Project Report: EEE 454 VLSI1 Group No: 27

Project Title: **Implementation of 8×8 Booth Encoded Multiplier**

Submitted By:

Student Name: Shamsuddin Ahamed

Student ID: 1506160

Student Name: Shadman Shahid

Student ID: 1506179

Date of Report: 21 – 09 - 2019

Table of Contents

1. Abstract:

A binary multiplier is an electronic circuit that is used in multiplication of two binary numbers in digital electronics such as computer. The traditional way of multiplication consists of multiplying each bit of multiplicand with multiplier and then summing up the partial products together. There are several algorithms that provide faster multiplication process than the traditional one. Booth's multiplication algorithm is one of them. In this project, we were assigned to design a '8 by 8 Booth encoded multiplier' using the Cadence Virtuoso platform.

The traditional multiplier needs to generate 8 partial products for multiplying two 8 bit binary numbers. Using Booth's multiplication algorithm, we can reduce it to 4. Thus the multiplication procedure becomes faster.

The main goal of this project is to design a circuit that implements Booth's multiplication process. Besides, minimizing the area of the layout is also a requirement. In multiplication process, partial products need to be added. For this purpose, adder circuit is used. There are several adder topologies available for instance Ripple- Carry, Carry- Look- Ahead, Carry- Select and Kogge- Stone. We were assigned to implement Ripple- Carry topology.

A Ripple- Carry adder is a logic circuit in which the carry-out of each full adder is the carry in of the succeeding next most significant full adder. It is called a ripple carry adder because each carry bit gets rippled into the next stage.

The provided block diagram was followed properly while designing the circuit. Each and every transistor level schematic of logic gates, building blocks of Booth encoder and decoder, adder circuit were checked and their functionalities were verified observing the timing diagram. Layout of each cell was made, DRC errors were removed using ASSURA and the layout was matched with corresponding schematic by LVS match. This DRC and LVS reports are provided in this report. We tried to minimize the total layout area in every possible way. The total area of layout is mentioned in this report.

2. Keywords:

Booth Multiplier, Virtuoso, Booth Encoding, Booth Decoding, Half Adder, Full Adder, Ripple Carry Adder, DRC - Design rule check LVS - Layout versus Schematic, Layout XL

3. Introduction:

In this project we have designed an 8x8 Booth-encoded Multiplier from the basic building blocks of digital logic circuits, i.e MOFETs. We have implemented the schematic and layout of the desired multiplier circuit.

A booth encoded multiplier multiplies two signed binary sequences and produces an output binary sequence. In processors multiplication is an importanct function and the effeciency of the process ensures faster processing in a processor.

By implementing this 8x8 Booth encoded multiplier in Cadence Virtuoso we have learned and furnished our skill in developing large scale integrated circuits using the associated software, i.e Cadence® Virtuoso® Analog Design Environment. We were able to work with this software for designing the entire circuit and getting a better understanding of the practical considerations that one has to take into account while working with this software. To design the schematic, we had to learn about the logic and algorithm involved in implementing the desired system. And thereafter, in designing the layout, we learned about the rules and cosiderations that one need to remain mindful of while designing. Also, with gradual practice, the design was minimized so that we could implement the system in a smaller space.

4. Theory

Booth's algorithm uses the fact that multiplication by a sequence of 1's can be computed simply with inversions and shifts. It is obviously simpler than addition operation.

Booths algorithm multiples two signed binary numbers by a repetitive process of adding and shifting. There are a few minor variations among a few of the Booth's algorithm. For the particular booth algorithm that we have implemented, is the most effecient one and uses minimum circuit elements.

Here Booth's algorithm for 8 bit by 8 bit multiplication is explained. The repetitive process can be extended for N bits. The algorithm is schematically depicted in the figure below:

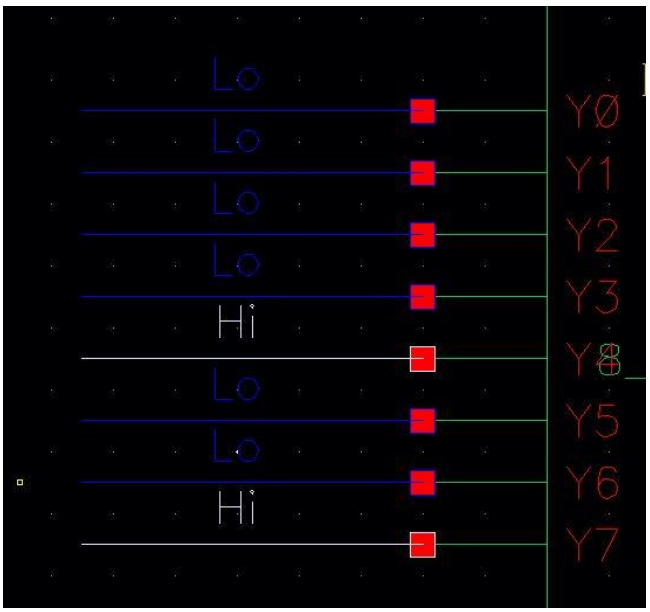
The two signed eight bit numbers being multiplied are X[0:7] and Y[0:7]. X[0:7] is the multiplier and Y[0:7] is the multiplicand. Consider X[-1]=0. So 3 bits from the multiplier gradually enters the Booth encoder and as per the bit sequence modifies and performs certain operations on The multiplicand Y[0:7]. In order to manipulate the multiplicand we have 9 bit booth decoders, which take in 8 bit multiplicands and give 9 bit results which we call 'partial products.

We take the 3 bits X[-1:1] and put it through booth encoder. Here, the 3 bits can lead to 2^3 or 8 possible outputs from the booth encoder. The 8 possible outputs are described in the table below:

Grouping	Partial product
000	0
001	Y
010	Y
011	2*Y
100	-2*Y
101	-Y
110	-Y
111	0

Example: Let X[0:7]=01100110=102

Y[0:7]=10010000=-112



And since no operation has been performed, PP=0.

Here, the signed binary number follows the 2's complement rule. So,

Y = 1 1001 0000

-Y = 0 0111 0000-

2*Y = 1 0010 0000

-2*Y= 0 1110 0000

Step 1: Let X[-1] = 0. So, we take X[-1:1] = 100 for the first 3 bit sequence. Which performs PP=PP+(-2*Y) = 0 1110 0000

Step 2: So, we take PP from previous step and send its 2 LSB - PP[0:1] - to the output result directly. And the MSBof this PP is extended to the two bit positions above the MSB. So,

PP (Modified) = [00]0 1110 00(00)

From the Multiplicand, we take the next group X[1:3] = 011 for the next 3 bit sequence (overlapping the last bit from previous step).

This performs PP=PP(Modified)+(+2*Y)

[00]0 1110 00(00)

+11 0 0100 00

PP = 1 1010 11000

Step 3: We take PP from previous step and send its two LSBs - PP[0:1] - to the output result directly. And the MSB of this PP is extended to the two bit positions above the MSB.

PP (Modified) = [11]1 1100 10(00)

From the Multiplicand, we take the next group X[3:5] = 100 for the next 3 bit sequence (overlapping the last bit from previous step).

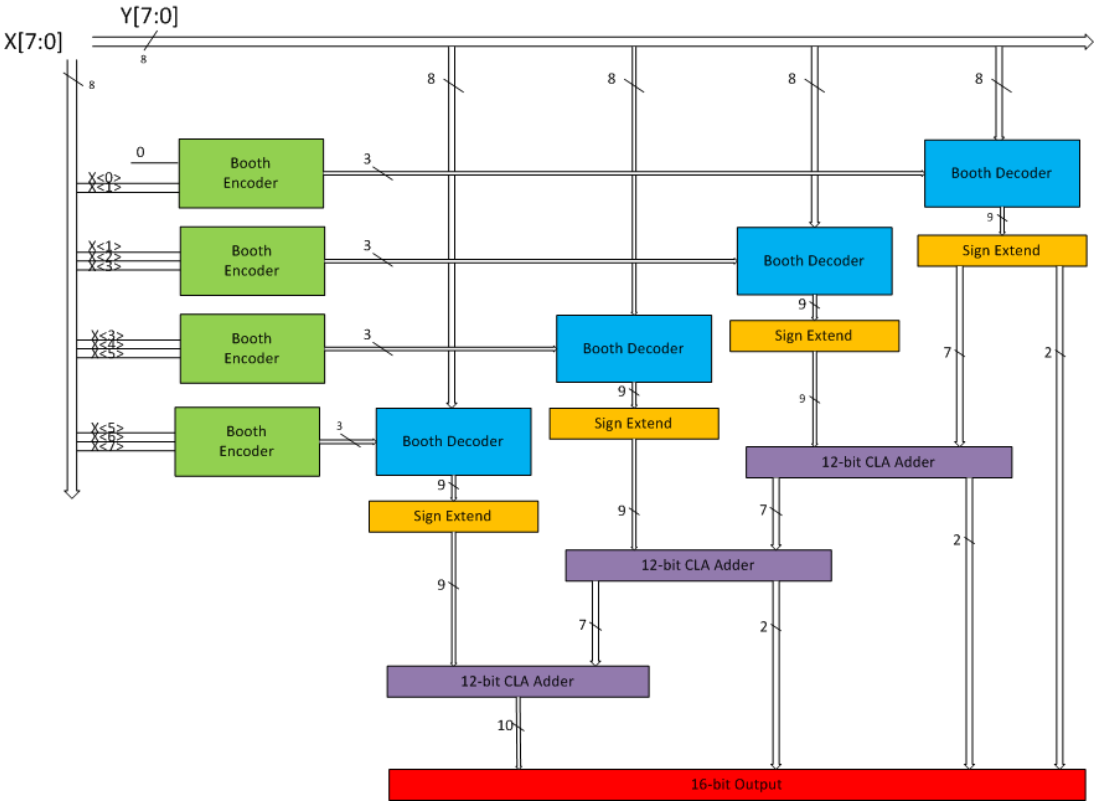
This performs PP=PP(Modified)+(-2*Y)

[11]1010 110(00)

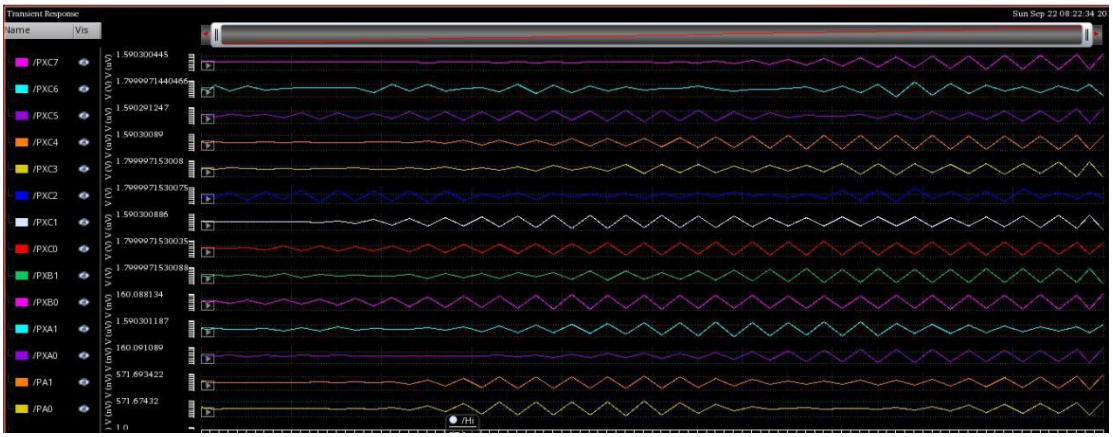
+01 1100 000

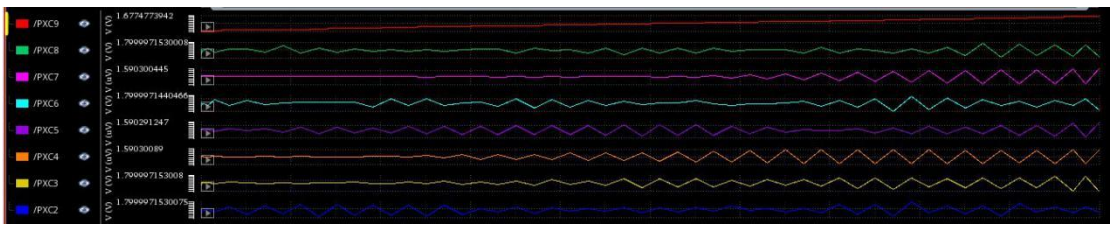
PP =0101101(10)

This process continues, as we find in the following diagram:



The final product is: 1101 1011 0110 0000





5. Results and verification:

The booth encoded multiplier that we have implemented here has the following blocks:

1. 8 to 3 bit Booth Encoders (4)
2. 8 to 9 bit Booth Decoder (4)
3. 12 bit Ripple Carry Adders (3)

Each Booth Decoder has the following components:

i. Nine (9) 1 bit Booth Decoder circuit cascaded
ii. Nine (9) half adder blocks (following the Decoder block)

Each Ripple Carry adder has the following blocks:

1. Twelve (12) Full adders cascaded

In order to implement the logic circuits, we used the following gates

1. NOT Gate
2. 2 input NAND Gate
3. 3 input NAND Gate
4. 2 input XOR Gate

All these blocks are shown below. We have demonstrated the following in this report:

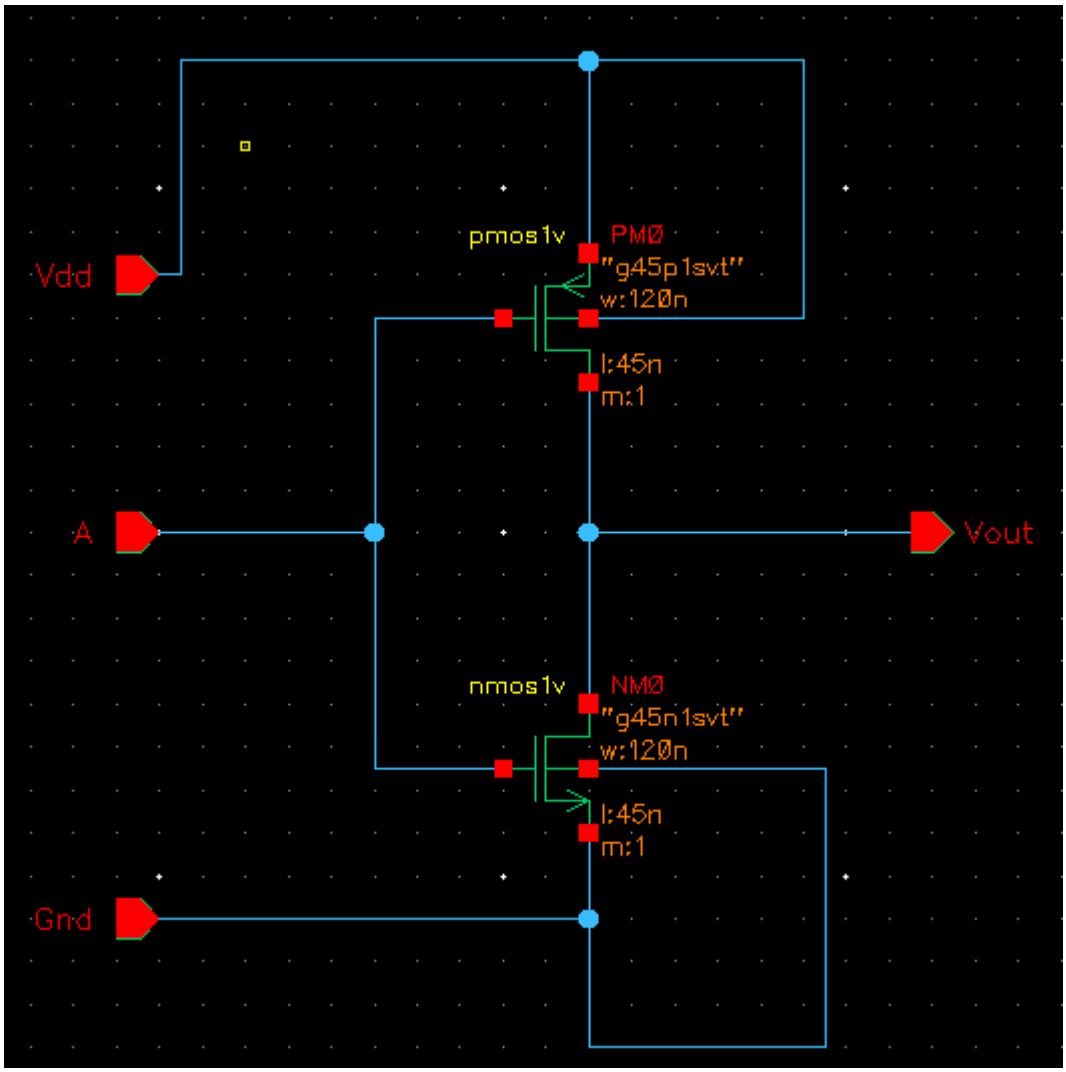
i) schematic diagrams
ii) truth tables

iii) Logic expression

iv) Layout verification

A. NOT Gate:

Schematic:

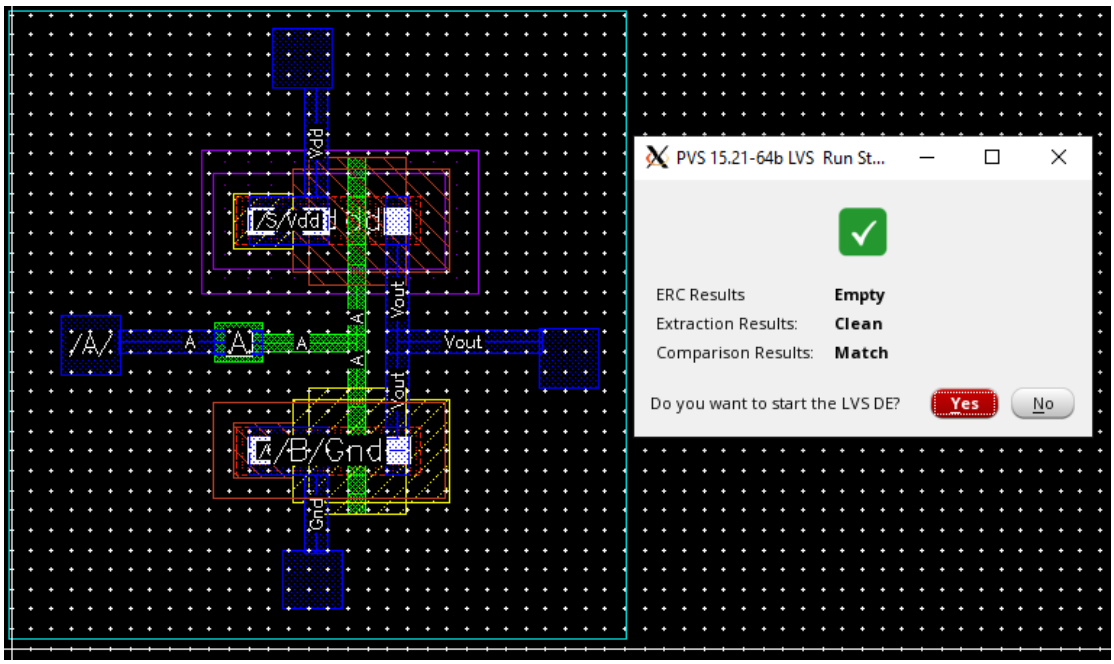


A symbol was generated with this logic circuit.

Truth Table

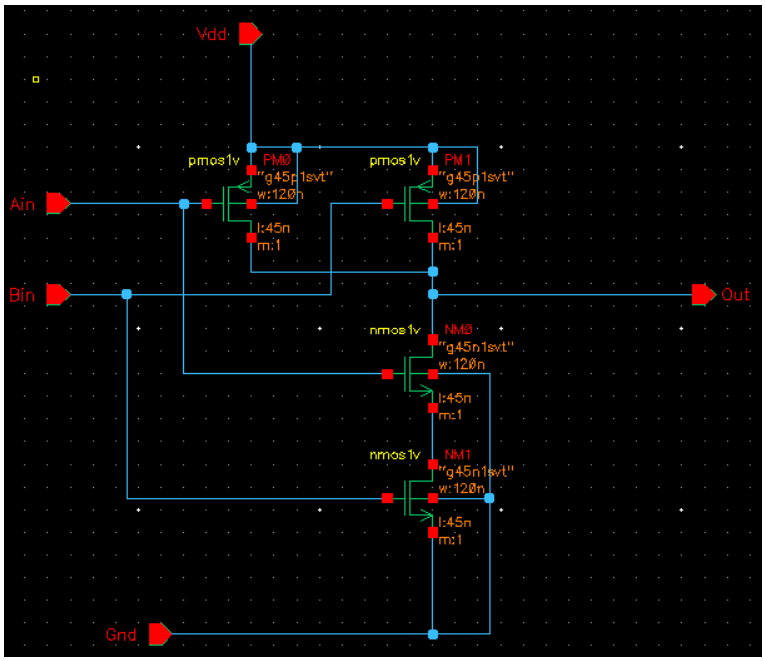
Input A	Output (Vout)
0	1
1	0

Fig. Truth Table of a NOT Gate



B. 2 input NAND Gate

Schematic:

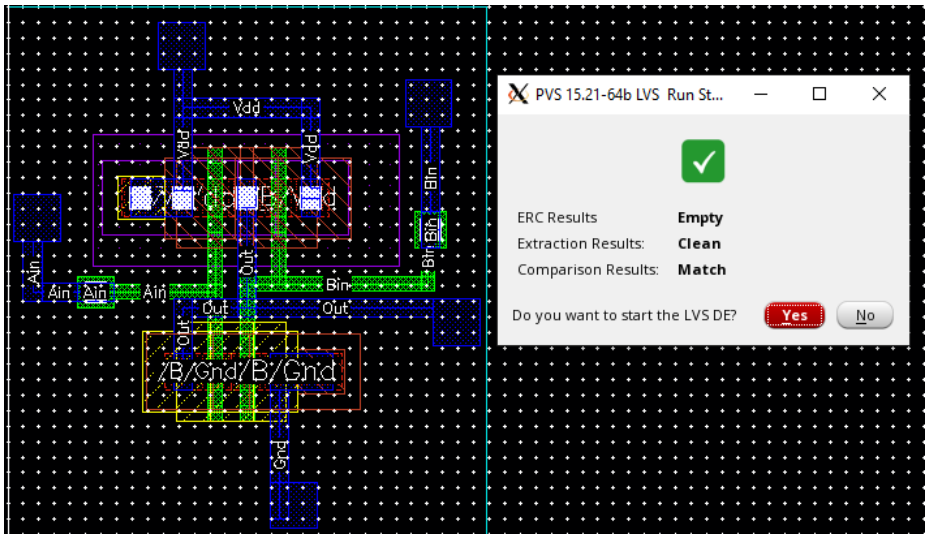


Truth Table

Input A	Input B	Output
0	0	1
0	1	1
1	0	1
1	1	0

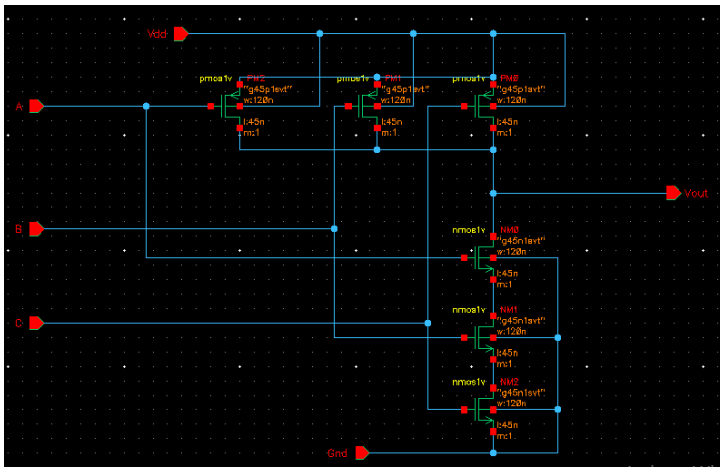
Fig. Truth Table of a 2 Input NAND Gate

Layout



C. 3 Input NAND Gate:

Schematic:

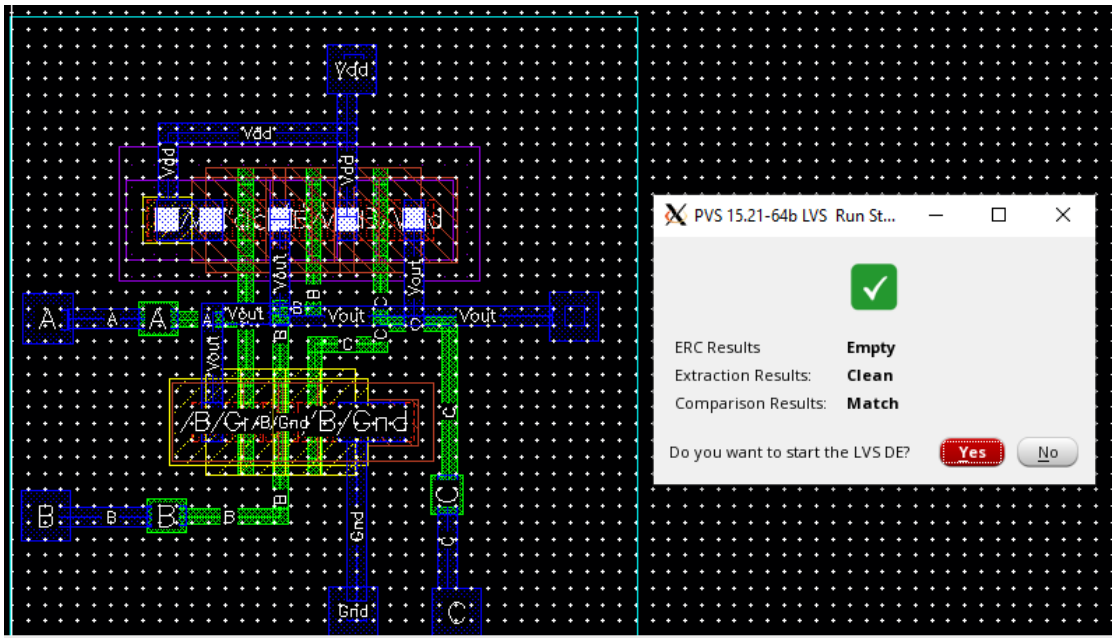


Truth Table

Input A	Input B	Input C	Output Vout
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

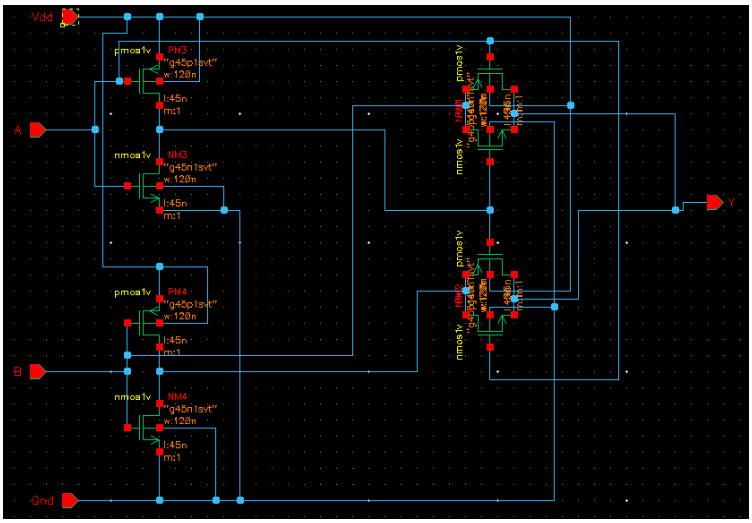
Fig. Truth Table of a 3 Input NAND Gate

Layout:

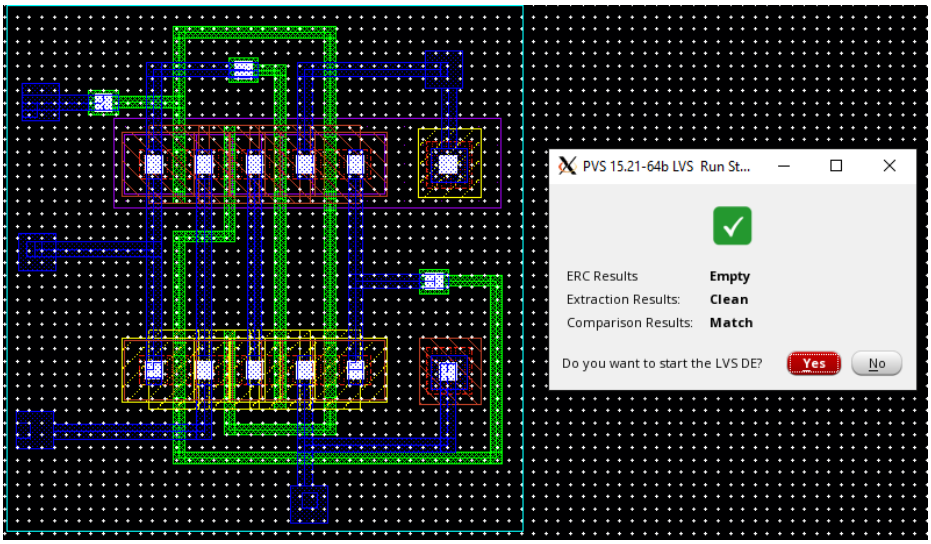


D. 2 input XOR:

Schematic



Layout:



Here we have used the transmission gate schematic to reduce gate numbers:

Truth Table

Input A	Input B	Output Y(Vout)
0	0	0
0	1	1
1	0	1
1	1	0

Fig. Truth Table of a 2 Input XOR Gate

E. Encoder:

Truth Table

X_{i+1}	X_i	X_{i-1}	SNG_i	DBL_i	NEG_i
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	0	1	1
1	0	1	1	0	1
1	1	0	1	0	1
1	1	1	0	0	1

Fig. Truth Table Of Booth Encoder

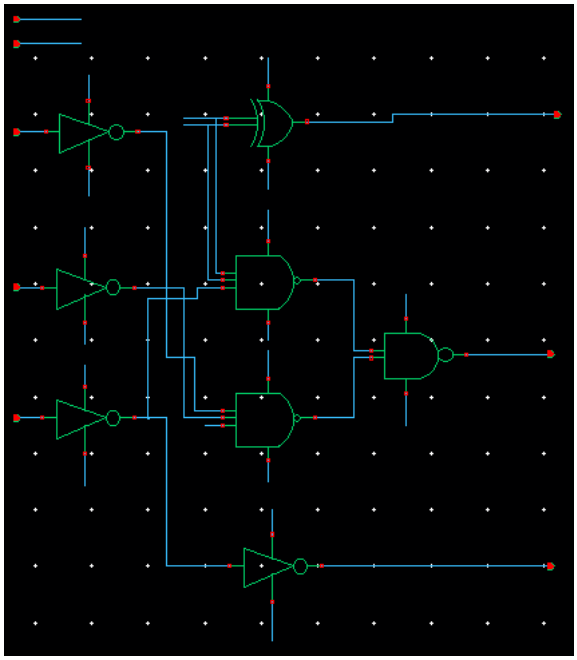
Logic expressions:

$$SNG = X_i \oplus X_{i-1}$$

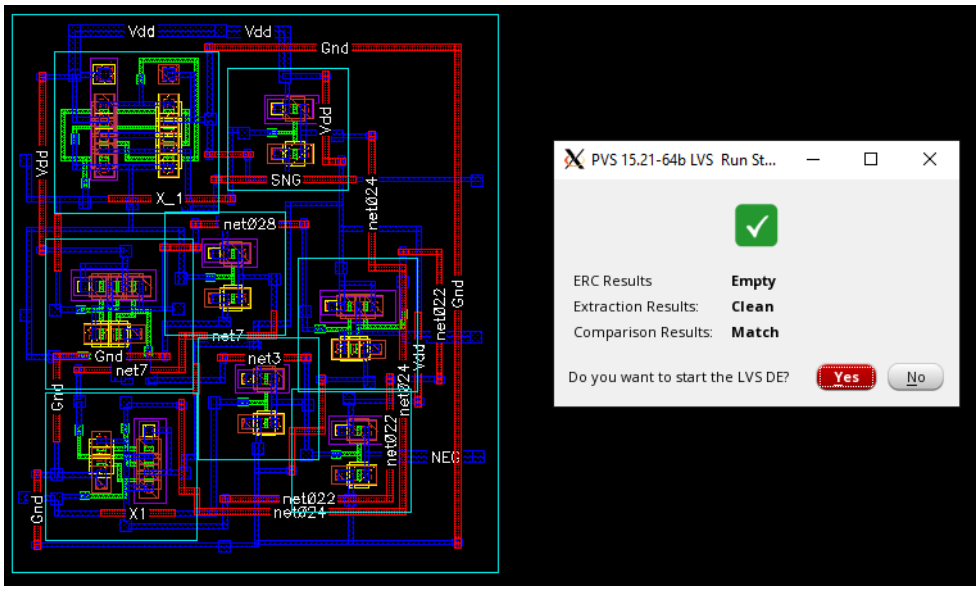
$$\begin{aligned} DBL &= \overline{X}_{i+1}.X_i.X_{i-1} + X_{i+1}.\overline{X}_i.\overline{X}_{i-1} \\ &= \overline{\overline{X}_{i+1}.X_i.X_{i-1} + X_{i+1}.\overline{X}_i.\overline{X}_{i-1}} \\ &= \overline{\overline{X}_{i+1}.X_i.X_{i-1}}.\overline{X_{i+1}.\overline{X}_i.\overline{X}_{i-1}} \end{aligned}$$

$$NEG = X_{i+1}$$

Schematic



Layout:



F. Decoder:

Truth Table:

SNG_i	DBL_i	Y_j	Y_{j-1}	PP_i (pre XOR)
0	0	0	0	0
0	0	0	1	0
0	1	0	0	0
0	1	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1

Fig. Truth Table of a Single Bit Booth Decoder

Logic expressions

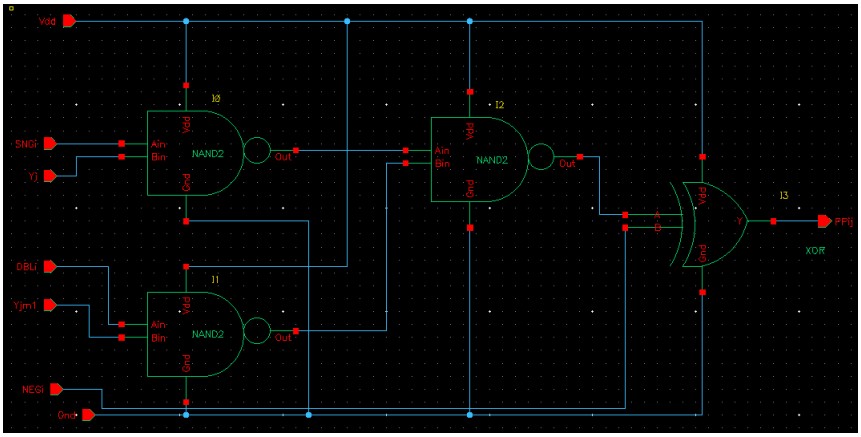
In SNG = 1, then PPi = Yi. This indicates that the 8 bits of Y are passed on to PPi without shifting.

If DBL = 1 then PPi = Yi-1. This indicates that the 8 bits of Y are left shifted on to PPi.

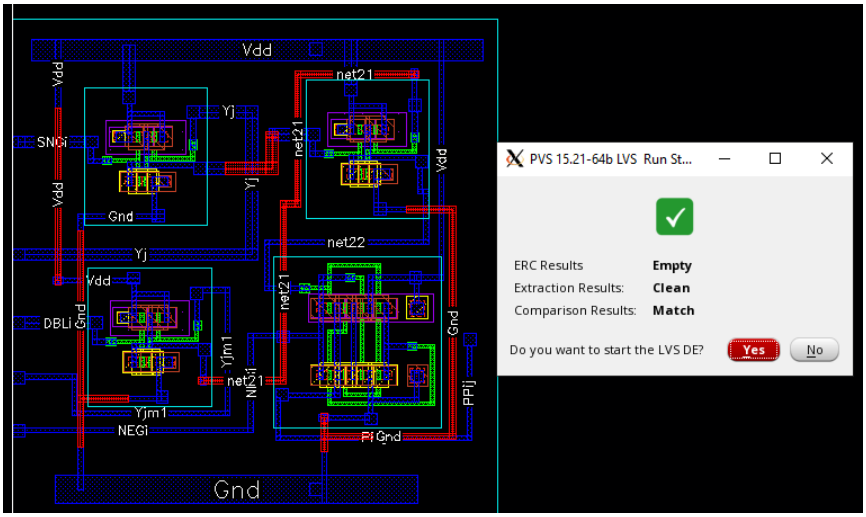
And for the NEG input, the PPi is inverted in the 1 bit decoder block. This inverted bit is passed onto a half adder block in the full nine bit decoder block, that performs 2's complement.

So, $PP_i = NEG_i \oplus (SNG_i.Y_j + DBL_i.Y_{j-1})$

Schematic:



Layout:



G. **Full Adder:**

X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

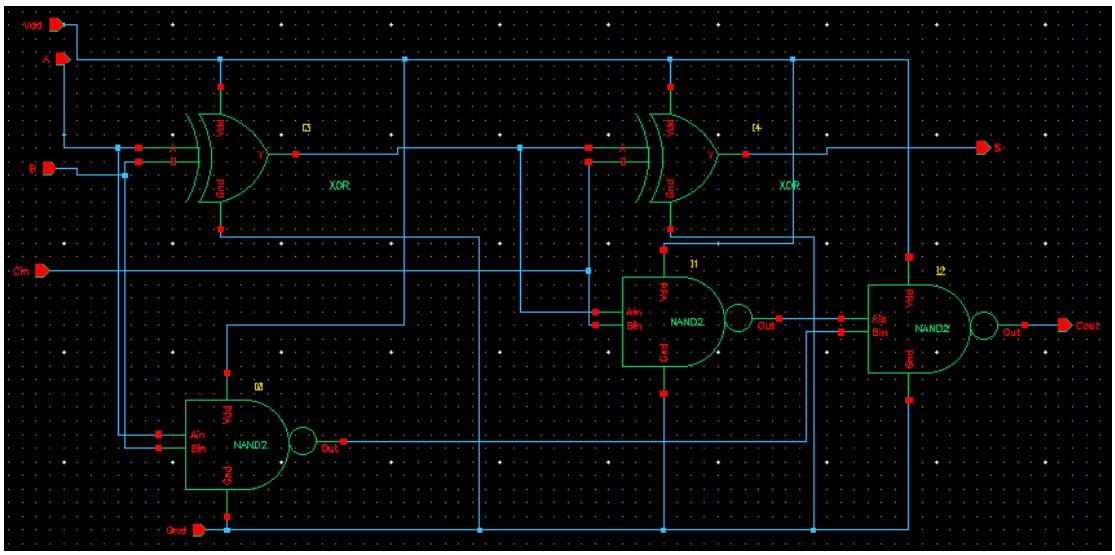
Fig. Truth Table of a Full Adder

Logic expressions:

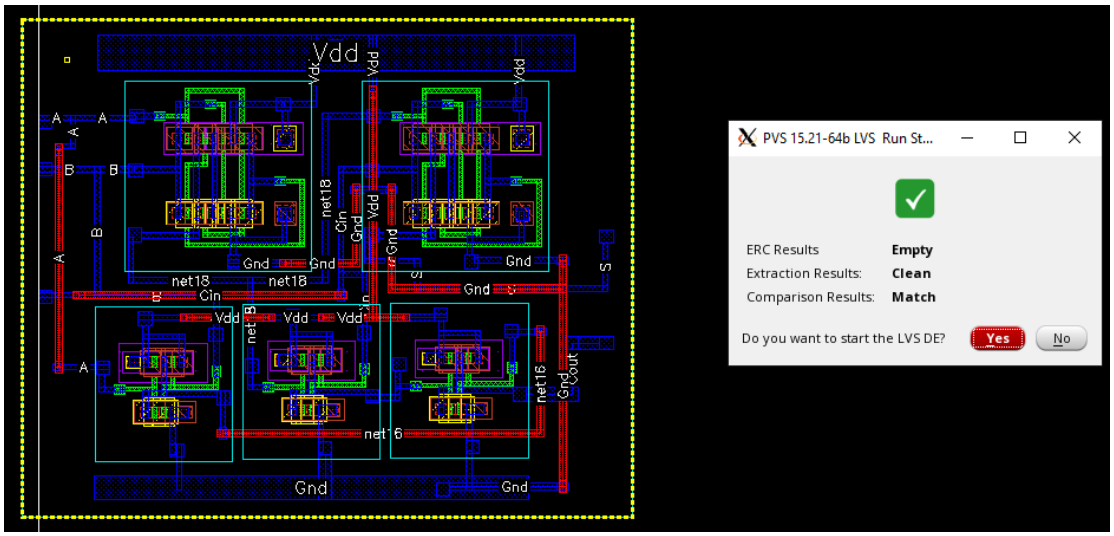
$$S = X \oplus Y \oplus C_{in}$$

$$C_{out} = X.Y + (X \oplus Y).C_{in}$$

Schematic:



Layout:


Vdd
Gnd
net18
Cin
net16
Gnd
PVS 15.21-64b LVS Run St...
ERC Results
Empty
Extraction Results:
Clean
Comparison Results:
Match
Do you want to start the LVS DE?
Yes
No

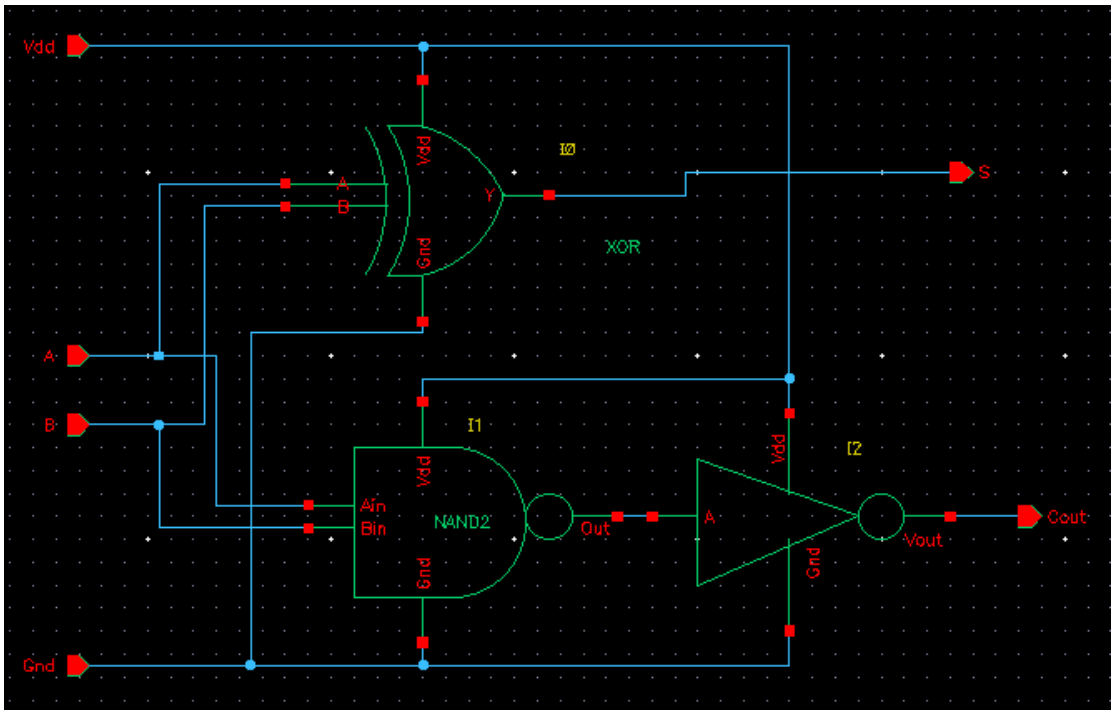
H. Half Adder:

X	Y	S	Cout
0	**0**	**0**	**0**
0	**1**	**1**	**0**
1	**0**	**1**	**0**
1	**1**	**0**	**1**

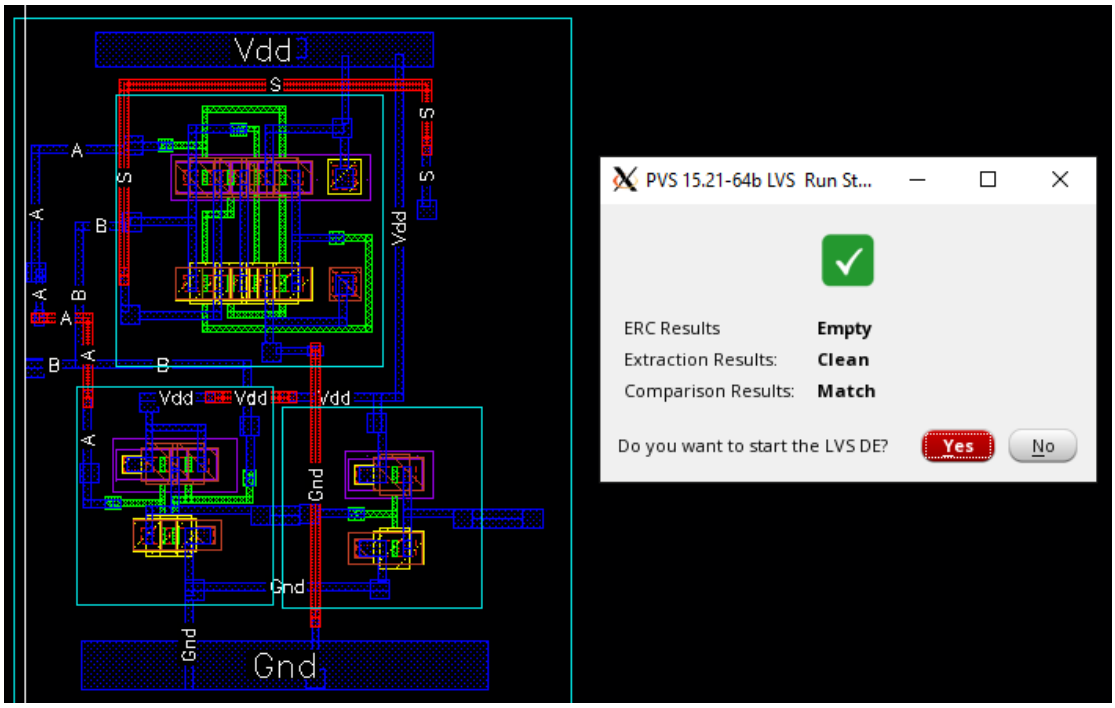
$S = X \oplus Y$

$C_{out} = X.Y$

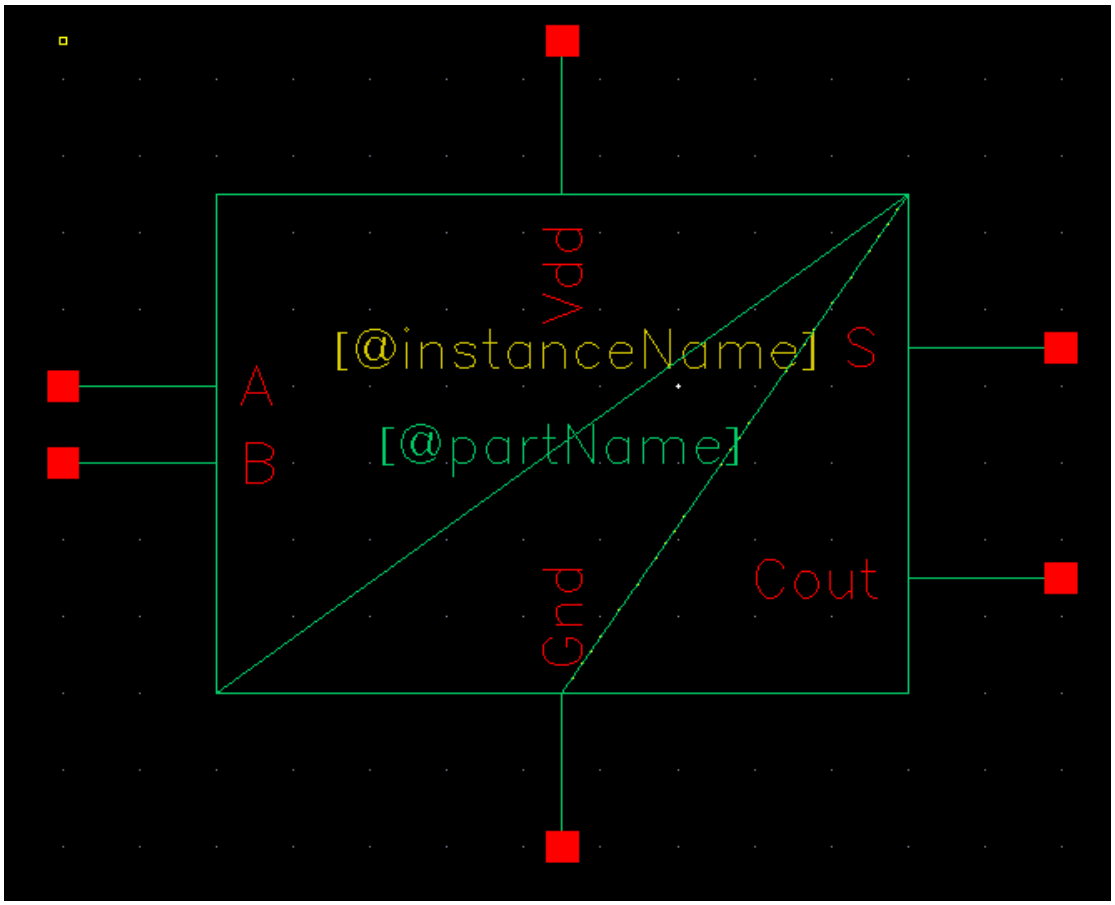
Schematic:



Layout:



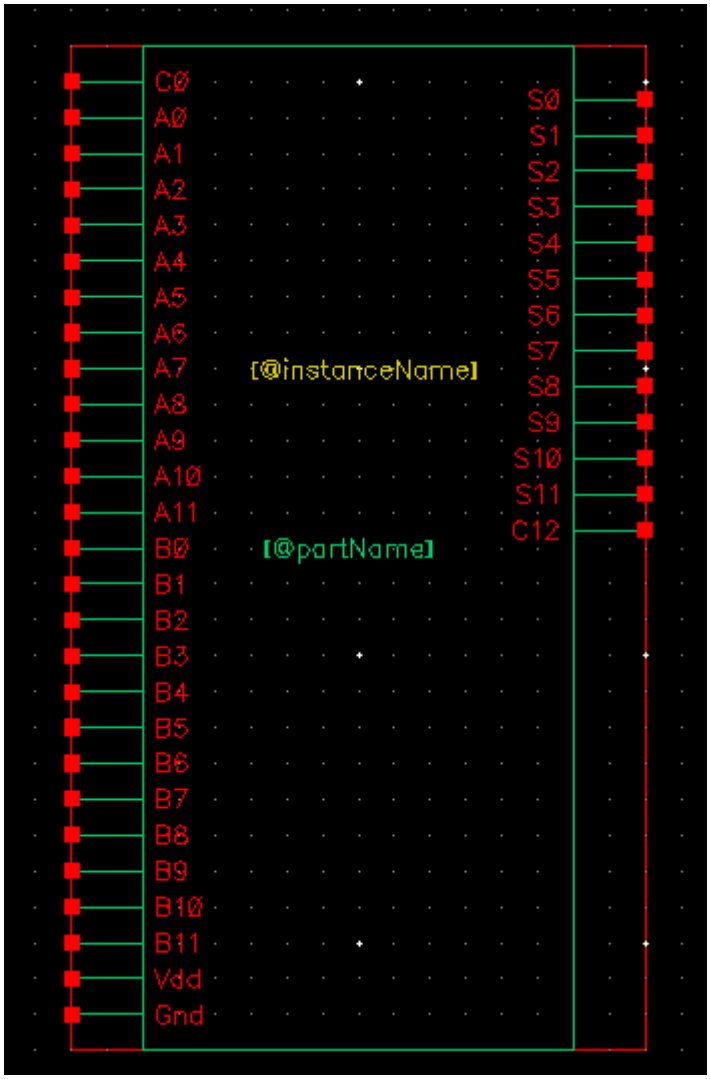
Circuit Symbol:



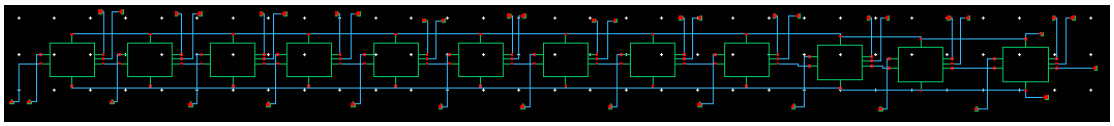
I. Ripple carry adder:

Here, 12 Full adders are cascaded in the following way to generate a 12 bit sum.

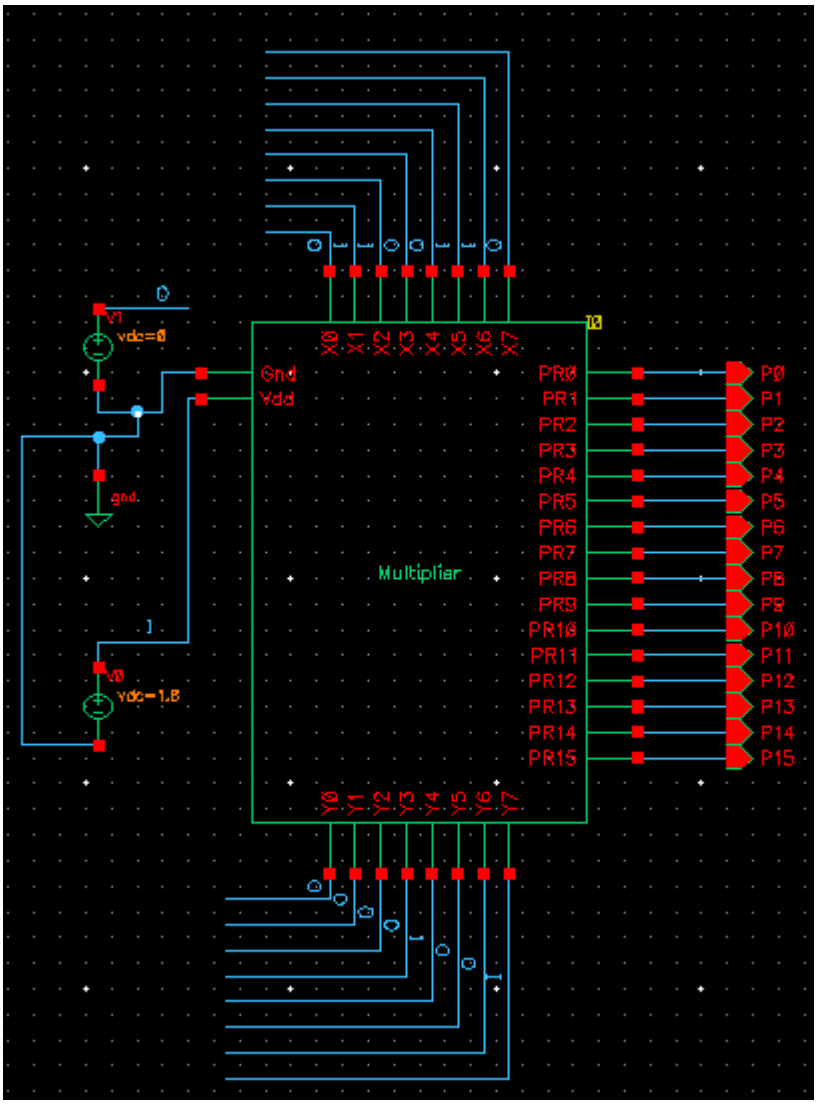
Schematic Symbol:



Layout:

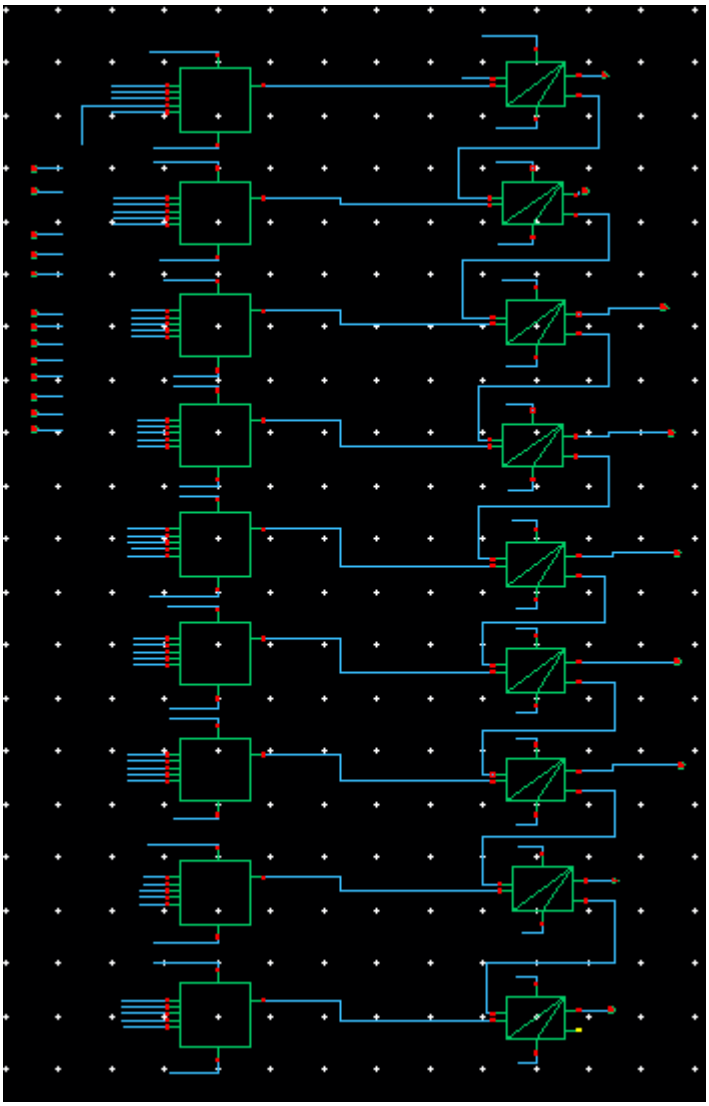


Simulation of 12 bit adder Circuit

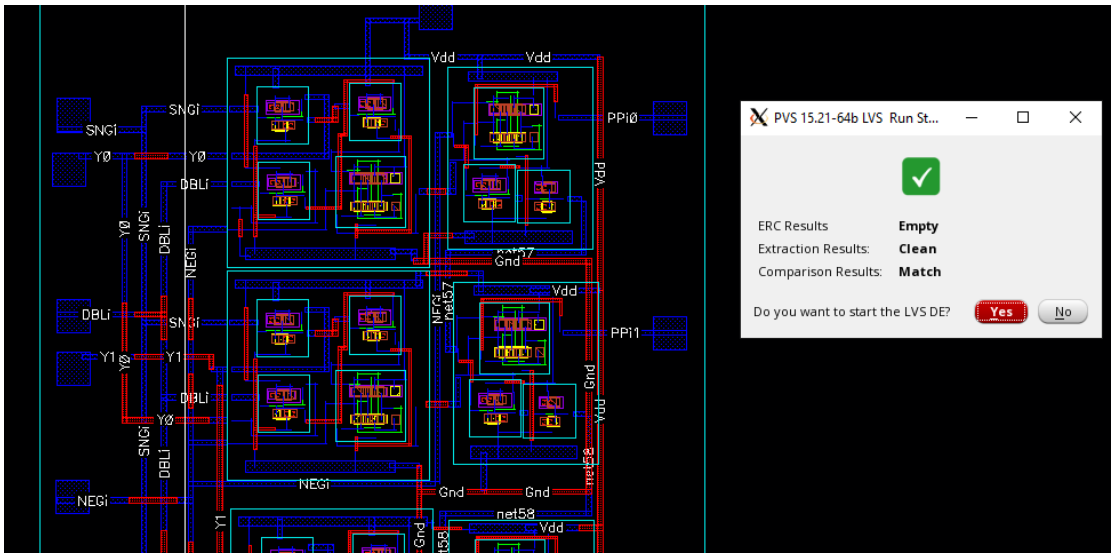


J. 9 bit Booth Decoder:

Schematic:

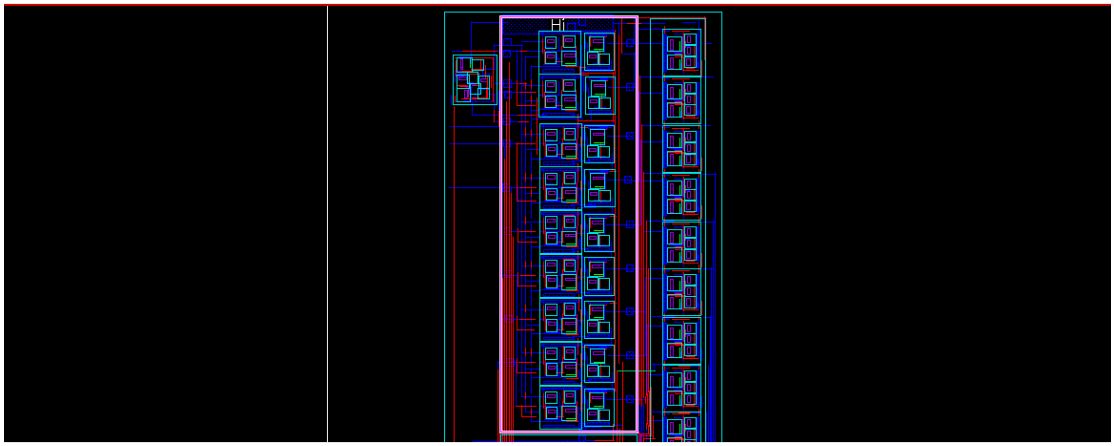


Layout:

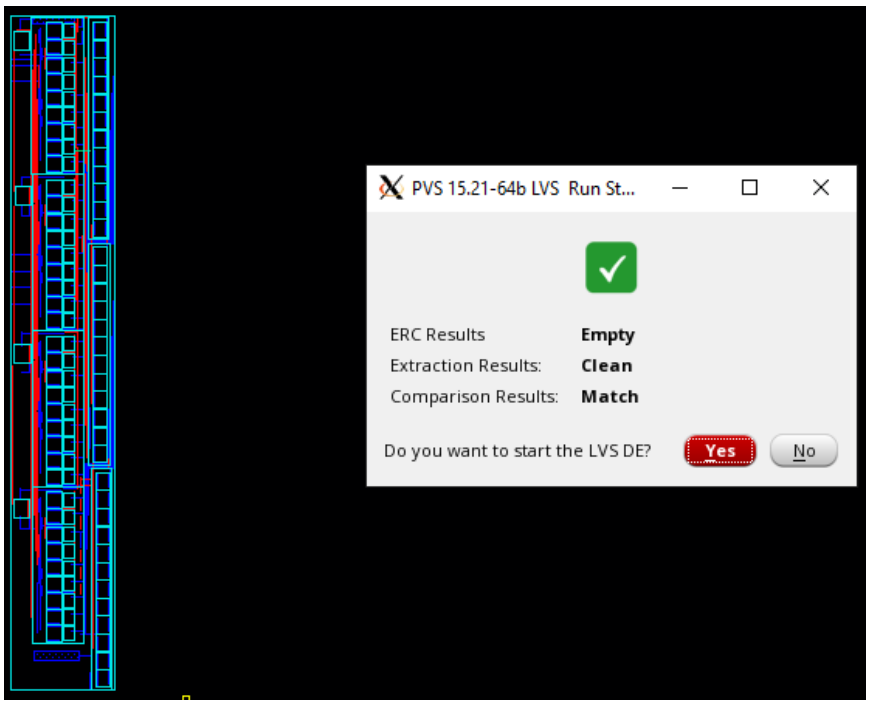

Vdd
Vdd
SNGi
SNGi
PPiØ
YØ
YØ
DBLi
PPi0
YØ
SNGi
DBLi
NEGi
Gnd
DBLi
Vdd
SNGi
NEGi
net57
PPi1
Y1
Y1
YØ
DBLi
Gnd
PPi1
YØ
SNGi
DBLi
net58
NEGi
Gnd
Gnd
Y1
net58
Gnd
net58
Vdd
PVS 15.21-64b LVS Run St...
ERC Results
Empty
Extraction Results:
Clean
Comparison Results:
Match
Do you want to start the LVS DE?
Yes
No

K. Full Multiplier circuit:

Layout



Full Layout:



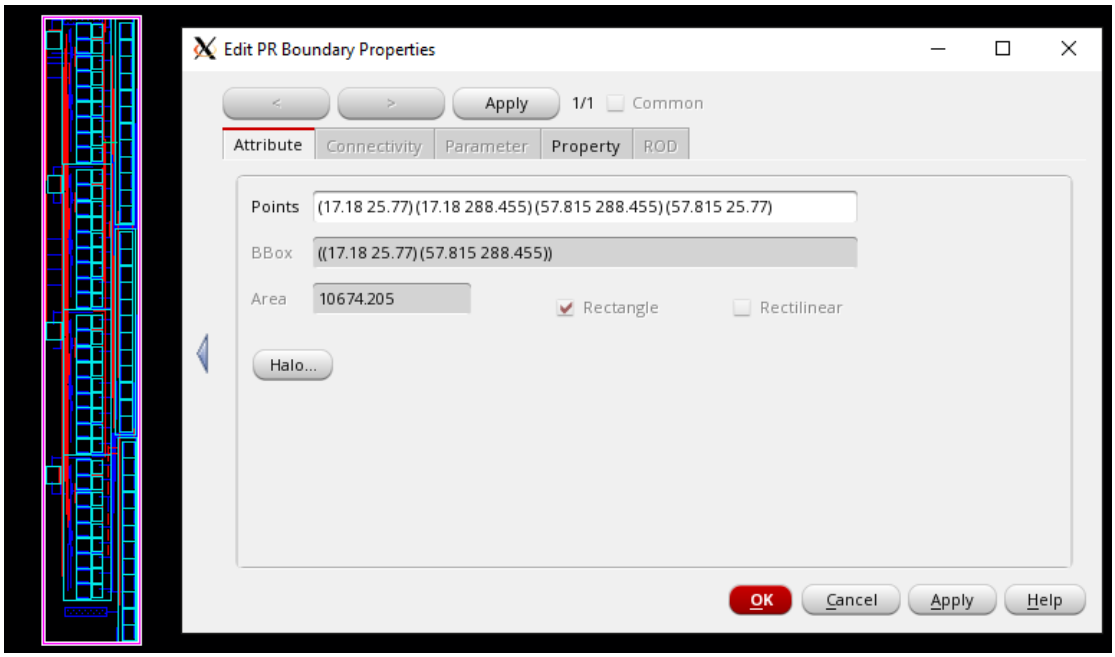
LVS Check results:

PVS LVS COMPARISON SUMMARY

```
                                    PVS LVS COMPARISON

Version                          :   15.21-s062
NVN Run Start                    :   Tue Sep 17 19:32:18 2019
ERC Summary File                 :   Multiplier3.sum
Extraction Report File           :   Multiplier3.rep
Comparison Report File           :   Multiplier3.rep.cls
Top Cell                         :   Multiplier3  <vs>  Multiplier3

Run Result                       :   MATCH

Run Summary                      :   [INFO]  ERC Results: Empty
                                 :   [INFO]  Extraction Clean

Layout Design                    :   Project Multiplier3 layout
Schematic File                   :   /home/vlsi27/TestsDRCLVS/Multiplier3.cdl (cdl)
Rules File                       :   .technology.rul
Pin Swap File                    :   Multiplier3.rep.cps

Extraction CPU Time              :   0h 0m 1s - (1s)
Extraction Exec Time             :   0h 0m 1s - (1s)
Extraction Peak Memory Usage     :   25.00MB
NVN CPU Time                     :   0h 0m 0s - (0s)
NVN Exec Time                    :   0h 0m 1s - (1s)
NVN Peak Memory Usage            :   219.71MB
LVS Total CPU Time               :   0h 0m 1s - (1s)
LVS Total Exec Time              :   0h 0m 2s - (2s)
LVS Total Peak Memory Usage      :   219.71MB
```

****ALL THE CIRCUIT COMPONENTS WERE DESIGN CHEK (DRC) VERIFIED AS WELL****

6. Layout Area:



The total area is about 0.010674 mm^2.

7. Conclusion:

Here, the booth multiplier that we have implemented is functionally correct as we can see from the timing diagram. Also the circuit area has been minimized, as we tried to make it as compact as possible. We used transmission gates for XOR gate, to minimize delay . Also, the circuit only has **2 Layers** of wires. These two layers also ensured that there was minimal propagation delay and less errors.

8. Further improvements:

- If more time was available then the circuit could have been made even more compact.
- The ripple of the output voltage was of pico range. This can be further reduced by reducing the amount of wiring.