EE5311: Digital IC Design, Jul - Nov 2022

Final Project, 8-bit Carry Save Multiplier Design

Group Members: Arun D A ee19b071 , Shashank Nag ee19b118 , Sai Gautham Ravipati ee19b053

Problem Statement:

Design a signed 8 bit carry save multiplier with a single stage pipeline. Show that the frequency of operation can be doubled and data can be fed to the multiplier at twice the rate through pipelining.

Results Summary:

Specifications	
Design Configuration	Carry - 3x, Sum - 1x, NAND/INV - 2x
CSM Layout Area	201.42386 μm^2
Non-pipelined Max Clock Frequency (Schematic)	2.27GHz
Pipelined Max Clock Frequency (Schematic)	3.63GHz
Pipeline Speedup (Schematic)	1.6x
Non-pipelined Max Clock Frequency (RC extracted netlist)	1.85GHz
Pipelined Max Clock Frequency (RC extracted netlist)	2.898GHz
Pipeline Speedup (RC extracted netlist)	1.56x
No. of Test Cases	13

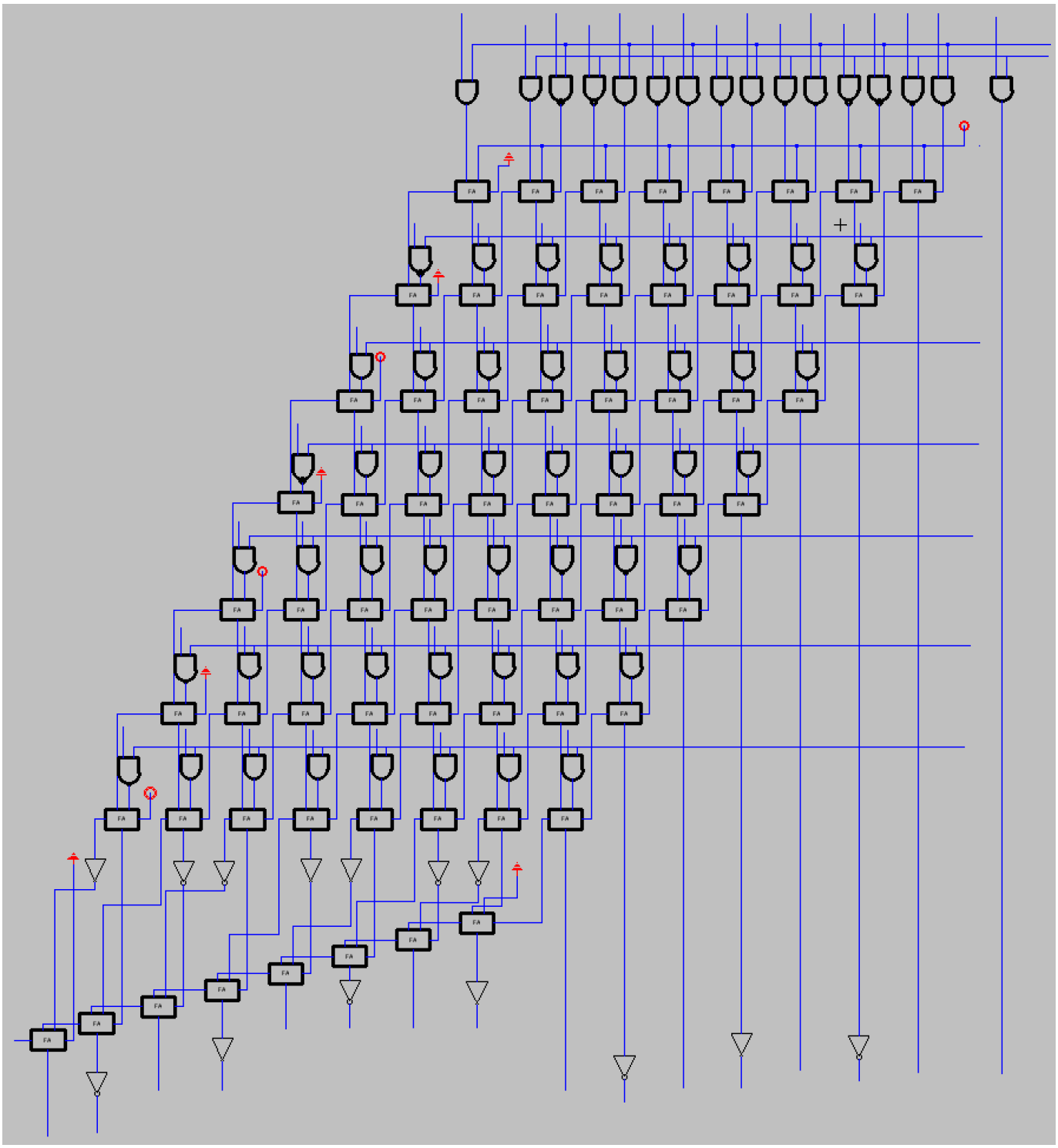
*All clock values are reported with input and output FFs

Design Configuration Choice:

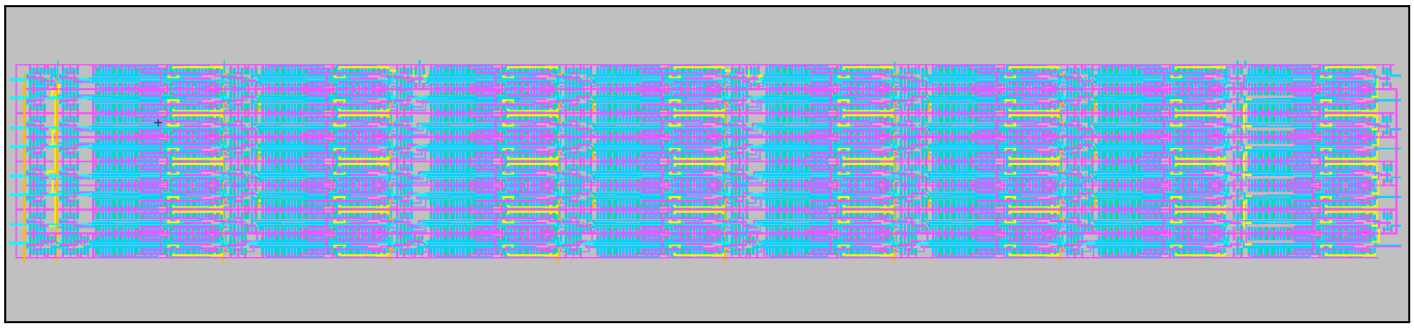
The critical path model of the CSM was analyzed to study the impact of the component size choices on the delay and area. Upon looking at the delay alone, the (carry - 3x , sum - 2x , nand/inv - 2x) configuration was found to be the optimal one. However, upon a closer observation, the gain in delay was not significant in comparison to the case of (carry - 3x, sum - 1x, nand/inv - 2x), while there was a high tradeoff in terms of the area. Hence, the latter configuration has been chosen for the design in this project.

Unpipelined Carry Save Multiplier:

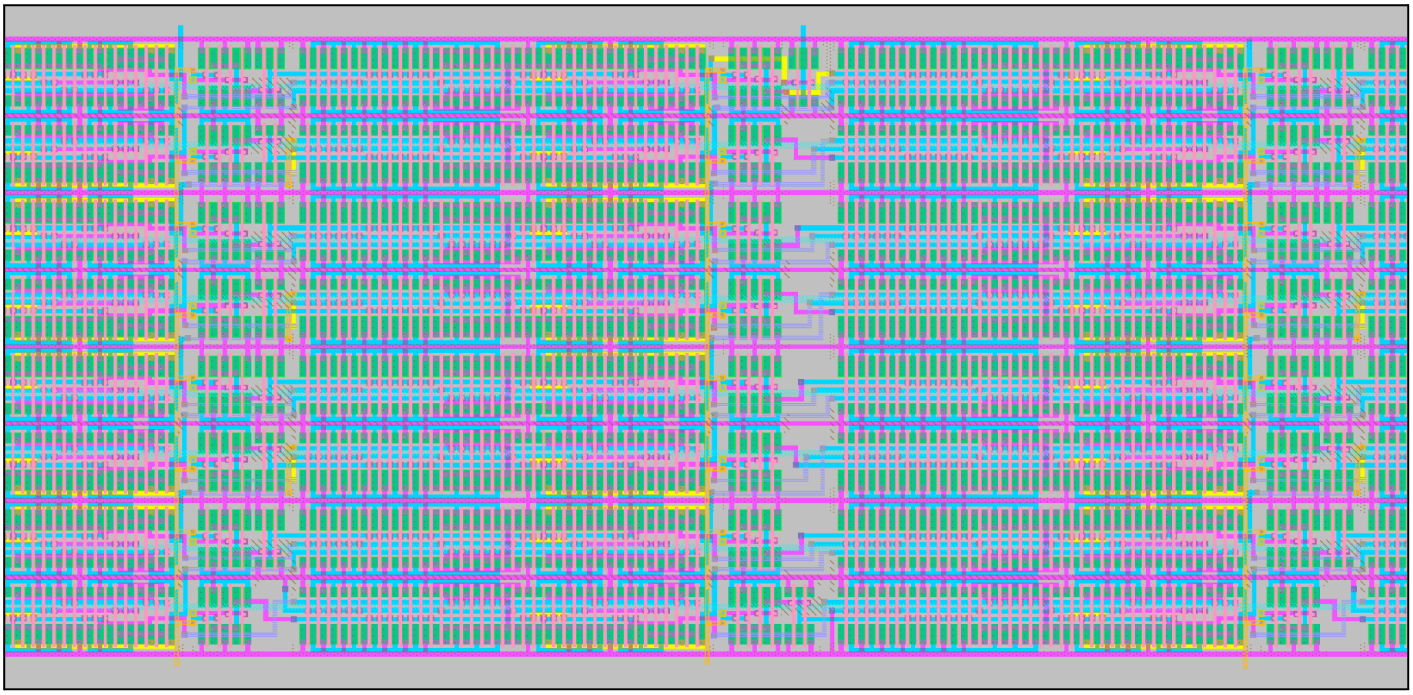
- **Carry Save Multiplier - Schematic:**



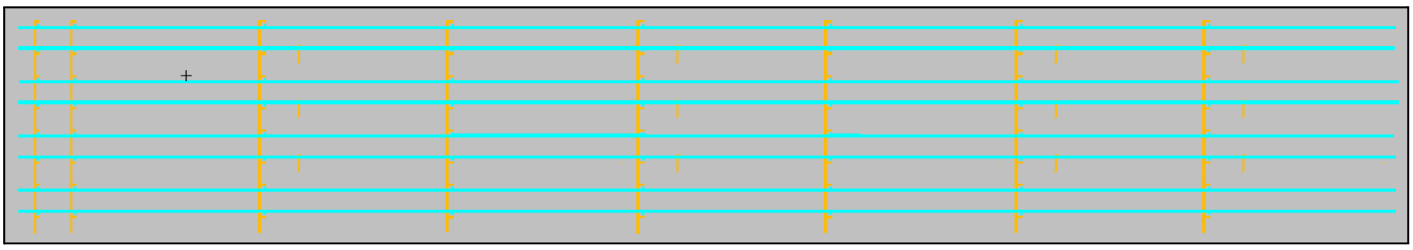
- **Carry Save Multiplier - Layout:**



Completete view of the layout



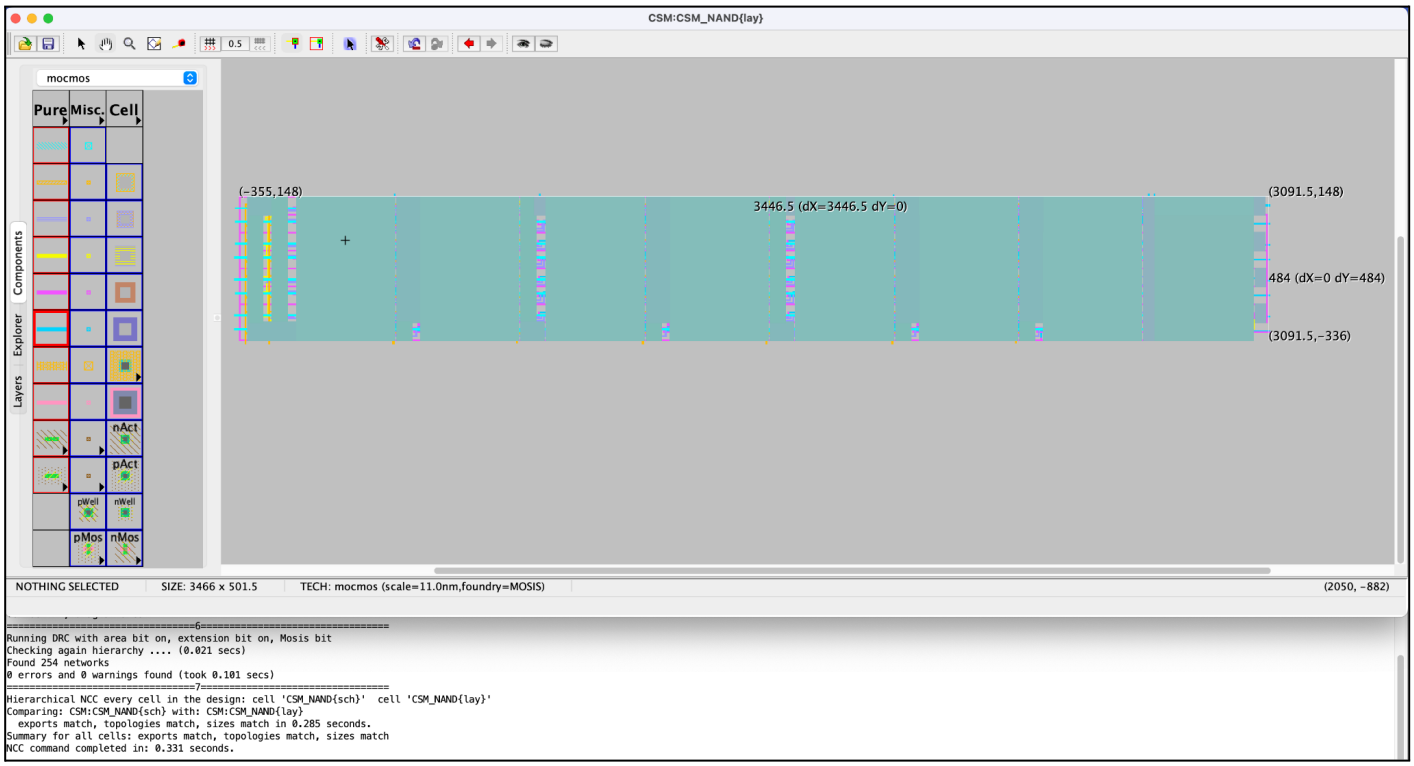
Closer view of the layout



Routing of inputs

- DRC, LVS Results and layout area:

- DRC - Pass, LVS - Pass (Appended in the image)
- Area of the layout: 3446.5λ * 483λ = 16,64,660λ^2 = 201.42386 μm^2



Layout area, DRC and LVS

- Delay results and Functional Tests:

X		Y		Result		Layout RC extracted delay (ps)	Cell RC extracted delay (ps)	Schematic Delay (ps)	Deviation factor
Decimal	Binary	Decimal	Binary	Decimal	Binary				
-1	11111111	-1	11111111	1	0000000000000001	522	503	396	1.32
1	00000001	-1	11111111	-1	1111111111111111	274	270	219	1.25
-1	11111111	1	00000001	-1	1111111111111111	492	478	361	1.36
-1	11111111	127	01111111	-127	1111111110000001	527	511	404	1.3
-128	10000000	-1	11111111	128	0000000010000000	338	324	242	1.4
127	01111111	127	01111111	16129	0011111100000001	530	514	412	1.29
-127	10000001	-127	10000001	16129	0011111100000001	320	309	229	1.4
127	01111111	-1	11111111	-127	1111111110000001	474	463	383	1.24
69	01000101	42	00101010	2898	0000101101010010	384	372	277	1.39
52	00110100	21	00010101	1092	0000010001000100	307	298	233	1.32
-11	11110101	-11	11110101	121	0000000001111001	513	504	382	1.34
5	00000101	36	00100100	180	0000000010110100	185	174	142	1.3
10	00001010	-128	10000000	-1280	1111101100000000	216	204	162	1.33

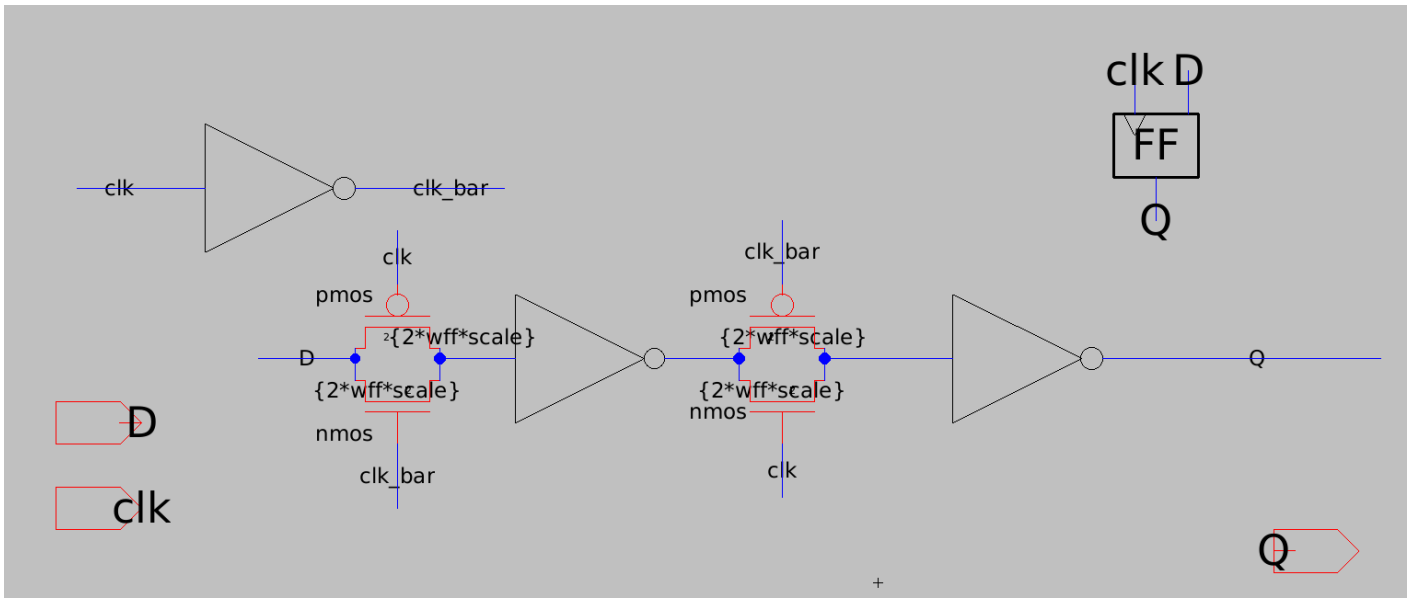
- In the layout, the input X provided from the left, input Y from the bottom, while output Z are taken from top and right. Metal-2 is used to make Vdd and gnd connections.
- Metal-4, Metal-5 and Metal-6 always run in a fixed direction while Metal-3 runs in both directions.
- Results of functional verification (all cases passed), have been provided at the end of the report.
- All adders used are inverting to optimize for delay, while the inputs fed are made inverting alternatively, given the fact that inverted inputs to inverting adder gives non-inverted outputs. For the first stage,

inputs to adders are inverted, so output is correct, so AND gated gates are used to feed non-inverted inputs and the outputs are inverted, resulting in usage of NAND gates for the next stage. For the vector-merge a similar task is performed for alternate adders using NOT gates.

Pipelining

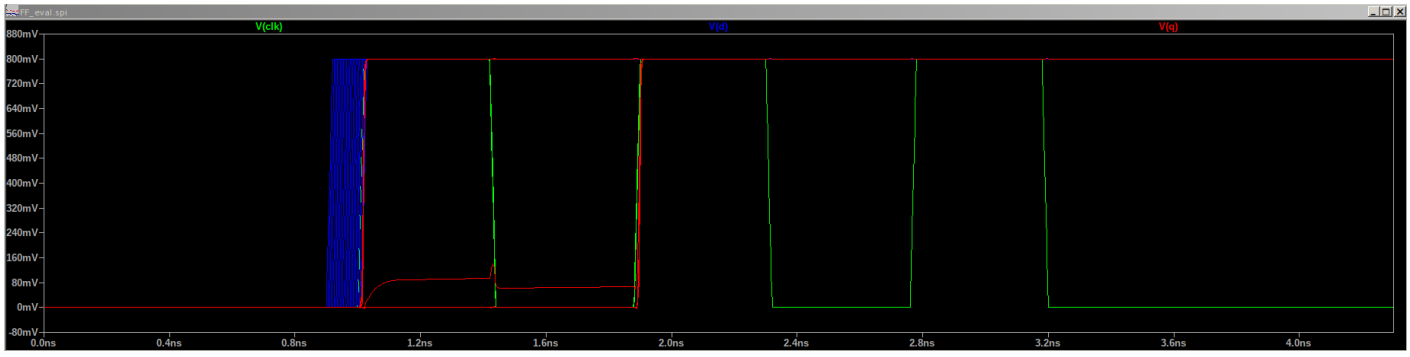
Designing & Characterizing a Flip Flop

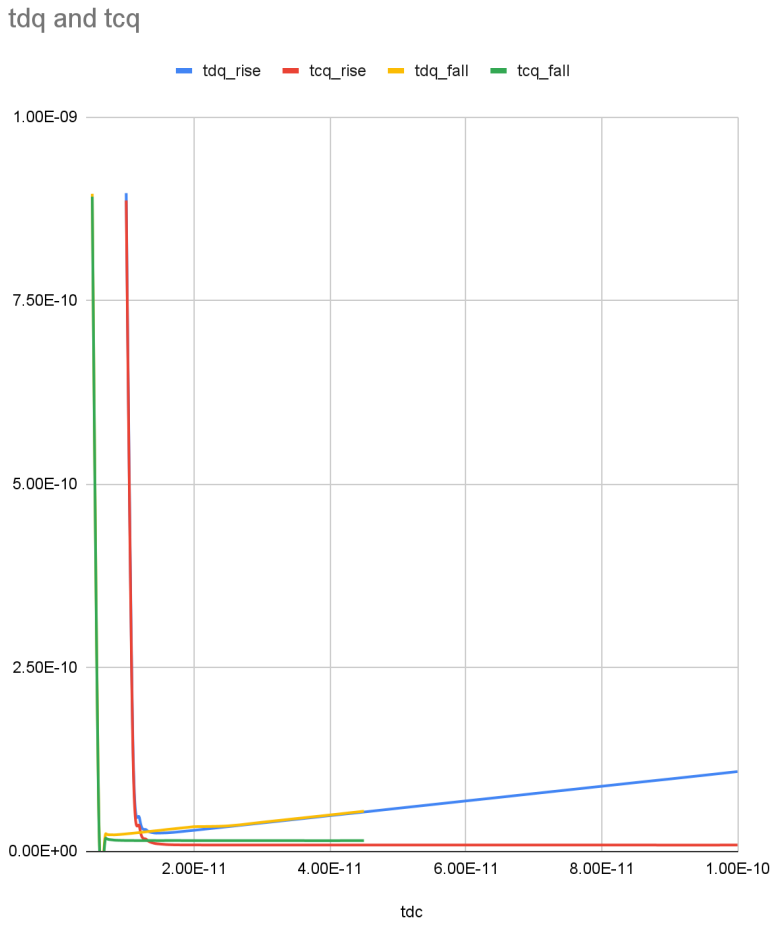
A transmission gate based D-Flip Flop was designed to be used for pipelining the CSM. Different combinations of gate sizes for the FFs were analysed, and the size corresponding to the lowest propagation delay was chosen.



This corresponds to wff = 8, in the above figure.

With the chosen size, to characterise the FF for setup time, the *tdq* and *tcq* were plotted against *tdc*

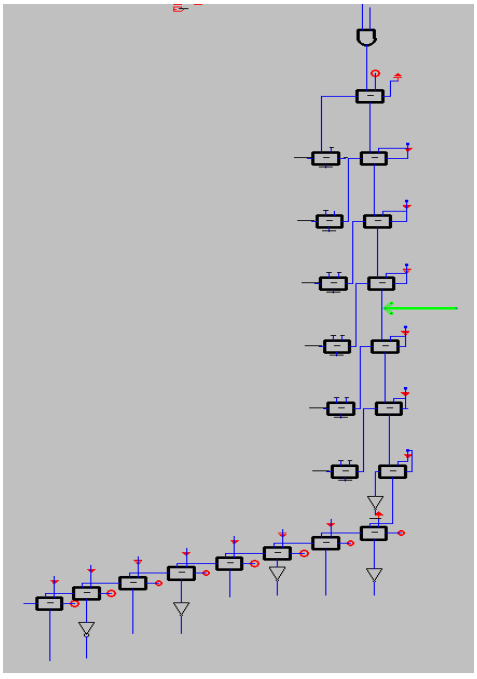




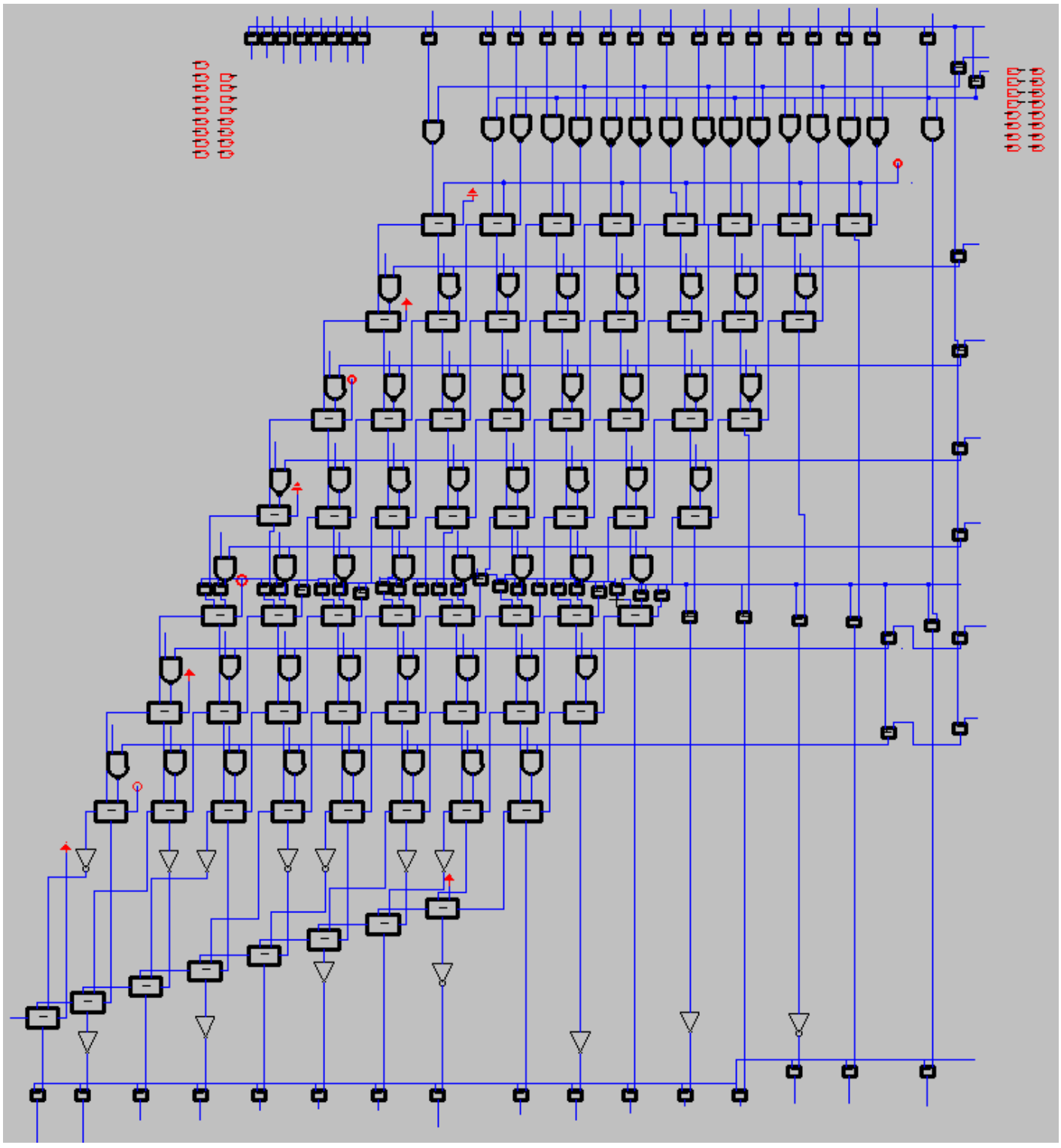
As observed from the plot, the *tsetup* corresponds to the *tdc* when the propagation delay, i.e., the *tdq*, is minimum. So, the *tsetup* for the chosen FF is 15ps.

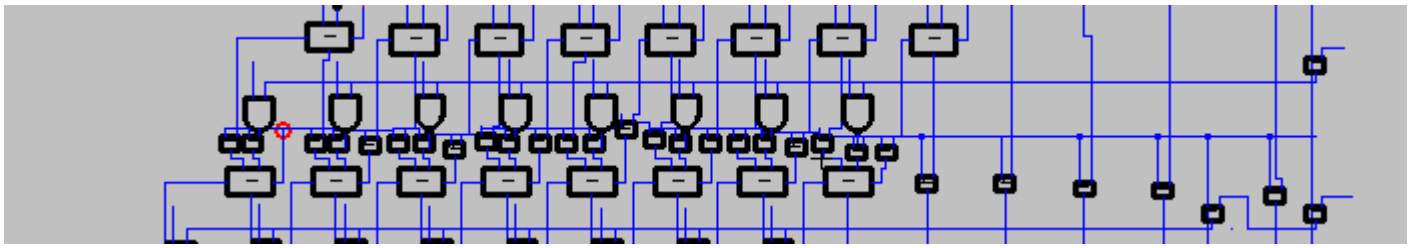
Pipelining the CSM for 2x throughput

In order to achieve nearly 2x speedup in terms of the throughput, the CSM should be converted to a two-stage pipelined unit, by placing FFs at appropriate positions in the datapath. For doing this, the critical path model was analysed, and the wire with nearly the half propagation delay was identified for inserting the pipeline FF.



As shown in the figure, the wire for inserting the pipeline FF was identified as the one after 4 full adders in the critical path. The corresponding cutset for this was identified, and the other FFs were inserted. In order to get a fair comparison, FFs were inserted at the inputs and outputs in both the pipelined and the non-pipelined cases.



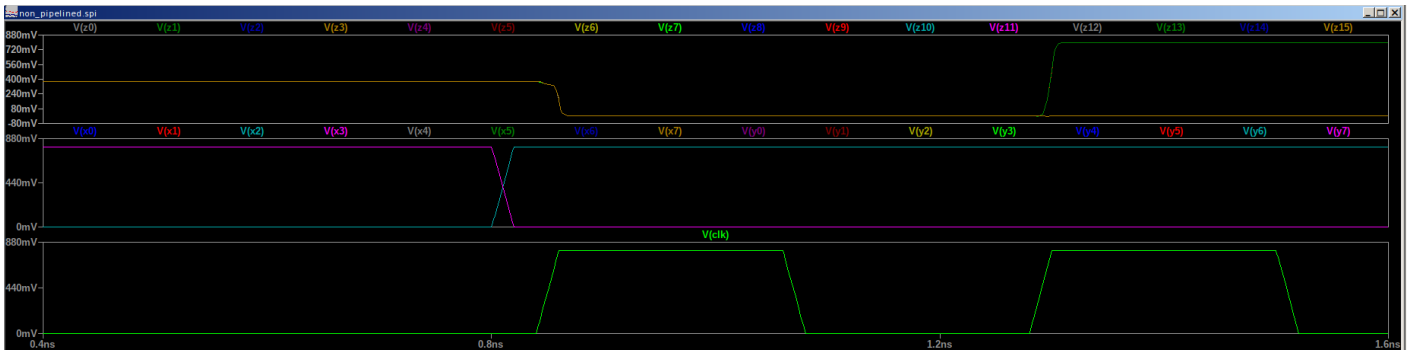


Simulating the Pipelined CSM

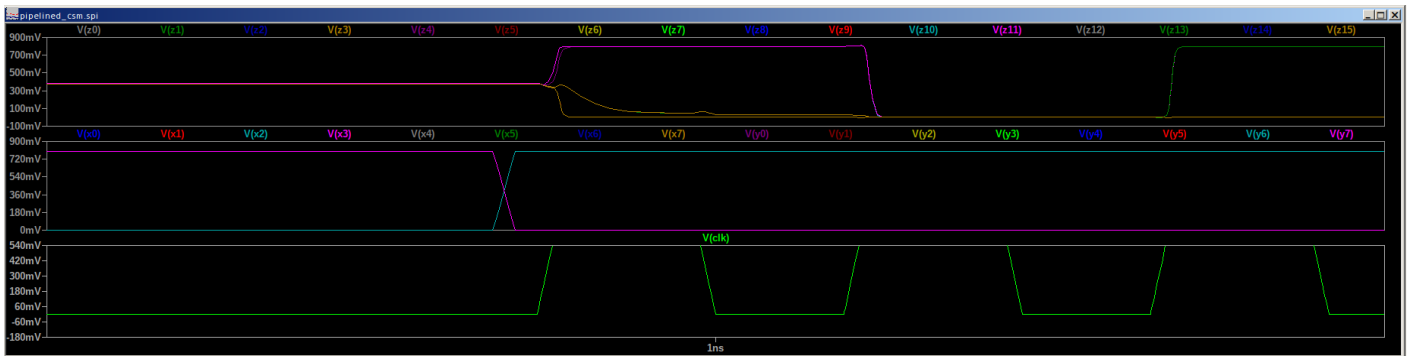
Since the pipelined CSM is a 2-stage pipeline, and we have a FF at the output, the output appears at the third clock pulse from the input being applied. In order to compute the permissible clock period for this pipelined case, the input combination giving the worst propagation delay in the non-pipelined case was considered and simulated. The clock period was progressively increased until all the outputs were correctly latched onto the FFs at the third clock edge, thus taking into account the setup constraints as well.

<u>127 x 127</u>

Non-pipelined : (for a clock period of 440ps) - Note that the output is valid after the second clock edge



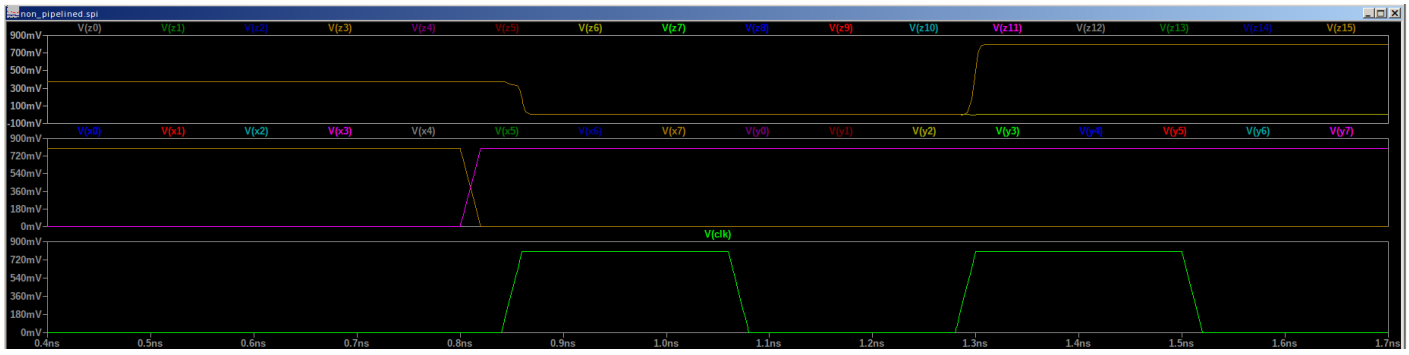
Pipelined : (for a clock period of 275ps) - Note that the output is valid after the third clock edge



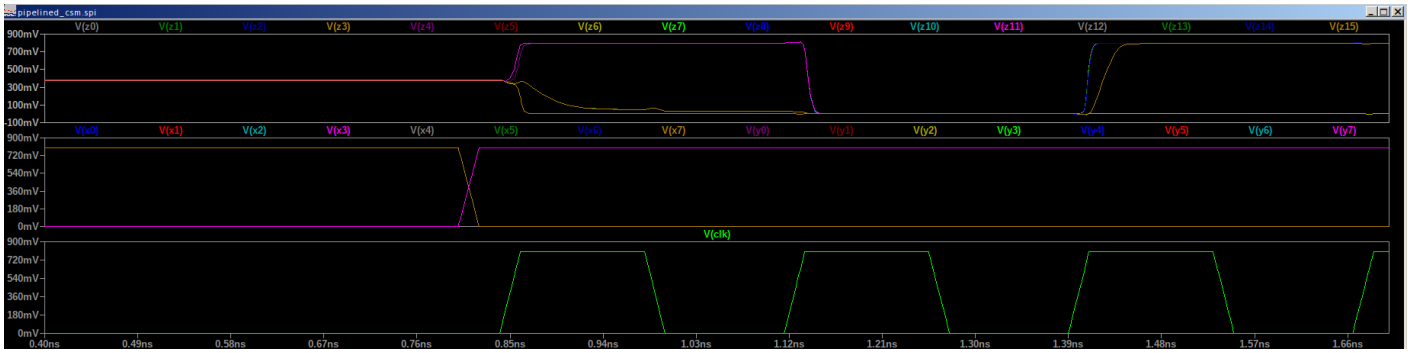
This resulted in a clock pulse of .

<u>127 x -1</u>

Non-pipelined : (for a clock period of 440ps) - Note that the output is valid after the second clock edge



Pipelined : (for a clock period of 275ps) - Note that the output is valid after the third clock edge



Clock Period/ Freq.	Non-Pipelined	Pipelined	Speedup
Schematic	440ps (2.27 GHz)	275ps (3.63 GHz)	1.6x
RC extracted netlist	540ps (1.85 GHz)	345ps (2.898 GHz)	1.56x

Theoretical Clock Period for 2-stage Pipeline

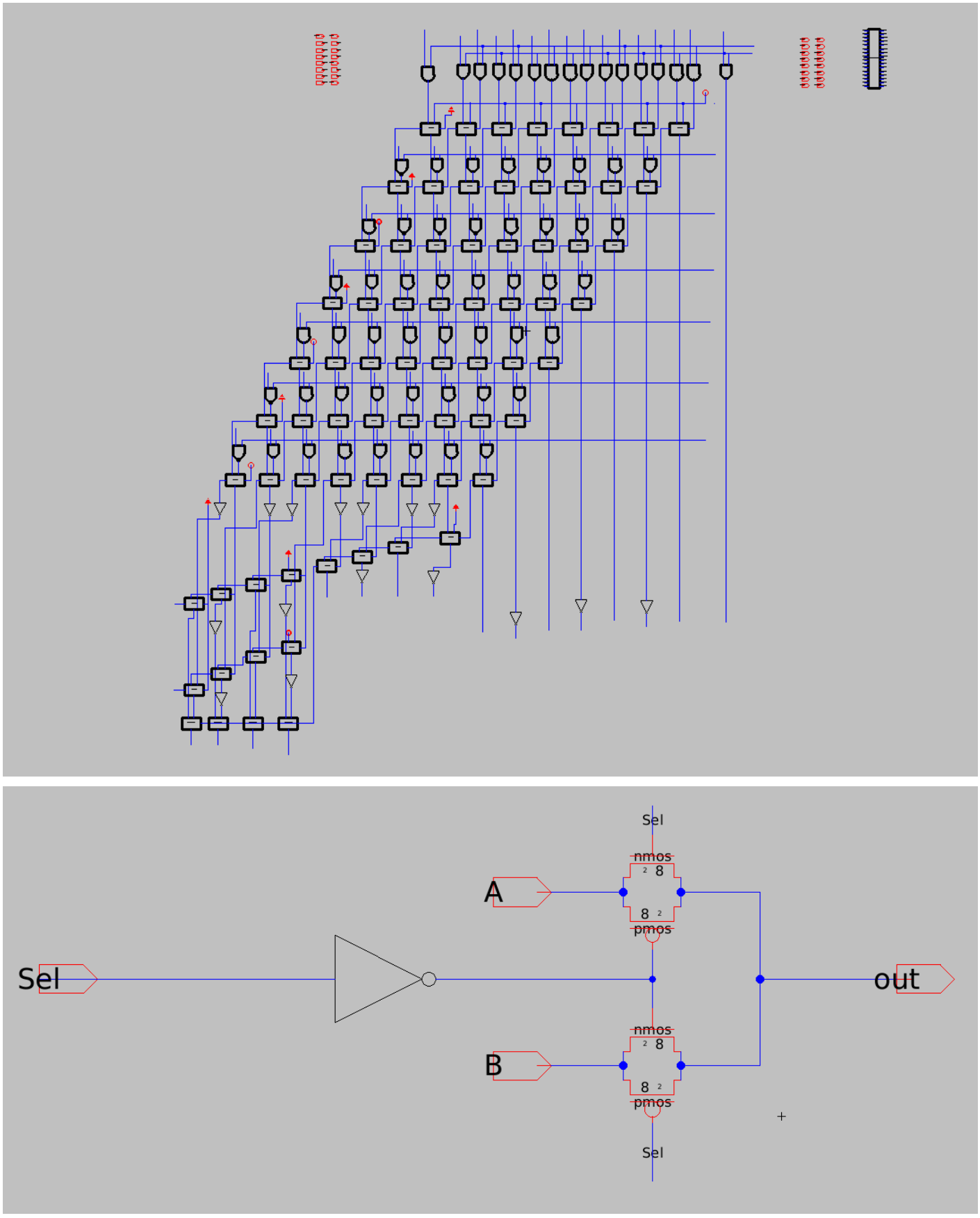
The critical path for the design, i.e., the worst case propagation delay for the design schematic was found to be 412ps. Since the pipeline FFs are expected to break the critical path nearly at the half, the new critical path would be 206ps. From the FF characterization, the sequential overhead was found to be, *tsetup + tcq = 15+9.36 = 24.36ps*. So, the expected operating clock period for the pipelined design's schematic would be *206+24.36 = **230.36ps***.

Alternate Vector Merge adders

Since the Vector Merge stage involves a series of Full Adders that propagate the carry through them, it is intended to explore alternate adder mechanisms that could cut down the time involved in this stage, thereby reducing the critical path. Some alternatives have been explored here.

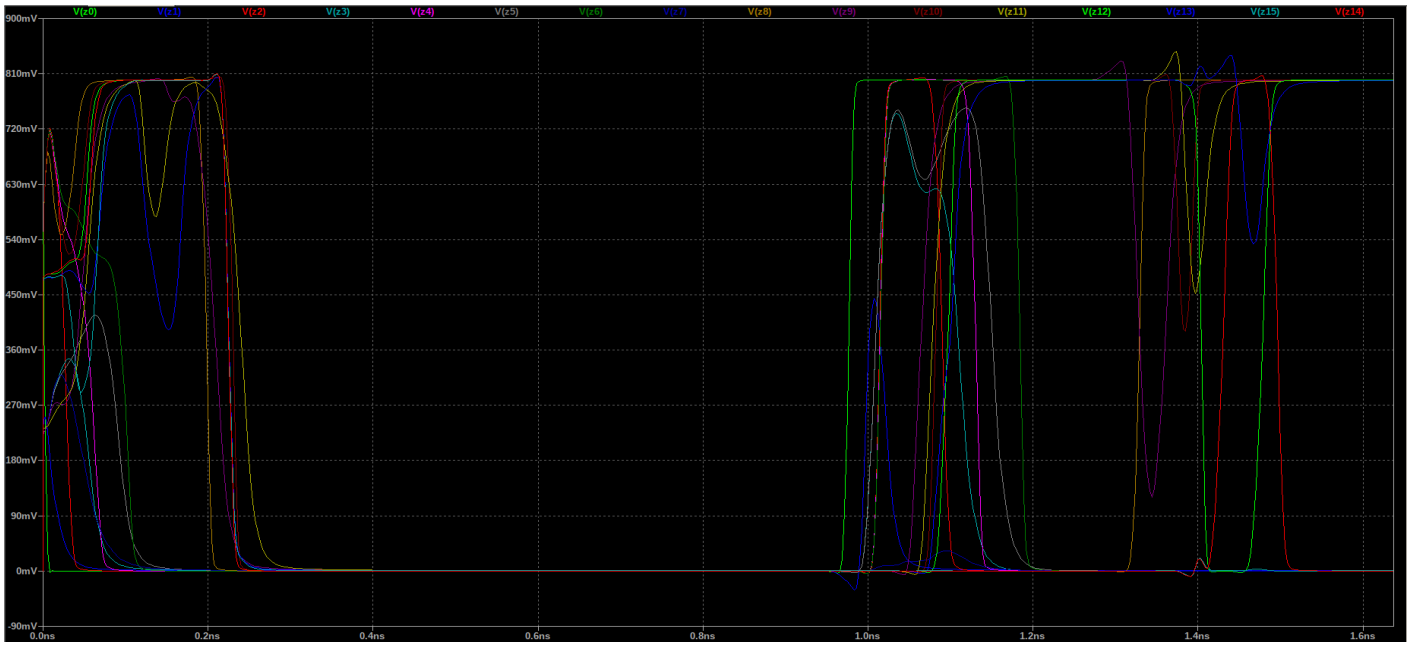
Linear Carry Select Adder

Linear Carry Select Adder involves an optimisation in terms of computing apriori the results for both cases of carry in, and selecting one of these based on the carry in, in order to reduce the propagation delay. This brings down the delay to O(√N). The schematic design for this, and the effect of incorporating it into the vector merge stage has been elucidated below.



2X1 Multiplexer

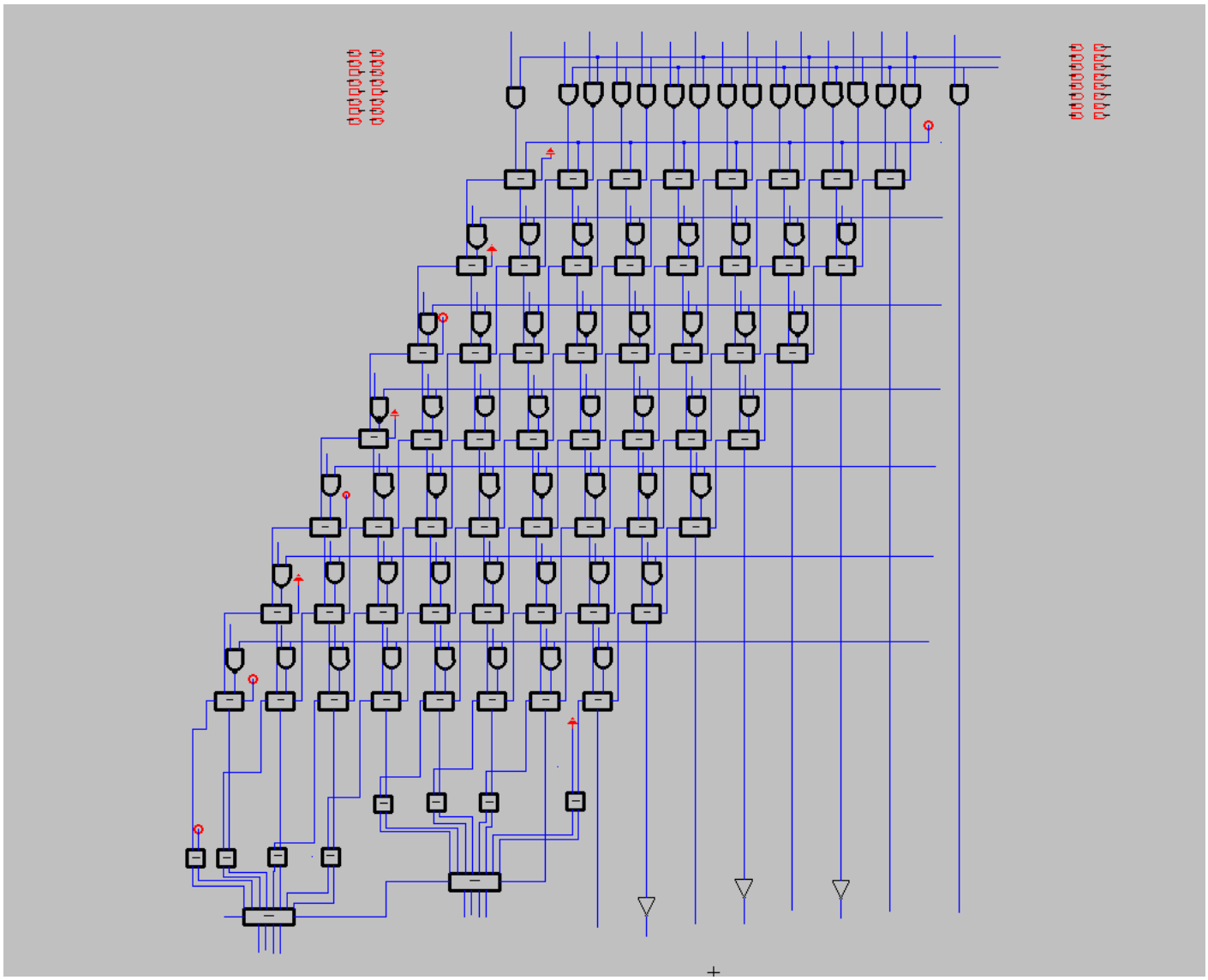
X		Y		Result		Ripple adder Cell RC extracted delay (ps)	Carry select adderCell RC extracted delay (ps)
Decimal	Binary	Decimal	Binary	Decimal	Binary		
-1	11111111	-1	11111111	1	0000000000000001	503	452
1	00000001	-1	11111111	-1	1111111111111111	270	214
-1	11111111	1	00000001	-1	1111111111111111	478	422
-1	11111111	127	01111111	-127	1111111110000001	511	542
-128	10000000	-1	11111111	128	0000000010000000	324	322
127	01111111	127	01111111	16129	0011111100000001	514	546
-127	10000001	-127	10000001	16129	0011111100000001	309	309
127	01111111	-1	11111111	-127	1111111110000001	463	464
69	01000101	42	00101010	2898	0000101101010010	372	368
52	00110100	21	00010101	1092	0000010001000100	298	297
-11	11110101	-11	11110101	121	0000000001111001	504	457
5	00000101	36	00100100	180	0000000010110100	174	174
10	00001010	-128	10000000	-1280	1111101100000000	204	148

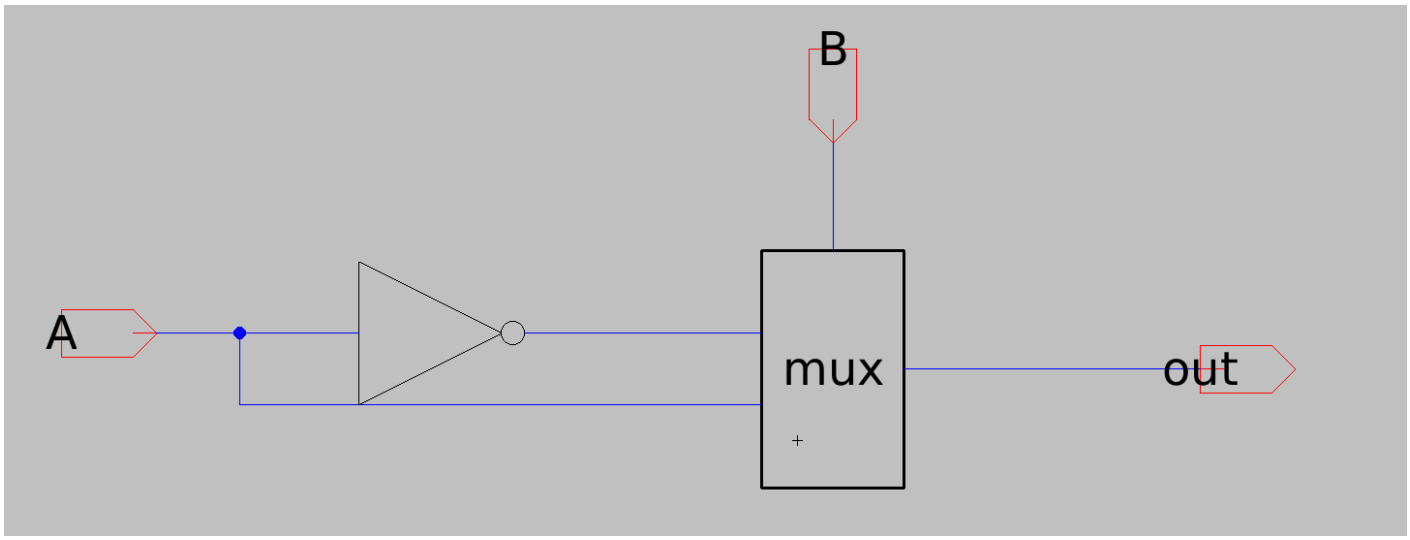


Worst Case delay using Carry Select Adder

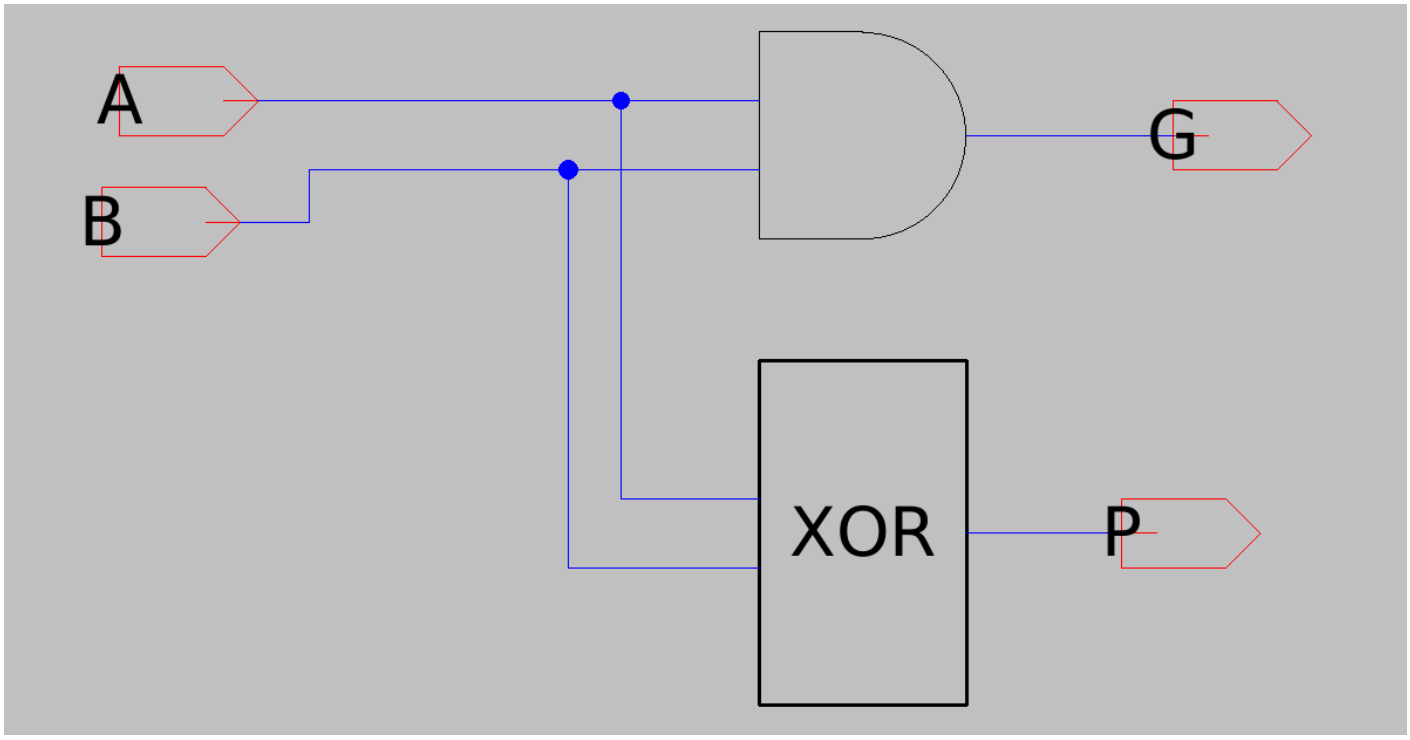
Carry Lookahead adder

The carry lookahead adder is a fast adder that computes the carry generation and propagation terms apriori, and then computes the sum using these terms. This cuts down the delay to O(log(N)).

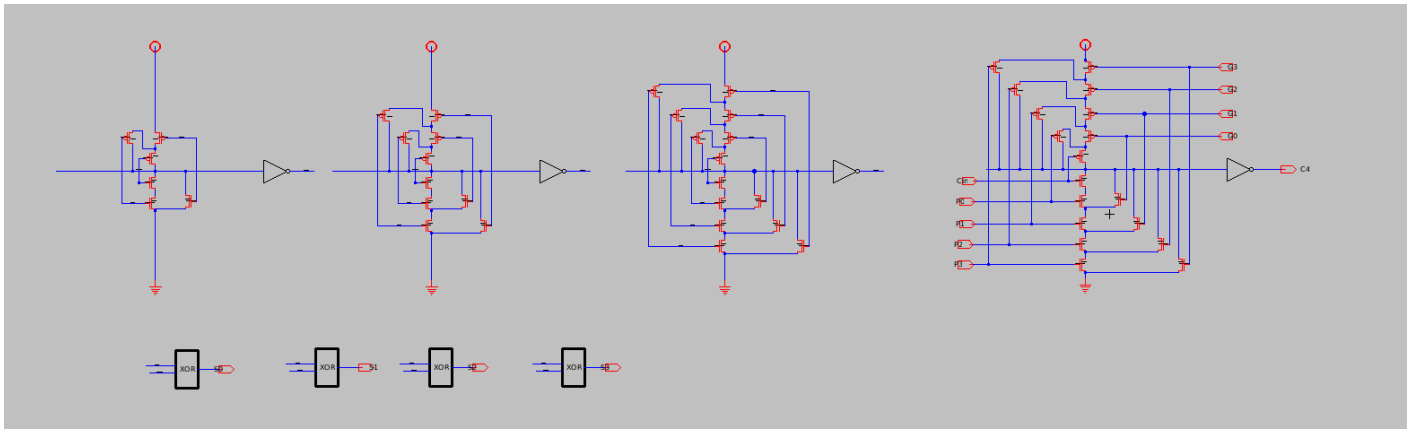




XOR gate

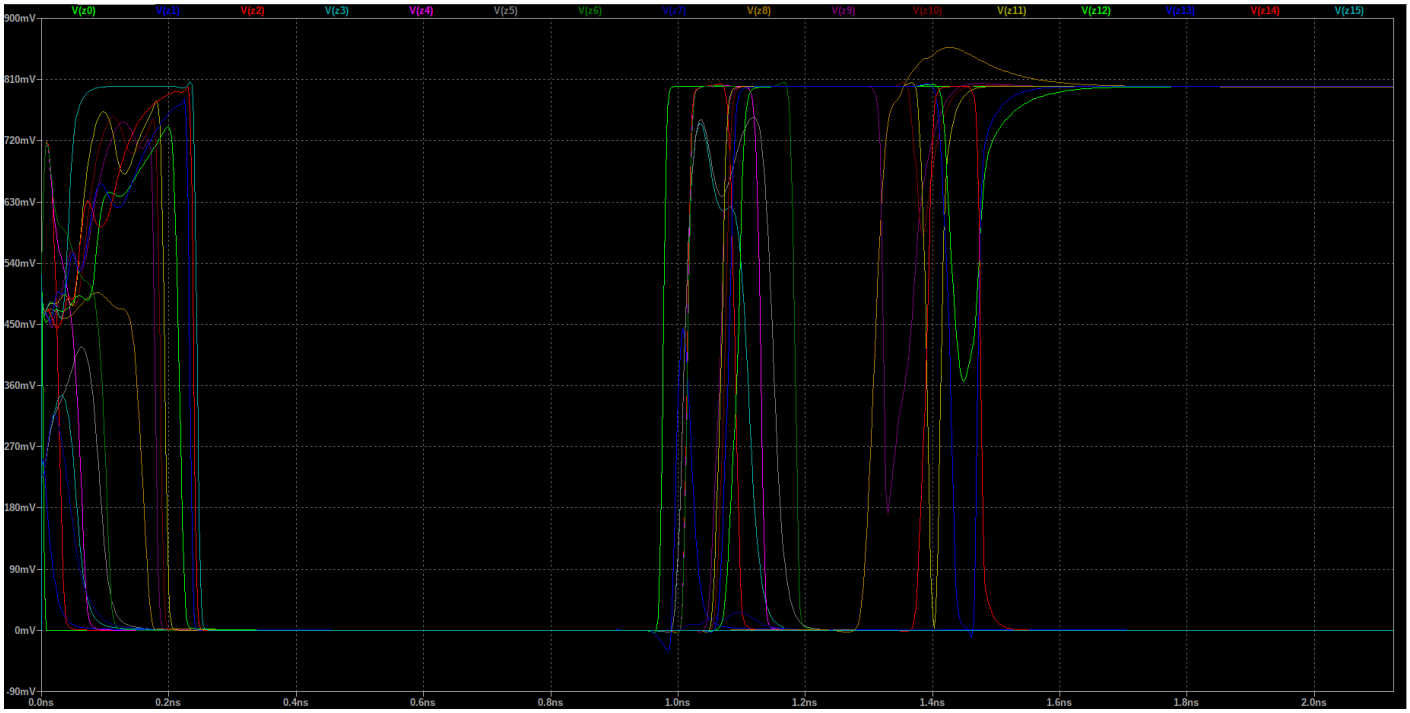


PG generation



CLA adder

X		Y		Result		Ripple adder Cell RC extracted delay (ps)	Carry Lookahead adder Cell RC extracted delay (ps)
Decimal	Binary	Decimal	Binary	Decimal	Binary		
-1	11111111	-1	11111111	1	0000000000000001	503	501
1	00000001	-1	11111111	-1	1111111111111111	270	253
-1	11111111	1	00000001	-1	1111111111111111	478	440
-1	11111111	127	01111111	-127	1111111110000001	511	503
-128	10000000	-1	11111111	128	0000000010000000	324	363
127	01111111	127	01111111	16129	0011111100000001	514	525
-127	10000001	-127	10000001	16129	0011111100000001	309	309
127	01111111	-1	11111111	-127	1111111110000001	463	402
69	01000101	42	00101010	2898	0000101101010010	372	390
52	00110100	21	00010101	1092	0000010001000100	298	289
-11	11110101	-11	11110101	121	0000000001111001	504	465
5	00000101	36	00100100	180	0000000010110100	174	181
10	00001010	-128	10000000	-1280	1111101100000000	204	172

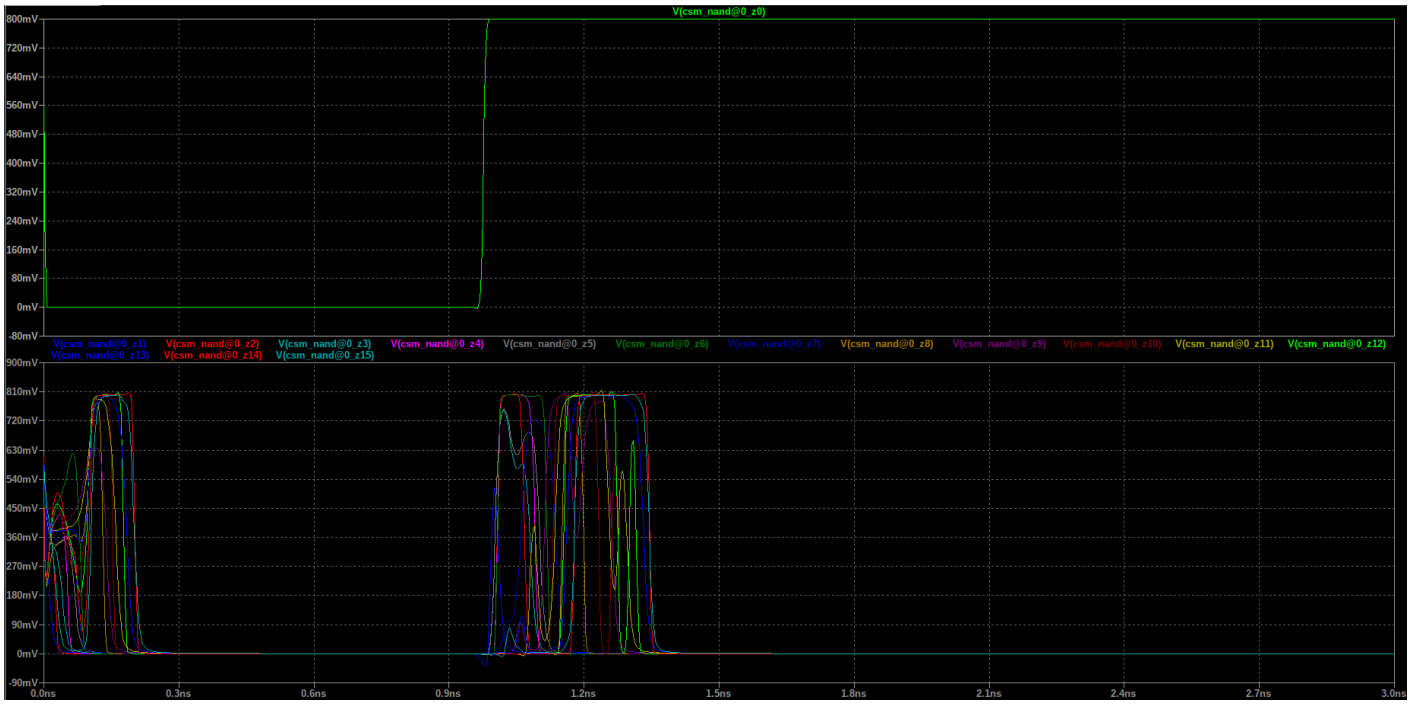


Worst Case delay using Carry Lookahead Adder

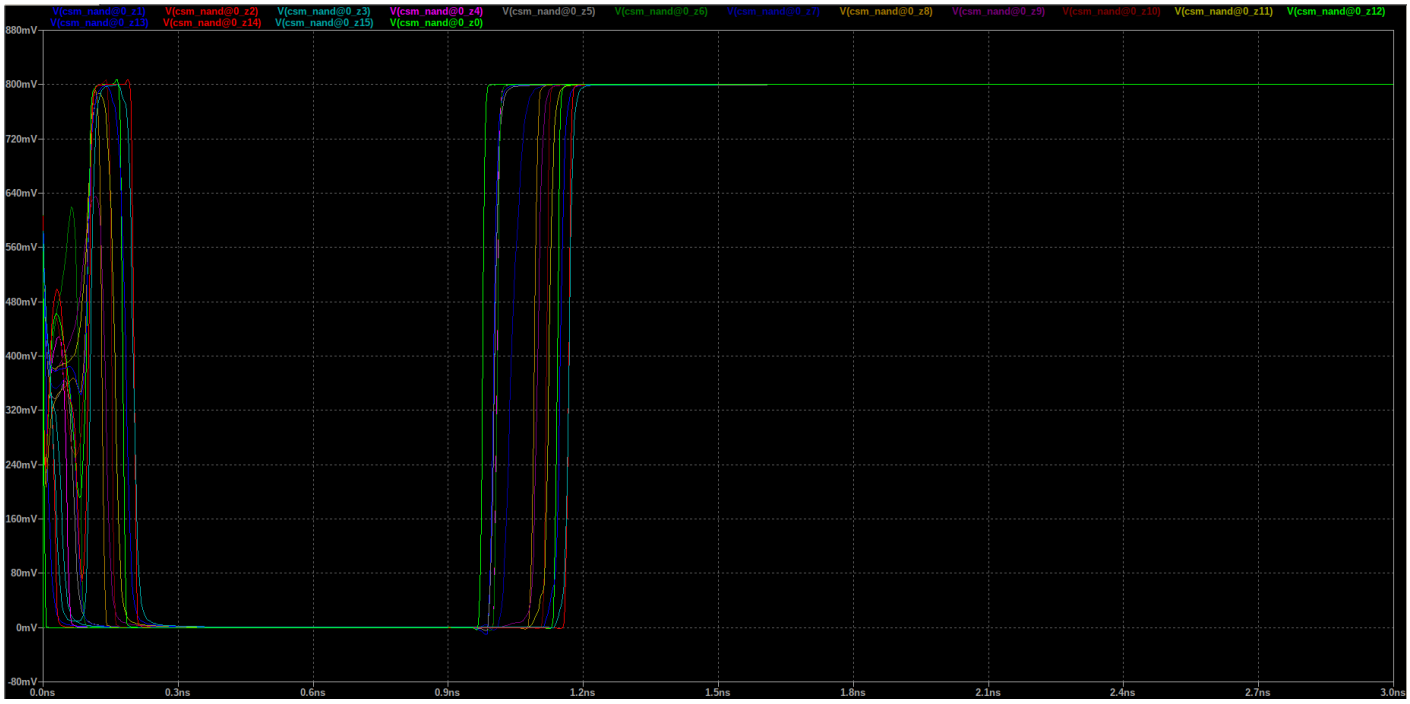
- Critical delay with ripple carry adder in vector merge = 412 ps (schematic)
  = 514 ps (layout)
- Critical delay with 4-bit carry select adder in vector merge = 443 ps (schematic)
  = 546 ps (layout)
- Critical delay with 4-bit carry lookahead adder in vector merge = 442 ps (schematic)
  = 525 ps (layout)

Functional Simulation

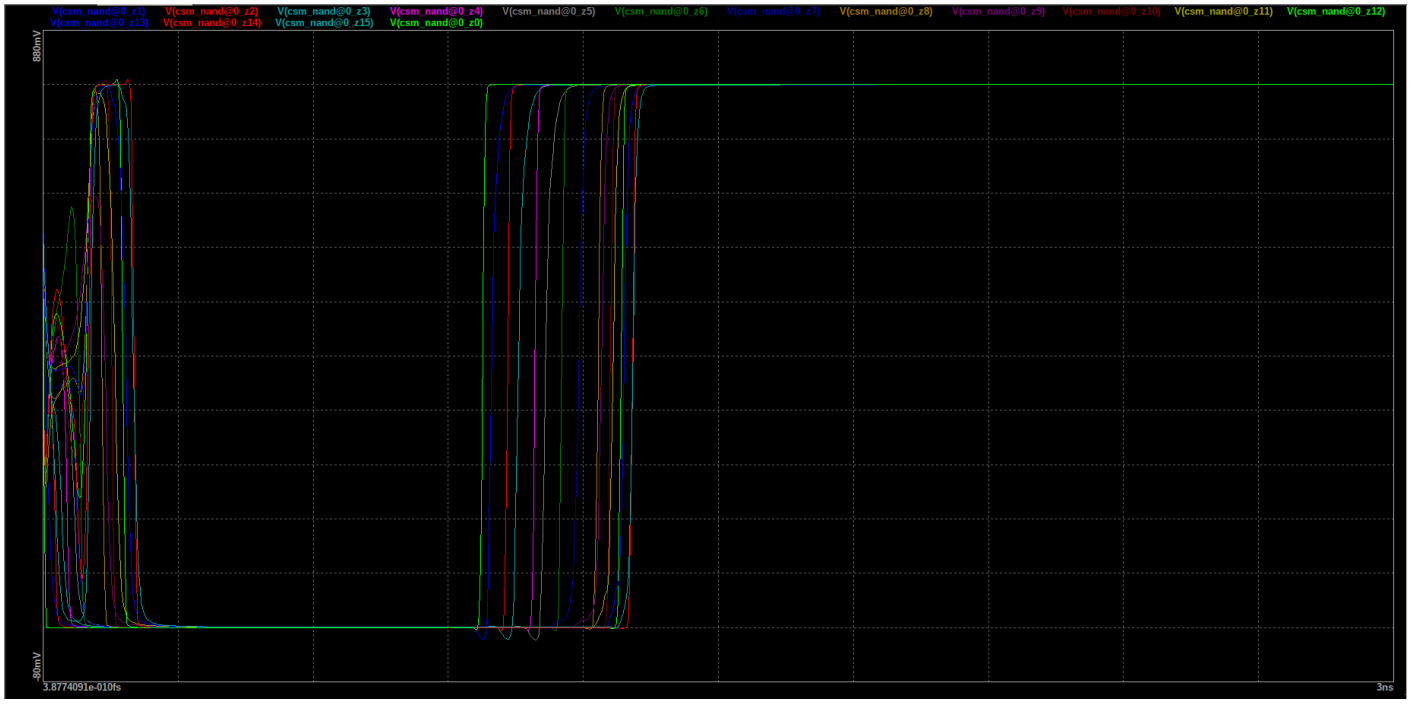
-1*-1 = 1



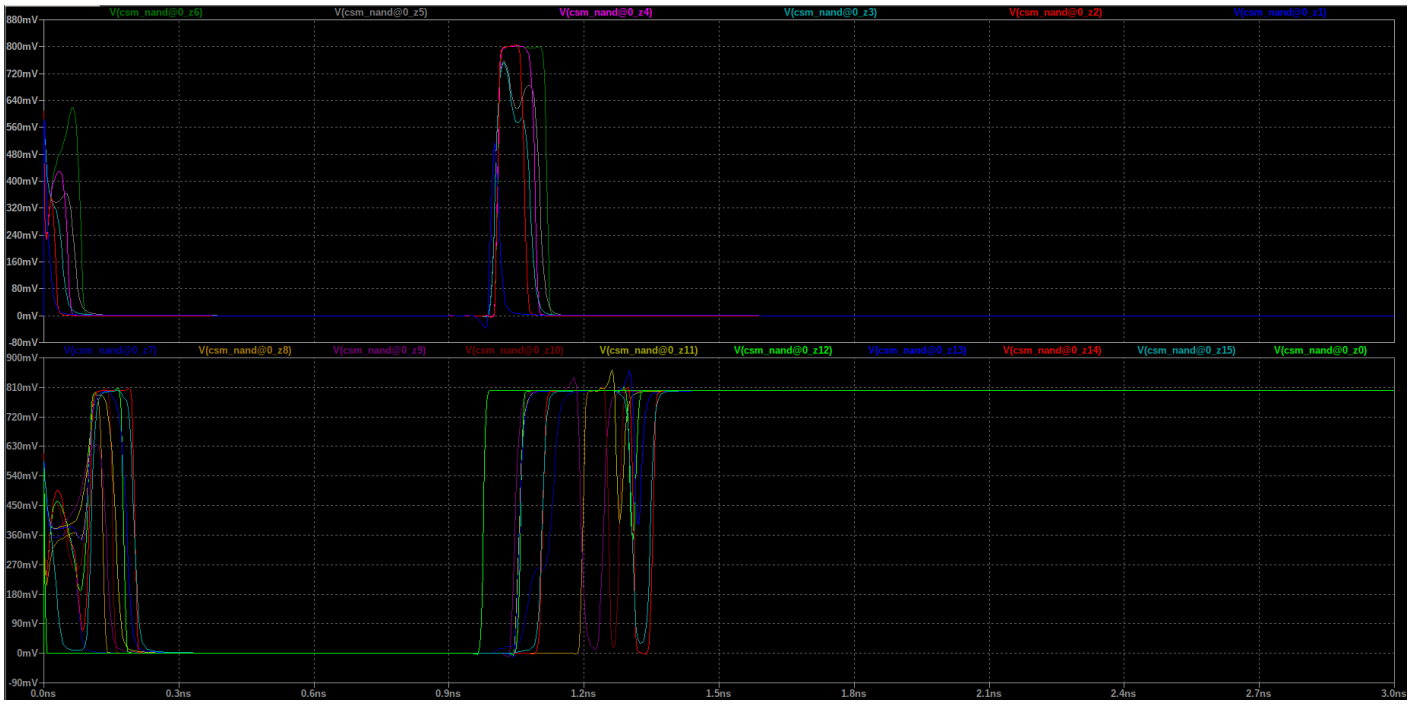
1*-1 = -1



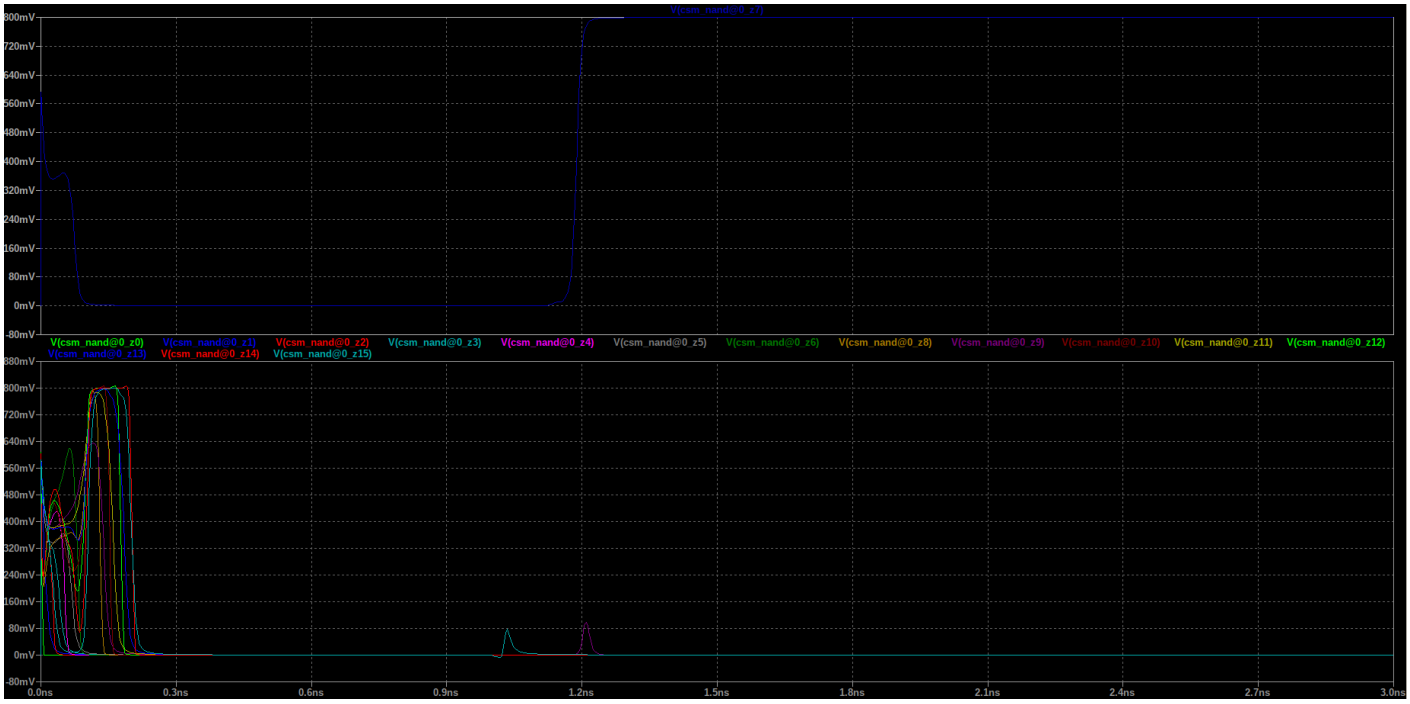
-1*1 = -1


V(csm_nand@0_z1) V(csm_nand@0_z2) V(csm_nand@0_z3) V(csm_nand@0_z4) V(csm_nand@0_z5) V(csm_nand@0_z6) V(csm_nand@0_z7) V(csm_nand@0_z8) V(csm_nand@0_z9) V(csm_nand@0_z10) V(csm_nand@0_z11) V(csm_nand@0_z12)
V(csm_nand@0_z13) V(csm_nand@0_z14) V(csm_nand@0_z15) V(csm_nand@0_z0)
880mV
-80mV
3.8774091e-010fs
3ns

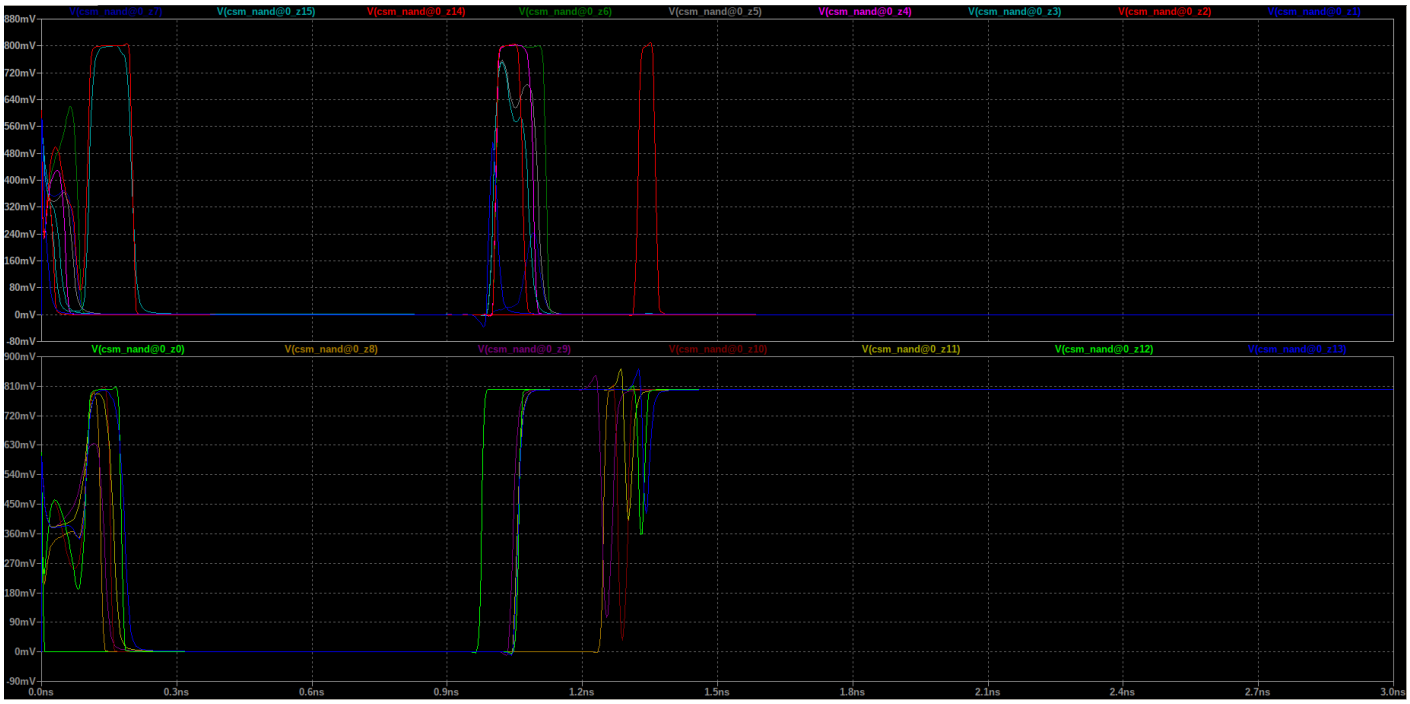
-1*127 = -127


V(csm_nand@0_z6) V(csm_nand@0_z5) V(csm_nand@0_z4) V(csm_nand@0_z3) V(csm_nand@0_z2) V(csm_nand@0_z1)
880mV 800mV 720mV 640mV 560mV 480mV 400mV 320mV 240mV 160mV 80mV 0mV -80mV
V(csm_nand@0_z7) V(csm_nand@0_z8) V(csm_nand@0_z9) V(csm_nand@0_z10) V(csm_nand@0_z11) V(csm_nand@0_z12) V(csm_nand@0_z13) V(csm_nand@0_z14) V(csm_nand@0_z15) V(csm_nand@0_z0)
900mV 810mV 720mV 630mV 540mV 450mV 360mV 270mV 180mV 90mV 0mV -90mV
0.0ns 0.3ns 0.6ns 0.9ns 1.2ns 1.5ns 1.8ns 2.1ns 2.4ns 2.7ns 3.0ns

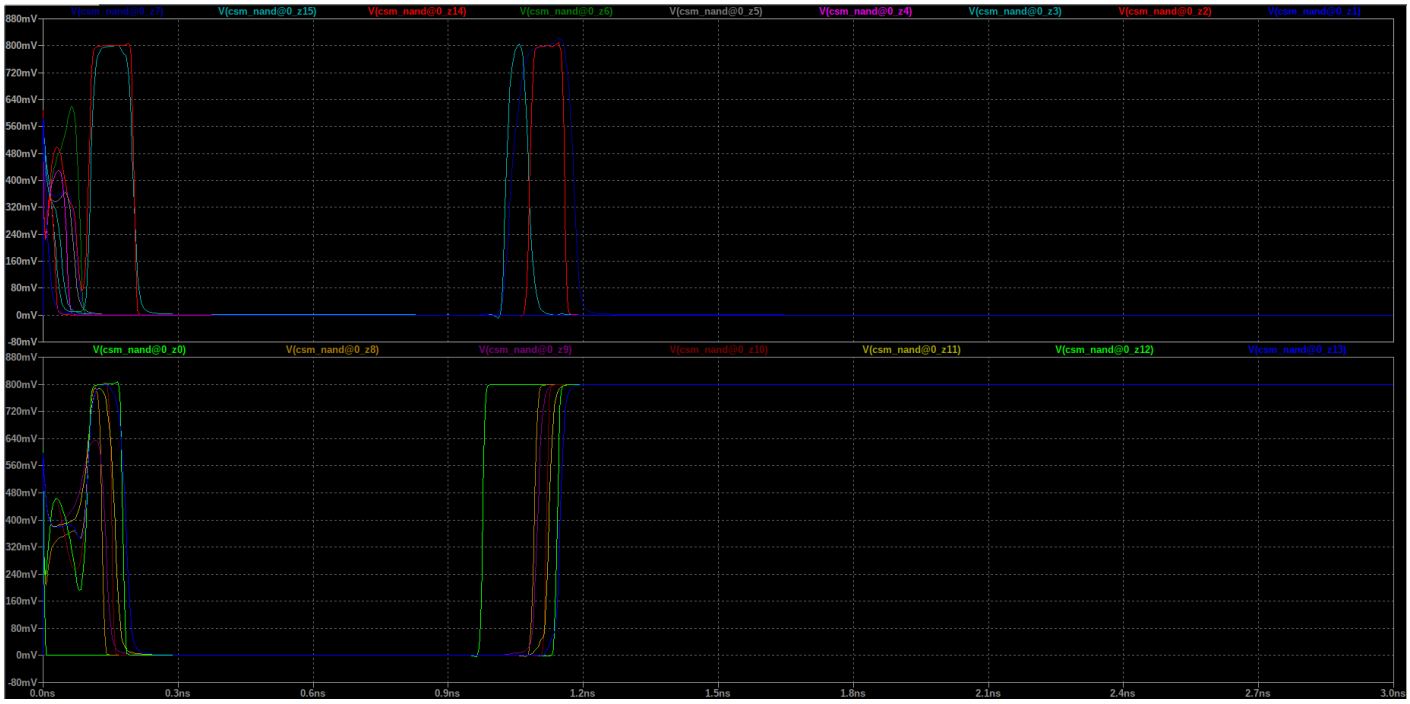
-128*-1 = 128



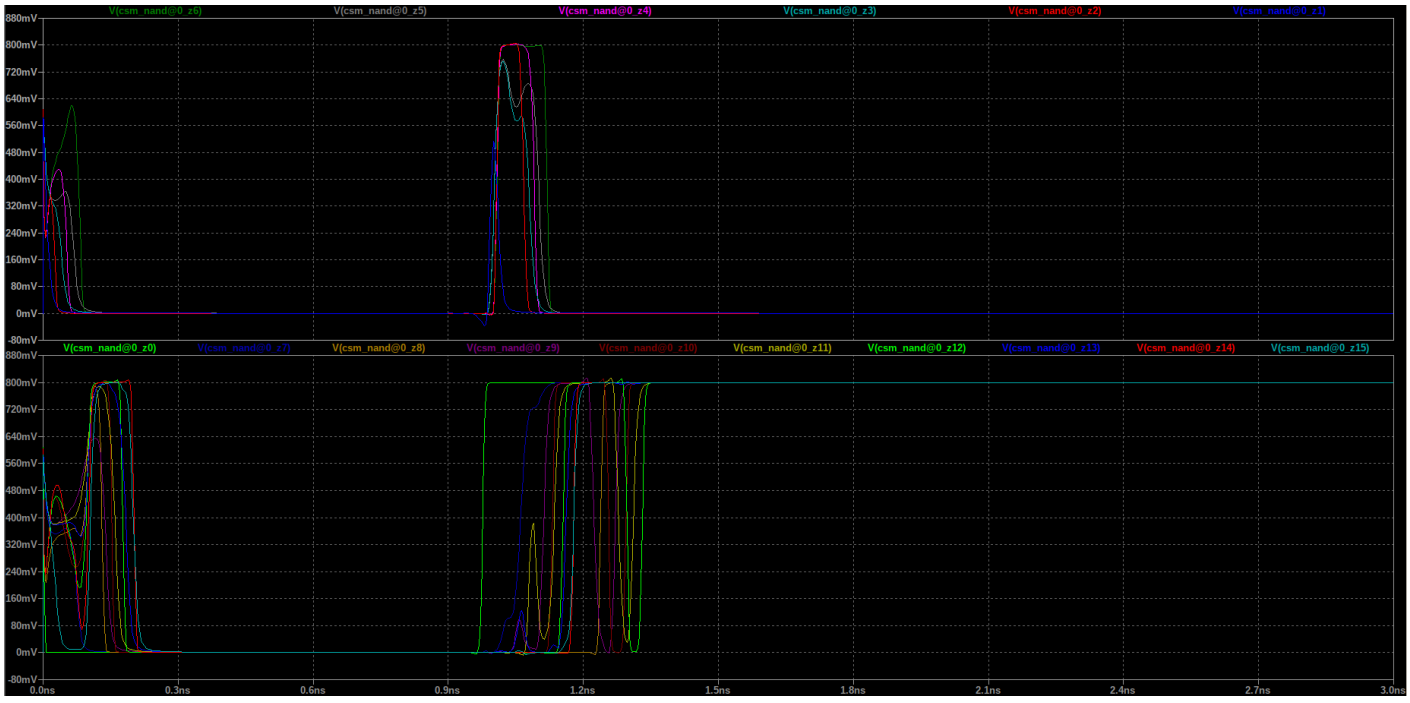
127*127 = 16129



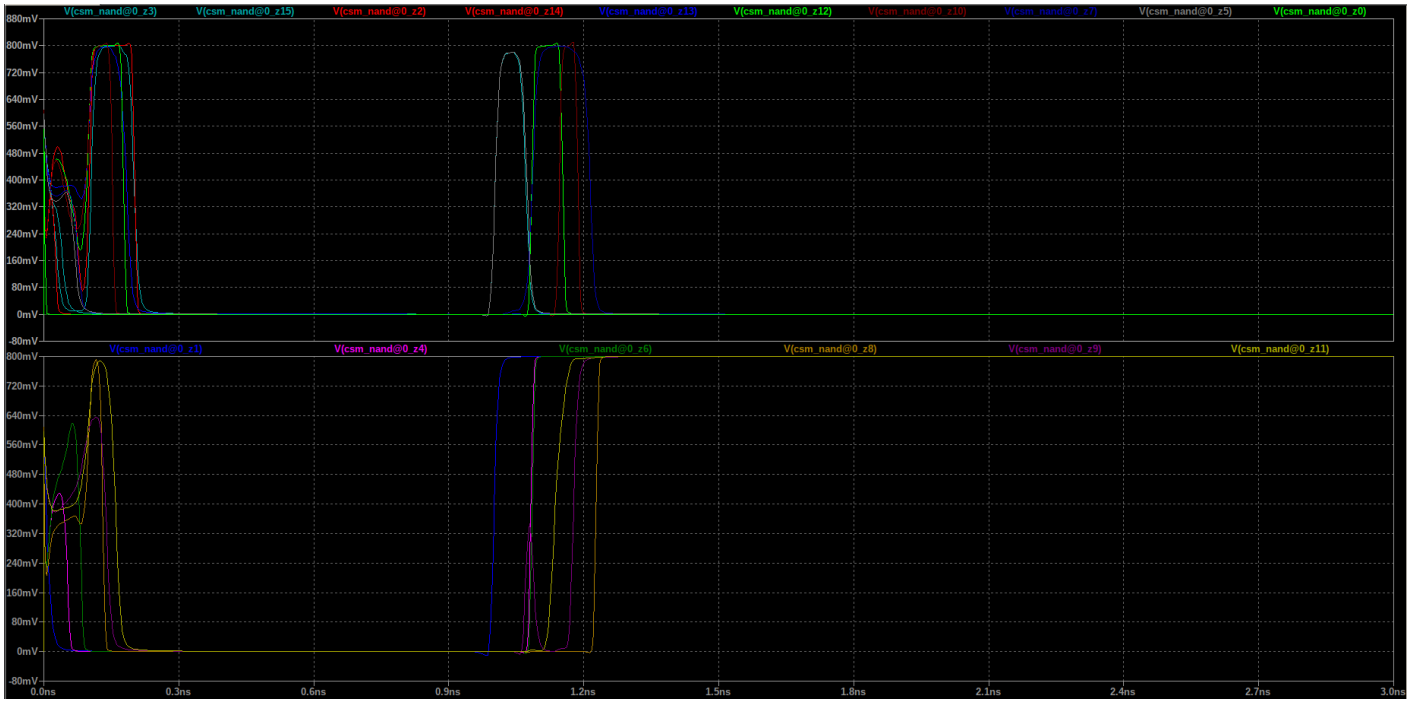
-127*-127 = 16129



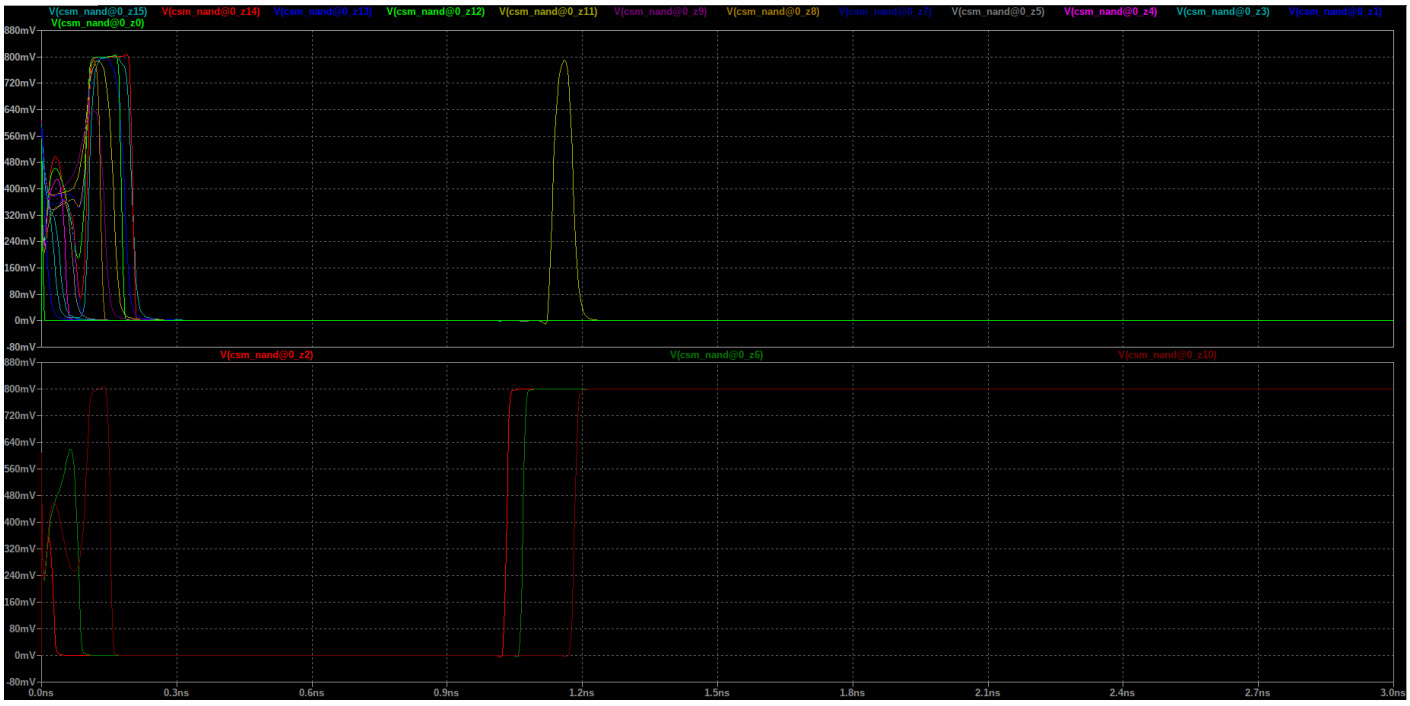
127*-1 = -127



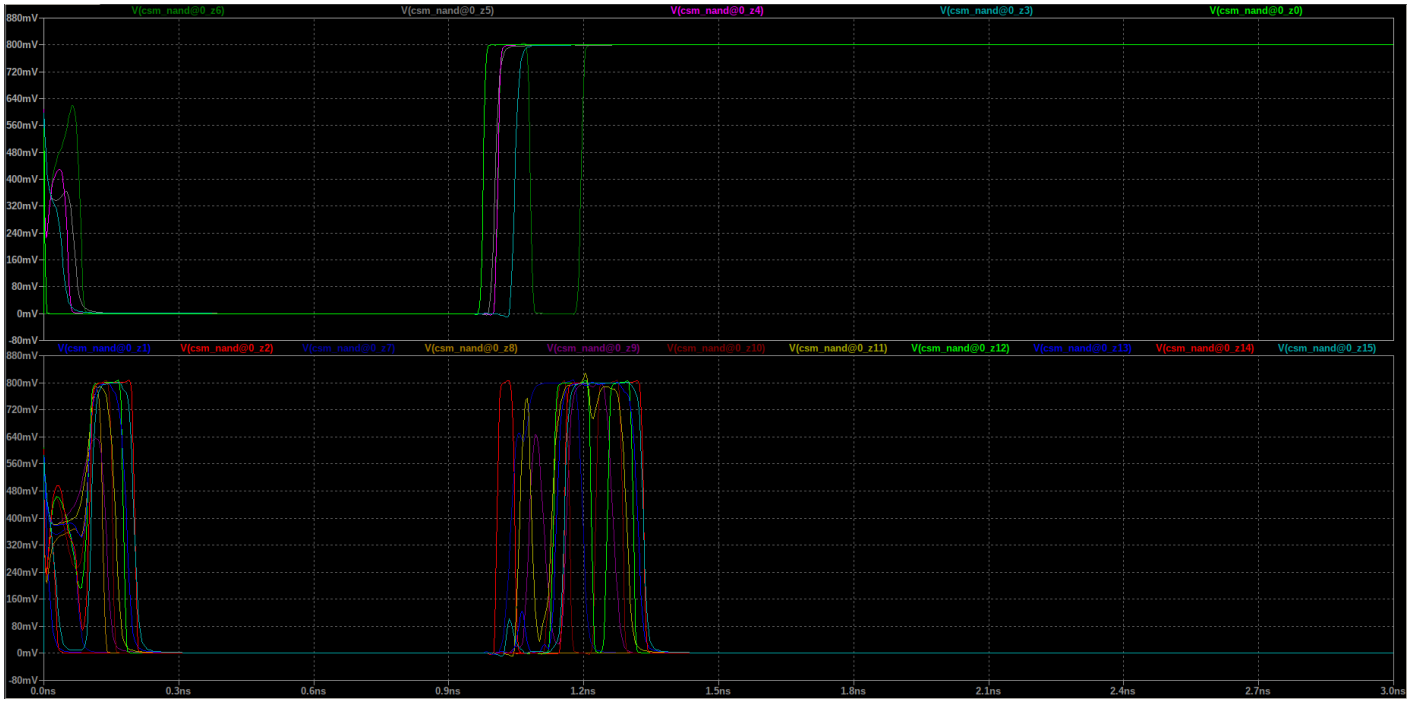
69*42 = 2898



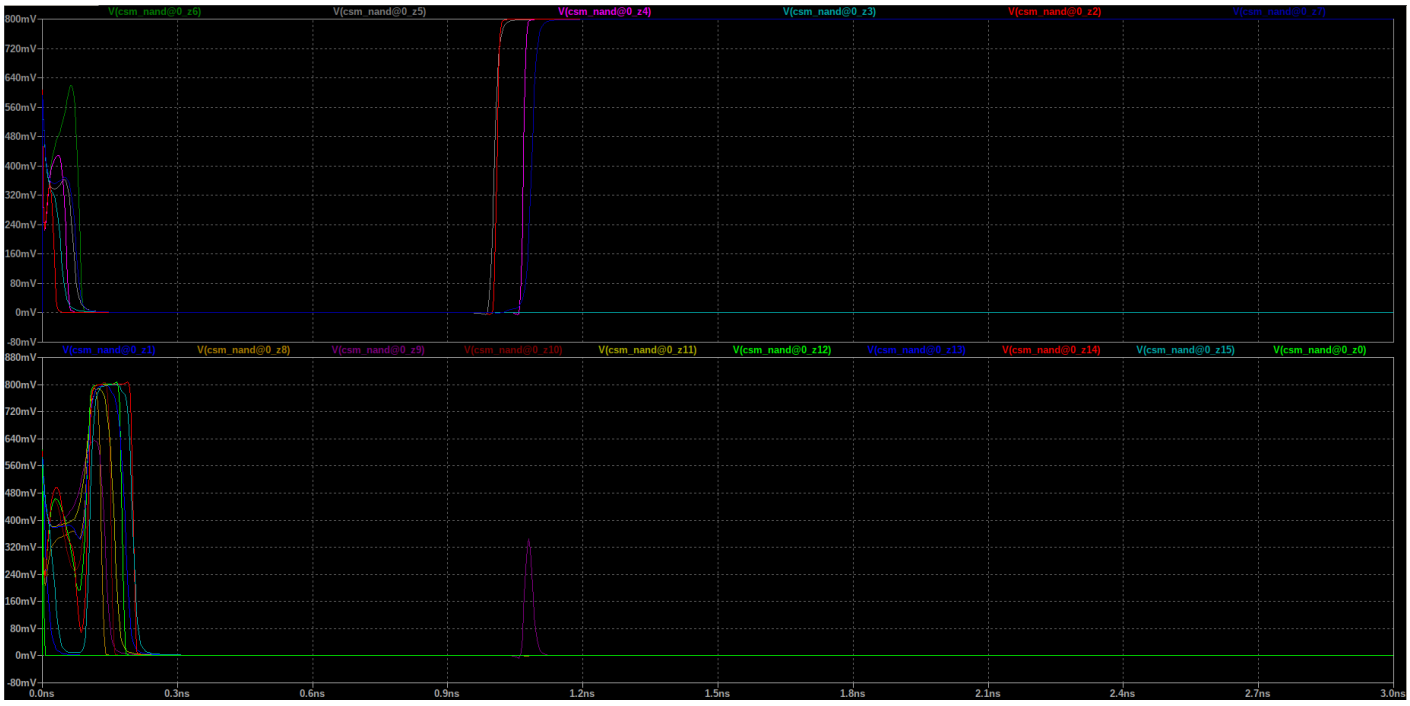
52*21 = 1092



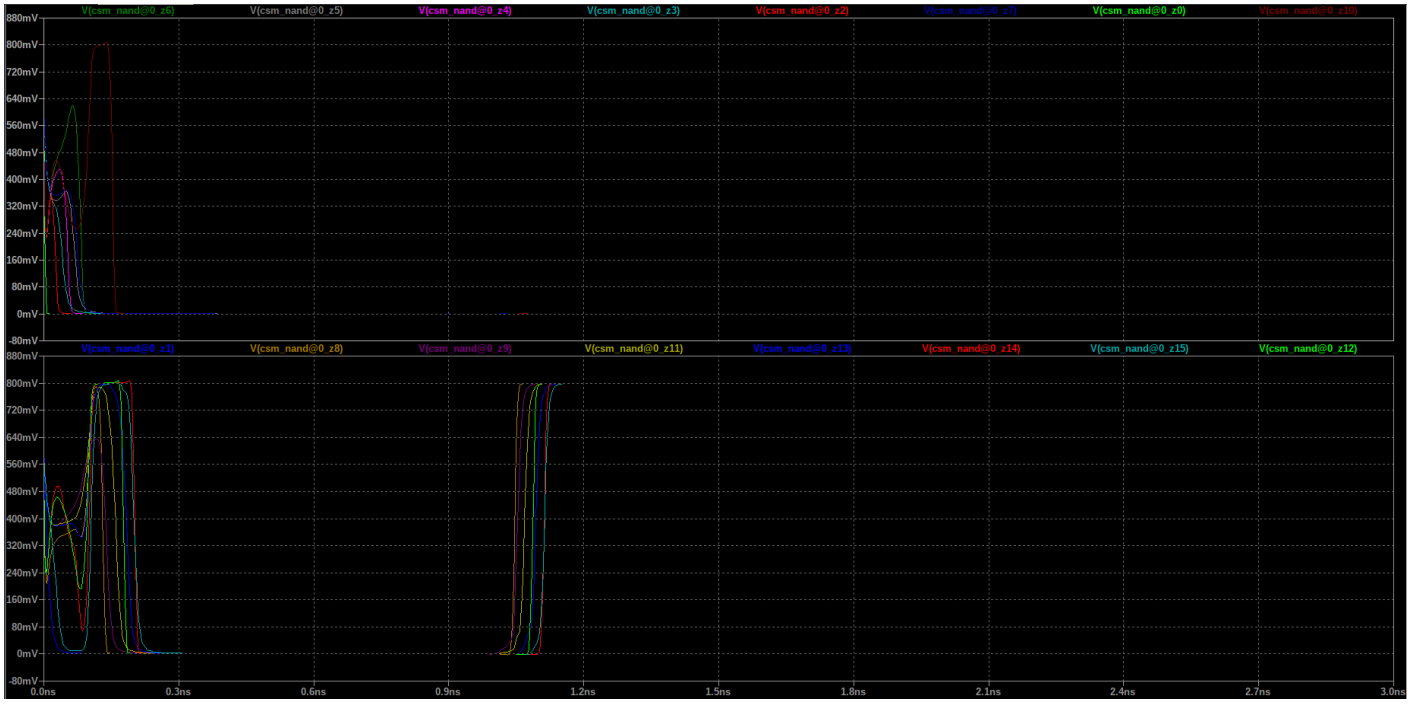
-11*-11 = 121



5*36 = 180



10*-128 = -1280


V(csm_nand@0_z6)
V(csm_nand@0_z5)
V(csm_nand@0_z4)
V(csm_nand@0_z3)
V(csm_nand@0_z2)
V(csm_nand@0_z7)
V(csm_nand@0_z0)
V(csm_nand@0_z10)
880mV
800mV
720mV
640mV
560mV
480mV
400mV
320mV
240mV
160mV
80mV
0mV
-80mV
V(csm_nand@0_z1)
V(csm_nand@0_z8)
V(csm_nand@0_z9)
V(csm_nand@0_z11)
V(csm_nand@0_z13)
V(csm_nand@0_z14)
V(csm_nand@0_z15)
V(csm_nand@0_z12)
0.0ns
0.3ns
0.6ns
0.9ns
1.2ns
1.5ns
1.8ns
2.1ns
2.4ns
2.7ns
3.0ns