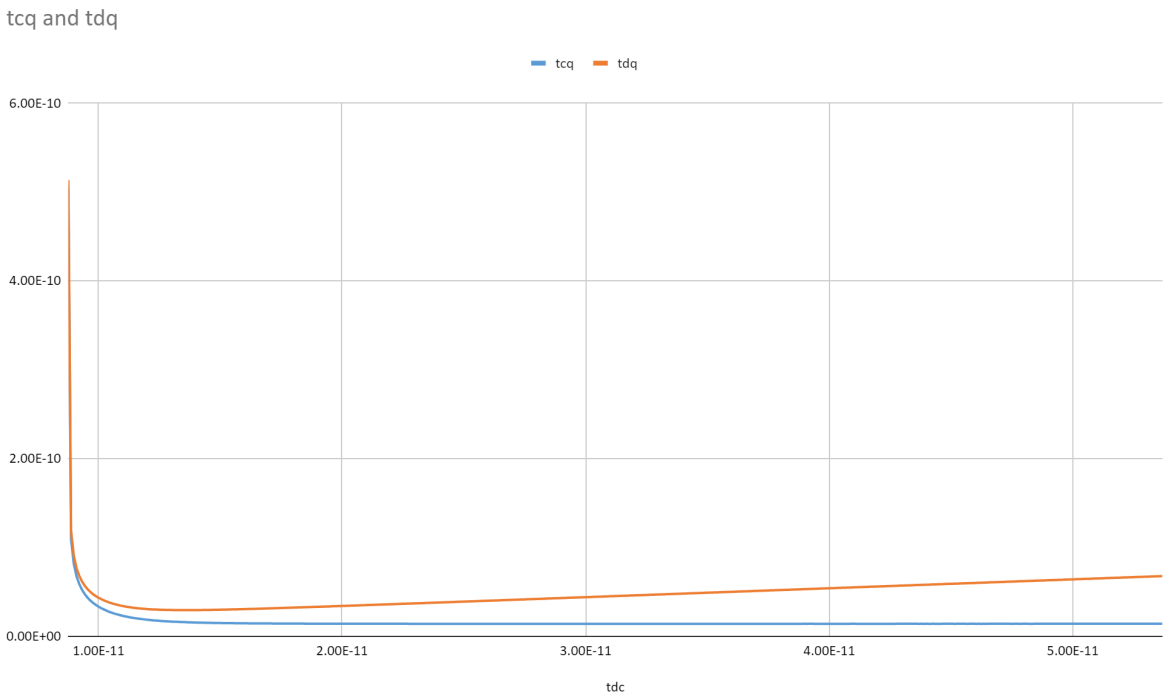
As we can see the minimum delay is seen for the 1x flip-flop. Hence, it was chosen for further use.

T_dq	3.46E-11
T_setup	1.70E-11
T_pcq	1.41E-11

The characterization graphs for the 1x flip-flop:
Fall:

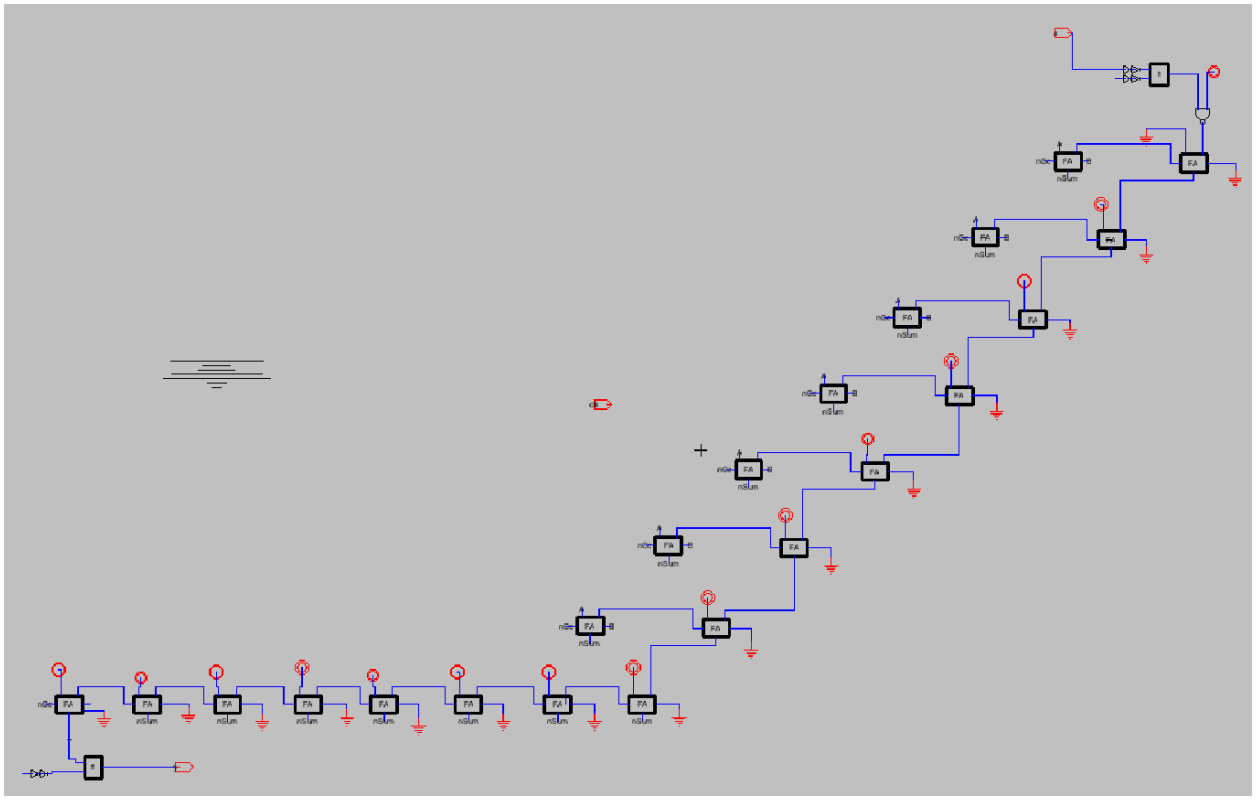


Rise:



Unpipelined CSM

For this, we needed to launch flops at the inputs and capture flops at the output. The time of the combinational circuit was taken and then we calculated a Theoretical Tclk with the flip-flop delays. We simulated both for the schematic as well as for the block RC-extracted schematic. Additionally, these simulation were run on the Critical Path Model.



Schematic:

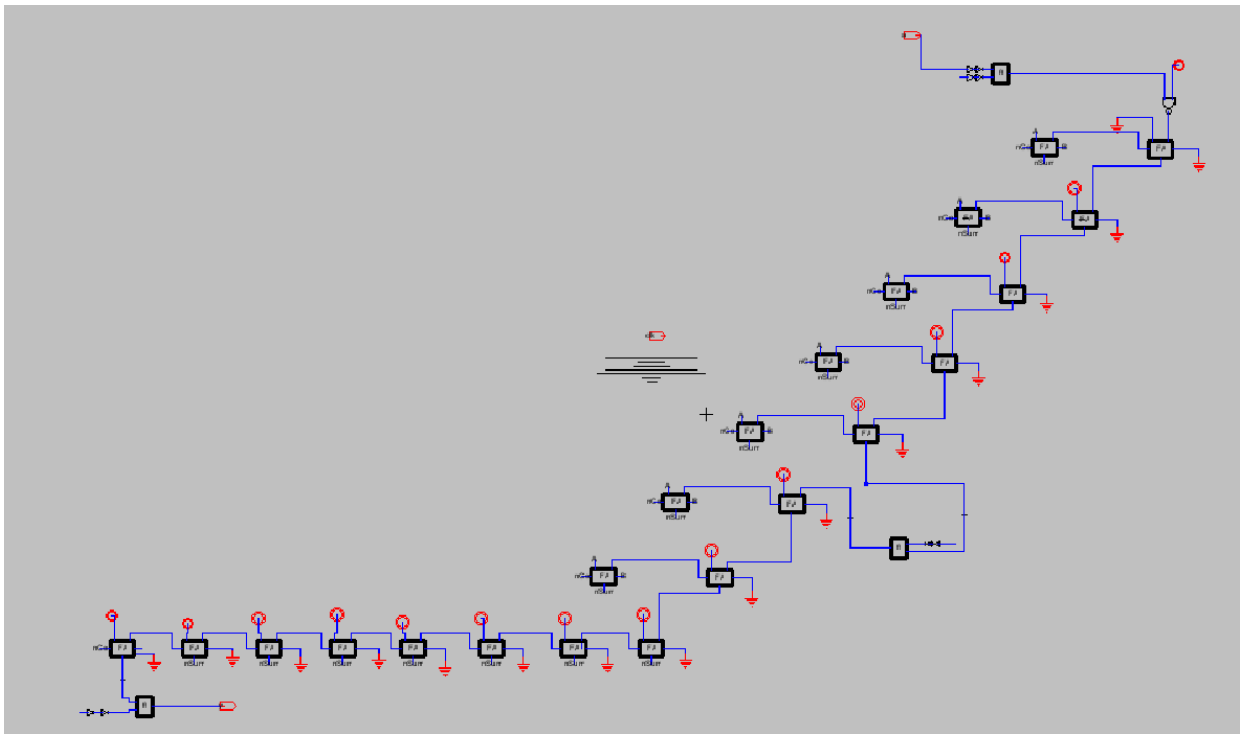
Tcomb	3.72E-10
Theoretical Minimum Tclk	4.07E-10
Obtained Minimum Tclk	4.16E-10

Block RC-Extracted:

Tcomb	5.29E-10
Theoretical Minimum Tclk	5.64E-10
Obtained Minimum Tclk	5.62E-10

Pipelined CSM

For this we first need to analyse at what point in the circuit we were getting an almost equal delay before and after it. For this we evaluated for several points in the system and finalised 2 full adders before the ripple carry adder stage as the most optimum point.The combinational delays were found for both schematic and for RC-extracted. Then an additional flop was inserted at this point to increase our cock frequency as well as the rate of our data sampling.



Combinational Delay:

	Schematic	Block RC-extracted
Td1	1.91E-10	2.76E-10
Td2	1.81E-10	2.54E-10

Schematic:

Tcomb	1.91E-10
Theoretical Minimum Tclk	2.26E-10
Obtained Minimum Tclk	2.29E-10

Block RC-Extracted:

Tcomb	2.76E-10
Theoretical Minimum Tclk	3.10E-10
Obtained Minimum Tclk	3.03E-10

The pipelined and unpipelined values were compared for both the setups and the increase in frequency was observed.

Schematic:

Theoretical Pipeline Gain	1.80E+00
Obtained Pipeline Gain	1.82E+00
Deviation from theoretical	0.82%

Block RC-Extracted:

Theoretical Pipeline Gain	1.82E+00
Obtained Pipeline Gain	1.85E+00
Deviation from theoretical	2.00%

From the above results we can see that we get an 1.82x increase in our frequency for the schematic based circuit and a 1.85x increase for our block rc-extracted circuit.

The final frequencies are

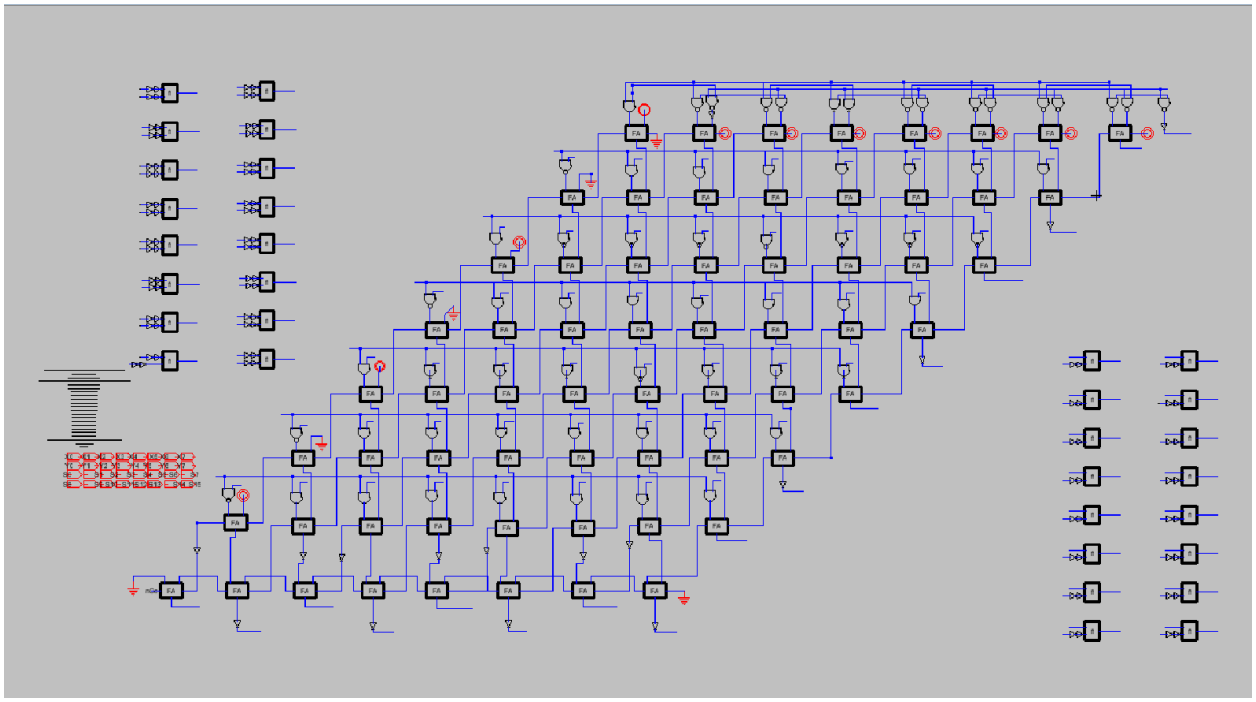
	Schematic	RC
unpipelined	2.40E+09	1.78E+09
pipelined	4.37E+09	3.30E+09

Comparing Schematic values to Block RC-extracted values:

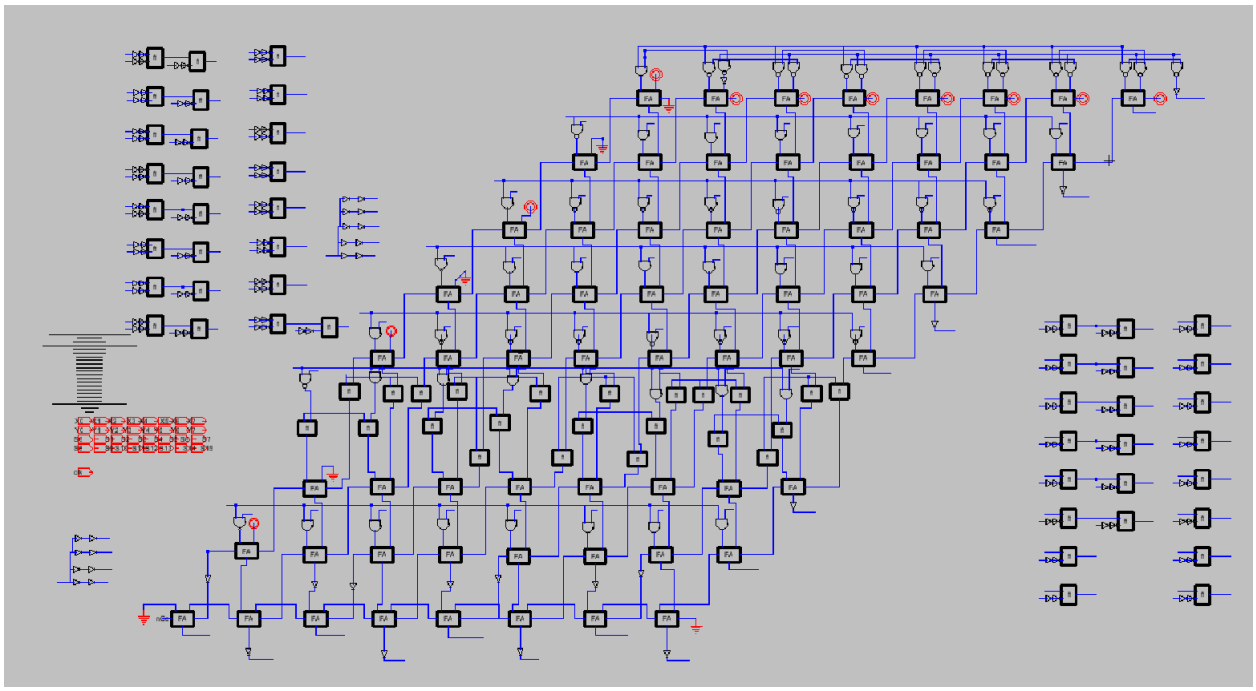
Unpipelined		Pipelined	
Tcomb	1.42E+00	Tcomb	1.44E+00
Theoretical Min. Tclk	1.39E+00	Theoretical Min. Tclk	1.37E+00
Obtained Min. Tclk	1.35E+00	Obtained Min. Tclk	1.32E+00

Full Schematics with Pipelinig

Unpipelined:



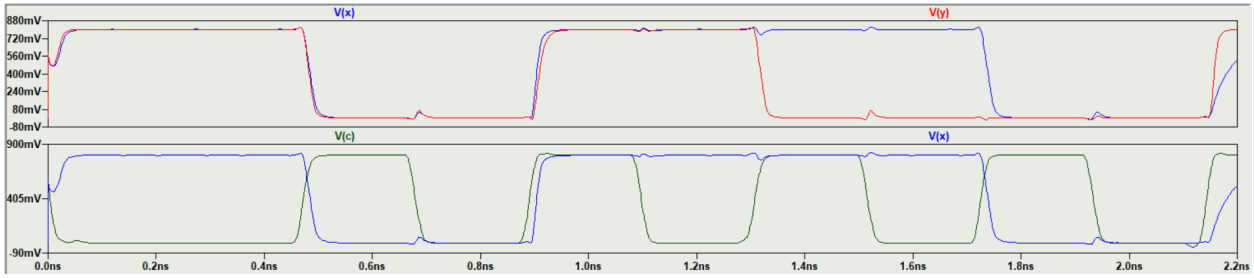
Pipelined:



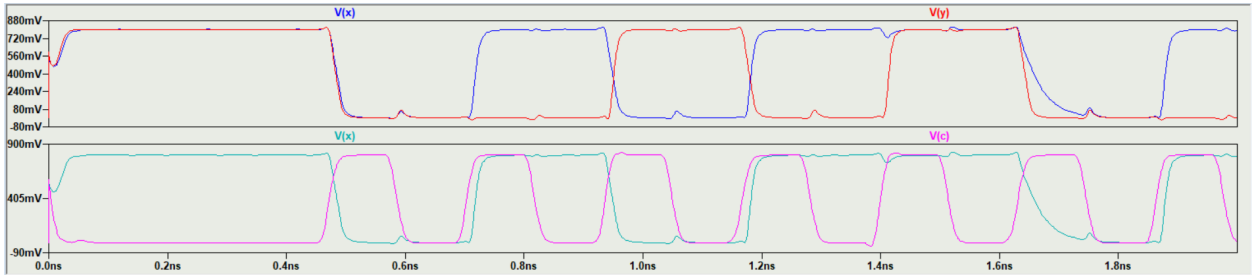
Spice Plots:

Schematic:

Unpipelined:

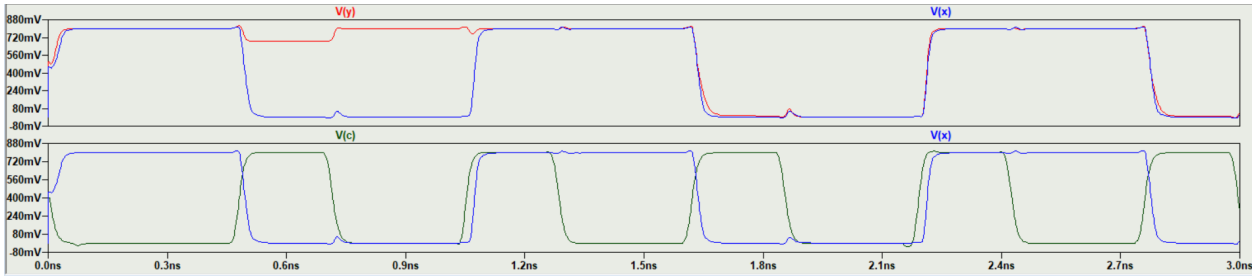


Pipelined:

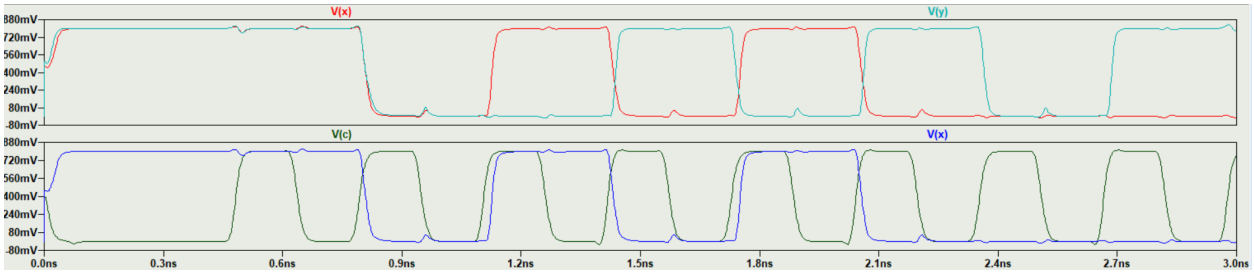


Block RC-extracted:

Unpipelined:

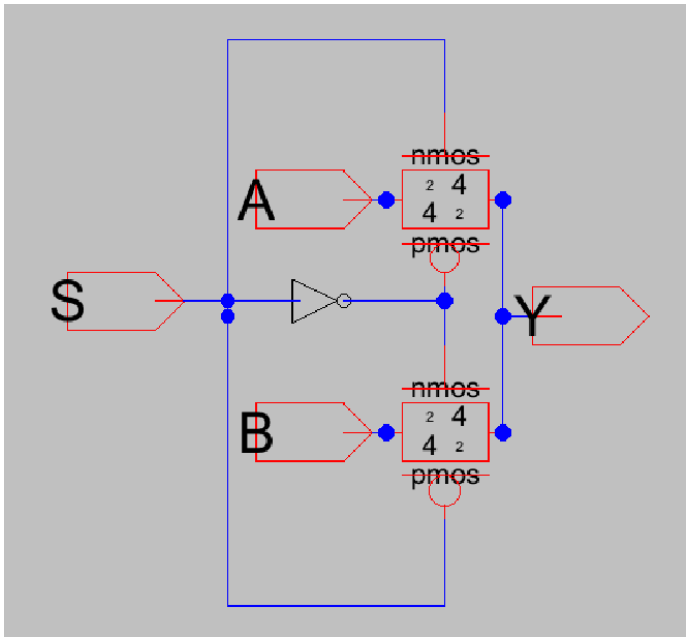


Pipelined:

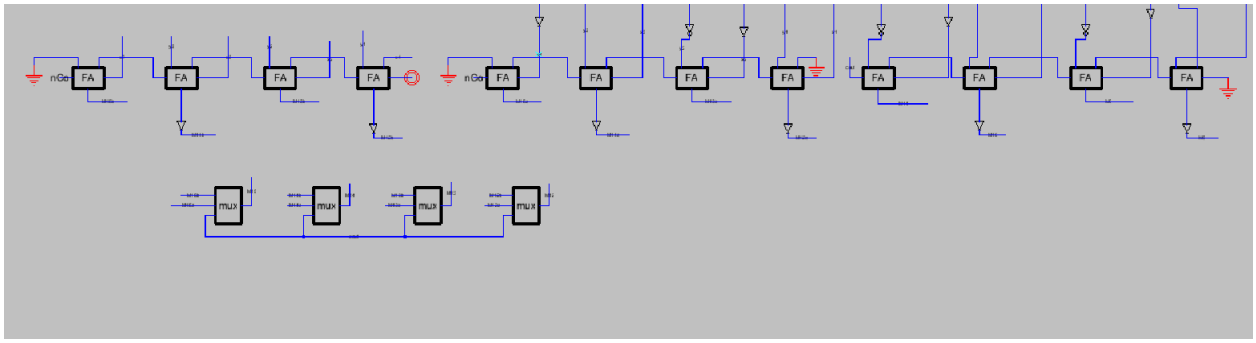


Alternate Vector Merge

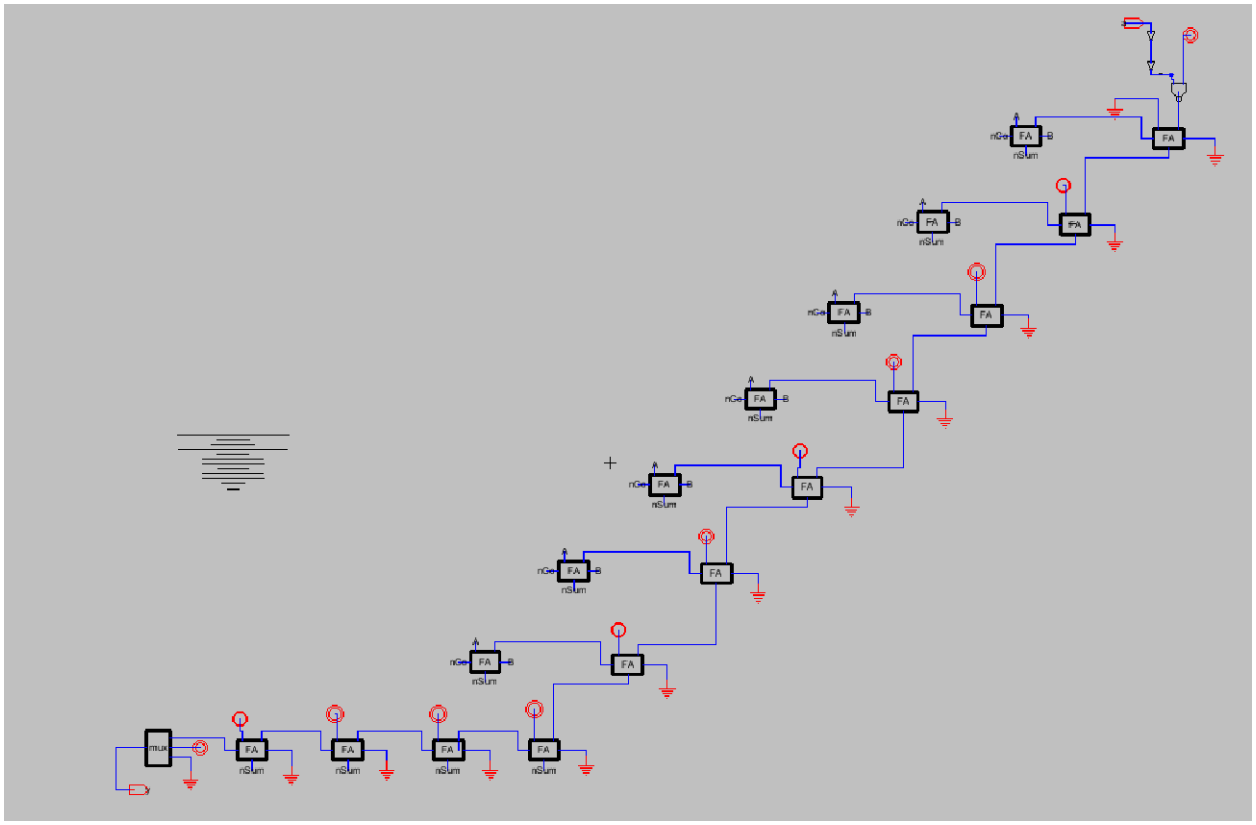
We chose Carry Select Adder. This was because we need an 8-bit adder and so even though the area increases it is not significant as the delay is reduced by a substantial amount. We needed a 2:! Multiplexer for this adder. This was designed as:



The final vector merge stage:



To analyse the exact delay we again used the critical path model.



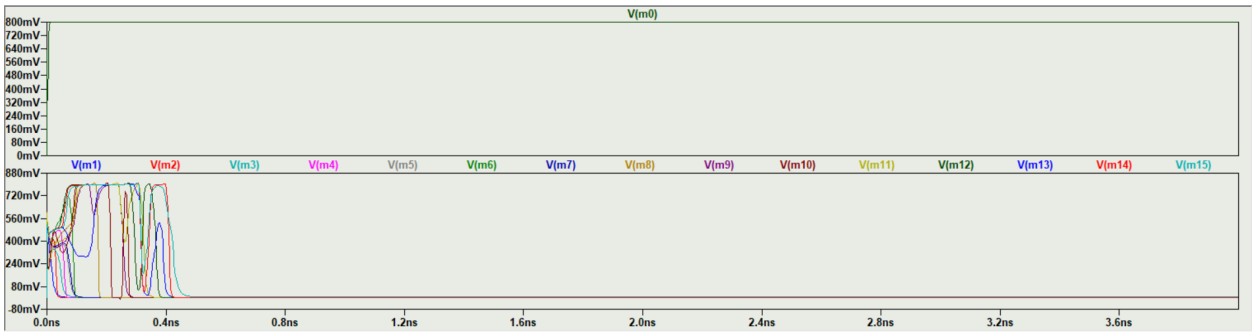
Delay:

	RCA	CSeA	Gain
RISE	3.72E-10	3.19E-10	1.17
FALL	3.56E-10	3.04E-10	1.17

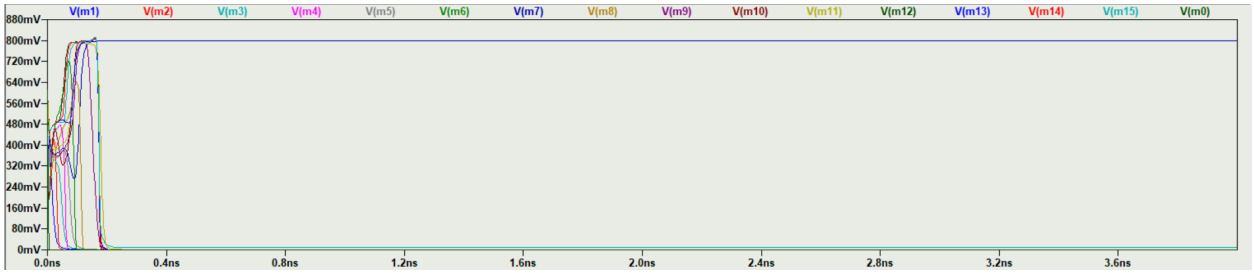
A		B		S		Ripple Carry VM Adder	Carry Select VM Adder
Decimal	Binary	Decimal	Binary	Decimal	Binary		
1	00000001	1	00000001	1	00000000 00000001	3.72E-10	4.21E-10
1	00000001	-1	11111111	-1	11111111 11111111	3.49E-10	1.73E-10
-1	11111111	1	00000001	-1	11111111 11111111	3.83E-10	3.75E-10
-1	11111111	-1	11111111	1	00000000 00000001	2.52E-10	1.67E-10

Functionality

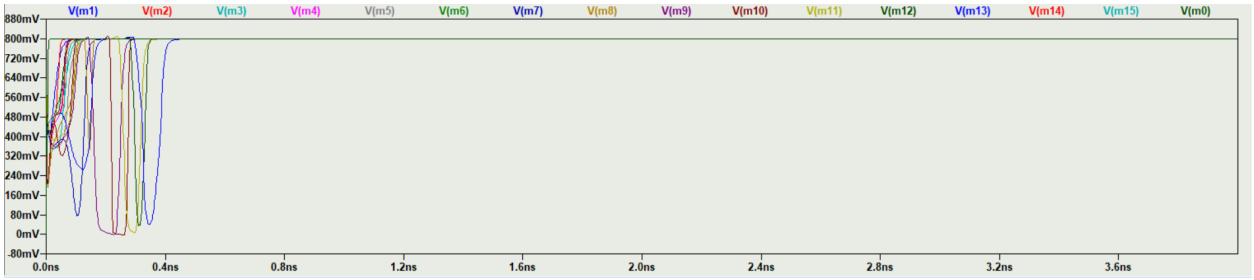
A=b’00000001 B=b’’00000001



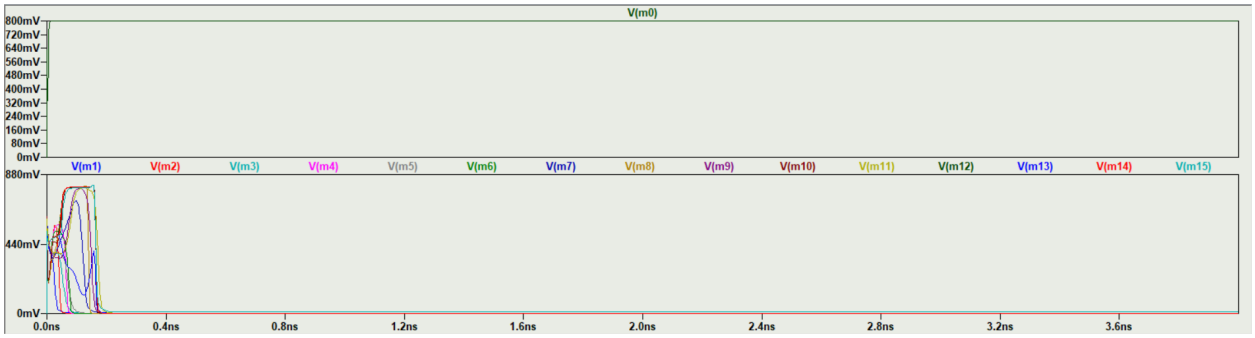
A=b’00000001 B=b’11111111



A=b’11111111 B=b’00000001



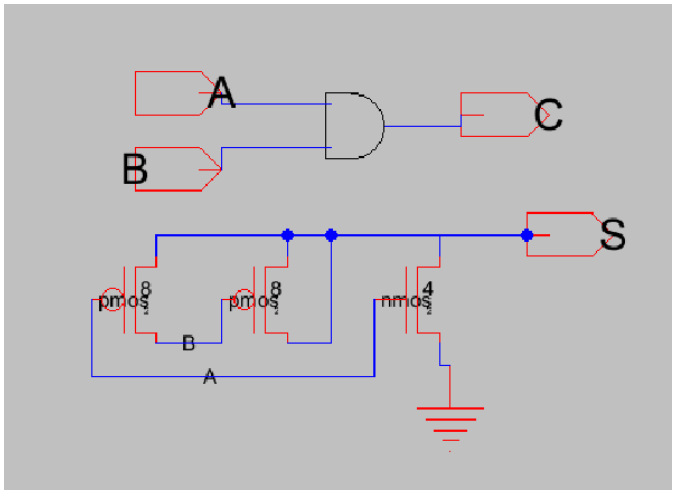
A=b’11111111 B=b’11111111



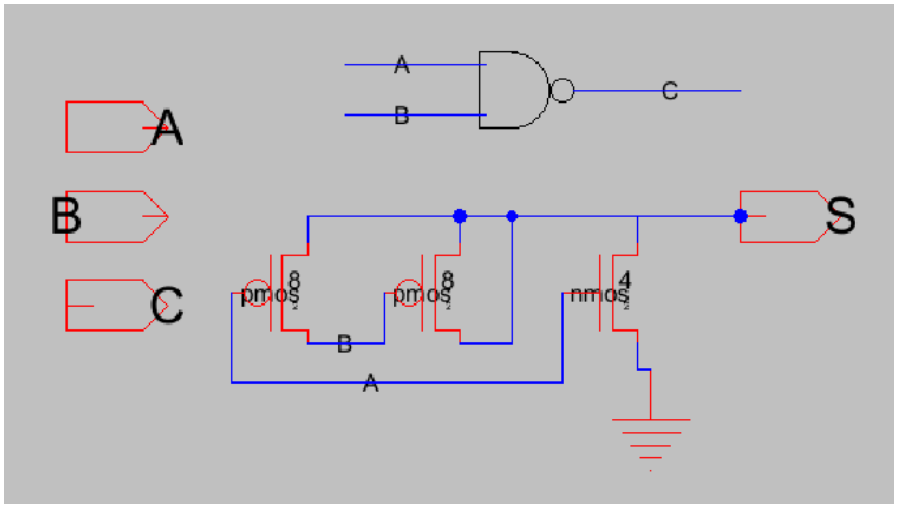
Further optimizations

We added 2 half adders in the path in place of full adders that were only adding 2 bits. We had to use 2 types of half adders. The half adders in the first row take in positive bits and send out positive sum and carry bits. For this, we also had to switch all the NANDs with Ands in the first row. As the second row expects a positive bit the rest is unaffected. The second half adder is used at the first adder of the vector merge stage This needs to propagate a negated carry bit for the rest of the circuit to be unaffected.

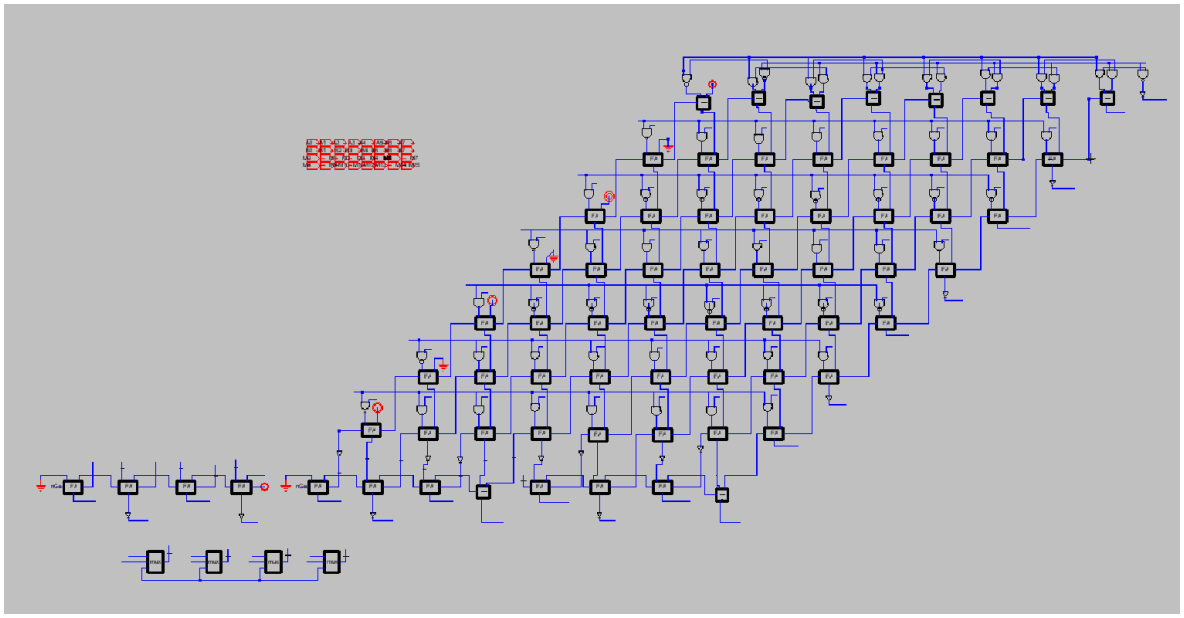
Half adder 1:



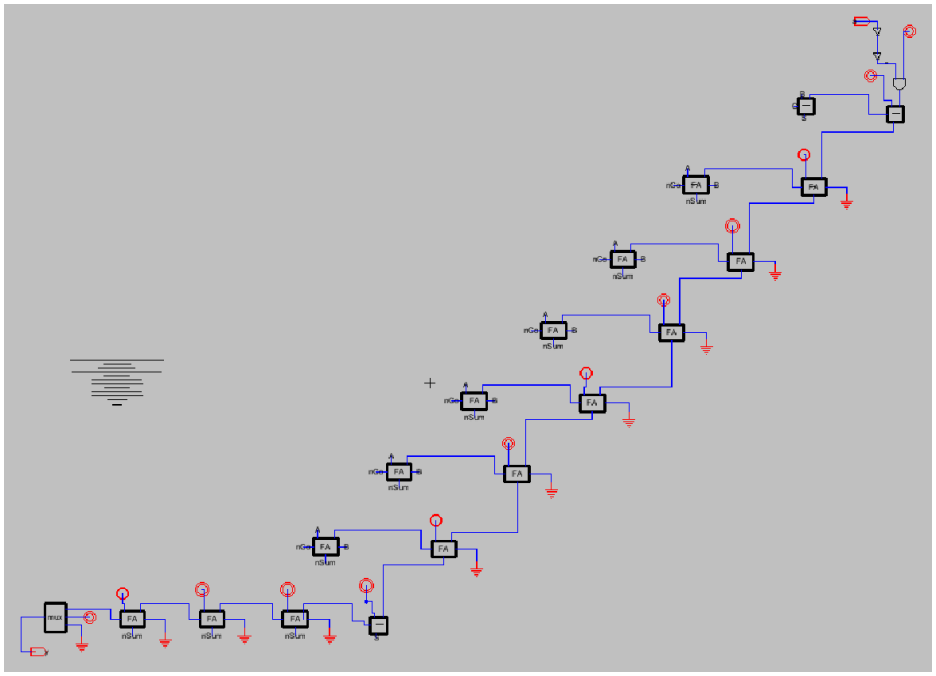
Half adder 2:



The modified Schematic:



To evaluate the delay again we used the critical path model.



Delay:

	RCA	CSeA	Modified	Gain
RISE	3.72E-10	3.19E-10	3.15e-10	1.18
FALL	3.56E-10	3.04E-10	2.95e-10	1.21