Jeff Grindel

16-bit x 16-bit Multiplication Using Split Path Data Driven Dynamic Logic

8/19/13

Table of Contents

Table of Figures

Table of Tables

Abstract

In this report a schematic and layout will be detailed for a 16x16 bit radix-4 Booth multiplier. Within this design a Wallace Tree Adder(combination of 3:2 and 4:2 reduction circuits), and a Parallel-prefix-adder (Kogge-stone) implementation will be implemented to calculate the final result. All designs are hierarchical and utilize sub-blocks to construct the final multiplier. The results for the Kogge-Stone adder and the full multiplier design will be discussed in terms of performance in power, leakage, and delay.

Introduction

The purpose of this project will be to implement a 16x16 bit multiplier using SPD3L in both schematic and layout and to analyze the results in terms of power, delay, and leakage. These 4-bit adders will be designed and implemented using hierarchical design methodology to construct the schematics, Synopsys HSPICE to measure the average power, and CSCOPE to obtain the delay timings of the operation of the full adders.

The schematic entry and creation of the circuits will be using the FreePDK45. This is a free and open source library that has different type of standard cell designs of PMOS and NMOS transistors. These standard cells are based on 45 nm technology.

The entire design is very hierarchical. The final schematic/layout for the multiplier will consists of multiple different cells in order to make the design simpler. In these cells most of them are also built in a hierarchal fashion, being built from the single bit implementations and replicated to get a full bus.

A majority of this design is based off an IEEE publication [1]. Other supplementing material, [2]-[4] is used to understand the algorithms and other detail of the circuits not described in [1].

Background

Split-Path Date Driven Dynamic Logic (SPD3L)

SPD3L splits up the series connected PMOS transistors in the PUN to multiple PUN's. This minimizes the delay and the energy penalty caused by the PUN, especially if there are a large number of leakage paths in the PDN. [1] To split up the PUN, the PDN is broken down into multiple circuits. For X leakage path in the PDN D3L there will be now be X SPD3L networks, one with each PDN path, with a single PMOS transistor for the pre-charge phase. Each of the circuits Dynamic Nodes are then fed into a NAND gate to obtain the correct functionality. The implementation of a SP-D3L circuit with the circuit equation $F = (I1,1 \ldots Ik,1) + \cdots + (I1,m \ldots In,m)$, can be seen below in Figure 1. [3]

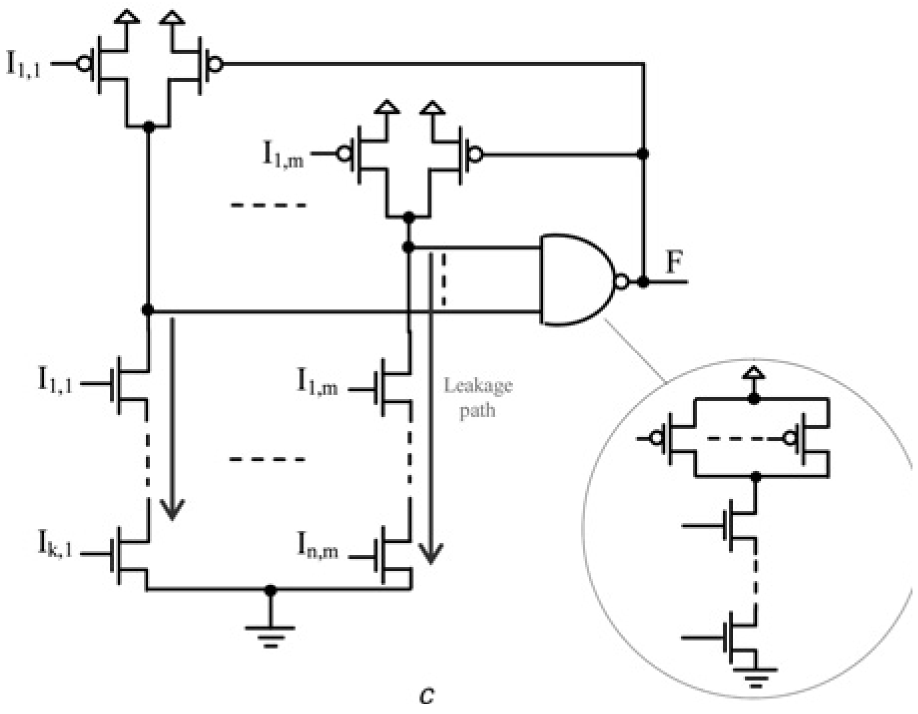
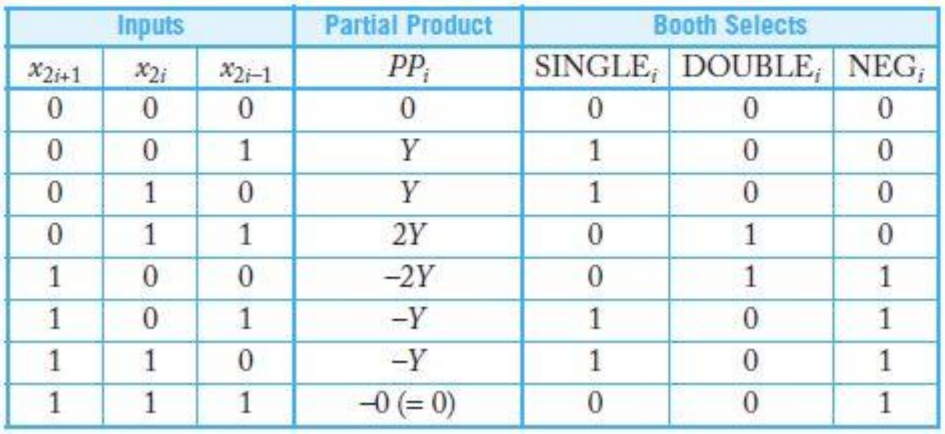


Figure 1: SP-D3L implementation of F

Booth Encoder

Booth's Algorithm reduces the number of partial products needed to calculate the final results, this reduces the hardware and delay required to sum the partial products. The radix-4 reduces the number of partial products to $\frac{n}{2} + 1$ (in the case of a 16x16 bit multiplier, the number of partial products generated will be 9). The booth encoder takes one of the inputs to the circuit and encodes it based on a 3-bit overlapping grouping of the bits. Grouping the bits together allows for a scale factor to be generated. The booth encoders produce 3 signals, SINGLE (ONE), DOUBLE (TWO), and NEG. These encoded signals are then passed to the PPG. The following figure [3] shows the 3-bit combinations that form the different scaling factor.

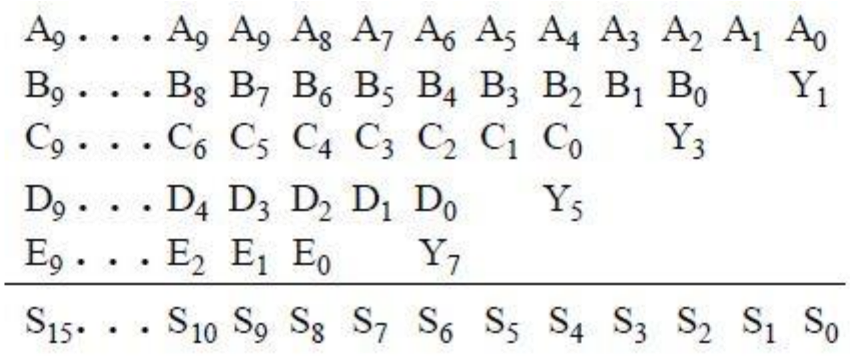


Inputs			Partial Product	Booth Selects		
x_{2i+1}	x_{2i}	x_{2i-1}	PP_i	SINGLE_i	DOUBLE_i	NEG_i
0	0	0	0	0	0	0
0	0	1	Y	1	0	0
0	1	0	Y	1	0	0
0	1	1	$2Y$	0	1	0
1	0	0	$-2Y$	0	1	1
1	0	1	$-Y$	1	0	1
1	1	0	$-Y$	1	0	1
1	1	1	-0 (= 0)	0	0	1

Figure 2: Booth Encoding Table

Partial Product Generator

The Partial products generator circuit takes in the outputs from Booth encoder and also inputs from the other data input. This produces a 16-bit partial product, the partial products are then sign extended, and depending on the sign of the partial product, the Y_{j-1} term is added below the 0 term of the partial product (this completed the 2's compliment if negative). A 16-bit version of this can be seen below in the figure [2].



A_9 . . .	A_9	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
B_9 . . .	B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0		Y_1
C_9 . . .	C_6	C_5	C_4	C_3	C_2	C_1	C_0		Y_3		
D_9 . . .	D_4	D_3	D_2	D_1	D_0		Y_5				
E_9 . . .	E_2	E_1	E_0		Y_7						
S_{15}. . .	S_{10}	S_9	S_8	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

Figure 3: Sign Extension of Partial Products

Wallace Tree Adder

Carry Ripple Adder

A Carry Ripple Adder (CRA) is the basic the basic building block for the Carry Compression Adders described in the next sections. A Full Adder takes three inputs, and generates two outputs, a sum and a carry out. The implementation of a general 1-bit adder can be seen below in Figure 4 [3].

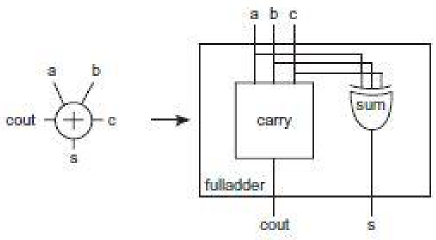


Figure 4: Full-Adder Implementation

In the SPD3L the inputs on the PUN require a monotonically rising input; this means for circuits that require both a logic input and the inverted signal of that input signal, it is required to construct a dual-rail implementation. In the case when both A and !A are needed in a PUN, both the A and !A need to be ensured to be monotonically rising. If these are cascaded from a previous stage it cannot simply be put through an inverter, but instead needs to be outputted from a previous stage SPD3L circuit. This causes the full adder to change slightly. This means the implementation of a full adder will be slightly altered. To cascade the full adders, a dual-rail for each of the outputs is implemented. This can be seen in the block diagram below in Figure 5.

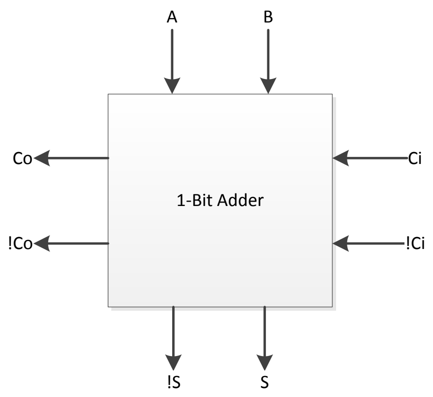


Figure 5: 1-Bit Full Adder D3L/SPD3L Block Diagram

[3:2] Carry Save Adder

A [p:2] Carry Save Adder (CSA) is a way to take p bit-vectors and reduce them to just 2 bit-vectors(VS, VC). After they are reduced to the 2 bit-vectors they can be put through a full adder to get the sum and the carry out. A [3:2] Carry Save Adder (CSA) is simply a modified full adder, in our case it takes in three 32-bit partial products and returns two 32-bit vectors. To implement the entire 32-bit adder implementation of the [3:2] CSA, the 1-bit full adder described in the previous section is replicated for each input bit. The CSA allows for the three bits to be added together through a full adder and to generate a VS, and a VC that carries twice the weight. That is it is shifted to the left one bit than the generated sum. The picture below in Figure 6 shows the implementation of a [3:2] CSA [4].

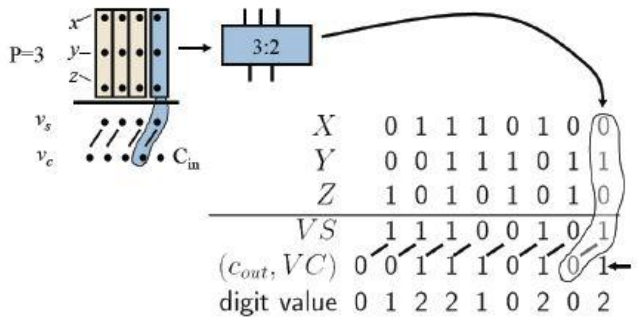


Figure 6: [3:2] Carry Save Adder Implementation

[4:2] Carry Save Adder

A [4:2] CSA is simply a modified full adder, in our case to take in four 32-bit partial products and return two 32-bit vectors. The CSA allows for the four bits to be added together through a full adder and to generate a VS, and a VC. In a [4:2] CSA there are two levels to it. These levels implement the [3:2] CSA, the first level takes 3 of the inputs, and generate the typical output for a [3:2] CSA. In the second level the fourth input is added to a sum from the previous level, and a carry from the previous bit of the previous level. The picture below in Figure 7 shows the implementation of a [4:2] CSA [4].

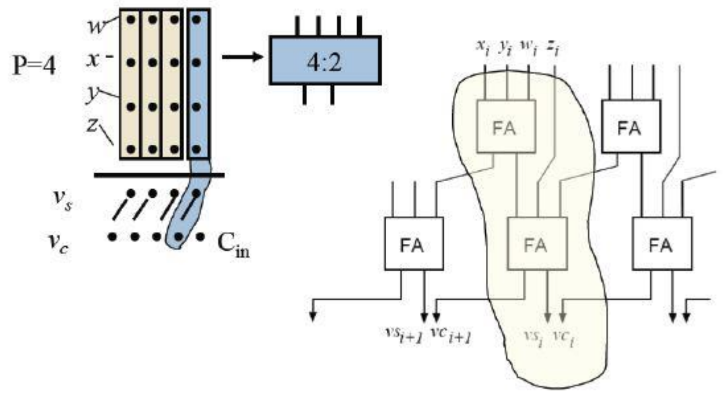


Figure 7: [4:2] Carry Save Adder Implementation

Linear/Tree Structure

The algorithm for a Radix-4 Booth Multiplier creates 9 partial product vectors outputted from the PPG blocks. To reduce the 9 operands down to 2 operands a 3-level Tree/Linear implementation was used, utilizing three [4:2] compressors and a single [3:2] compressor.

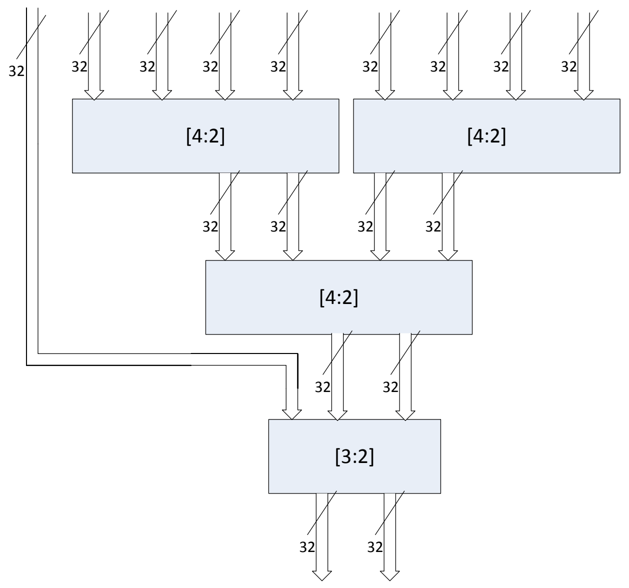


Figure 8: Wallace Tree Adder Implementation

Parallel Prefix Adder (Kogge Stone Adder)

The Kogge-Stone tree adder is an algorithm to add two numbers together in a way that the carry chain is reduced, making the Kogge-stone adder a very fast adder design. The adder consists of three different stages, the PG-Cell, the kogge-stone, and finally the final sum. The overall implementation of the PPA can be seen below [1].

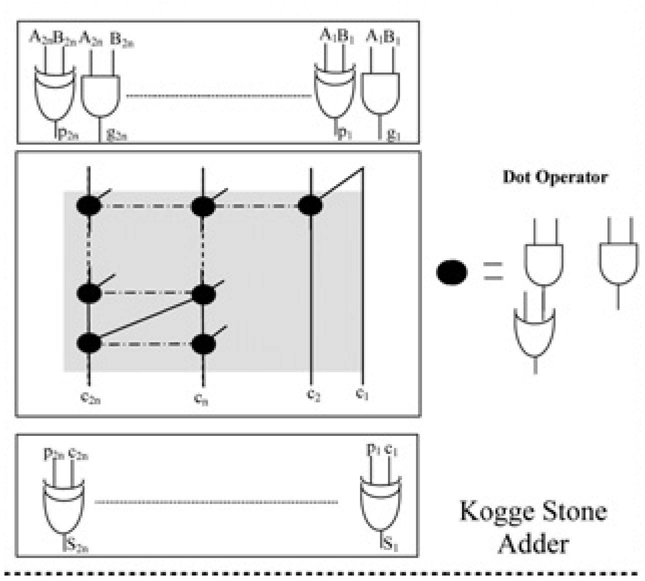


Figure 9: PPA Implementation

The second level or the kogge-stone implementation consists of grey and black cells, which can be seen above in Figure 9 on the right hand side (Dot Operator). For a more detailed flow of the kogge-stone adder, the following figure[3] was used and extended for the 32-bit adder implemented in this project.

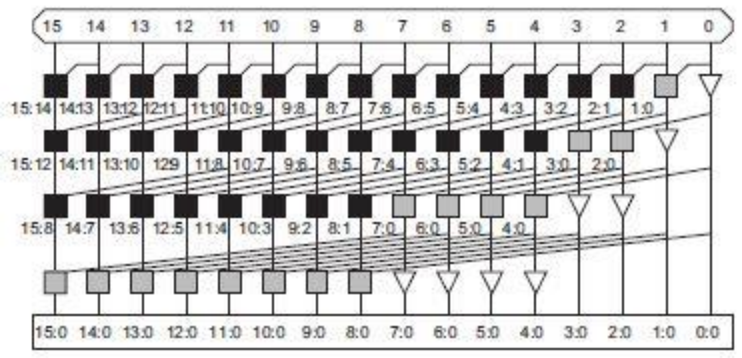


Figure 10: Detailed Kogge-Stone Adder

Circuit Details

In general, NAND2 and NAND3 gates that combine the split paths in the circuits were implemented using normal CMOS logic. When a single path was needed, a typical INV gate was also implemented using normal CMOS logic. The rest of the logic is all implemented in SPD3L technique.

Booth Encoder

In [1], they detail a POS signal coming from an inversion of the Y_{j+1} signal, since both the inverted signal and the non-inverted signal already need to be supplied to Booth Encoder for the other logic, this signal was already taken care of.

The logic to produce the ONE (and ONE_b) signal was an XOR_DR gate, which can be seen below in Figure 11.

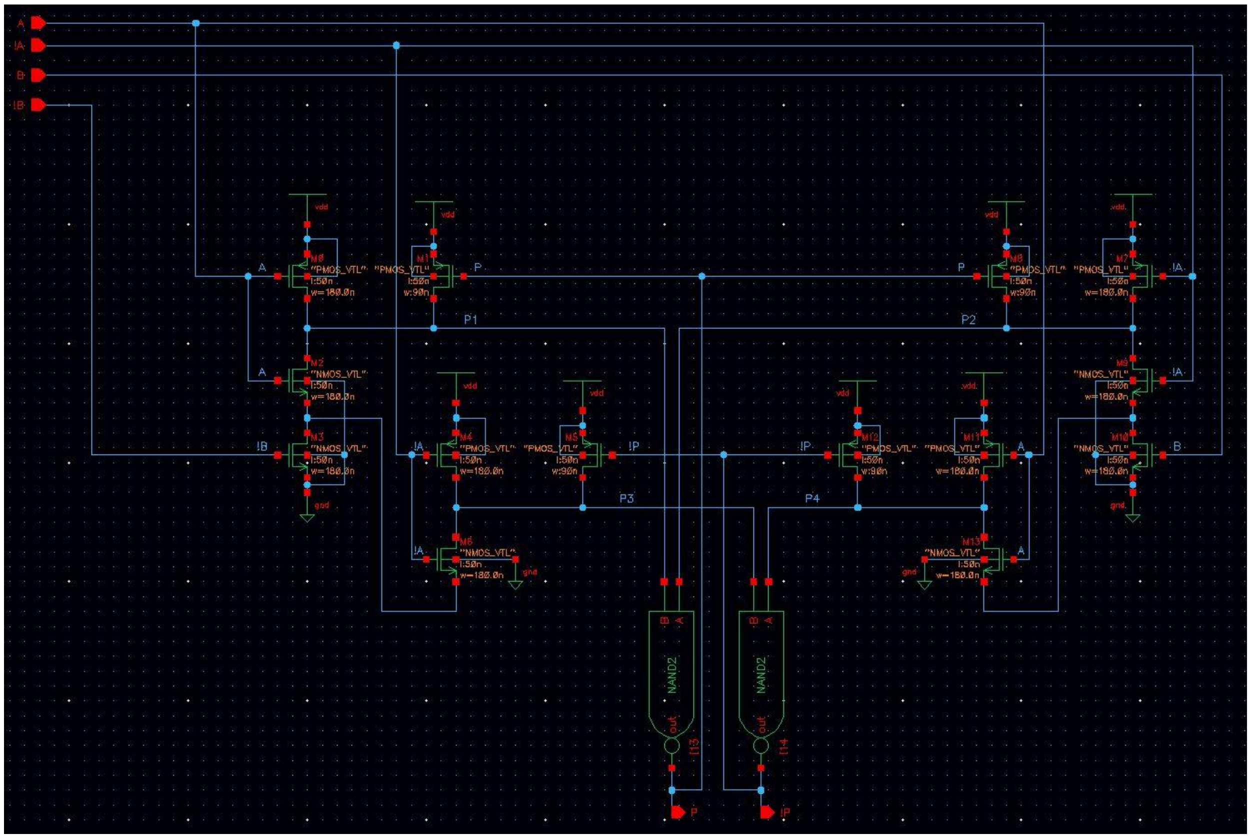


Figure 11: 2-input XOR/XNOR

The logic to produce the TWO (and TWO_b) signal was a combination of 3-input AND, and a 2-input OR gate (both gates producing a dual-rail). Figure 12-14 show the circuits for 3-input AND, 3-input NAND, 2-input OR, and 2-input NOR respectably.

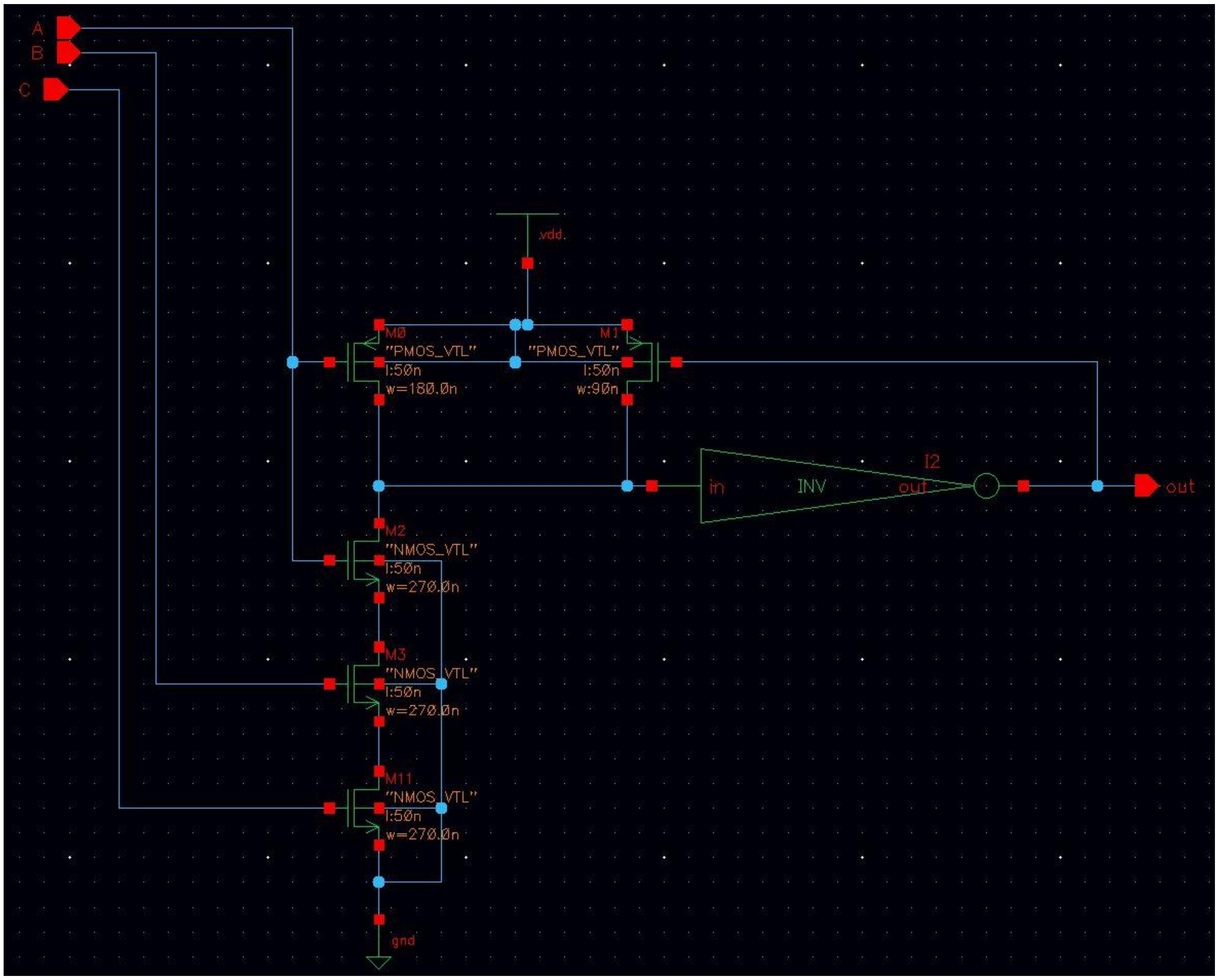


Figure 12: 3-input AND

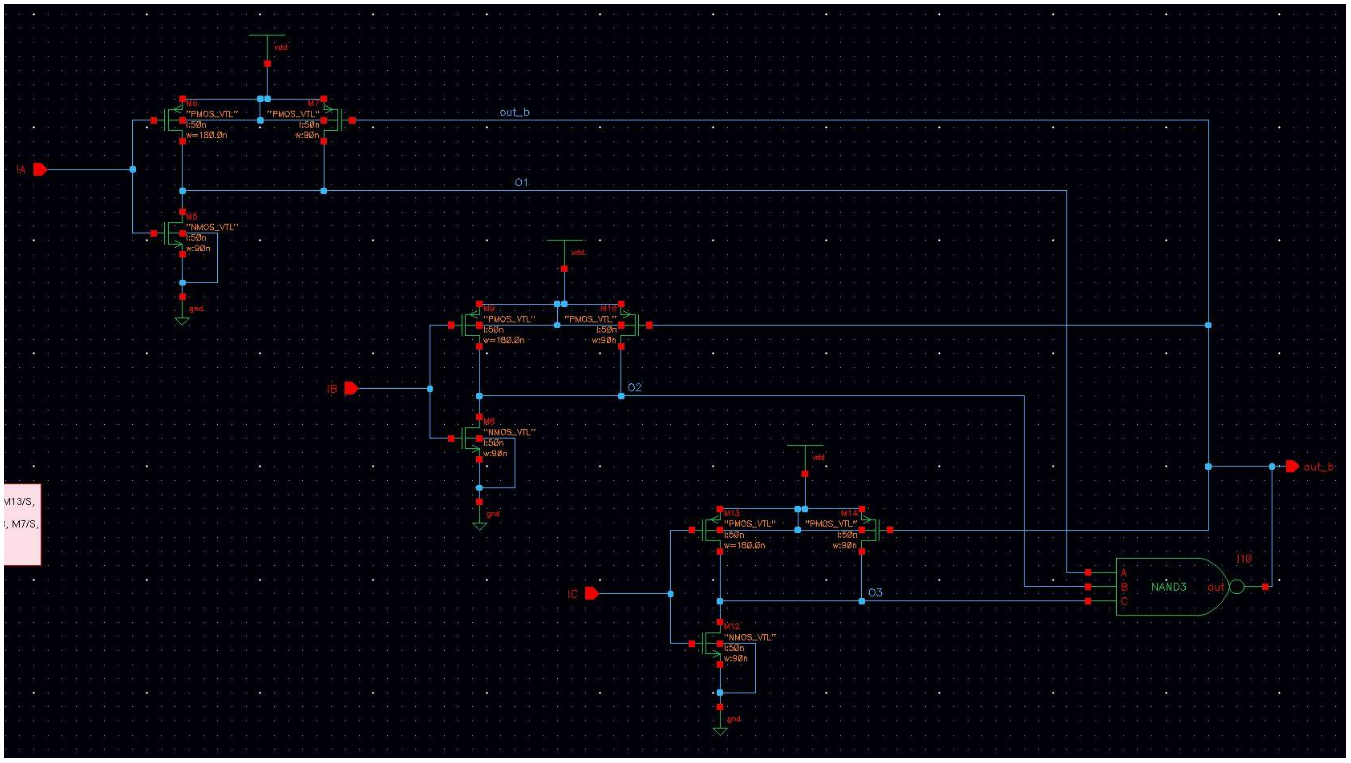


Figure 13: 3-input NAND

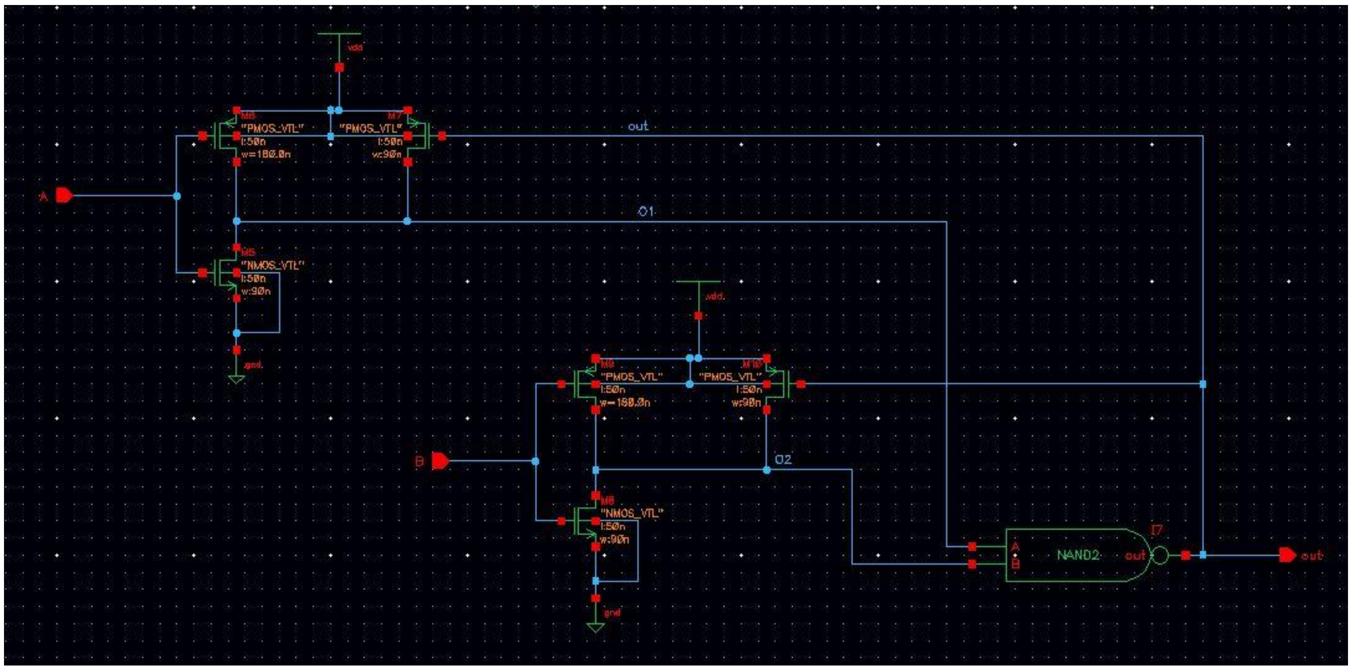


Figure 14: 2-input OR

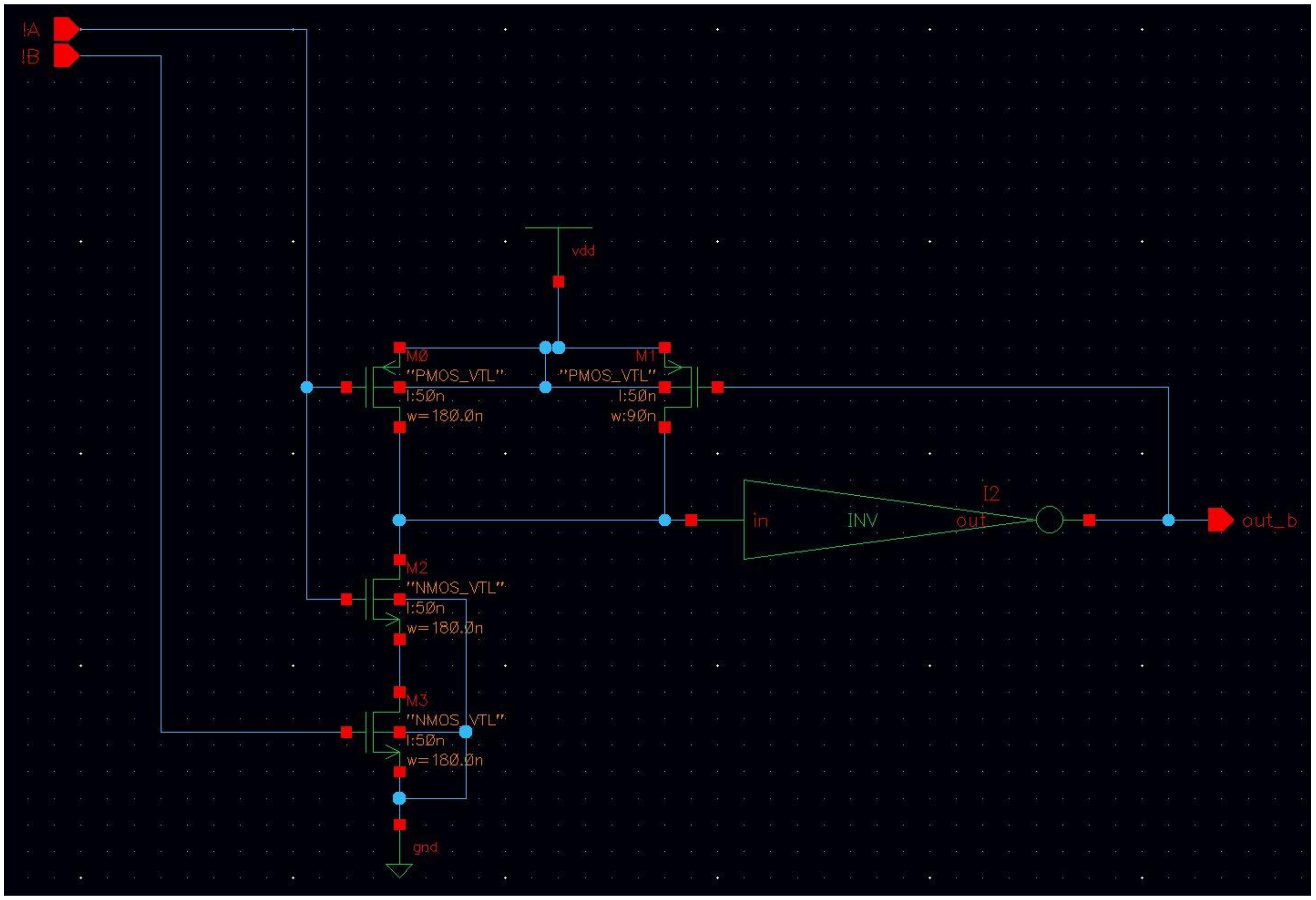


Figure 15: 2-input NOR

Finally these circuits were combined into the Booth Encoder Cell. Which can be seen below in Figure 16.

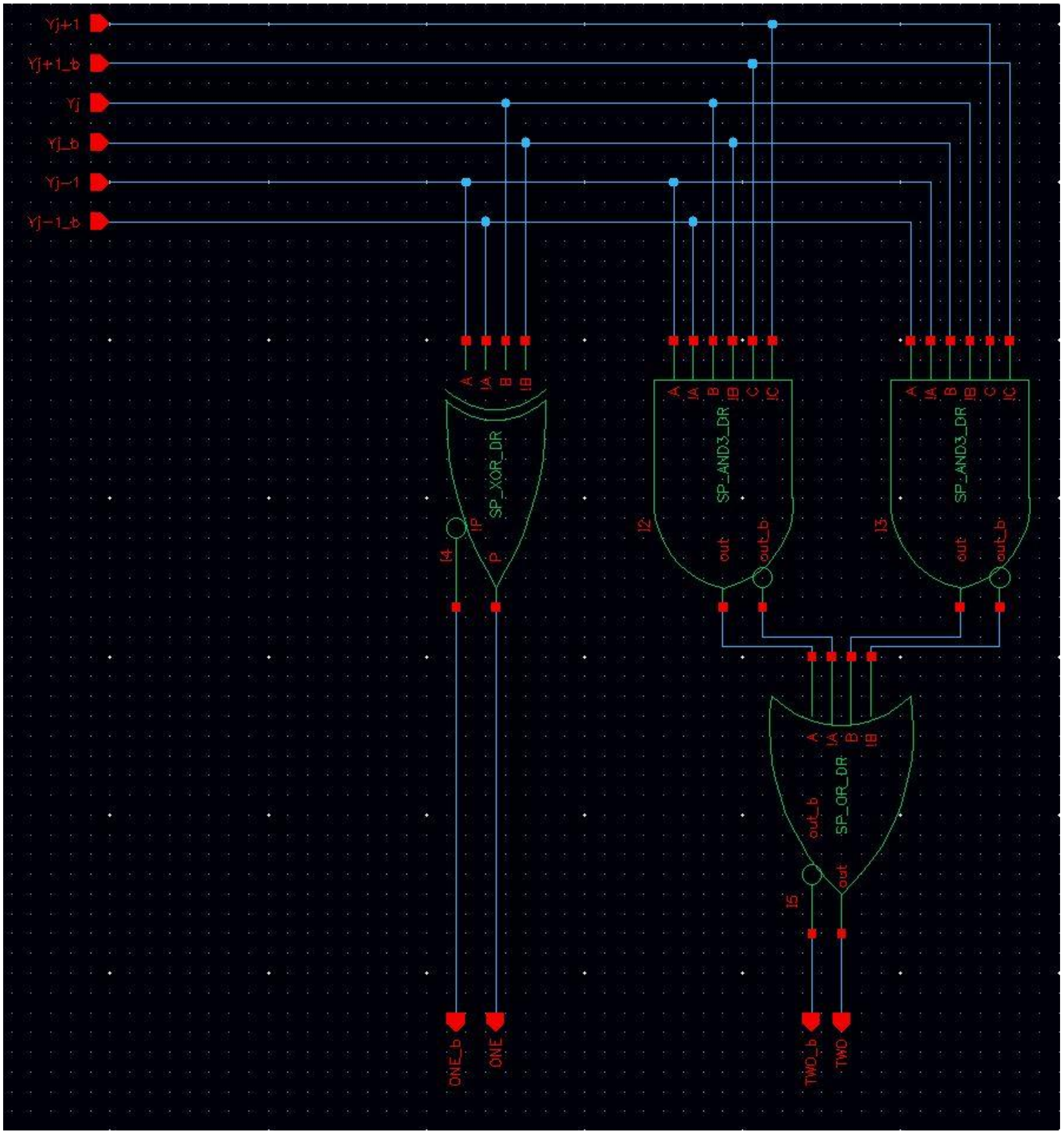


Figure 16: BE Cell

Partial Product Generator

The PPG consists of 2-input AND/NAND, 2-input OR/NOR, and 2-input XOR/XNOR gates. The 2-input OR, 2-input NOR circuits can be found in Figure 14 and 15, respectively. The 2-input XOR/XNOR can be found in Figure 11. The 2-input AND and NAND can be found below in Figure 17 and 18, respectively. And in Figure 19, the overall cell for the PPG can be seen. To form the PPG block for the entire bus, the PPG Cell was replicated multiple times.

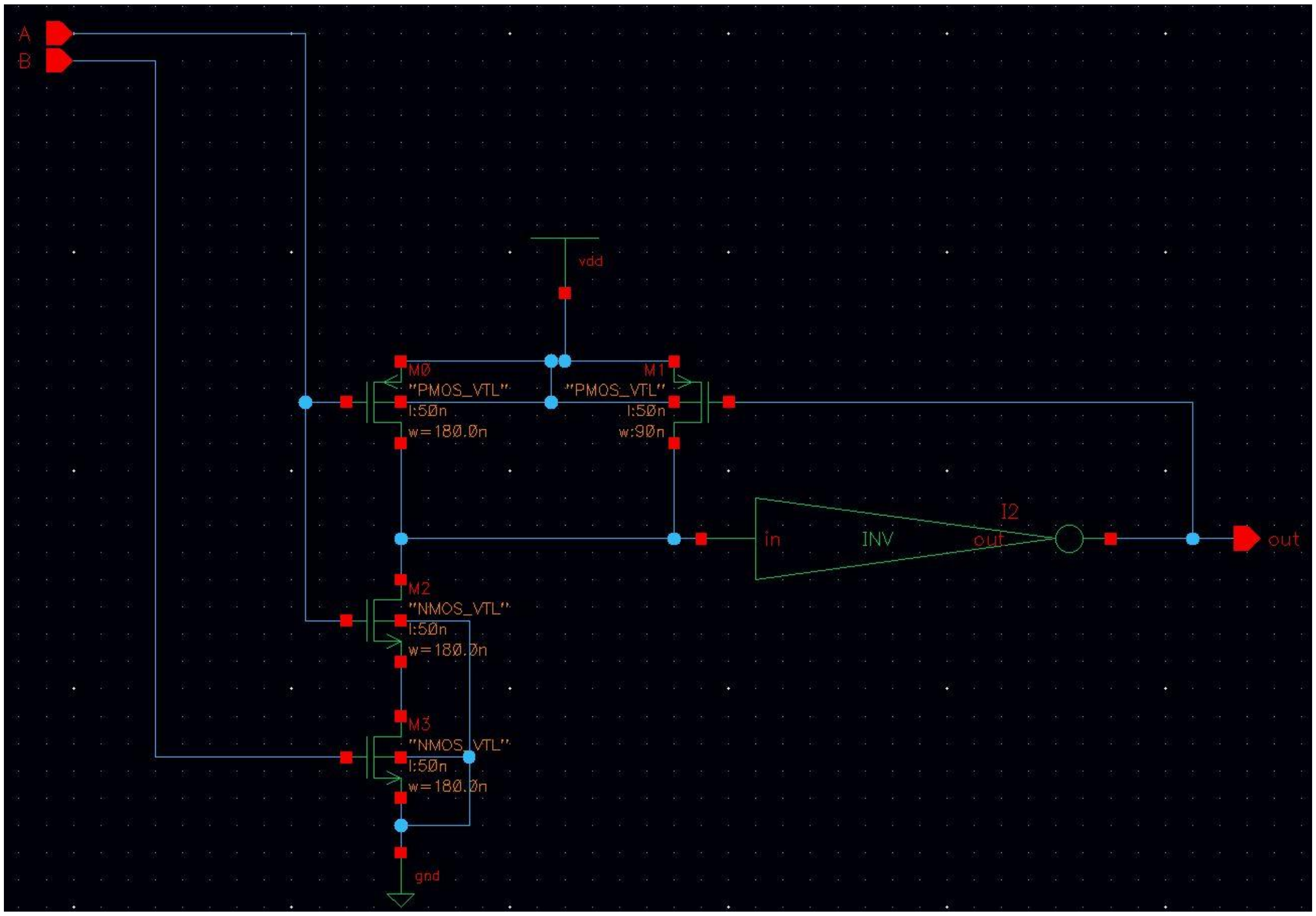


Figure 17: 2-input AND

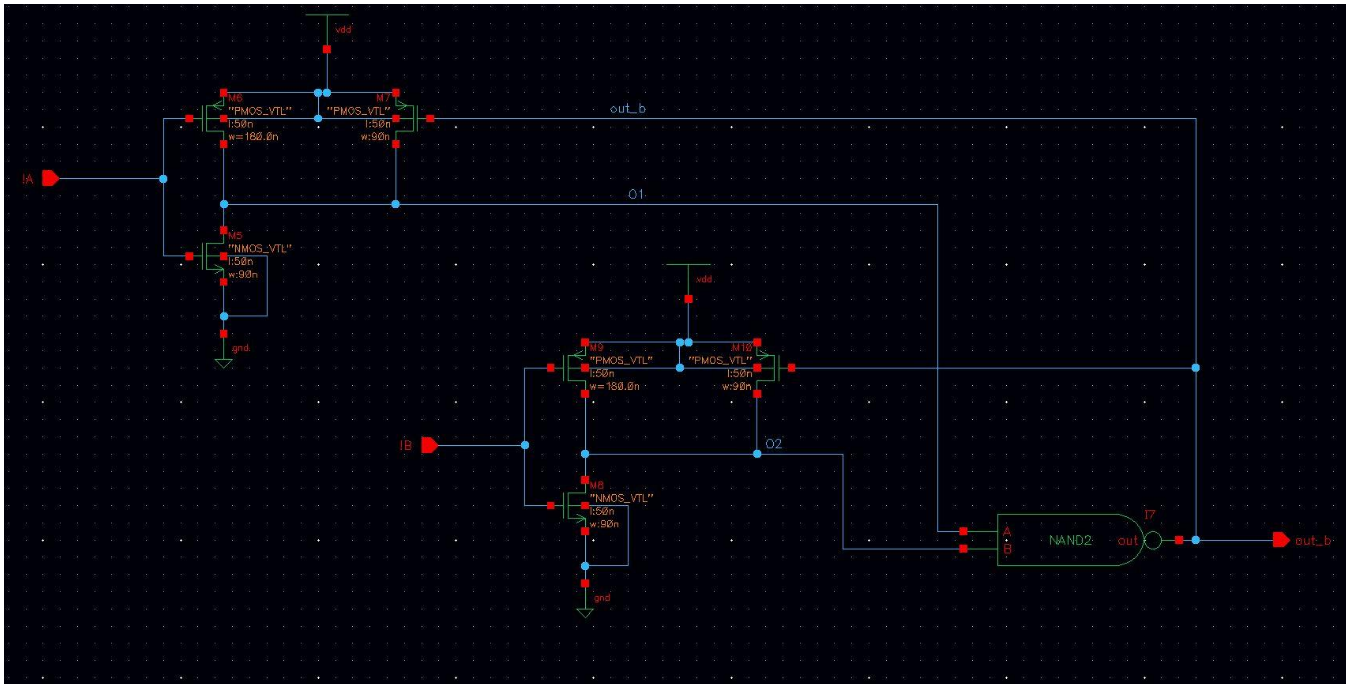


Figure 18: 2-input NAND

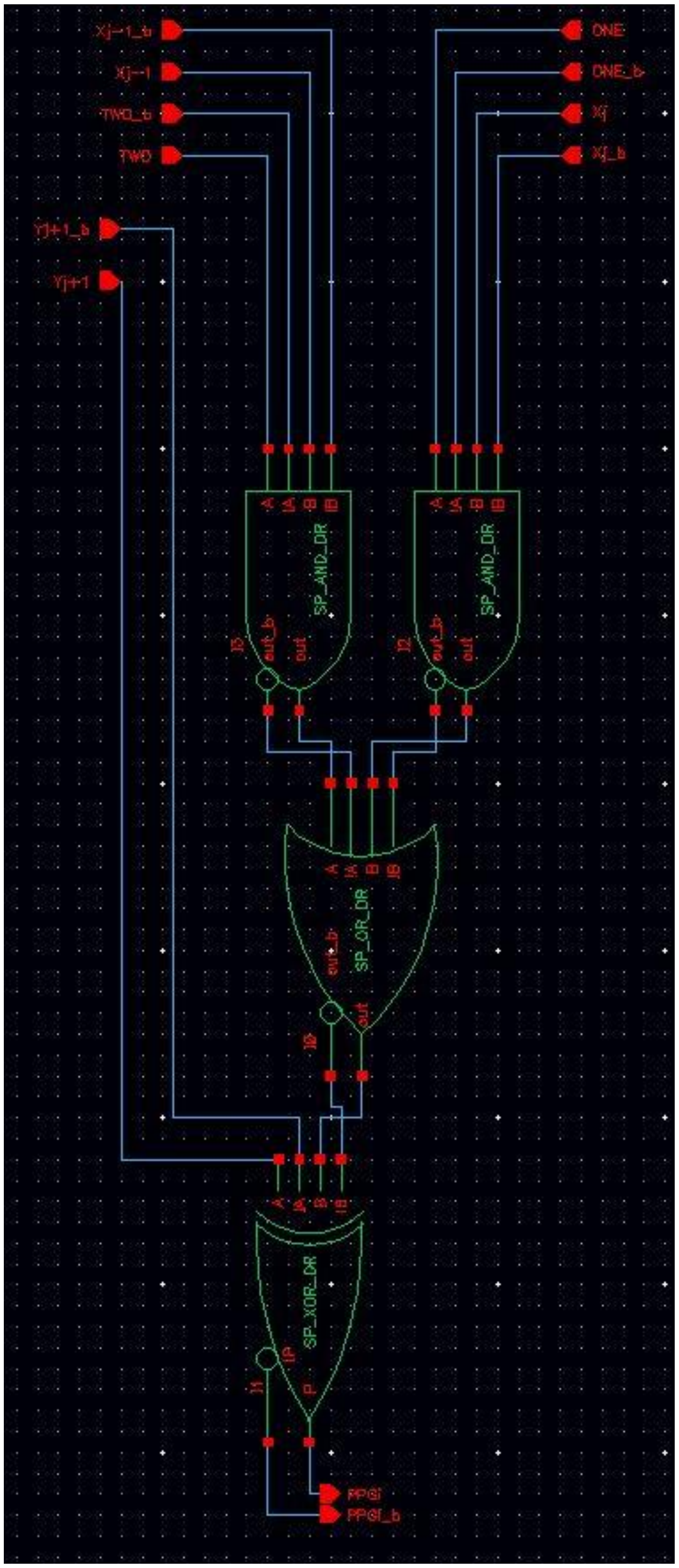


Figure 19: PPG Cell

Wallace Tree Adder

This is constructed of three [4:2] compressors, and a [3:2] compressor. Both of the compressors contain a full adder. The full-adder design consists of a 3-input XOR/XNOR Gate seen in Figure 20, and a Majority Circuit to calculate the Carry in Figure 21. Figure 22 shows the Dual-rail implementation of the Majority circuit to provide both the Co and Co_b, and finally Figure 23 shows the Dual Rail Full Adder combined into the full circuit.

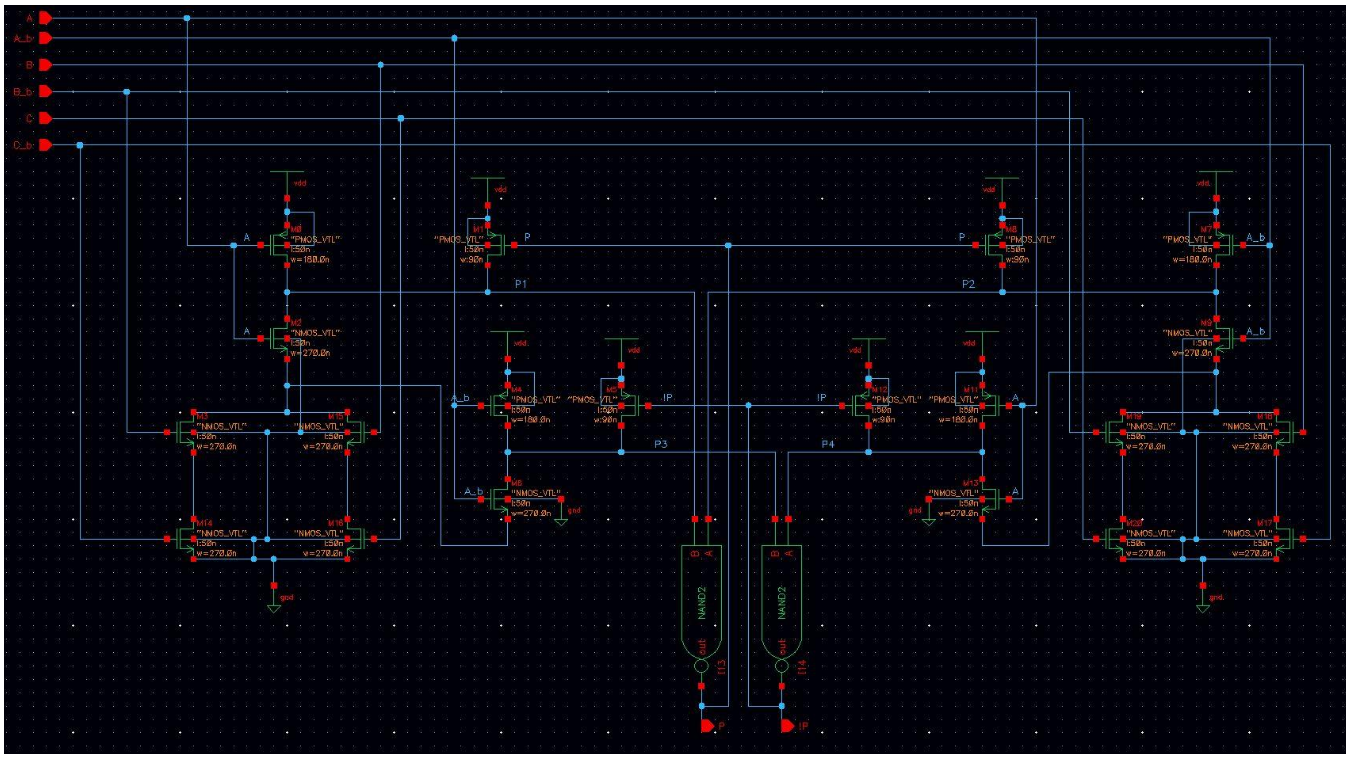


Figure 20: 3-input XOR/XNOR

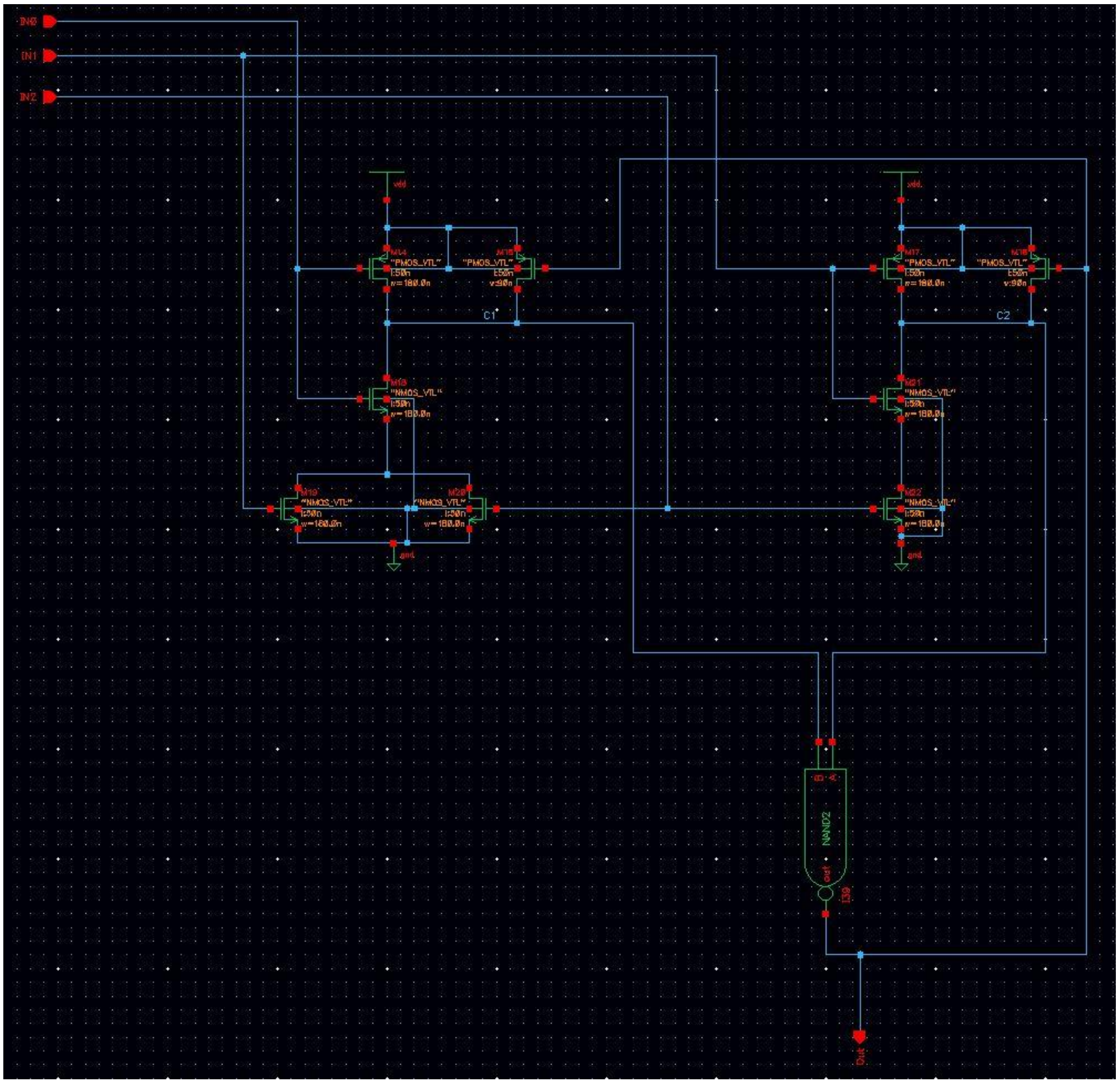


Figure 21: Majority Circuit

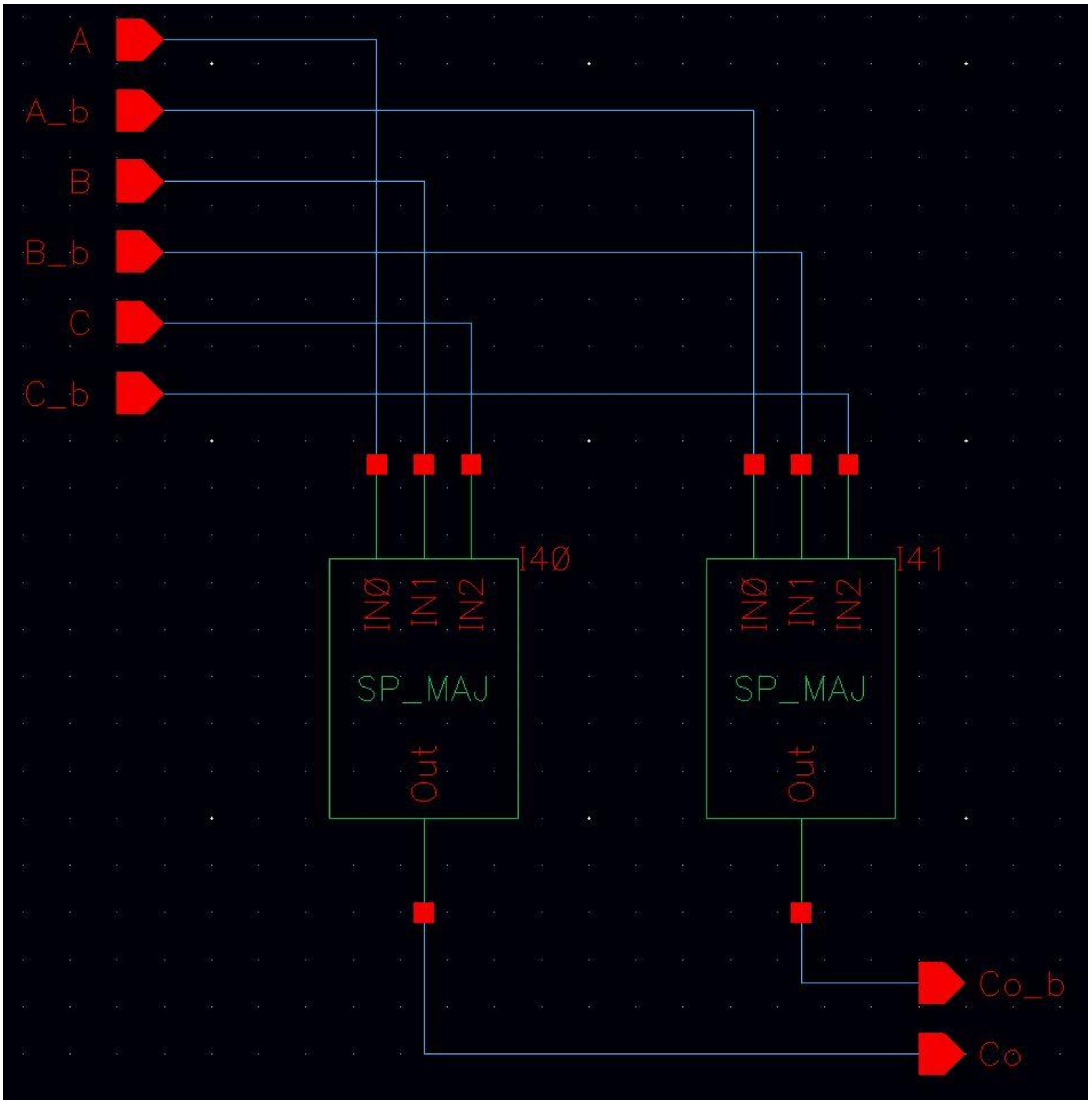


Figure 22: Dual-Rail Majority Circuit

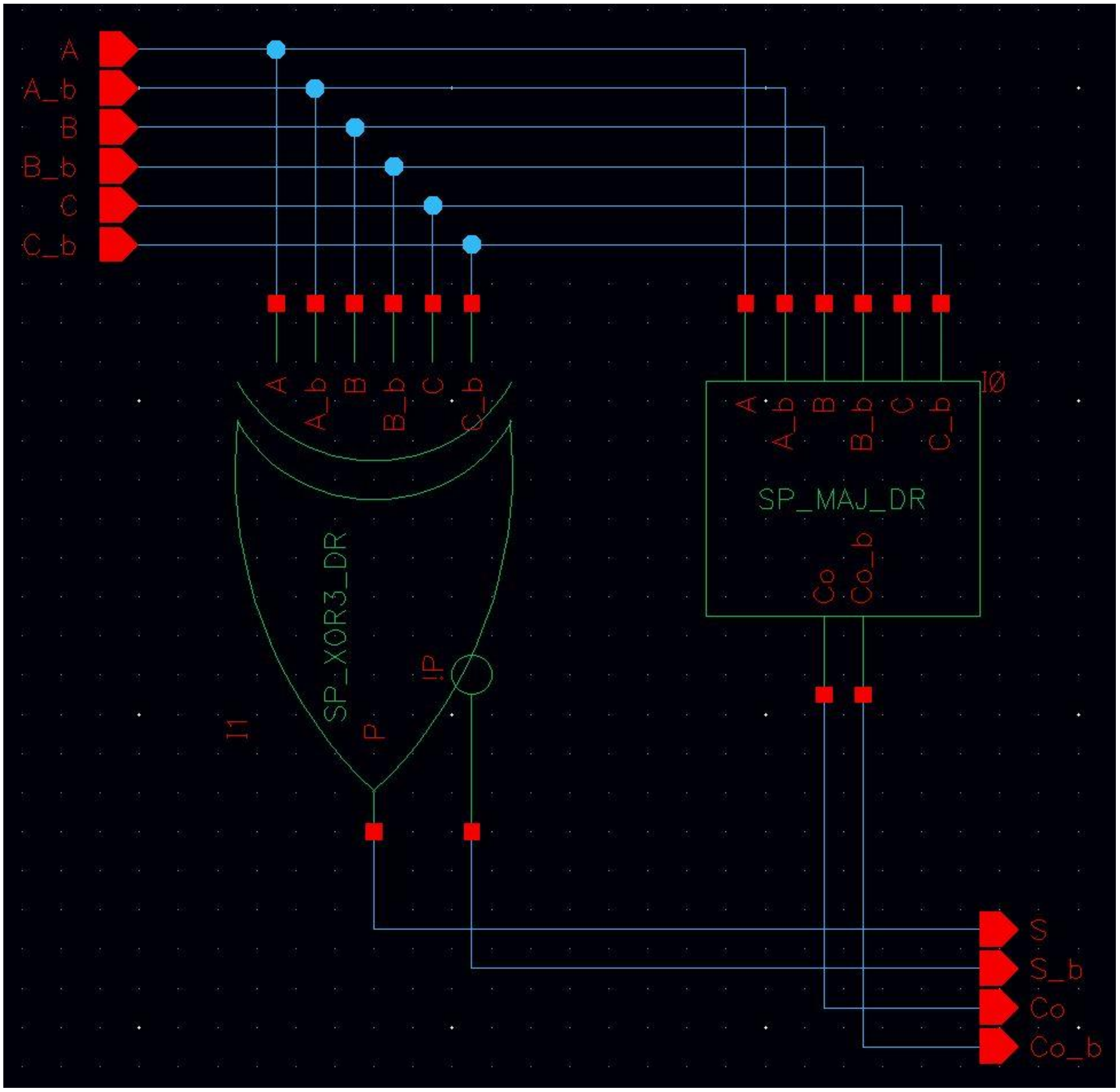


Figure 23: Dual-Rail Full Adder

The [3:2] compressor was constructed as detailed in Figure 6, and the [4:2] compressor was constructed as detailed in Figure 7. And finally the entire Wallace Tree implementation of 9-operand addition can be seen in Figure 8.

Parallel Prefix Adder

The first stage of the PPA, the initial P and G signals are constructed from a 2-input XOR/XNOR gate, seen in Figure 11, and a 2-input AND/NAND gate, seen in Figure 17 and 18. The Full PG Block can be seen in Figure 24. The second stage consisted of Grey and Black Cells. The Grey Cell Schematics can be seen in Figure 25 and 26, and Black Cell, which consisted of a Grey Cell, and a 2-input AND/NAND gate seen in figure 17 and 18, can be seen in Figure 27. And finally the third stage of the PPA was simply a 2-

input XOR circuit seen in Figure 28. These circuits were then used to constructed the entire PPA seen in both Figure 9 and Figure 10.

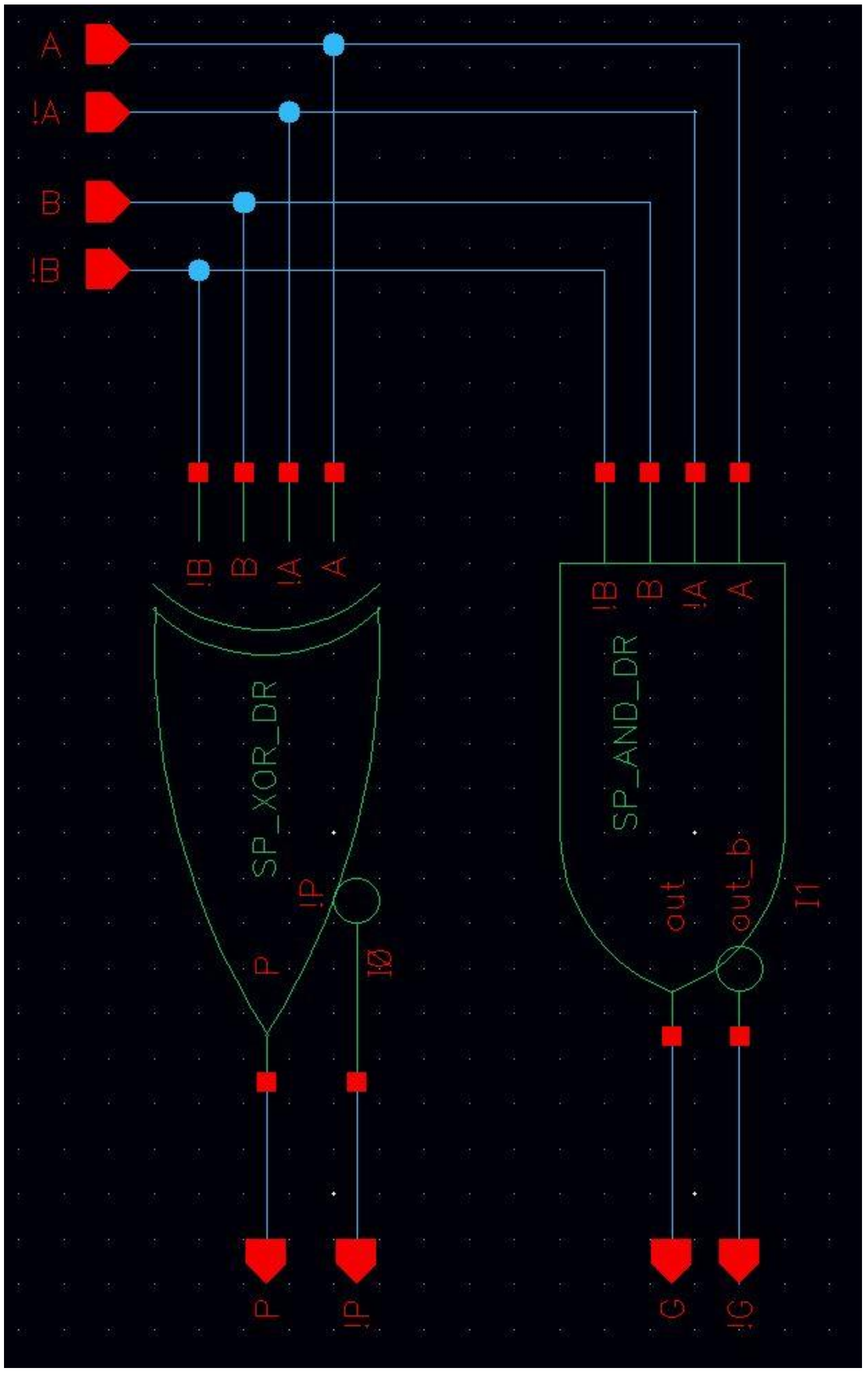


Figure 24: PG Block

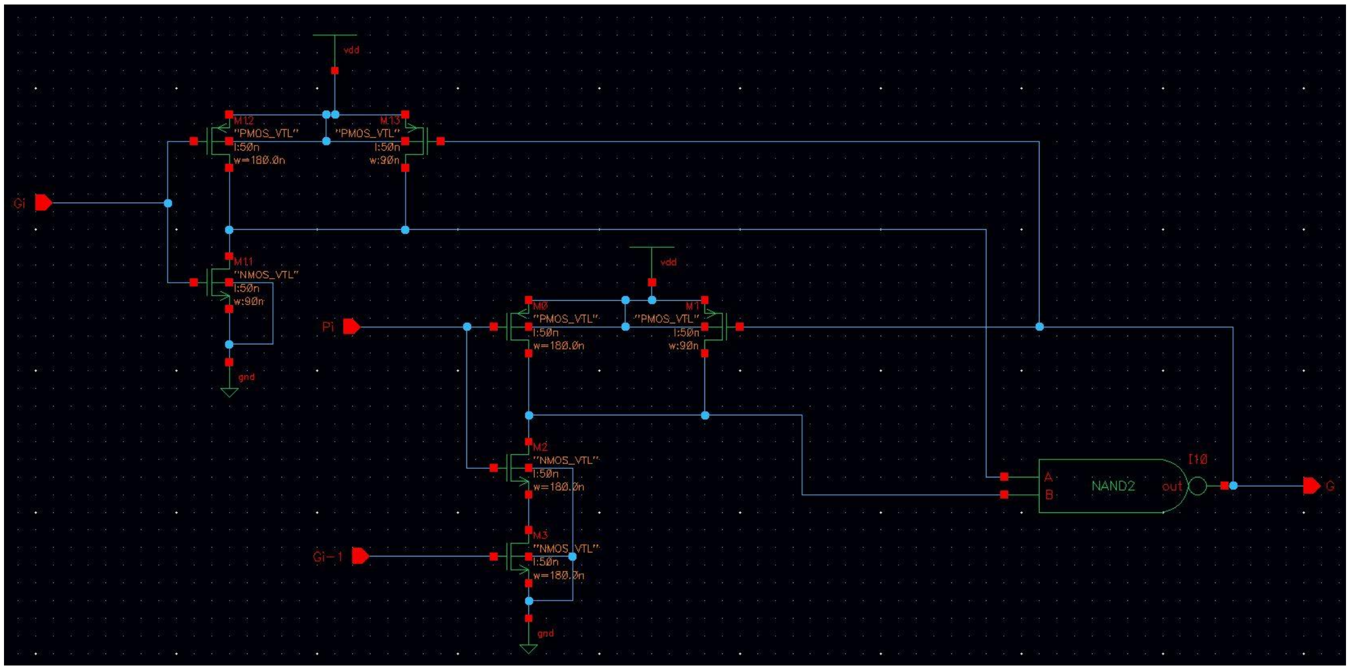


Figure 25: Grey Cell

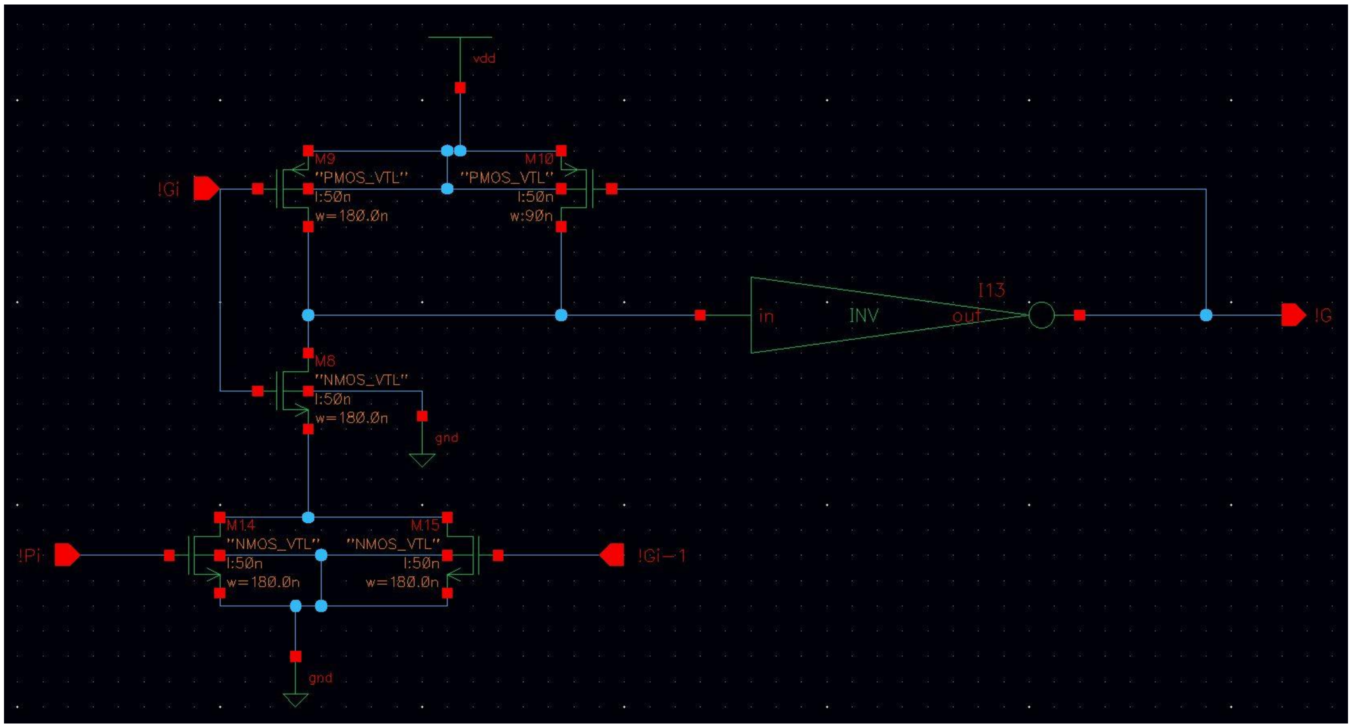


Figure 26: Grey_b Cell

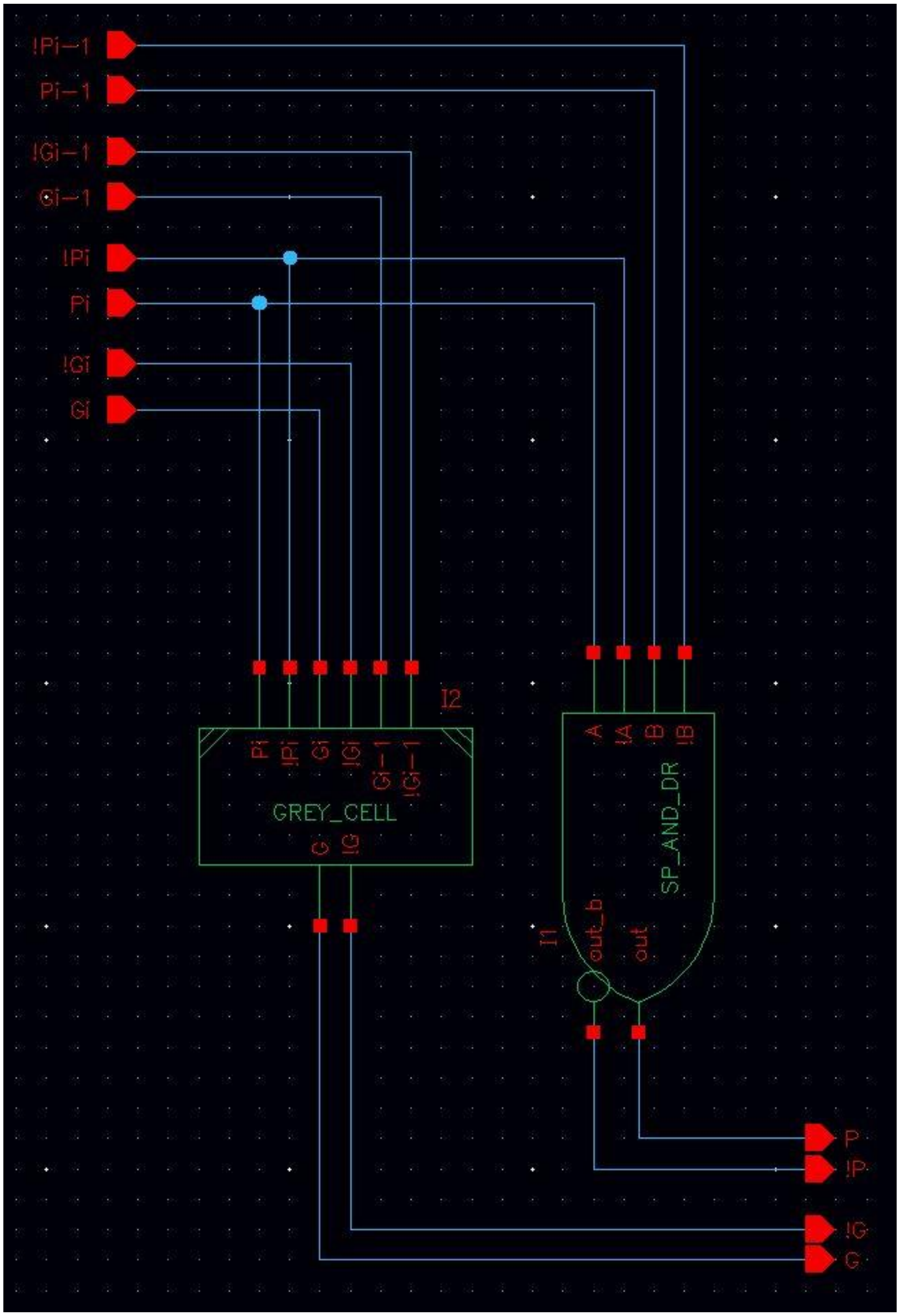


Figure 27: Black Cell

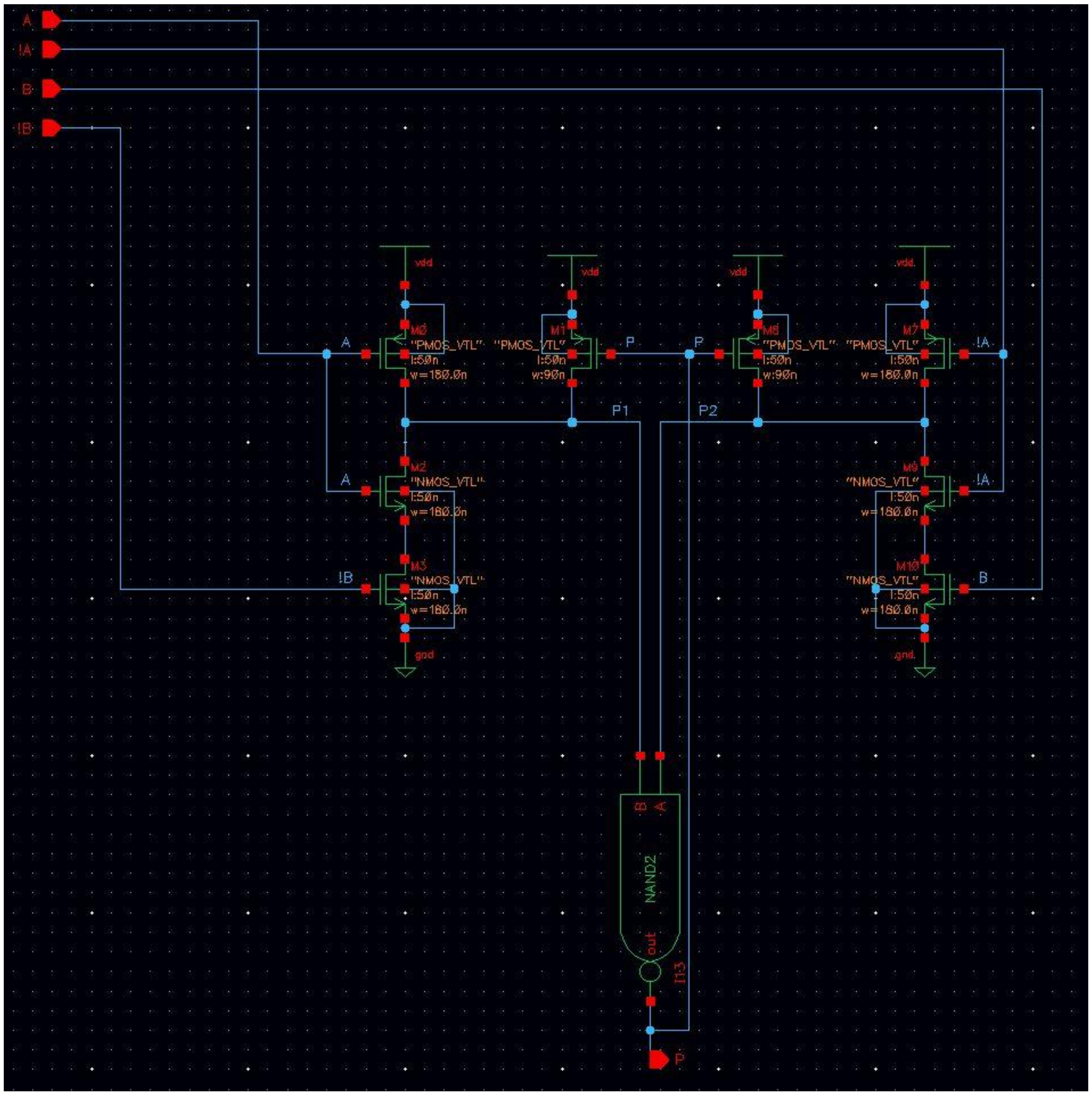


Figure 28: 2-input XOR

Functional Validation and Verification

Testing Methodology

For SPD3L to function correctly, instead of using a clock, a pre-charge signal is used. The circuit has to be pre-charged before it enters the evaluation mode. This is modeled by a PC signal. In general the timing diagram seen in Figure 29 is how the input stimulus should look like.

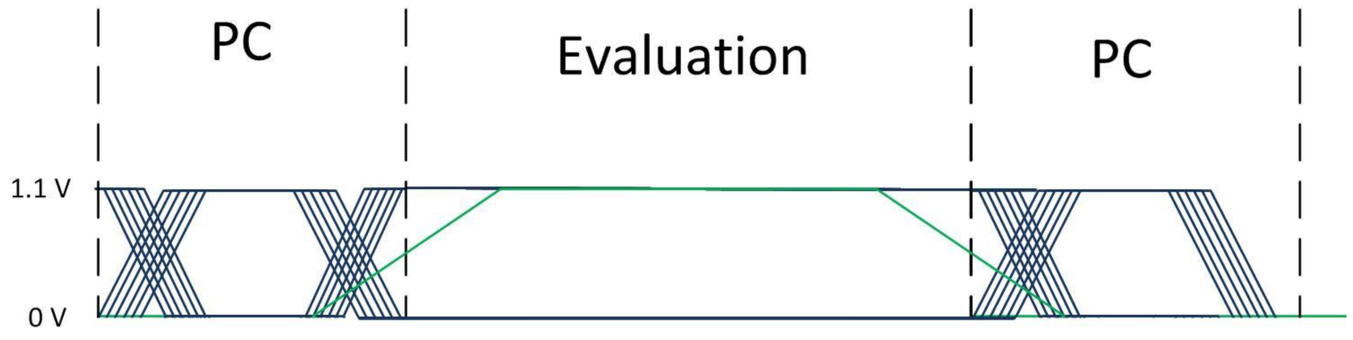


Figure 29: Input Signal Definition

The inputs (blue waveforms) are free to change during the PC (green waveform) modes due to the circuit being in pre-charge, the pre-charge circuitry is driving all of the inputs to 0. The pre-charge circuit is simply an implementation of an AND between PC and the Input stimulus. In order to ensure proper functionality of the circuit the inputs should not change during the Evaluation mode. This is shown in Figure 29, once the PC signal reaches the threshold of the transistor the inputs are held constant throughout the evaluation mode. After the evaluation mode the circuit goes back into Pre-charge and the inputs are then free to change.

In simulation the PC signal is 0 for the first 60ns of simulation, this gives the circuit enough time to propagate the pre-charge signal through-out the circuit. Figure 30 shows a single period of the multiplication circuit.

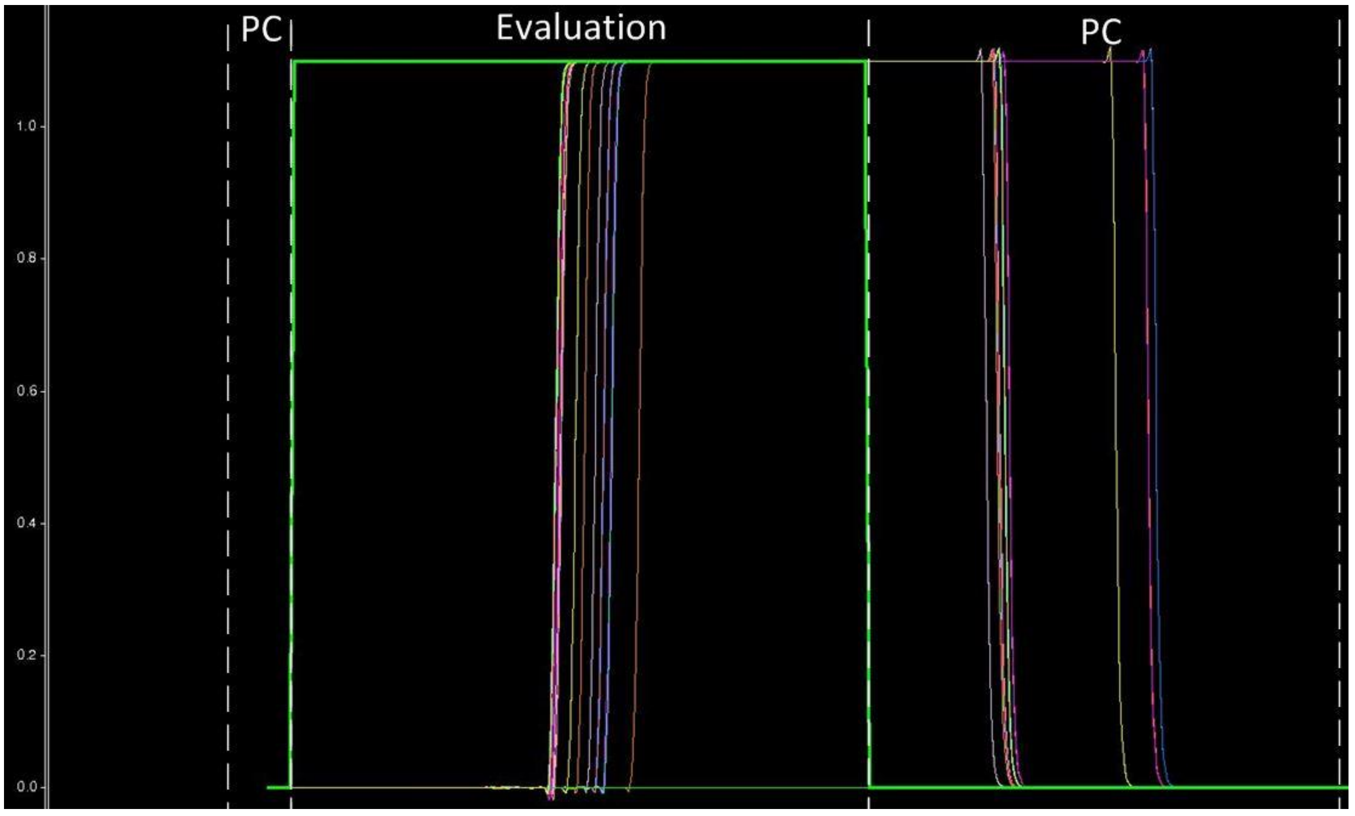


Figure 30: Output Signal Definitions

Parallel Prefix Adder

Schematic

For an intermediate step, the Kogge-Stone was tested to ensure functionality. Multiple inputs were tested with passing results, but since it was an intermediate step one was chosen at random to document. The test case being 0x386E 96BC + 0x610F 3278, resulting in an addition of 0x 997D C934. The output can be seen in the following figure.

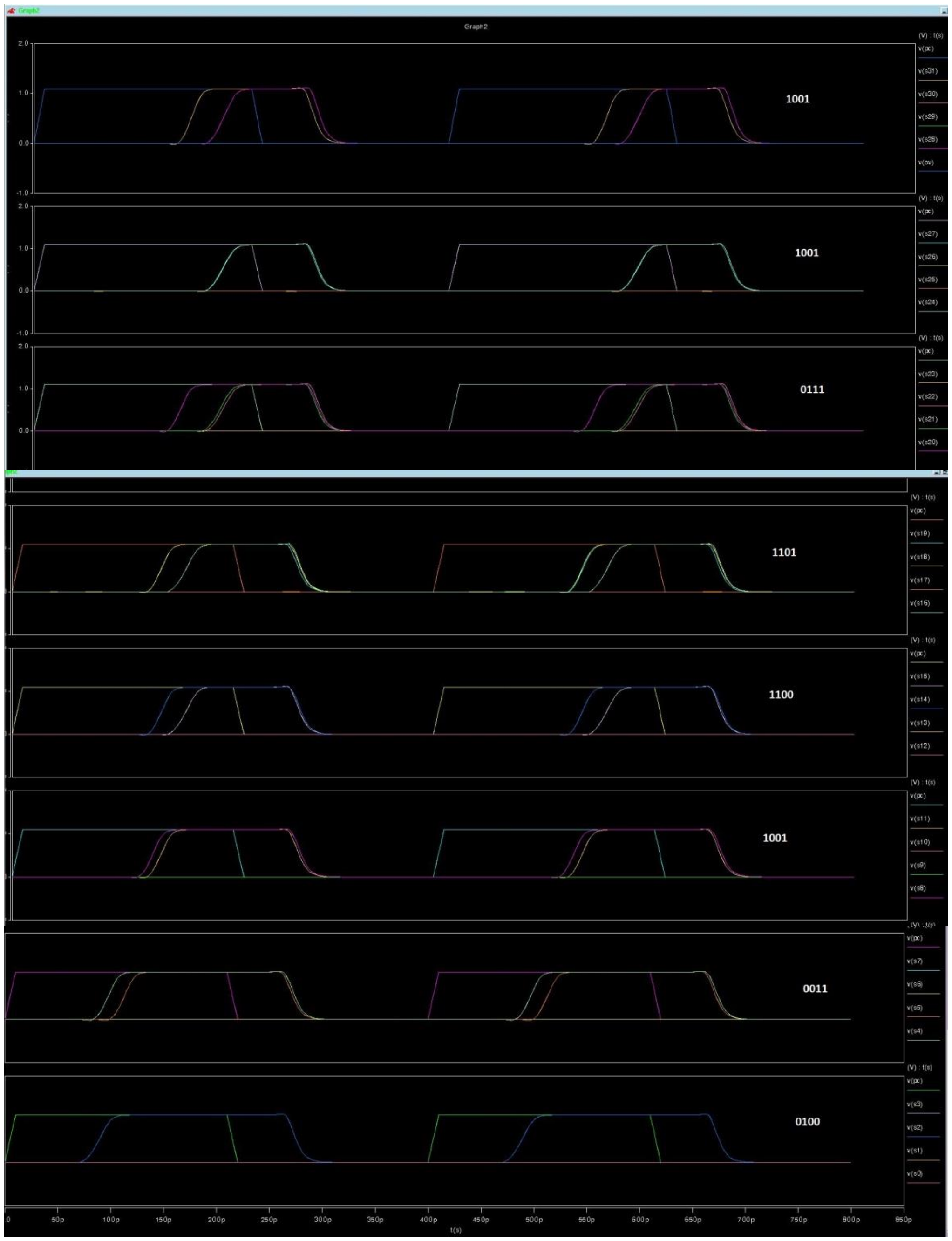


Figure 31: PPA Schematic Verification

Layout

The same test run for the schematic was run during after the circuit was done in layout. Figure 32 shows the output post-layout simulation.

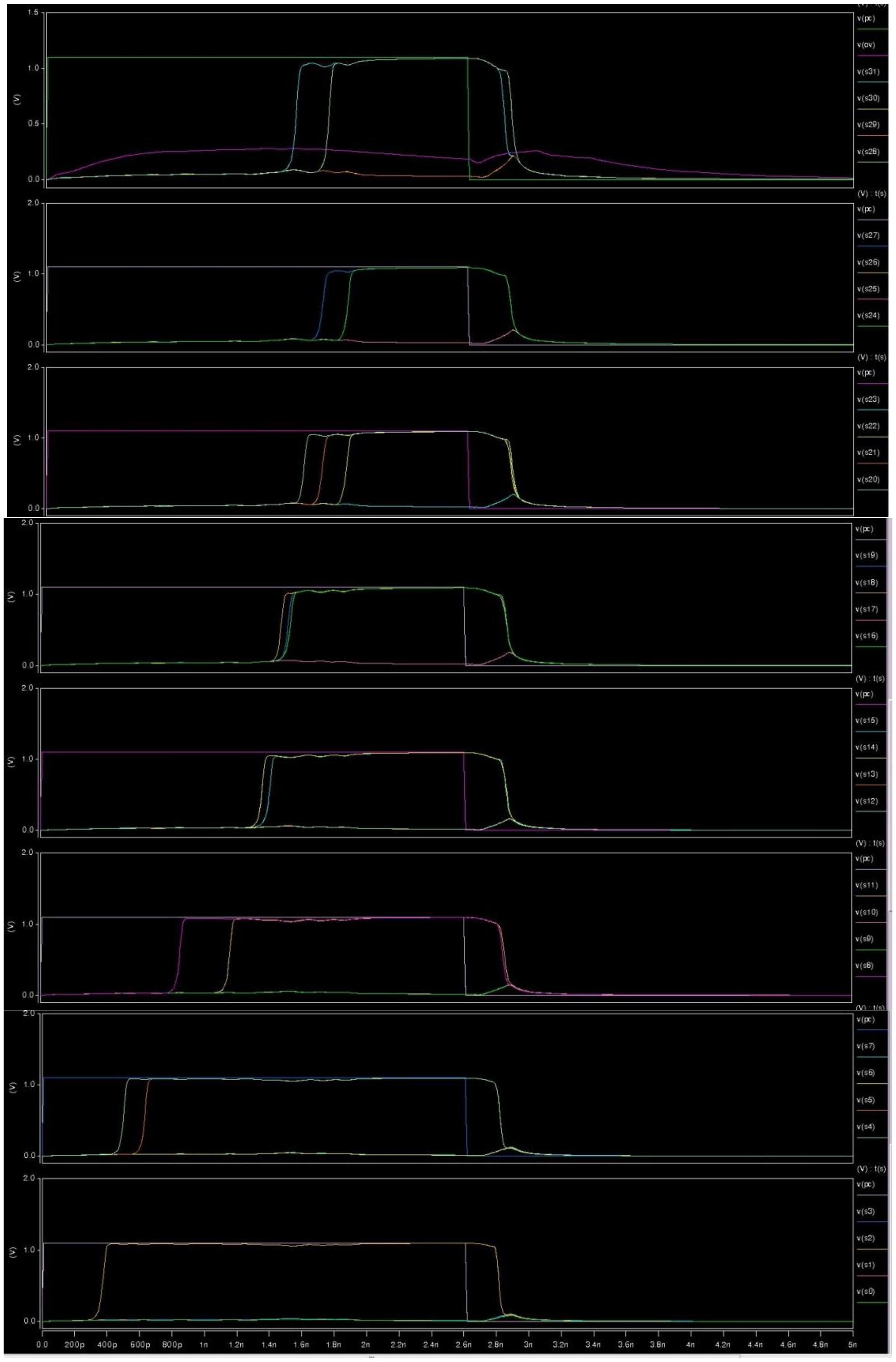


Figure 32: PPA Layout Verification

16x16 Radix-4 Booth Multiplier

For the final verification, a series of measure statements calculated the voltage of the output signals. These results were outputted into a .MT0 file which was post-processed using Microsoft Excel to turn the voltage calculations into binary data, which could then be interrupted as final output values.

Schematic

Using the post-processing of the .MT0 file, the following verification of the schematic multiplication was obtained for 10 randomly generated inputs.

Table 1: Multiplier Schematic Verification

X (Hex)	Y (Hex)	X*Y (Hex)	Expected (Dec)	Actual(Dec)
8811	E5F1	7A374D01	2050444545	2050444545
A5DC	21AF	15D2BD64	366132580	366132580
1693	6BAB	097E8531	159286577	159286577
9081	ED2A	85DF222A	2245993002	2245993002
55CA	A078	35C676B0	902198960	902198960
564A	324D	10F46842	284452930	284452930
5C20	AEE3	3EEF7060	1055879264	1055879264
8475	078E	03E8ABE6	65580006	65580006
1CFE	0B89	014E6DEE	21917166	21917166
304C	5B1B	11301C04	288365572	288365572

Layout

Table 2: Multiplier Layout Verification (coming soon)

Performance Results

Parallel Prefix Adder

Average Power

For each of the cases tested the average power was calculated in both schematic and layout for nine random inputs to the adder. The results can be seen in Table 3.

Table 3: PPA Average Dynamic Power Results

Case	Schematic Avg Pwr (mW)	Layout Avg Pwr (mW)
Random Case 1	0.61	1.3615
Random Case 2	0.78583	1.3551
Random Case 3	0.80882	1.3639
Random Case 4	0.77038	1.3649
Random Case 5	0.76068	1.3667
Random Case 6	0.61679	1.355
Random Case 7	0.66912	1.3468
Random Case 8	0.74724	1.3462
Random Case 9	0.6693	1.3334
Average Avg Pwr	0.715	1.355

Leakage Current

For calculating the leakage the inputs were held constant through simulation. This allows for the leaking transistors to be observed. For the PPA two cases were tested, all inputs asserted, and all inputs non-asserted. This gives us the cases where all pull up transistors leaking, and all pull down transistors leaking. The results can be seen in Table 4.

Table 4: PPA Average Static Power Results

Case	Schematic Avg Pwr	Layout Avg Pwr
All 0's (Pull Down Leaking)	0.58894	1.3402
All 1's (Pull Up Leaking)	0.8572	1.3737
Average Avg Pwr	0.723	1.357

Delay Results

The worst case for delay calculation in the PPA would be when the carry has to propagate through the entire circuit addition circuitry resulting in the OV signal being asserted. This test case is when 0x0000 0001 and 0x FFFF FFFF are added together resulting in the result of 0x 1 0000 0000. The max evaluation delay for the PPA would be from the rising edge of the PC signal to the rising edge of the OV signal. The signals can be seen below in figure 33, and the results can be seen in Table 5.

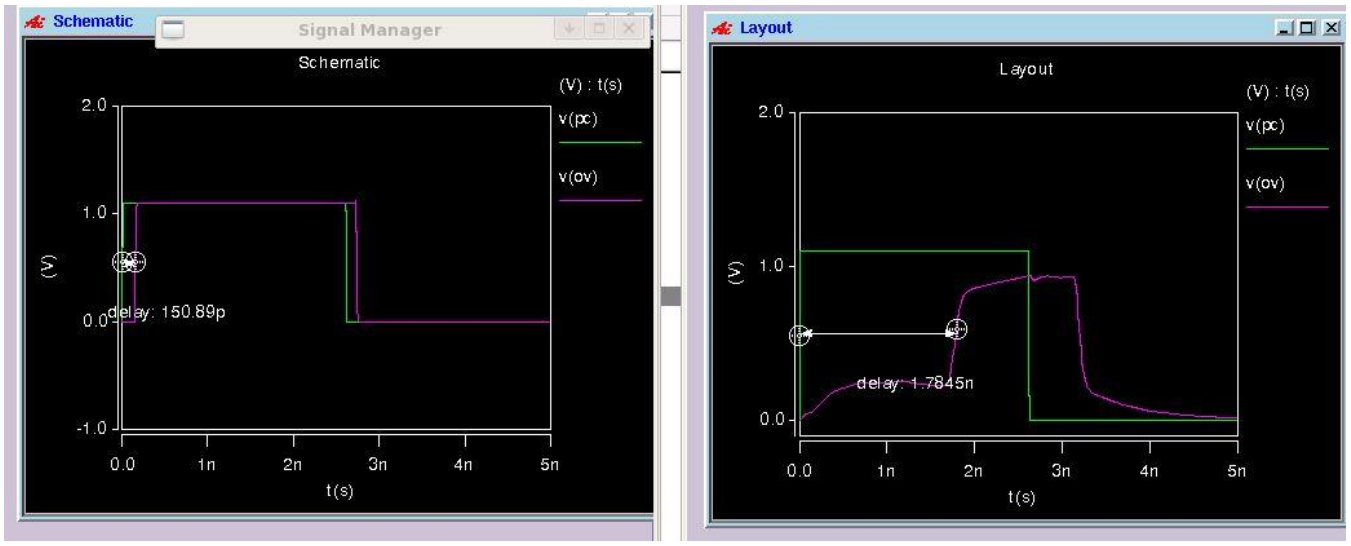


Figure 33: PPA Delay Waveforms

Table 5: PPA Delay Results

Case	Schematic Eval. (pS)	Layout Eval. (pS)
0x0000 0000 + 0xFFFF FFFF	150.89	1078

16x16 Radix-4 Booth Multiplier

Average Power

For the multiplier circuit, ten randomly chosen inputs were calculated and the average powers were measured through simulation. In the table below the results can be seen.

Table 6: Multiplier Dynamic Power Results

X (Hex)	Y (Hex)	Schematic Avg Pwr (mW)	Layout Avg Pwr (mW)
8811	E5F1	6.96	
A5DC	21AF	7.00	
1693	6BAB	7.31	
9081	ED2A	7.05	
55CA	A078	6.78	
564A	324D	6.74	
5C20	AEE3	7.03	
8475	078E	6.73	
1CFE	0B89	6.69	
304C	5B1B	6.78	
Average Avg Pwr		6.91	

Leakage Current

Similarly to the PPA, the leakage was measured while holding the inputs at constant value, this allowed for the measuring of leakage. Since the entire multiplier circuit consists of over 30,000 circuits it was very difficult to observe the true worst leakage case. Instead, similarly to the Dynamic power tests, ten randomly generated inputs were tested with constant input. The results can be seen in the table below.

Table 7: Multiplier Static Power Results

X (Hex)	Y (Hex)	Schematic Avg Pwr (mW)	Layout Avg Pwr (mW)
C647	C27A	6.96	
118B	23FF	7.05	
FF13	ED95	6.87	
94F4	8B0C	6.87	
830D	D8E1	6.76	
7E6A	9501	6.81	
36C2	1B89	7.27	
6C22	7C7C	6.69	
63FD	6A5E	6.75	
AA18	9B9C	6.80	
Average Avg Pwr		6.89	

Delay Results

For the delay results for the multiplier circuit a series of 25 random cases were chosen to observe the average evaluation delay, max evaluation delay, average pre-charge delay, and max pre-charge delay. The evaluation delay was measuring from the rising edge of the PC signal to the time when the final product was computed, and the max being the longest delay path. A similar methodology was taken for the Pre-charge signal. The circuit entered pre-charge again after the falling edge of the PC signal, and the circuit would be ready again when the last output was deserted, both the average and the max were calculated for the pre-charge time as well. These are summarized for the schematic results in Table 8 and in Table 9 are the final averages and max.

Table 8: Multiplier Schematic Delay Results

X (Hex)	Y (Hex)	Avg Ev Delay	Max Ev Delay	Avg PC Delay	Max PC Delay
8D72	5B9F	6.17E-10	6.42E-10	3.42E-10	3.94E-10
7A8E	37B3	6.02E-10	6.23E-10	2.89E-10	3.23E-10
D273	25A8	6.12E-10	6.21E-10	3.35E-10	3.79E-10
5F1E	1A69	6.67E-10	8.83E-10	3.44E-10	3.78E-10
D60D	9F02	6.69E-10	8.44E-10	3.03E-10	3.74E-10
6C71	0A3B	6.07E-10	6.15E-10	3.46E-10	3.89E-10
C021	C801	6.09E-10	6.36E-10	3.40E-10	3.90E-10
8E57	5D13	6.04E-10	6.19E-10	3.13E-10	3.74E-10
3C7A	465C	6.42E-10	7.10E-10	2.89E-10	3.27E-10
81AE	D4B0	6.22E-10	7.06E-10	3.55E-10	3.91E-10
36E3	B0D8	6.25E-10	7.27E-10	3.52E-10	3.96E-10
8C48	5F91	6.24E-10	6.42E-10	3.39E-10	3.92E-10
E9D3	DF19	6.32E-10	8.54E-10	3.21E-10	3.85E-10
4A1A	A817	6.60E-10	7.26E-10	3.10E-10	3.71E-10
3203	2C1E	6.85E-10	7.33E-10	3.51E-10	3.98E-10
70AE	97F8	6.93E-10	8.79E-10	3.03E-10	3.71E-10
2D79	8848	6.48E-10	8.18E-10	3.34E-10	3.83E-10
8370	45C9	6.04E-10	6.13E-10	3.45E-10	3.95E-10
B825	8F85	6.23E-10	6.37E-10	3.62E-10	3.91E-10
A209	34C6	6.22E-10	6.32E-10	3.31E-10	3.89E-10
BC63	612C	6.65E-10	8.92E-10	3.67E-10	3.82E-10
228D	663B	6.57E-10	9.08E-10	3.57E-10	3.90E-10
EA5A	A8BD	6.01E-10	6.14E-10	3.16E-10	3.74E-10
A0BA	3FC0	6.14E-10	6.29E-10	3.18E-10	3.90E-10
F4BB	7DA2	6.47E-10	8.81E-10	3.26E-10	3.78E-10

Table 9: Multiplier Layout Delay Results (Coming soon)

Table 10: Multiplier Combined Delay Results

Test	Schematic Time	Layout Time
Average Evaluation Delay	6.34E-10	
Max Evaluation Delay	9.08E-10	
Average Pre-charge Delay	3.32E-10	
Max Pre-Charge Delay	3.98E-10	

Conclusion and Future Work

References

[1] F. Frustaci M. Lanuzza P. Zicari S. Perri P. Corsonello, "Low-power split-path data-driven dynamic logic", IET Circuits Devices Syst., 2009, Vol. 3, Iss. 6, pp. 303–312

[2] EE 486 lecture 7: Integer Multiplication, M.J Flynn, Stanford University.

[3] CMOS VLSI Design: A Circuits and Systems Perspective. Neil Weste, David Harris. 2011.

[4] ECE 429, Fall 2012 ECE 429 Final Project Guide. Dr. Erdal Oruklu. 11/20/2012