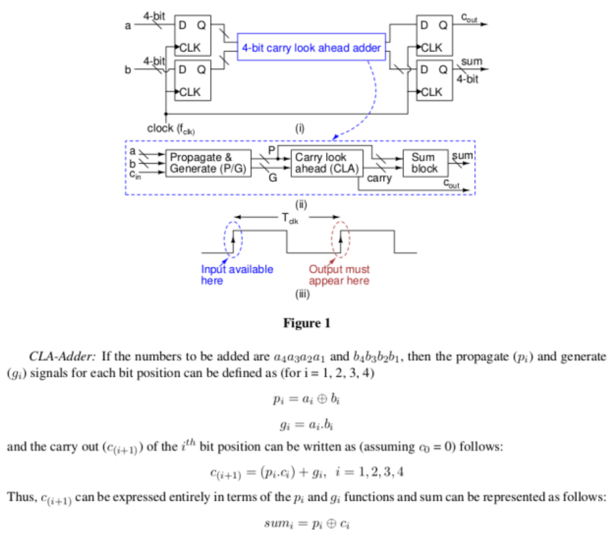
VLSI PROJECT REPORT

Jal Parikh
2023102066

I. Structure of Adder

As shown in Fig. 1, the structure of the Carry Look-Ahead Adder (CLA) is detailed below:



CLA-Adder: If the numbers to be added are $a_4a_3a_2a_1$ and $b_4b_3b_2b_1$, then the propagate (p_i) and generate (g_i) signals for each bit position can be defined as (for i = 1, 2, 3, 4)

$$p_i = a_i \oplus b_i$$

$$g_i = a_i.b_i$$

and the carry out ($c_{(i+1)}$) of the i^{th} bit position can be written as (assuming $c_0 = 0$) follows:

$$c_{(i+1)} = (p_i.c_i) + g_i, \;\; i = 1, 2, 3, 4$$

Thus, $c_{(i+1)}$ can be expressed entirely in terms of the p_i and g_i functions and sum can be represented as follows:

$$sum_i = p_i \oplus c_i$$

Fig. 1: Basic structure and working of CLA.

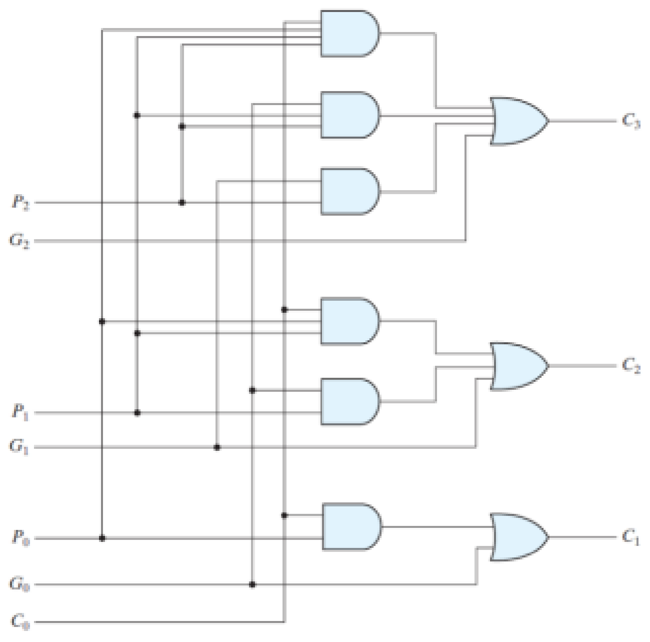


Fig. 2: CLA Generator Logic.

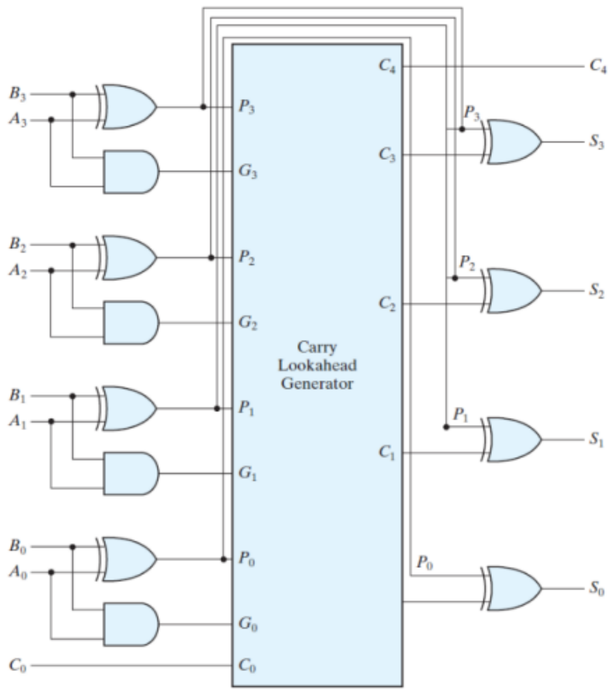


Fig. 3: CLA Adder Logic.

II. Topology and Sizing of each block

All gates have been implemented using static CMOS topologies. For Flip-Flops, the True Single-Phase Clocked (TSPC) topology has been utilized. The sizing used for the TSPC Flip-Flop is

$$\frac{20\lambda}{10\lambda}, \quad \text{where } \lambda = 0.09\,\mu\text{m}.$$

The sizing used for the gates throughout the circuit is

$$\frac{W_p}{W_n} = 2.$$

III. Stick Diagrams

The stick diagrams of gates used are shown below

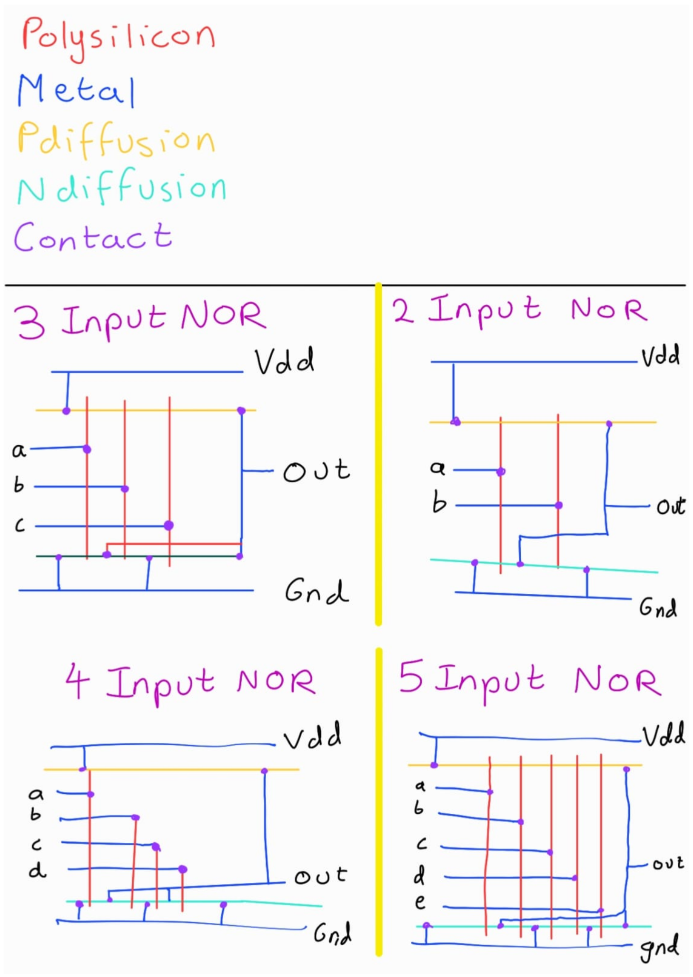


Fig. 4: Enter Caption

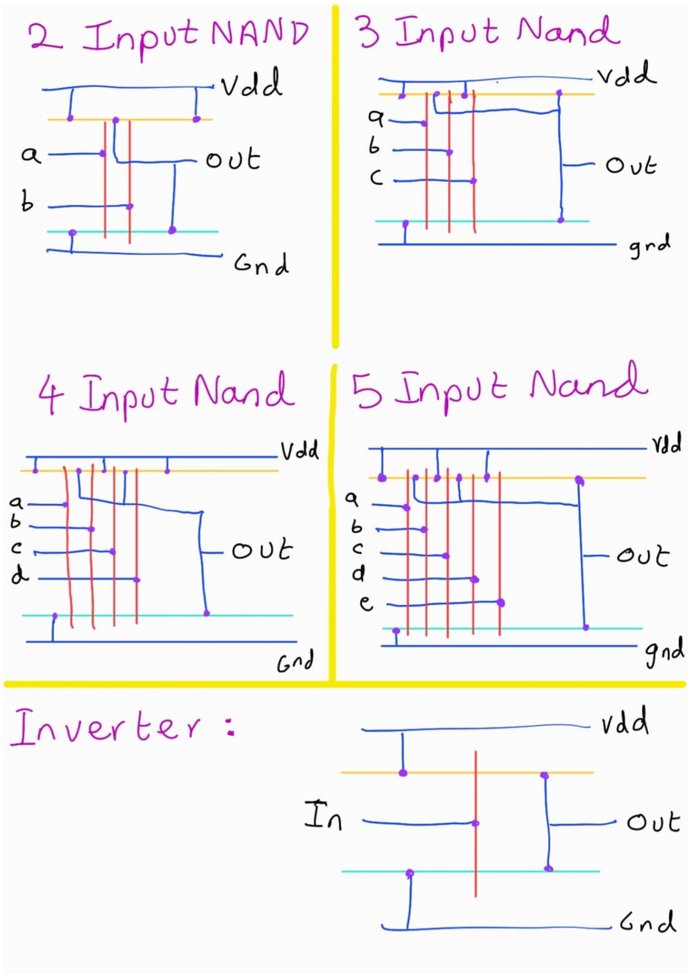
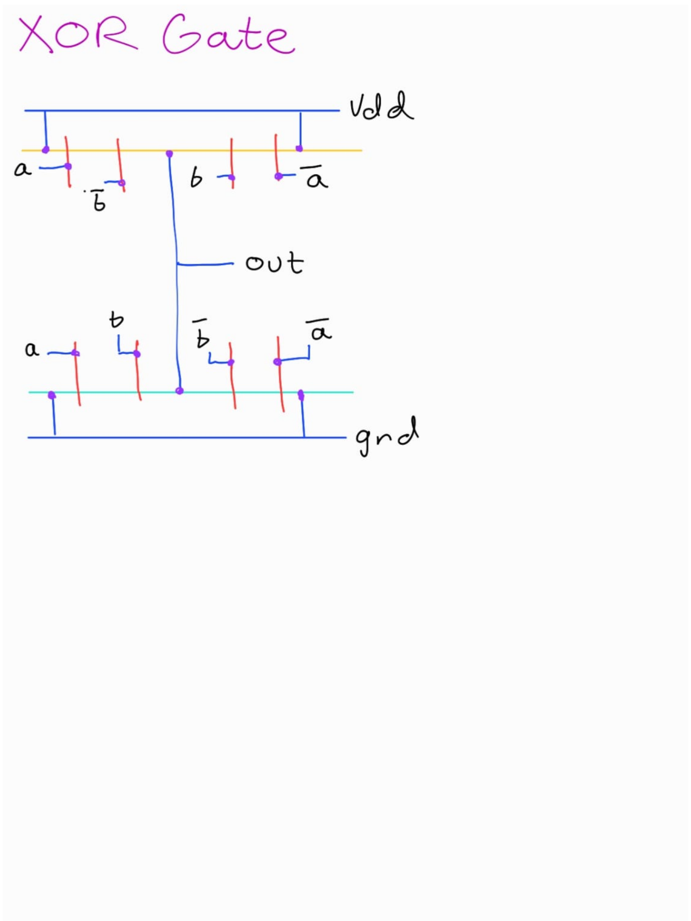


Fig. 5: Enter Caption



IV. NGSPICE Simulations

A. Gate Simulations

1) Inverter: Inverter implemented using a pmos and nmos connected in series.

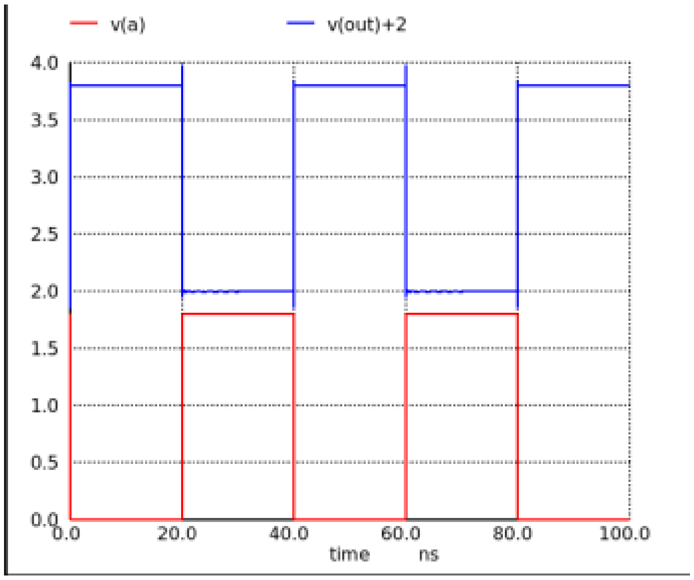
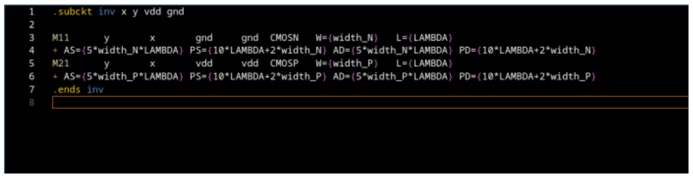


Fig. 7: Inverter Simulation



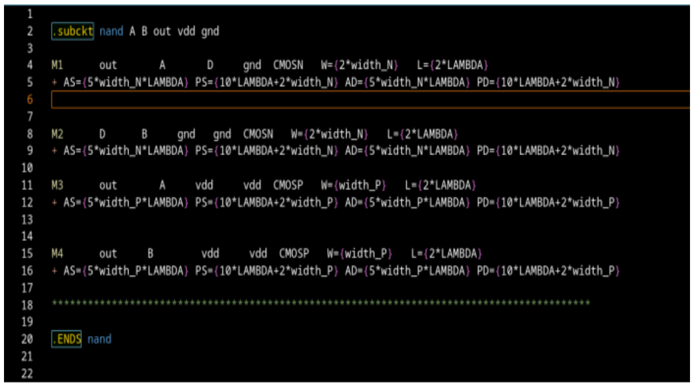
```
.subckt inv x y vdd gnd

M11       y       x       gnd      gnd  CMOSN   W={width_N}    L={LAMBDA}
+ AS={5*width_N*LAMBDA} PS={10*LAMBDA+2*width_N} AD={5*width_N*LAMBDA} PD={10*LAMBDA+2*width_N}
M21       y       x       vdd      vdd  CMOSP   W={width_P}    L={LAMBDA}
+ AS={5*width_P*LAMBDA} PS={10*LAMBDA+2*width_P} AD={5*width_P*LAMBDA} PD={10*LAMBDA+2*width_P}
.ends inv

```

Fig. 8: Inverter Subckt

2) AND Gate: For the two-input AND gate, I have implemented a CMOS style NAND gate connected to an inverter. A CMOS NAND gate uses parallel pMOS transistors in the pull-up network and series nMOS transistors in the pull-down network. The subckt for the NAND gate is shown in Fig. 9 and the simulation for the AND gate is shown in Fig. 10.



```

.subckt nand A B out vdd gnd

M1      out       A       D      gnd  CMOSN   W={2*width_N}    L={2*LAMBDA}
+ AS={5*width_N*LAMBDA} PS={10*LAMBDA+2*width_N} AD={5*width_N*LAMBDA} PD={10*LAMBDA+2*width_N}


M2      D      B     gnd    gnd  CMOSN   W={2*width_N}    L={2*LAMBDA}
+ AS={5*width_N*LAMBDA} PS={10*LAMBDA+2*width_N} AD={5*width_N*LAMBDA} PD={10*LAMBDA+2*width_N}

M3      out       A     vdd     vdd  CMOSP   W={width_P}    L={2*LAMBDA}
+ AS={5*width_P*LAMBDA} PS={10*LAMBDA+2*width_P} AD={5*width_P*LAMBDA} PD={10*LAMBDA+2*width_P}


M4      out     B        vdd     vdd  CMOSP   W={width_P}    L={2*LAMBDA}
+ AS={5*width_P*LAMBDA} PS={10*LAMBDA+2*width_P} AD={5*width_P*LAMBDA} PD={10*LAMBDA+2*width_P}

************************************************************************************

.ENDS nand


```

Fig. 9: NAND Gate

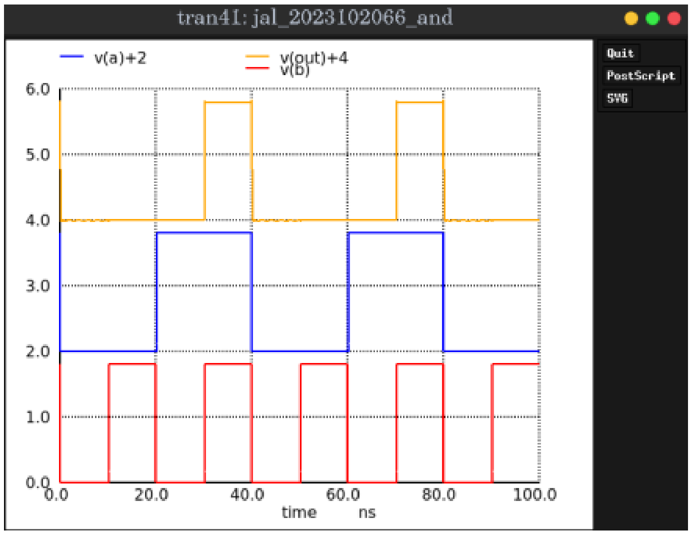
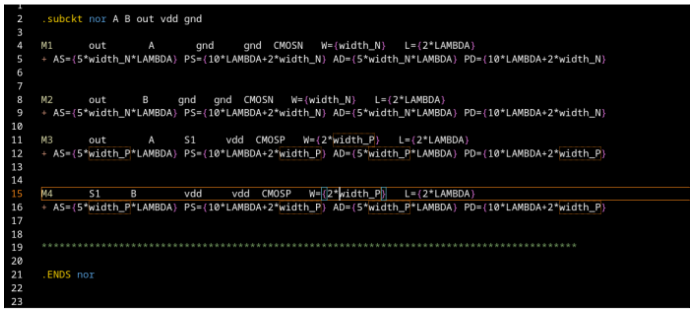


Fig. 10: AND Simulation

3) OR Gate: For the two-input OR gate, I have implemented a CMOS style NOR gate connected to an inverter. A CMOS NOR gate uses series pMOS transistors in the pull-up network and parallel nMOS transistors in the pull-down network. The subckt for the NOR gate is shown in Fig. 11 and the simulation for the OR gate is shown in Fig. 12.



```
.subckt nor A B out vdd gnd

M1       out       A       gnd      gnd  CMOSN   W={width_N}    L={2*LAMBDA}
+ AS={5*width_N*LAMBDA} PS={10*LAMBDA+2*width_N} AD={5*width_N*LAMBDA} PD={10*LAMBDA+2*width_N}


M2       out      B      gnd    gnd  CMOSN   W={width_N}    L={2*LAMBDA}
+ AS={5*width_N*LAMBDA} PS={10*LAMBDA+2*width_N} AD={5*width_N*LAMBDA} PD={10*LAMBDA+2*width_N}

M3       out       A      S1      vdd  CMOSP   W={2*width_P}    L={2*LAMBDA}
+ AS={5*width_P*LAMBDA} PS={10*LAMBDA+2*width_P} AD={5*width_P*LAMBDA} PD={10*LAMBDA+2*width_P}


M4       S1      B         vdd      vdd  CMOSP   W={2*width_P}    L={2*LAMBDA}
+ AS={5*width_P*LAMBDA} PS={10*LAMBDA+2*width_P} AD={5*width_P*LAMBDA} PD={10*LAMBDA+2*width_P}


************************************************************************************

.ENDS nor


```

Fig. 11: NOR subckt

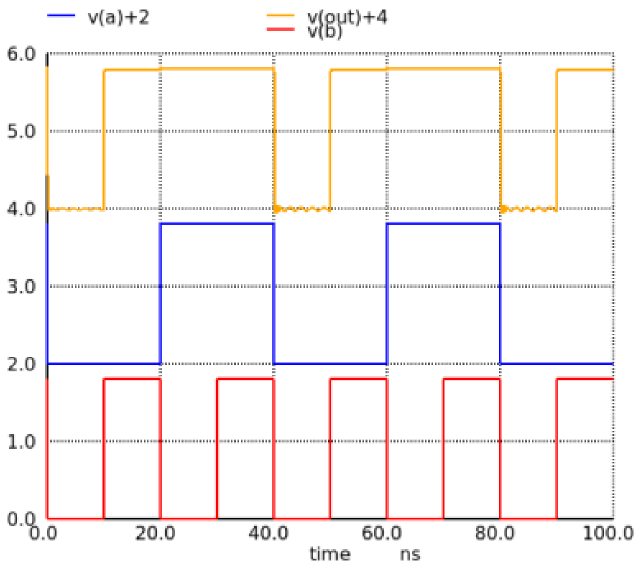
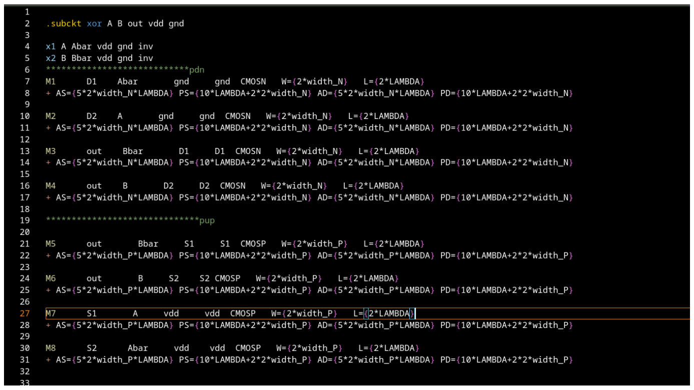


Fig. 12: OR Simulation

4) XOR Gate: For the two-input XOR Gate, I have implemented a CMOS style XOR gate. It consists of 4 PMOS transistors in the pull-up network and 4 NMOS transistors in the pull-down network as shown in the stick diagram of the XOR gate.



```

.subckt xor A B out vdd gnd

x1 A Abar vdd gnd inv
x2 B Bbar vdd gnd inv
****************************pdn
M1      D1    Abar       gnd     gnd  CMOSN   W={2*width_N}   L={2*LAMBDA}
+ AS={5*2*width_N*LAMBDA} PS={10*LAMBDA+2*2*width_N} AD={5*2*width_N*LAMBDA} PD={10*LAMBDA+2*2*width_N}

M2      D2    A       gnd     gnd  CMOSN   W={2*width_N}   L={2*LAMBDA}
+ AS={5*2*width_N*LAMBDA} PS={10*LAMBDA+2*2*width_N} AD={5*2*width_N*LAMBDA} PD={10*LAMBDA+2*2*width_N}

M3      out    Bbar       D1     D1  CMOSN   W={2*width_N}   L={2*LAMBDA}
+ AS={5*2*width_N*LAMBDA} PS={10*LAMBDA+2*2*width_N} AD={5*2*width_N*LAMBDA} PD={10*LAMBDA+2*2*width_N}

M4      out    B       D2     D2  CMOSN   W={2*width_N}   L={2*LAMBDA}
+ AS={5*2*width_N*LAMBDA} PS={10*LAMBDA+2*2*width_N} AD={5*2*width_N*LAMBDA} PD={10*LAMBDA+2*2*width_N}

******************************pup

M5      out       Bbar     S1     S1  CMOSP   W={2*width_P}   L={2*LAMBDA}
+ AS={5*2*width_P*LAMBDA} PS={10*LAMBDA+2*2*width_P} AD={5*2*width_P*LAMBDA} PD={10*LAMBDA+2*2*width_P}

M6      out       B     S2     S2 CMOSP   W={2*width_P}   L={2*LAMBDA}
+ AS={5*2*width_P*LAMBDA} PS={10*LAMBDA+2*2*width_P} AD={5*2*width_P*LAMBDA} PD={10*LAMBDA+2*2*width_P}

M7      S1       A     vdd     vdd  CMOSP   W={2*width_P}   L={2*LAMBDA}
+ AS={5*2*width_P*LAMBDA} PS={10*LAMBDA+2*2*width_P} AD={5*2*width_P*LAMBDA} PD={10*LAMBDA+2*2*width_P}

M8      S2      Abar     vdd    vdd  CMOSP   W={2*width_P}   L={2*LAMBDA}
+ AS={5*2*width_P*LAMBDA} PS={10*LAMBDA+2*2*width_P} AD={5*2*width_P*LAMBDA} PD={10*LAMBDA+2*2*width_P}

```

Fig. 13: XOR subckt

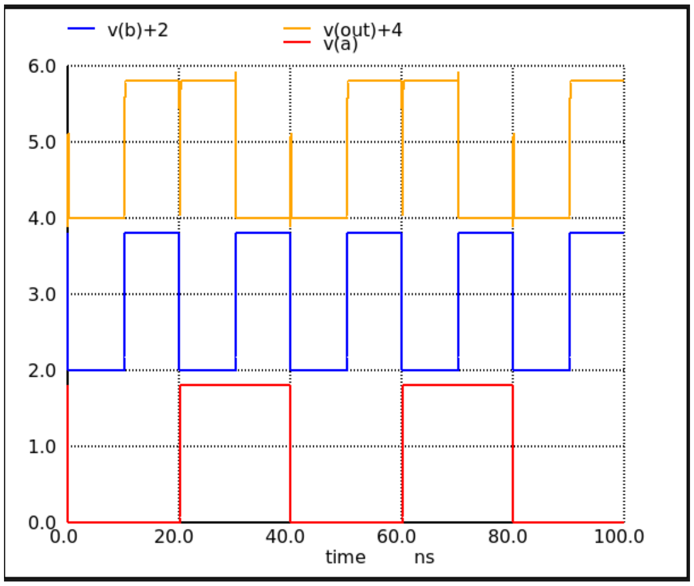


Fig. 14: XOR Simulation

5) Flip-Flop: For the flip-flop, I chose the True Single Phase Clock (TSPC) topology since it's fast and has zero hold time. Its circuit can be seen in the figure below. We connect the output of the negative latch with input of the positive latch to make positive edge triggered flip flop.

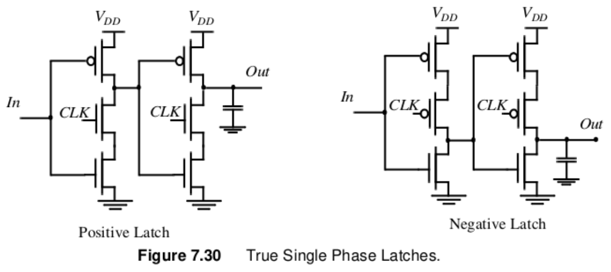
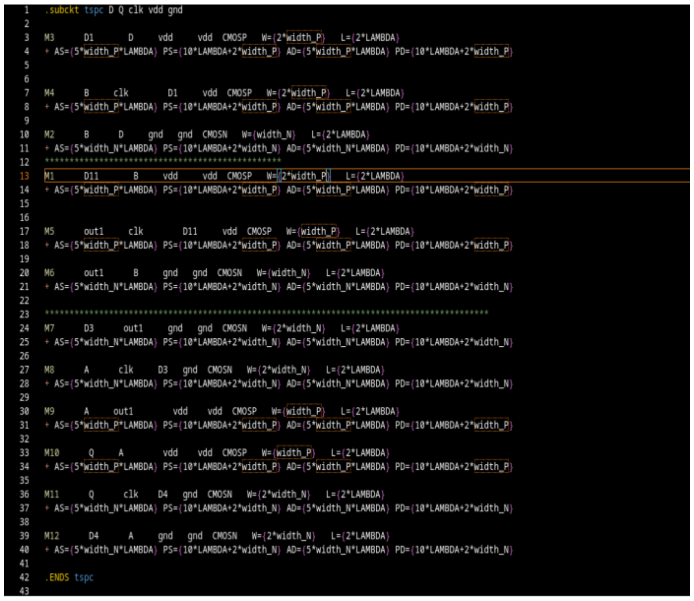


Fig. 15: TSPC Latches



```
.subckt tspc D Q clk vdd gnd

M3      D1       D     vdd     vdd  CMOSP   W={2*width_P}   L={2*LAMBDA}
+ AS={5*width_P*LAMBDA} PS={10*LAMBDA+2*width_P} AD={5*width_P*LAMBDA} PD={10*LAMBDA+2*width_P}


M4      B     clk       D1     vdd  CMOSP   W={2*width_P}   L={2*LAMBDA}
+ AS={5*width_P*LAMBDA} PS={10*LAMBDA+2*width_P} AD={5*width_P*LAMBDA} PD={10*LAMBDA+2*width_P}

M2      B      D     gnd   gnd  CMOSN   W={width_N}   L={2*LAMBDA}
+ AS={5*width_N*LAMBDA} PS={10*LAMBDA+2*width_N} AD={5*width_N*LAMBDA} PD={10*LAMBDA+2*width_N}
**********************************************
M1      D11       B     vdd     vdd  CMOSP   W={2*width_P}   L={2*LAMBDA}
+ AS={5*width_P*LAMBDA} PS={10*LAMBDA+2*width_P} AD={5*width_P*LAMBDA} PD={10*LAMBDA+2*width_P}


M5      out1     clk       D11     vdd  CMOSP   W={width_P}   L={2*LAMBDA}
+ AS={5*width_P*LAMBDA} PS={10*LAMBDA+2*width_P} AD={5*width_P*LAMBDA} PD={10*LAMBDA+2*width_P}

M6      out1      B     gnd   gnd  CMOSN   W={width_N}   L={2*LAMBDA}
+ AS={5*width_N*LAMBDA} PS={10*LAMBDA+2*width_N} AD={5*width_N*LAMBDA} PD={10*LAMBDA+2*width_N}

*******************************************************************************************
M7      D3      out1     gnd   gnd  CMOSN   W={2*width_N}   L={2*LAMBDA}
+ AS={5*width_N*LAMBDA} PS={10*LAMBDA+2*width_N} AD={5*width_N*LAMBDA} PD={10*LAMBDA+2*width_N}

M8      A      clk     D3   gnd  CMOSN   W={2*width_N}   L={2*LAMBDA}
+ AS={5*width_N*LAMBDA} PS={10*LAMBDA+2*width_N} AD={5*width_N*LAMBDA} PD={10*LAMBDA+2*width_N}

M9      A     out1        vdd    vdd  CMOSP   W={width_P}   L={2*LAMBDA}
+ AS={5*width_P*LAMBDA} PS={10*LAMBDA+2*width_P} AD={5*width_P*LAMBDA} PD={10*LAMBDA+2*width_P}

M10      Q     A        vdd    vdd  CMOSP   W={width_P}   L={2*LAMBDA}
+ AS={5*width_P*LAMBDA} PS={10*LAMBDA+2*width_P} AD={5*width_P*LAMBDA} PD={10*LAMBDA+2*width_P}

M11      Q      clk    D4   gnd  CMOSN   W={2*width_N}   L={2*LAMBDA}
+ AS={5*width_N*LAMBDA} PS={10*LAMBDA+2*width_N} AD={5*width_N*LAMBDA} PD={10*LAMBDA+2*width_N}

M12      D4      A     gnd    gnd  CMOSN   W={2*width_N}   L={2*LAMBDA}
+ AS={5*width_N*LAMBDA} PS={10*LAMBDA+2*width_N} AD={5*width_N*LAMBDA} PD={10*LAMBDA+2*width_N}

.ENDS tspc

```

Fig. 16: TSPC subckt

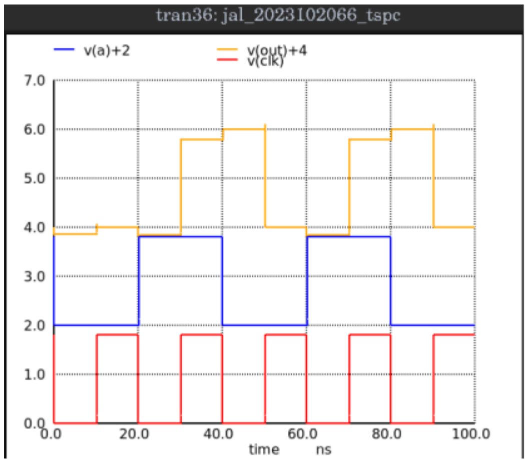
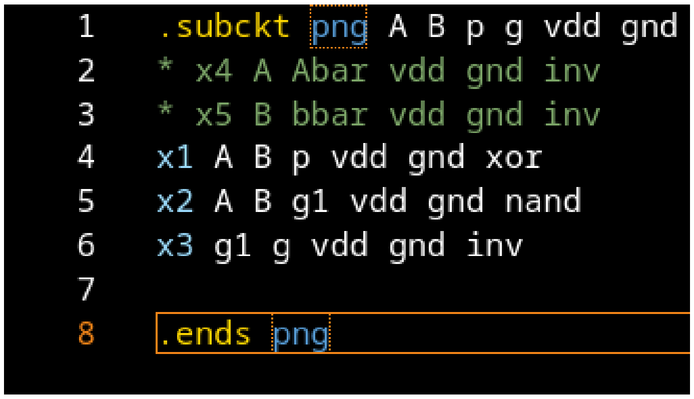


Fig. 17: TSPC Simulation

B. Block Simulations

1) Propagate and Generate Block: The subckt and simulation for png block is shown. This block generates propagate(p) and generate (g) which are necessary for cla generator block.



```
.subckt png A B p g vdd gnd
* x4 A Abar vdd gnd inv
* x5 B bbar vdd gnd inv
x1 A B p vdd gnd xor
x2 A B g1 vdd gnd nand
x3 g1 g vdd gnd inv

.ends png
```

Fig. 18: png subckt

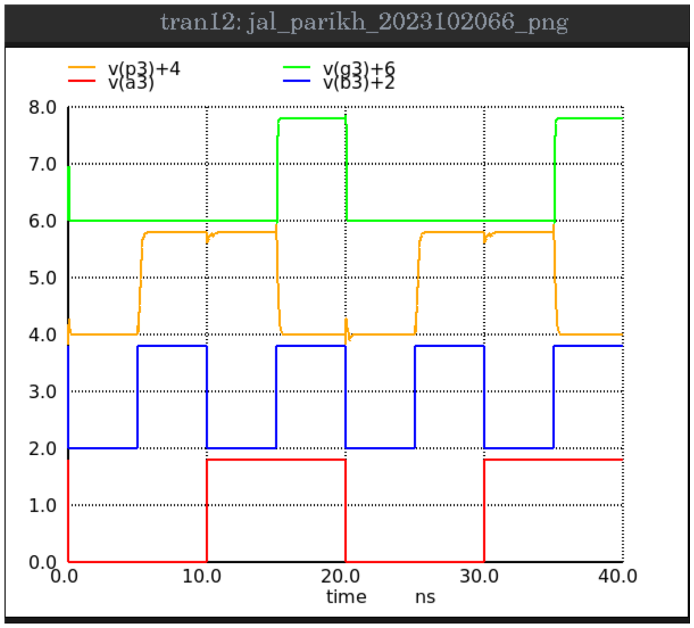


Fig. 19: png simulation

2) *CLA Generator:* This block generates the carry. The formula for all different carry is given by:

$$
\begin{aligned}
C_1 &= G_0 + P_0 C_{in} \\
C_2 &= G_1 + P_1 C_1 \\
&= G_1 + P_1 G_0 + P_1 P_0 C_{in} \\
C_3 &= G_2 + P_2 C_2 \\
&= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in} \\
C_4 &= G_3 + P_3 C_3 \\
&= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in}
\end{aligned}
$$

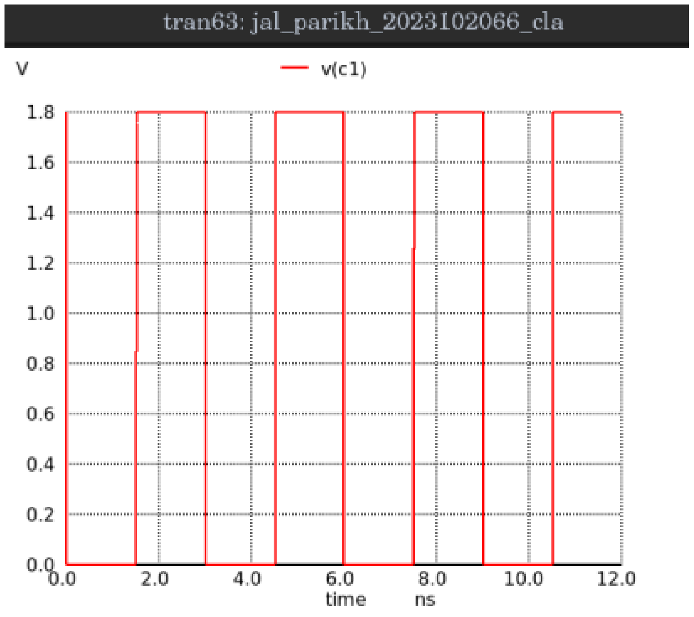


Fig. 20: Cin

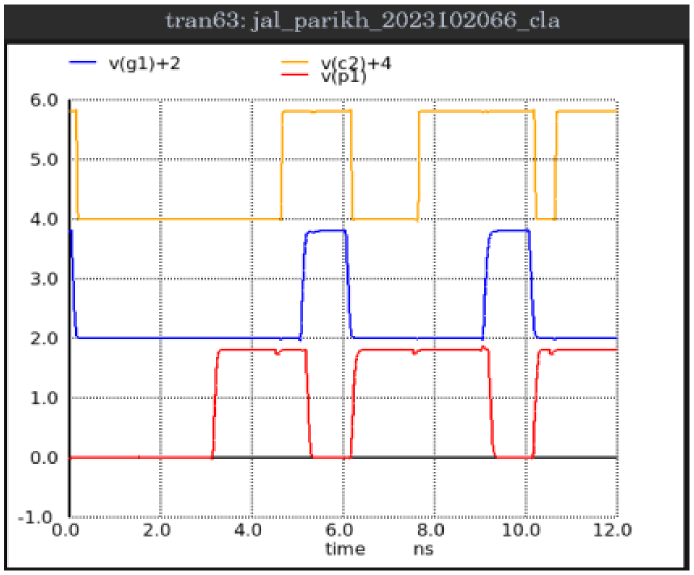


Fig. 21: C2

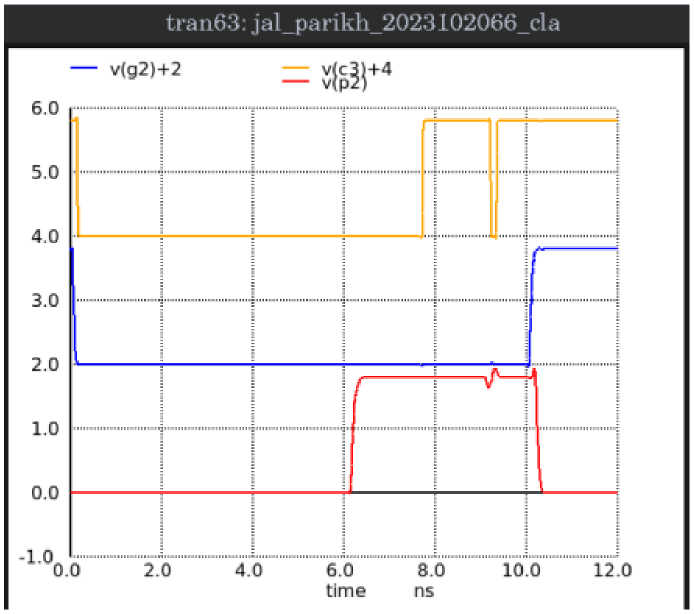


Fig. 22: C3

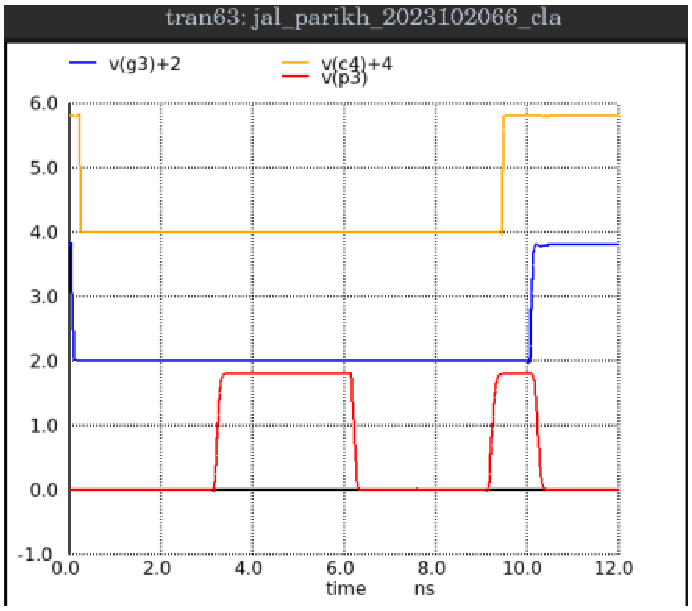


Fig. 23: C4

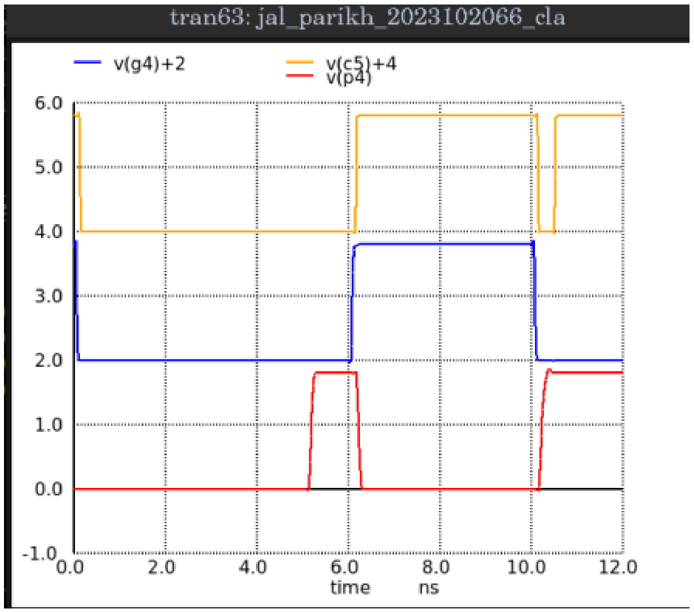


Fig. 24: Cout

3) Sum block: This block gives the final output. The simulations are provided in fig 24, 25 26, 27.

$$\begin{array}{ccccc} & a_4 & a_3 & a_2 & a_1 \\ + & b_4 & b_3 & b_2 & b_1 \\ \hline c_5 & s_4 & s_3 & s_2 & s_1 \end{array}$$

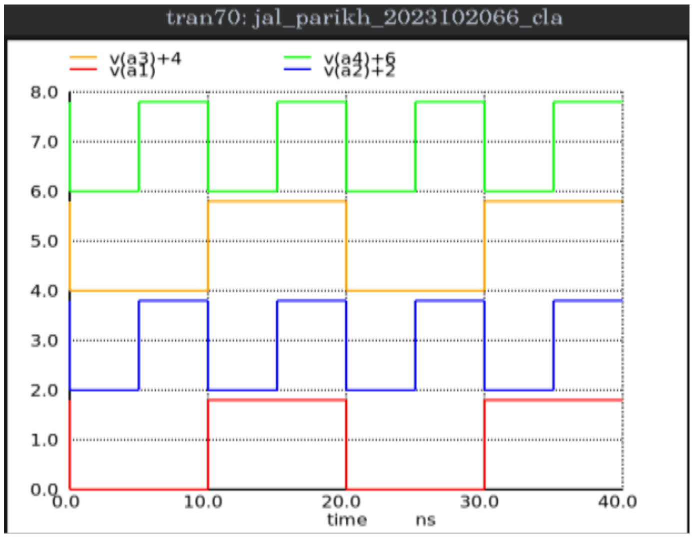


Fig. 25: Input A

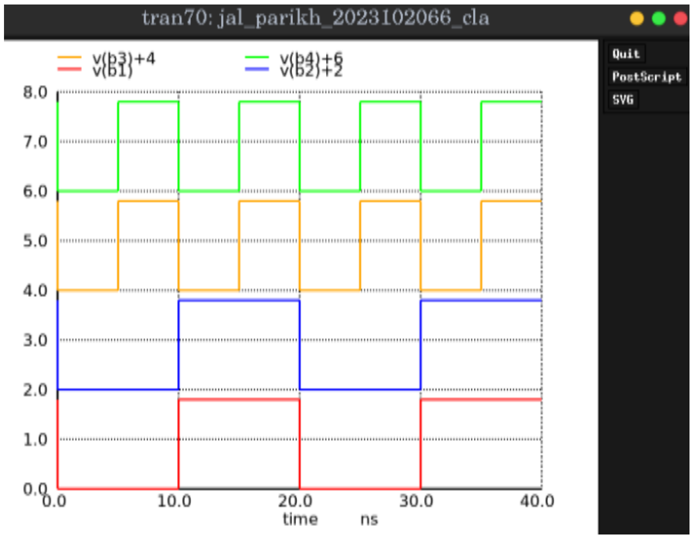


Fig. 26: Input B

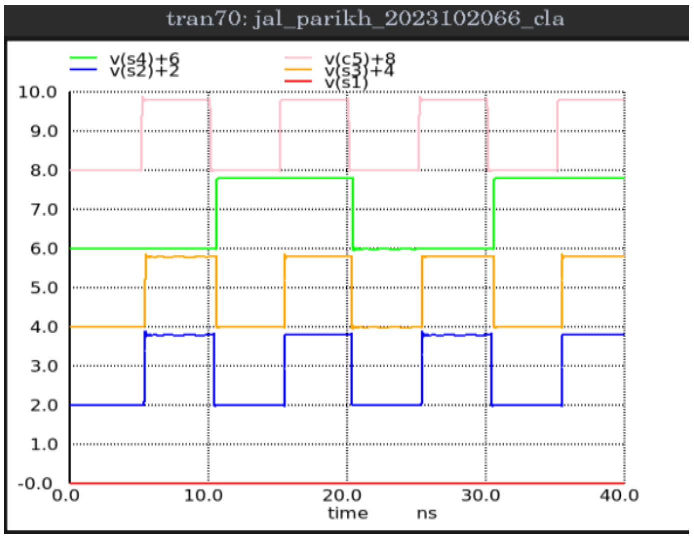
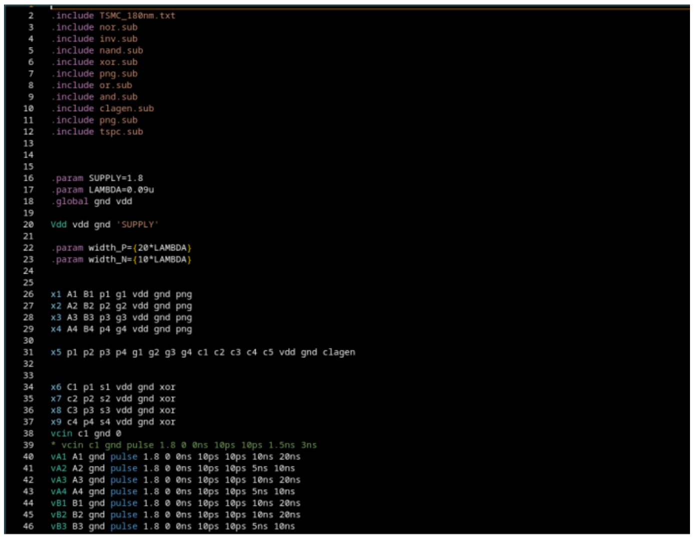


Fig. 27: Sum output



```
.include TSMC_180nm.txt
.include nor.sub
.include inv.sub
.include nand.sub
.include xor.sub
.include png.sub
.include or.sub
.include and.sub
.include clagen.sub
.include png.sub
.include tspc.sub



.param SUPPLY=1.8
.param LAMBDA=0.09u
.global gnd vdd

Vdd vdd gnd 'SUPPLY'

.param width_P={20*LAMBDA}
.param width_N={10*LAMBDA}


x1 A1 B1 p1 g1 vdd gnd png
x2 A2 B2 p2 g2 vdd gnd png
x3 A3 B3 p3 g3 vdd gnd png
x4 A4 B4 p4 g4 vdd gnd png

x5 p1 p2 p3 p4 g1 g2 g3 g4 c1 c2 c3 c4 c5 vdd gnd clagen


x6 C1 p1 s1 vdd gnd xor
x7 c2 p2 s2 vdd gnd xor
x8 C3 p3 s3 vdd gnd xor
x9 c4 p4 s4 vdd gnd xor
vcin c1 gnd 0
* vcin c1 gnd pulse 1.8 0 0ns 10ps 10ps 1.5ns 3ns
vA1 A1 gnd pulse 1.8 0 0ns 10ps 10ps 10ns 20ns
vA2 A2 gnd pulse 1.8 0 0ns 10ps 10ps 5ns 10ns
vA3 A3 gnd pulse 1.8 0 0ns 10ps 10ps 10ns 20ns
vA4 A4 gnd pulse 1.8 0 0ns 10ps 10ps 5ns 10ns
vB1 B1 gnd pulse 1.8 0 0ns 10ps 10ps 10ns 20ns
vB2 B2 gnd pulse 1.8 0 0ns 10ps 10ps 10ns 20ns
vB3 B3 gnd pulse 1.8 0 0ns 10ps 10ps 5ns 10ns
```

Fig. 28: Sum Subckt

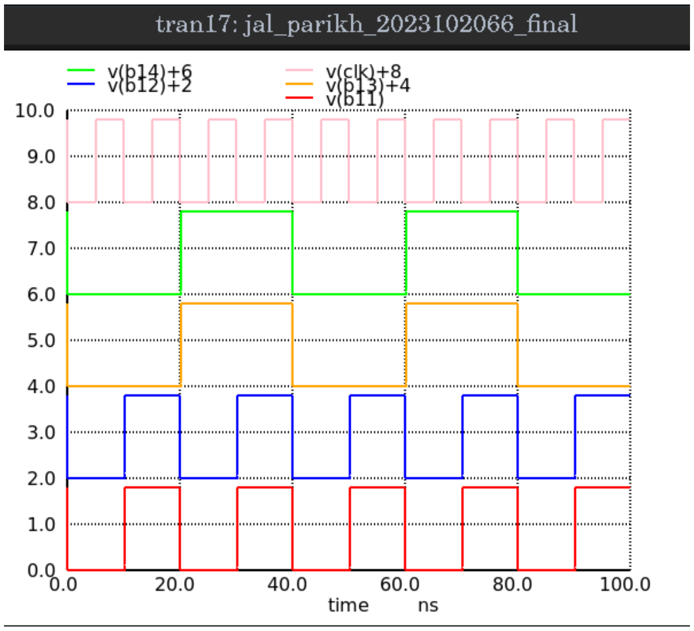


Fig. 30: Input B

C. Final Circuit

After adding flipflops to input and output bits, my circuit follows clock and output bits arrive at next rising edge of the cycle as shown in fig 28,29,30.

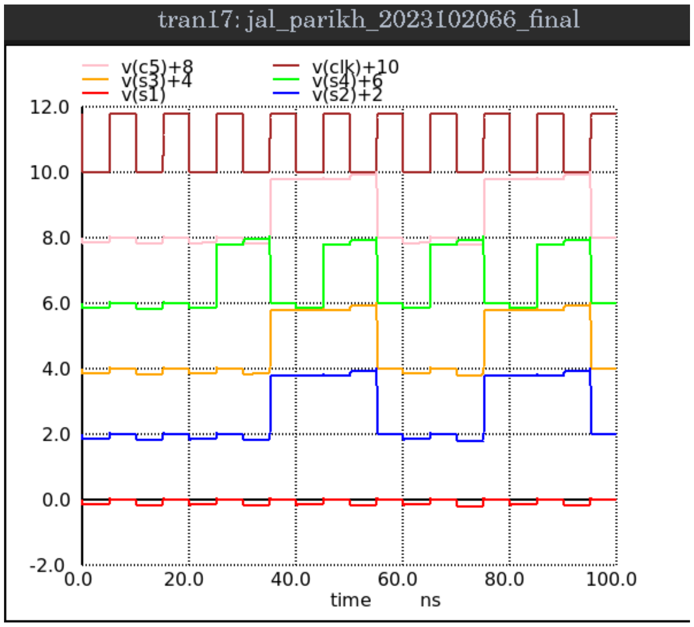


Fig. 31: sum bits

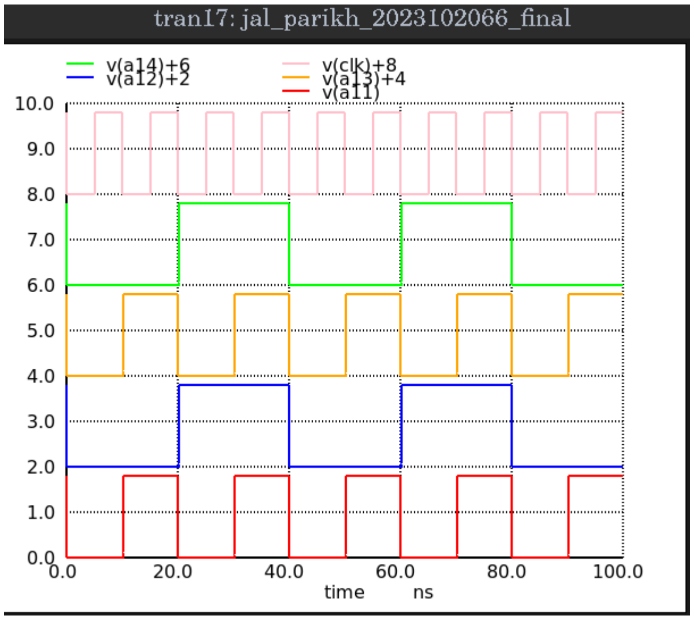


Fig. 29: Input A

V. Magic Layout

A. Gates

Layout for all gates are shown below

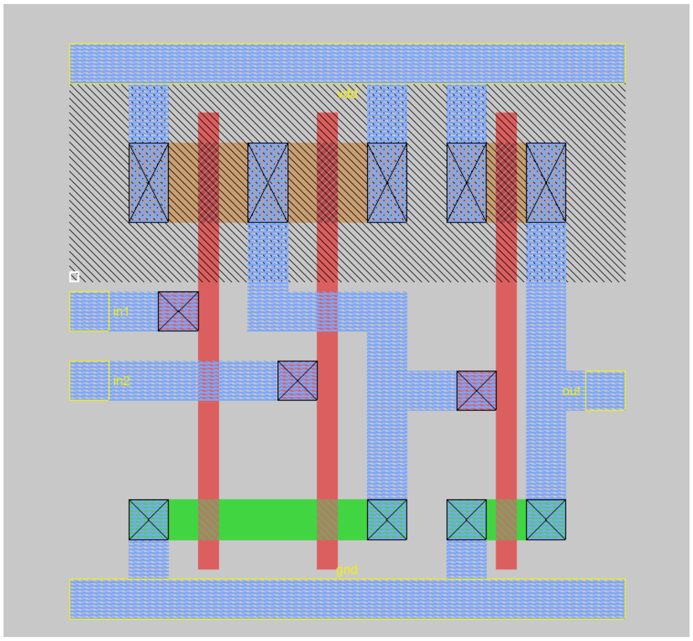


Fig. 32: And Gate

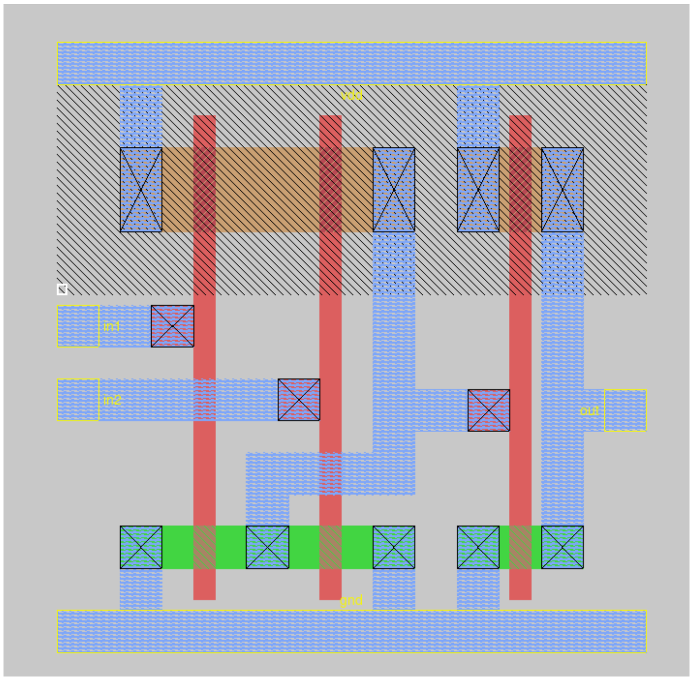


Fig. 33: Or Gate

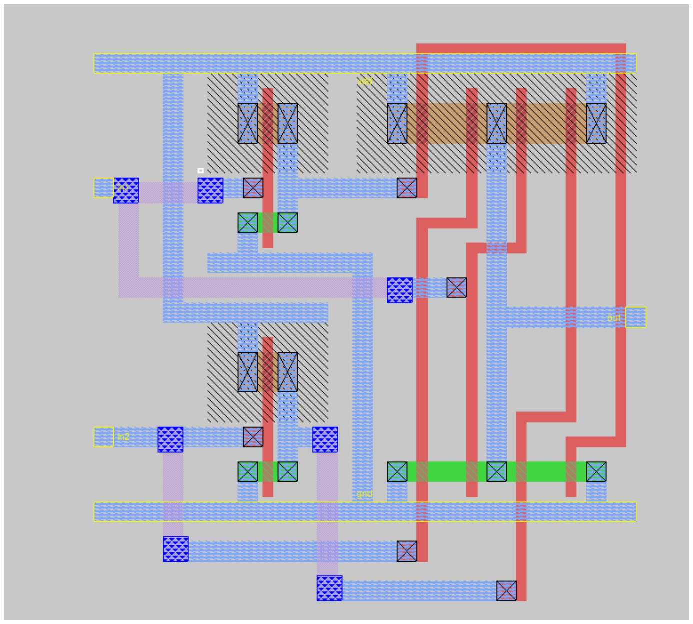


Fig. 34: Xor Gate

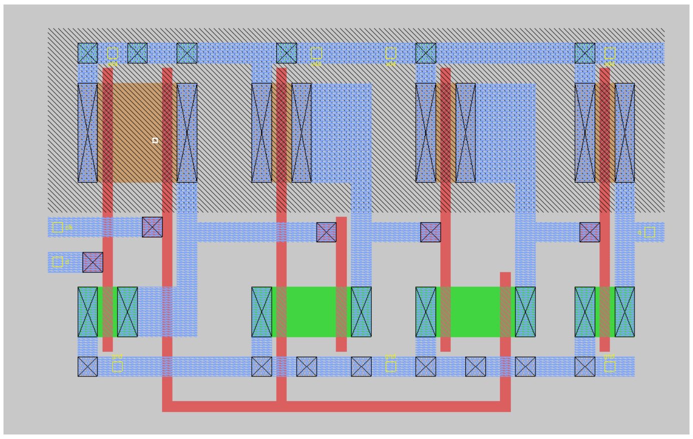


Fig. 35: TSPC flipflop

B. Blocks

Layout for the blocks are:

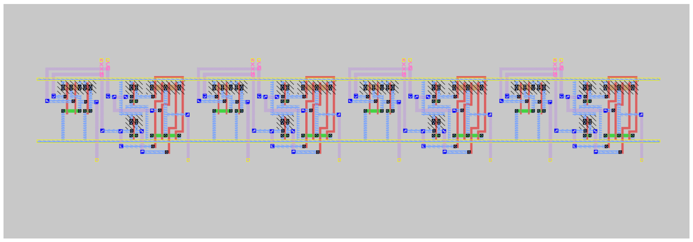


Fig. 36: png block

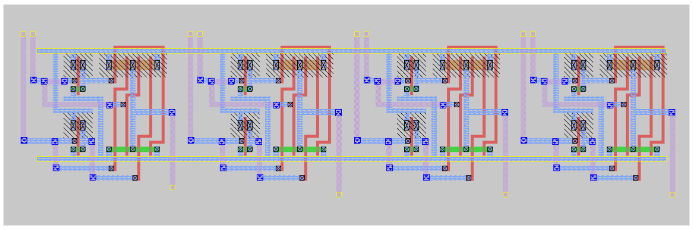


Fig. 37: sum block

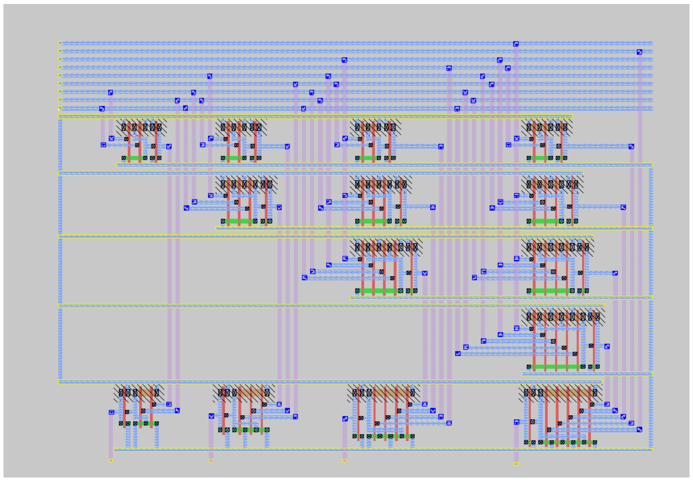


Fig. 38: Cla generator block

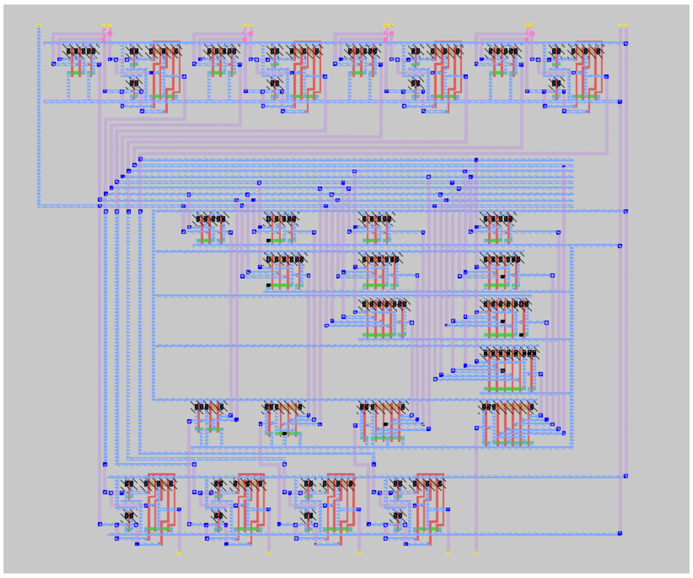


Fig. 39: Adder Block(All 3 combined)

C. Floor Plan of the Layout

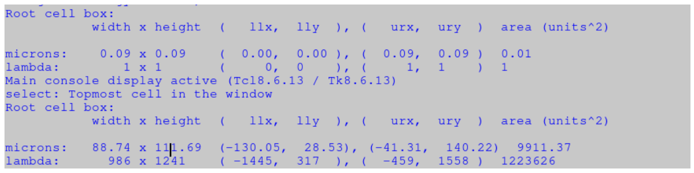
```
Root cell box:
   width x height ( llx, lly ), ( urx, ury )
       area (units^2)

microns: 0.09 x 0.09 ( 0.00, 0.00 ), ( 0.09,
    0.09 ) 0.01
lambda: 1 x 1 ( 0, 0 ), ( 1, 1 ) 1

Main console display active (Tcl8.6.13 / Tk8
    .6.13)
select: Topmost cell in the window

Root cell box:
   width x height ( llx, lly ), ( urx, ury )
       area (units^2)

microns: 88.74 x 111.69 ( -130.05, 28.53), (
    -41.31, 140.22) 9911.37
lambda: 986 x 1241 (-1445, 317), ( -459, 1558 )
     1223626
```


```
Root cell box:
           width x height  (   llx,  lly  ), (   urx,  ury  )  area (units^2)

microns:    0.09 x 0.09    (  0.00,  0.00 ), (  0.09,  0.09 )  0.01
lambda:        1 x 1       (     0,  0    ), (     1,  1    )  1
Main console display active (Tcl8.6.13 / Tk8.6.13)
select: Topmost cell in the window
Root cell box:
           width x height  (   llx,  lly  ), (   urx,  ury  )  area (units^2)

microns:   88.74 x 111.69  (-130.05,  28.53), (-41.31,  140.22)  9911.37
lambda:      986 x 1241    ( -1445,  317  ), (  -459,  1558 )  1223626
```
Fig. 40: Floor plan

VI. Post Layout Simulations

A. Gates

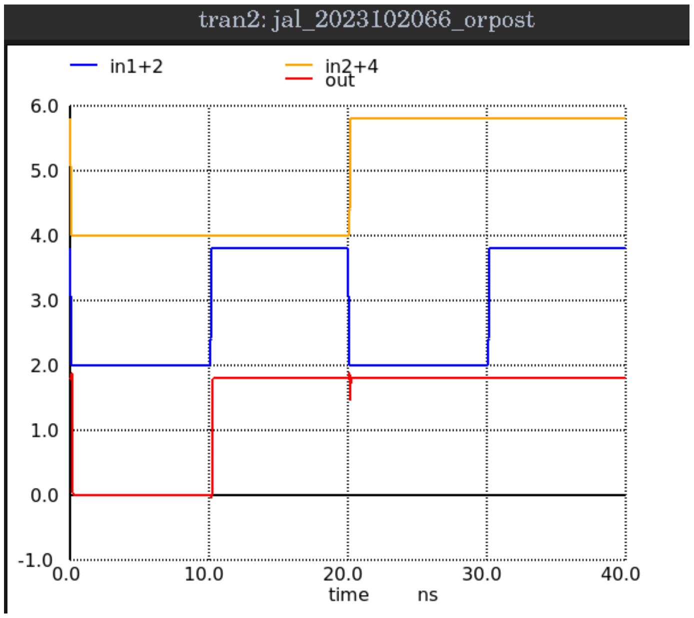


Fig. 41: post layout Or gate simulation

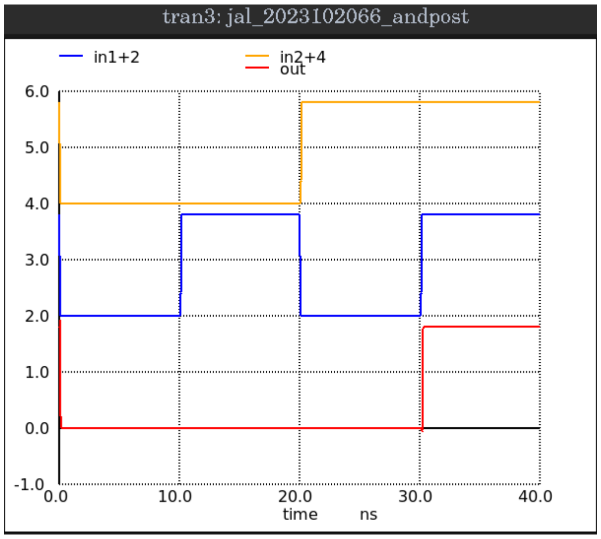


Fig. 42: Post layout AND gate simulation

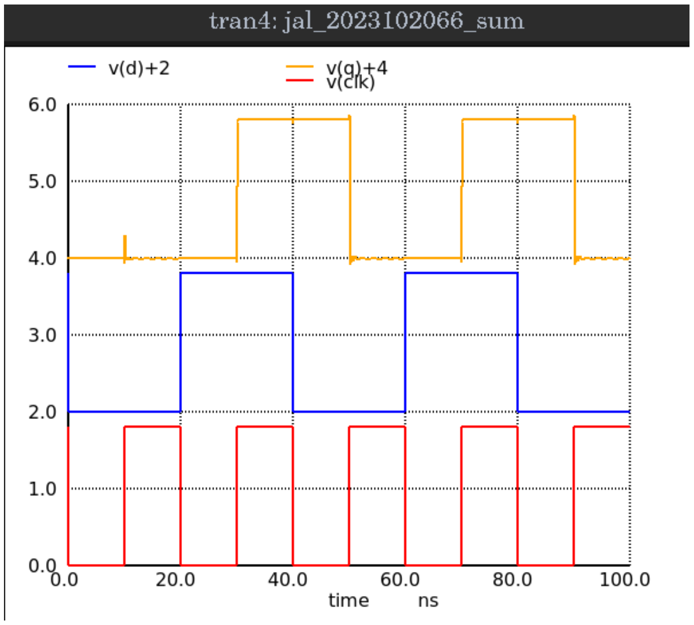


Fig. 43: post layout TSPC flipflop simulation

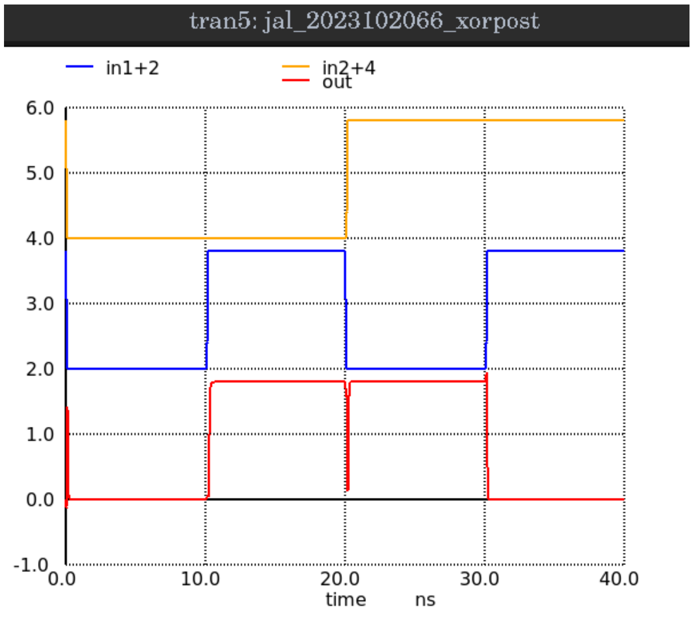


Fig. 44: post layout XOR gate simulation

B. Blocks

1) png block: The simulations of p3 and g3 are shown in fig 40. Other Pi and Gi are similar to this.

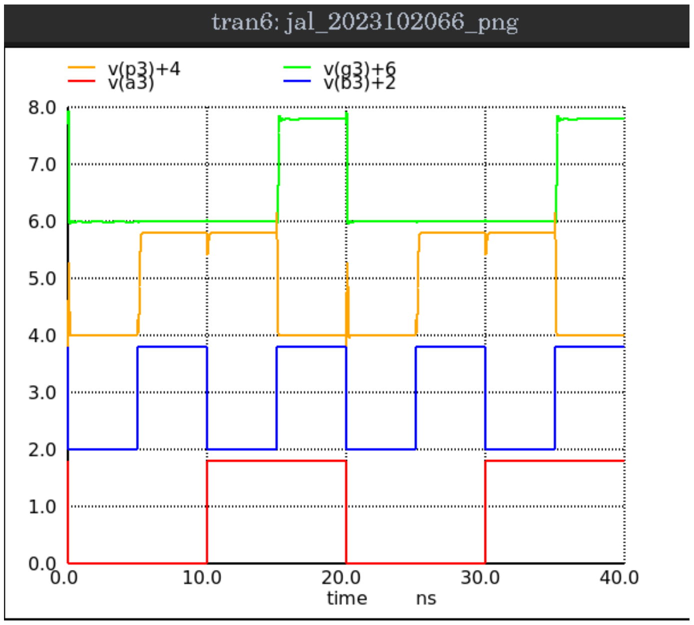


Fig. 45: post layout simulation of png block

2) CLA gen outputs: The simulations of Cin,C1,C2,C3,C4 are shown in fig 45-49.

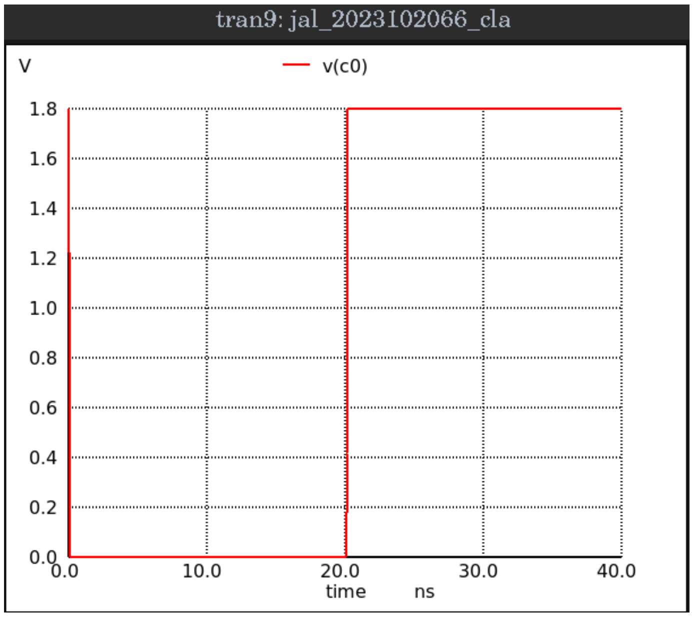


Fig. 46: Cin

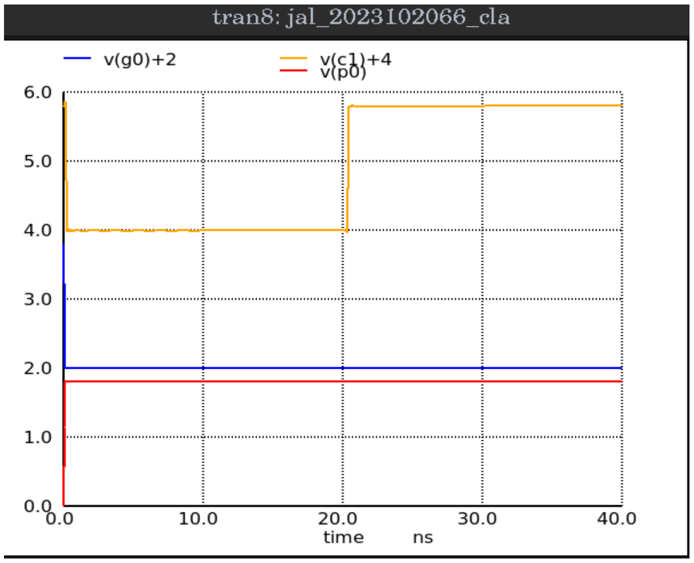


Fig. 47: C1

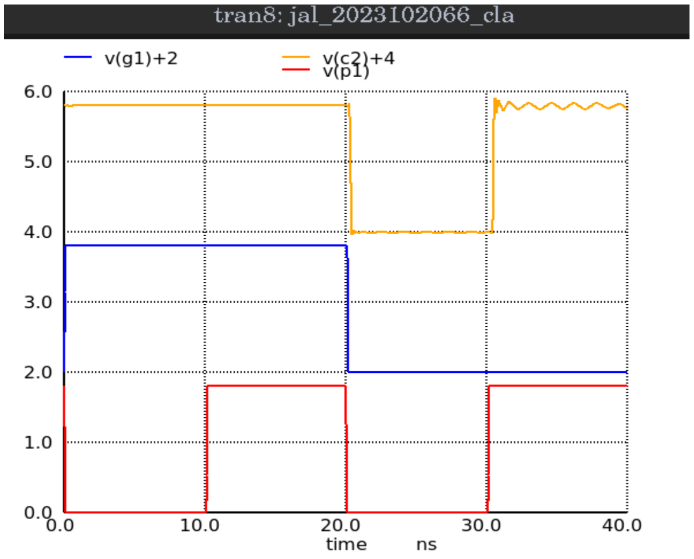


Fig. 48: C2

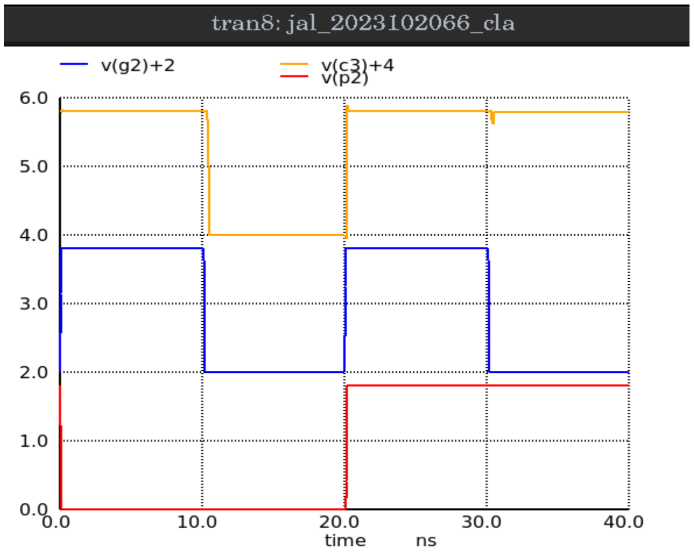


Fig. 49: C3

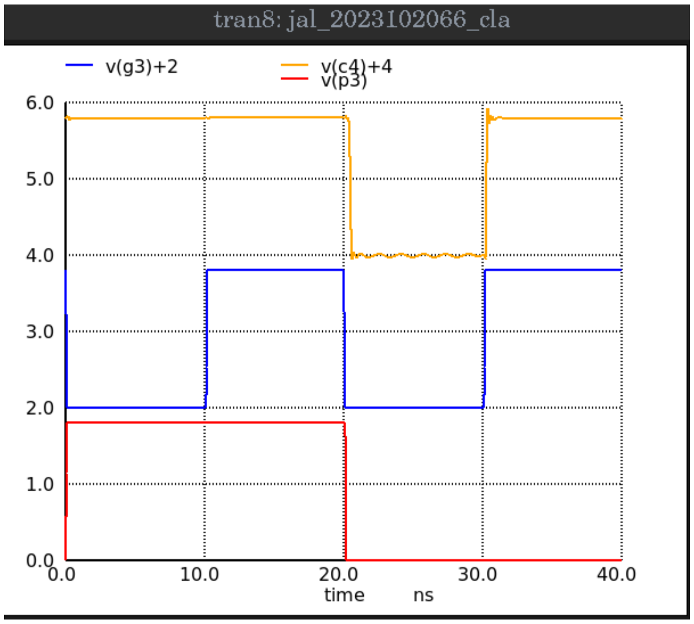


Fig. 50: Cout

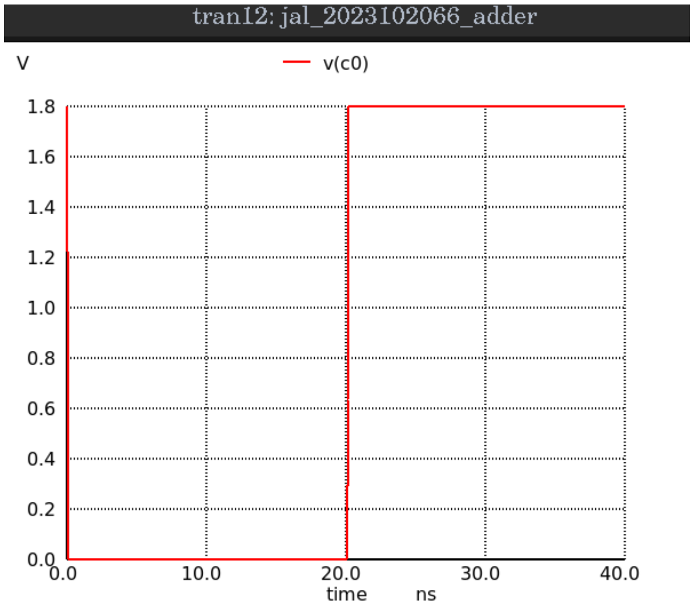


Fig. 52: Cin

3) *Sum Block:* For sum block, we need P and C only as seen in the diagram earlier. Simulation for S2 is shown here, other follow the same type.

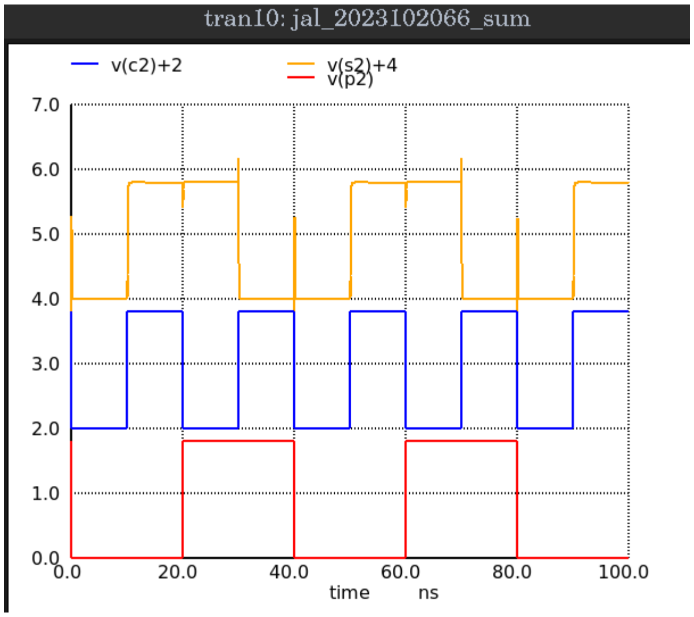


Fig. 51: post layout simulation of sum block

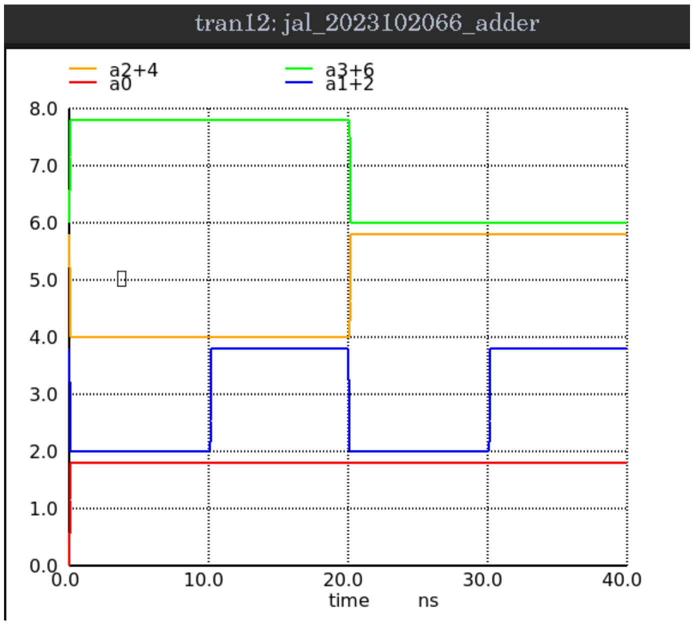


Fig. 53: Input A

4) *Adder block - All 3 combined:* Post layout simulations of adder block without using flipflops are shown in fig 51-54.

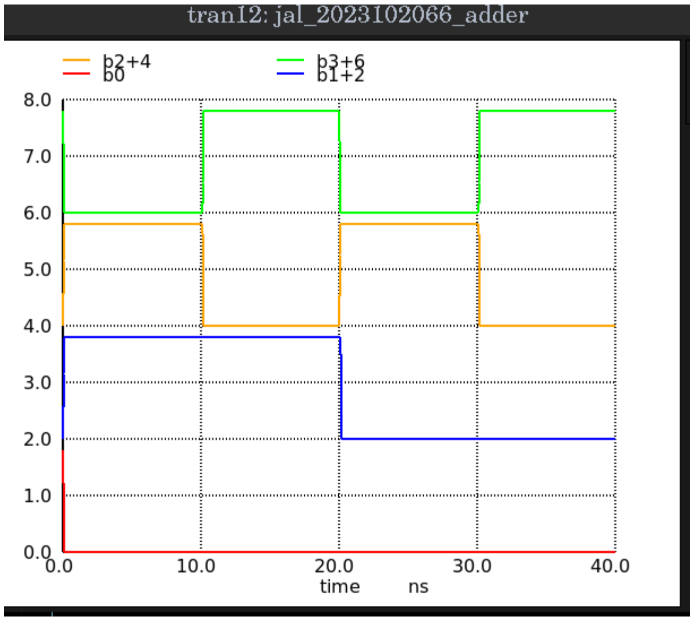


Fig. 54: Input B

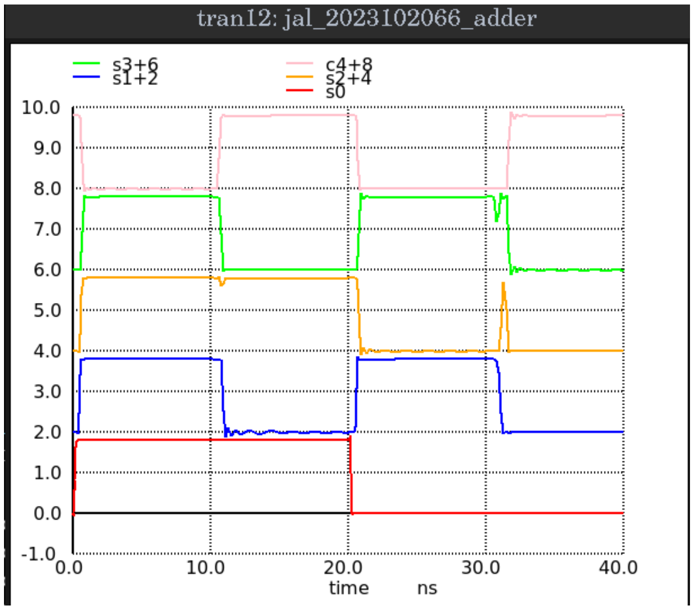


Fig. 55: Output Sum Bits

5) Final Block: The post layout simulations after adding flipflops and connecting all blocks are shown below

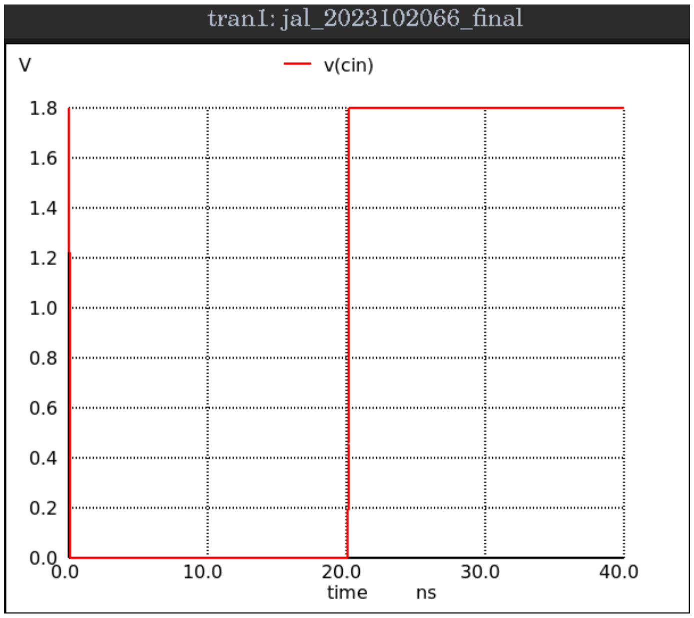


Fig. 56: Cin

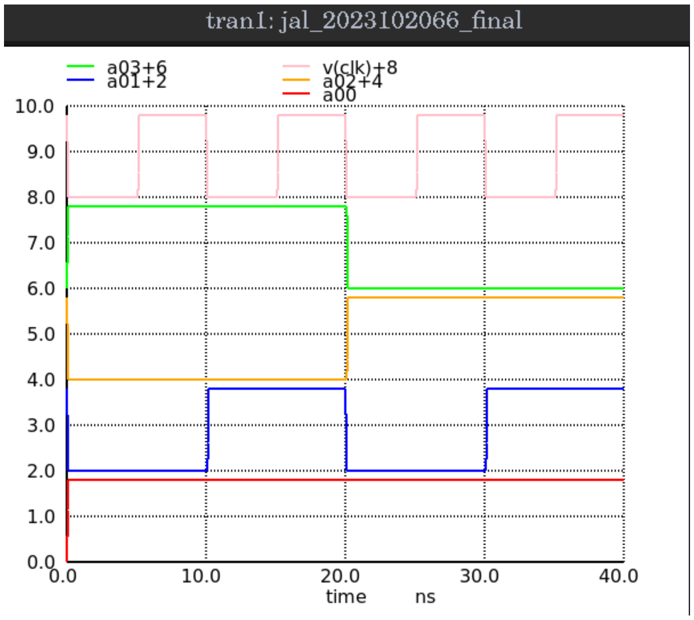


Fig. 57: Input A

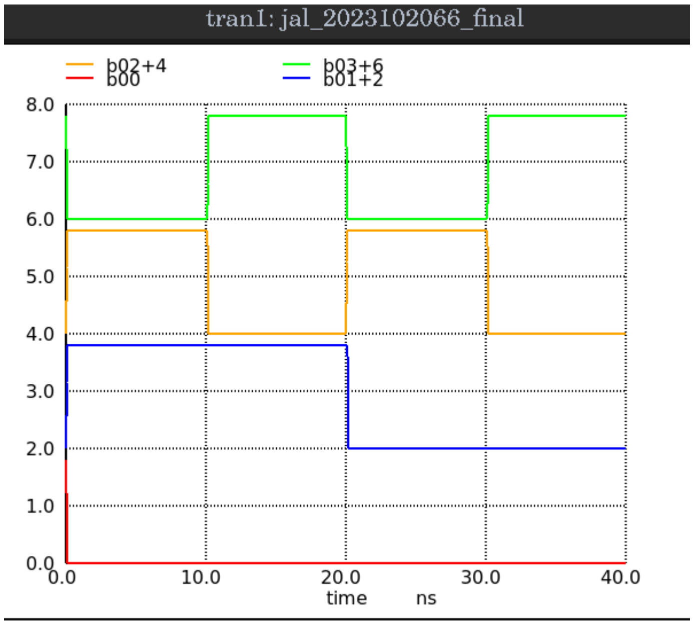


Fig. 58: Input B

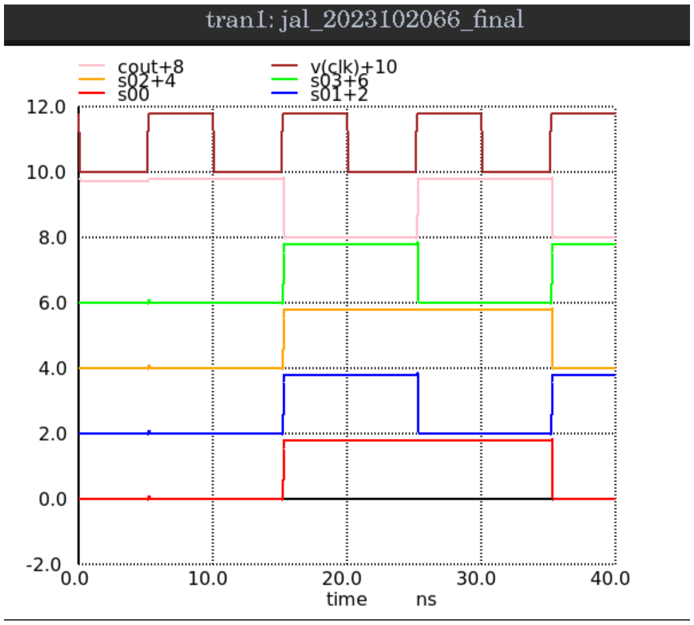


Fig. 59: Sum

VII. Verilog and GTKWave Simulations

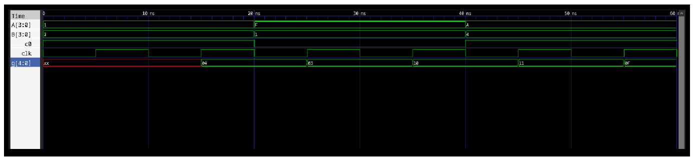
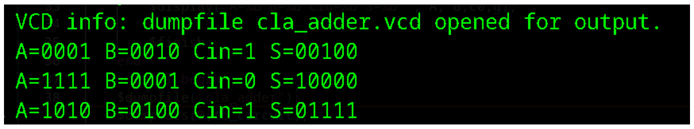


Fig. 60: GTKWave



```
VCD info: dumpfile cla_adder.vcd opened for output.
A=0001 B=0010 Cin=1 S=00100
A=1111 B=0001 Cin=0 S=10000
A=1010 B=0100 Cin=1 S=01111
```

Fig. 61: Terminal Results

VIII. Timing Analysis

A. Setup Time

1) Prelayout and Postlayout Timing: I found out setup time by changing input just before clock edge appeared. I continued to bring input close to clock edge and stopped when I didn't get desired output. The setup time for TSPC flipflop came out to be **0.135ns**.

The postlayout setup time came out to be **0.125ns**

B. Hold Time

1) Prelayout and Postlayout Timing: Since I implemented TSPC flipflop logic and we know TSPC has 0 hold time so the hold time is **0ns**

C. Clock to Q Delay

1) Prelayout Timing and Postlayout Timing : I found out Tpcq by measuring the delay between clock's positive edge and output change. I did it using T_{pcq} using:

```
meas tran clk_to_Q_delay trig v(clk) val=0.9
rise=1 targ v(Q) val=0.9 rise=1
```

T_{pcq} came out to be 0.077 ns. Similarly the post layout simulation timing came out to be **0.086ns**.

D. Adder Delay

1) Prelayout Timing and Postlayout Timing: For adder delay, I observed that output bit S_4 had the maximum delay, and it came out to be 0.396 ns while going from 0 to 1 (rise) and 0.353 ns while going from 1 to 0 (fall). So for the worst-case scenario, I am considering 0.396 **ns**.

Similarly, the post-layout simulation timing came out to be 0.571 ns while going from 0 to 1 (rise) and 0.638 ns while going from 1 to 0. Hence, for the worst-case scenario, I considered T_{pcq} as 0.638 **ns**.

Command used for finding delay:

```
meas tran worst_case_delay trig v(A0) val=0.9
fall=1 targ v(S2) val=0.9 fall=1
```

E. Maximum Clock Speed

$$T_{\text{clk}} \geq T_{\text{su}} + T_{\text{pcq}} + T_{\text{pd}}$$

$$T_{\text{su}} = 0.135\,\text{ns}$$

$$T_{\text{pcq}} = 0.077\,\text{ns}$$

$$T_{\text{pd}} = 0.396\,\text{ns}$$

$$T_{\text{clk}} \geq 0.135 + 0.077 + 0.396 \approx 0.608\,\text{ns}$$

$$f_{\text{clk}} \leq \frac{1}{T_{\text{clk}}}$$

$$\text{Maximum Frequency} = 1.6 \times 10^9\ \text{Hz}$$

For postlayout

$$T_{\text{su}} = 0.125\,\text{ns}$$

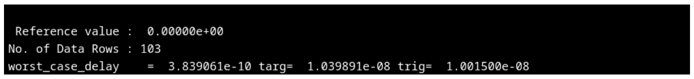
$$T_{\text{pcq}} = 0.0886\,\text{ns}$$

$$T_{\text{pd}} = 0.638\,\text{ns}$$

$$T_{\text{clk}} \geq 0.125 + 0.088 + 0.638 \approx 0.851\,\text{ns}$$

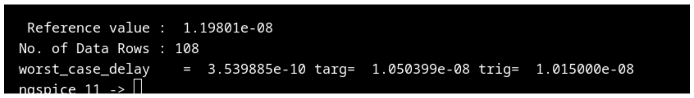
$$f_{\text{clk}} \leq \frac{1}{T_{\text{clk}}}$$

$$\text{Maximum Frequency} = 1.17 \times 10^9\,\text{Hz}$$



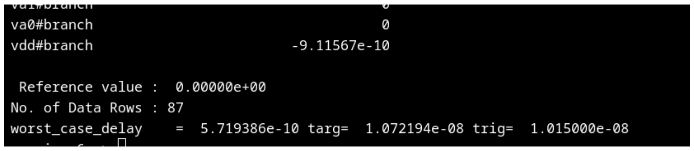
```
 Reference value :  0.00000e+00
No. of Data Rows : 103
worst_case_delay    =  3.839061e-10 targ=  1.039891e-08 trig=  1.001500e-08
```

Fig. 62: prelayout rise adder delay



```
 Reference value :  1.19801e-08
No. of Data Rows : 108
worst_case_delay    =  3.539885e-10 targ=  1.050399e-08 trig=  1.015000e-08
ngspice 11 ->
```

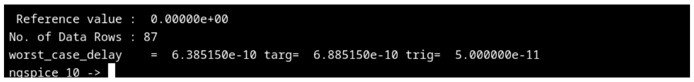
Fig. 63: prelayout fall adder delay



```
va0#branch                              0
vdd#branch                   -9.11567e-10

 Reference value :  0.00000e+00
No. of Data Rows : 87
worst_case_delay    =  5.719386e-10 targ=  1.072194e-08 trig=  1.015000e-08
```

Fig. 64: Postlayout rise adder delay



```
 Reference value :  0.00000e+00
No. of Data Rows : 87
worst_case_delay    =  6.385150e-10 targ=  6.885150e-10 trig=  5.000000e-11
ngspice 10 ->
```

Fig. 65: Postlayout fall adder delay

IX. References

1) *Digital Logic and Computer Design* by Morris Mano
2) *Digital Integrated Circuits* by Jan M. Rabaey
3) GeeksforGeeks
4) DSM Slides of Aftab Sir