CONTENTS

VLSI Design Course Project
Designing and Implementing 5 bit CLA

Nagamalla Sai Abhinav
2024102037
Department of Electronics and Communication Engineering
IIIT Hyderabad
saiabhinav.nagamalla@students.iiit.ac.in

***Abstract*—This project is about designing an efficient 5-bit Carry Look-Ahead Adder (CLA) using TSMC 180 nm technology. A CLA implementation of an adder is a massive improvement over the conventional Ripple Carry Adder (RCA) because the carry bits are precomputed in CLA unlike in RCA where we need to wait for the carry to propagate for higher bits, making it slower. We discuss the design proposal, verify the functionality, do performance analysis for metrics like delay, power, etc. Simulations are done with the help of NGSPICE and MAGIC Layout tool. Logic verification is done with the help of Verilog and the hardware implementation is done with the help of an FPGA using Vivado and verifying the output with oscilloscope.**

I. Introduction

Addition is a fundamental operation and is the basis for digital circuits. Traditional RCAs, though straightforward to implement, are slower due to the linear growth of delay as the number of bits increases. The CLA improves this by using a parallelized approach, enabling faster computation and eliminating the ripple effect. This is achieved by precomputing the carry bits, which saves propagation time for the higher bits. The proposed CLA takes two 5 bit numbers as inputs and outputs the 5 bit sum and the carry bit. The design integrates the CLA logic with D-flip-flops for synchronizing inputs and outputs. Inputs are available on the rising edge of the clock and the output will be ready on the immediate rising edge.

The initial carry input i.e. c_{in} is taken to be 0 i.e. the adder just adds two 5 bit numbers and no initial carry is considered.

II. Design Proposal

- V_{DD} = 1.8 V
- Technology node = $90n$

A. Generate, Propagate, Carry and Sum

The CLA uses two signals called as Generate (denoted by g) and propagate (denoted by p). For the two i^{th} bits a_i and b_i they are given as

$$g_i = a_i b_i,$$
$$p_i = a_i \oplus b_i.$$

The generate signal determines if a carry will be generated at the i^{th} bit, regardless of the input carry. The propagate signal indicates if a carry from the previous stage will propagate through the i^{th} bit.

So, the carry out (denoted by c_i) is as follows:

$$c_i = g_i + (p_i c_{i-1})$$

where c_{i-1} denotes the carry from the previous stage.

Once these carry signals are determined, the sum at i^{th} bit i.e. s_i is evaluated as follows.

$$s_i = p_i \oplus c_{i-1}$$

Now, we can evaluate all the carry bits using the above relations recursively and we get the following equations:

$$c_1 = g_0 + p_0 c_0 = g_0,$$
$$c_2 = g_1 + p_1 g_0,$$
$$c_3 = g_2 + p_2 g_1 + p_2 p_1 g_0,$$
$$c_4 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0,$$
$$c_5 = g_4 + p_4 g_3 + p_4 p_3 g_2 + p_4 p_3 p_2 g_1 + p_4 p_3 p_2 p_1 g_0.$$

where c_i denotes the carry to be added to the i^{th} bit and c_5 is the final carry bit (c_{out}) after the addition.

We see from the above equations that realizing these expressions directly requires high fan-in logic, such as four-input and five-input AND and OR gates. Such gates introduce larger delays because the delay increases with the logical effort of the gate, which becomes higher as the number of inputs increases.

Furthermore, g_0 appears in every carry expression, which results in a large fan-out. This increases the load driven by g_0 and further contributes to the delay of the circuit.

These limitations motivate the use of a hierarchical or group-based CLA structure which is discussed in the upcoming section.

B. Optimization

We realize that the propagate signal can be generated with an OR gate instead of an XOR gate to simplify the layout because CMOS style XOR is too complicated to implement and PTL style XOR requires both inputs and it's complements which will again complicate the circuitry.

This is because the propagate signal only needs to indicate whether a carry would pass through the i^{th} bit if it were to arrive. The condition for propagation is satisfied whenever at least one of the inputs is high. Hence,

$$p_i = a_i + b_i,$$

which can be implemented using a simple OR gate. When both a_i and b_i are 1, the bit already generates a carry ($g_i = a_i b_i = 1$), so the propagate value in that case does not affect the correctness of the carry logic. Thus, using an OR gate for p_i reduces complexity and improves delay without altering functionality.

We will still need XOR gates to generate the sum bits: the sum at the ith position is

$$s_i = (a_i \oplus b_i) \oplus c_{i-1},$$

which can be implemented using two XOR stages (one to form $a_i \oplus b_i$, and a second to combine that result with c_{i-1}). This XOR-based sum block is local to each bit and does not interfere with the carry-generation network; hence, replacing the propagate function by an OR gate only simplifies the carry logic and layout without eliminating the XORs required for sum computation.

As discussed previously, due to higher input gates and large fan-out, it is not a great idea to implement the above logic blindly. Therefore, following an idea used in the research paper [1], we can play around with De Morgan's law and obtain the following equations:

The following equations represent the expansion and simplification of carry terms (c_1 to c_5) using Generate (g) and Propagate (p) logic.

$$\begin{aligned}
c_1 &= g_0 \\
c_1 &= \overline{\overline{g_0}}
\end{aligned}$$

$$\begin{aligned}
c_2 &= g_1 + p_1 g_0 \\
\overline{c_2} &= \overline{g_1 + p_1 g_0} \\
&= \overline{g_1} \cdot \overline{p_1 g_0} \\
&= \overline{g_1} \cdot (\overline{p_1} + \overline{g_0}) \\
c_2 &= \overline{\overline{g_1} \cdot (\overline{p_1} + \overline{g_0})}
\end{aligned}$$

$$\begin{aligned}
c_3 &= g_2 + p_2 g_1 + p_2 p_1 g_0 \\
&= g_2 + p_2 g_1 + p_2 p_1 c_1 \\
\overline{c_3} &= \overline{g_2 + p_2 g_1 + p_2 p_1 c_1} \\
&= (\overline{g_2})(\overline{p_2 g_1})(\overline{p_2 p_1 c_1}) \\
&= (\overline{g_2})(\overline{p_2 g_1})(\overline{p_2 p_1} + \overline{c_1}) \\
c_3 &= \overline{(\overline{g_2})(\overline{p_2 g_1})(\overline{p_2 p_1} + \overline{c_1})} \\
&= \overline{(\overline{g_2})(\overline{p_2 g_1})} + \overline{\overline{p_2 p_1} + \overline{c_1}} \\
c_3 &= \overline{(\overline{g_2})(\overline{p_2} + \overline{g_1})} + \overline{\overline{p_2} + \overline{p_1}} \cdot c_1
\end{aligned}$$

Performing similar simplification for the higher-order terms:

$$\begin{aligned}
c_4 &= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 \\
&= g_3 + p_3 g_2 + p_3 p_2 (g_1 + p_1 g_0) \\
&= g_3 + p_3 g_2 + p_3 p_2 c_2 \\
c_4 &= \overline{\overline{g_3}(\overline{p_3} + \overline{g_2})} + \overline{\overline{p_3} + \overline{p_2}} \cdot c_2
\end{aligned}$$

$$\begin{aligned}
c_5 &= g_4 + p_4 g_3 + p_4 p_3 g_2 + p_4 p_3 p_2 g_1 + p_4 p_3 p_2 p_1 g_0 \\
&= g_4 + p_4 g_3 + p_4 p_3 (g_2 + p_2 g_1 + p_2 p_1 g_0) \\
&= g_4 + p_4 g_3 + p_4 p_3 c_3 \\
c_5 &= \overline{\overline{g_4}(\overline{p_4} + \overline{g_3})} + \overline{\overline{p_4} + \overline{p_3}} \cdot c_3
\end{aligned}$$

The fan-out of each term is limited to 3, and only 2-input gates are required. Although higher-order carry bits algebraically depend on lower-order carries, parallel computation of intermediate generate and propagate signals allows partial evaluation in advance, avoiding idle time. So, it is still a CLA only and not RCA because it doesn't wait for all the lower carries to be evaluated to evaluate higher carry.

So, our optimized circuit is much better/faster than the trivial implementation of the equations directly because of lower fan-out and the fact that only 2 input gates are used. The circuit diagram for the following is given below:

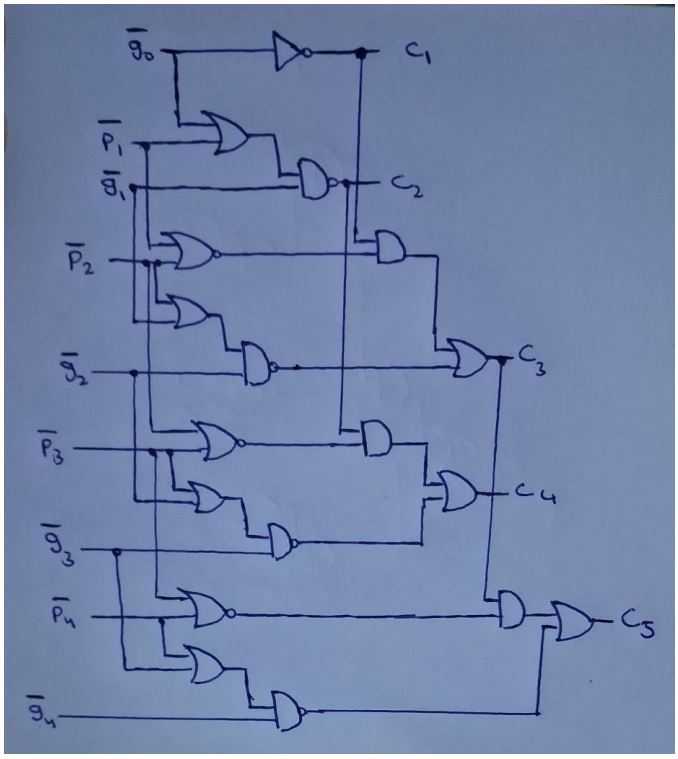


Fig. 1: Carry Generator Circuit diagram

III. Design Details

From the above circuit diagram, we see that we need an Inverter, 2 input NAND, NOR, OR, AND gates. We also need a 2 input XOR for the sum block and D Flip Flop for the synchronization. With the modifications we made, we need $\overline{g_i}$

and $\overline{p_i}$ instead of g_i and p_i. So to generate these signals, we can use 2 input NAND and NOR respectively. Getting these signals in terms of NAND and NOR will save layout area because OR and AND take up extra area due to inverter at the end. These generated carry bits are then XORed with the output of $a \oplus b$ to get the final sum bit.

To size these, the worst case resistance of these in both PUN and PDN are equated to that of the reference CMOS inverter which is given whose size is : $W_n = 10\lambda$ and $W_p = 20\lambda$.

To design D Flip flop, I used the design given in [2]. It uses True Single Phase Clock (TSPC) style. The circuit diagram for it is:

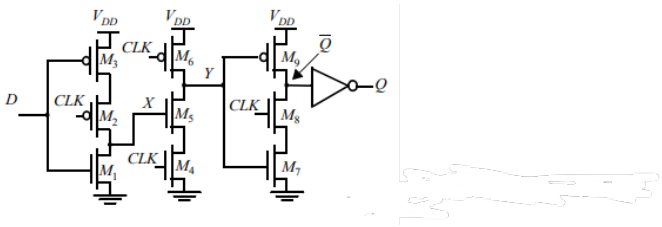


Fig. 2: Circuit Diagram of TSPC style D Flip Flop

It is sized such that at all the 4 stages of the flip flop, the sizing condition is met. But, glitches are possible [2] which can be avoided by making $M_4 - M_5$ pull down path stronger and/or $M_7 - M_8$ pull down path weaker. I chose the former option by doubling the theoretically calculated widths. So, my final sizing is as follows (M_{10} is the NMOS in the CMOS inverter and M_{11} is the PMOS in the same)

Transistor	Size (in λ)
M_1	10
M_2	40
M_3	40
M_4	40
M_5	40
M_6	20
M_7	20
M_8	20
M_9	20
M_{10}	10
M_{11}	20

To make NAND, NOR and NOT gates, I used static CMOS style logic and the sizing is done accordingly. To make AND and OR, I added an Inverter at the outputs of NAND and NOR gates respectively. So, I am only mentioning the sizes of the former 3 gates in the table below.

Gate	NMOS width (in λ)	PMOS width (in λ)
NAND	20	20
NOR	10	40
NOT	10	20

To make the XOR gate, I used PTL logic style. But, it degrades logic high voltage levels which can affect the following stages. So, I added a skewed Inverter at the output to recognize the HIGH at $V_{DD} - |V_{TP}|$ only. But, it will invert the output. So, I am making a XNOR gate with PTL followed by an Inverter which not only restores logic levels, but also inverts the logic back to XOR. The circuit diagram is as follows:

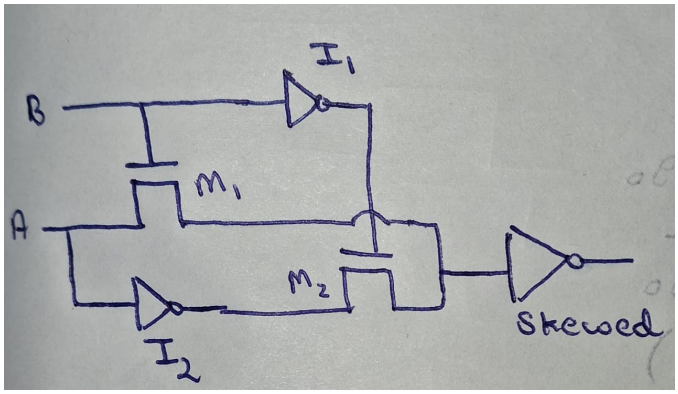


Fig. 3: PTL Implementation of XOR gate

I_1 and I_2 are normal CMOS inverters while the inverter at the end is skewed to decrease NM_H which is done by making NMOS stronger. So, it's $W_P = 20\lambda$ and $W_N = 20\lambda$ (to make NMOS stronger). The Pass Transistors M_1 and M_2 are controlled switches and thus can be sized arbitrarily. I sized them with 20λ, not too small that the resistance is larger or not too big that it takes up a lot of area.

IV. NGSPICE Simulation

A. Inverter

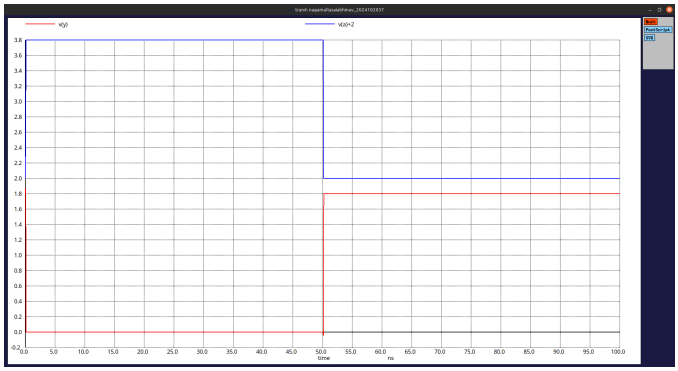


Fig. 4: Inverter Plot : Pre-Layout

B. NAND Gate

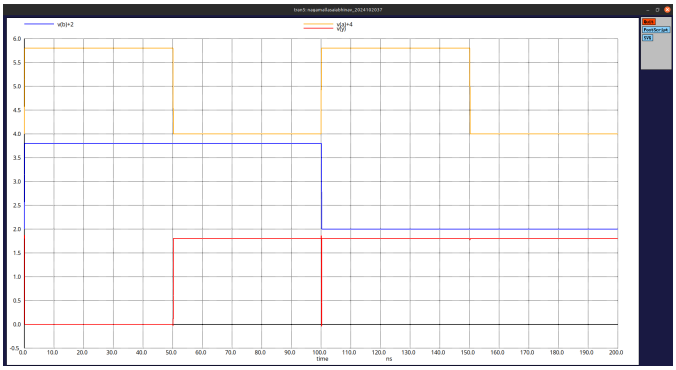


Fig. 5: 2 input NAND Plot : Pre-Layout

C. NOR Gate

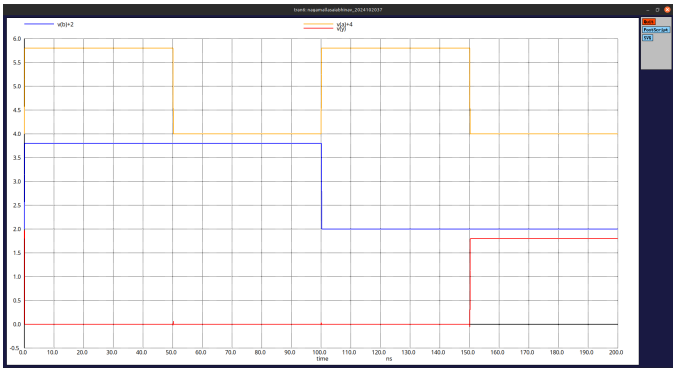


Fig. 6: 2 input NOR Plot : Pre-Layout

D. AND Gate

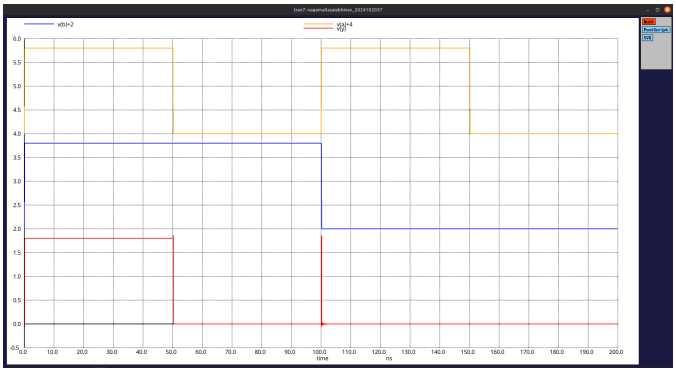


Fig. 7: 2 input AND Plot : Pre-Layout

E. OR Gate

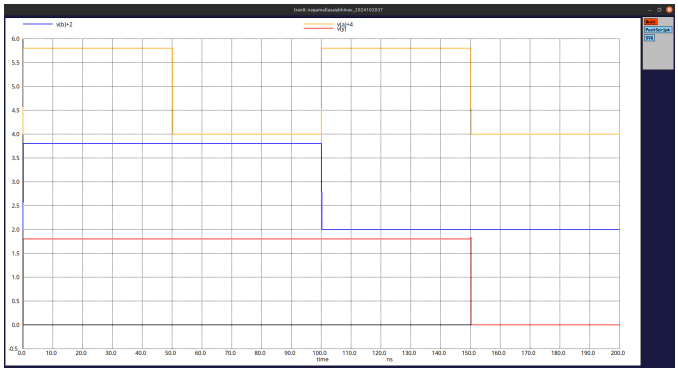


Fig. 8: 2 input OR Plot : Pre-Layout

F. XOR Gate

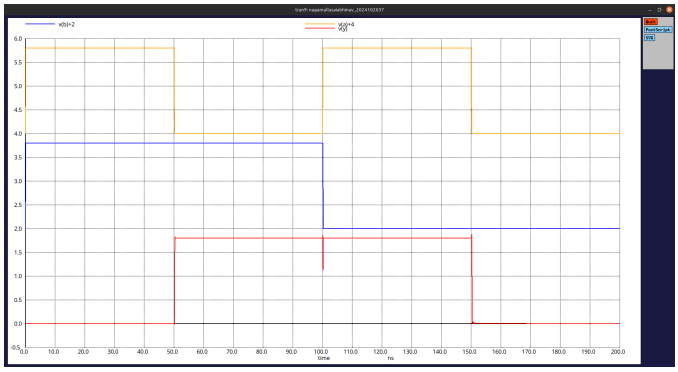


Fig. 9: 2 input XOR Plot : Pre-Layout

G. D Flip Flop

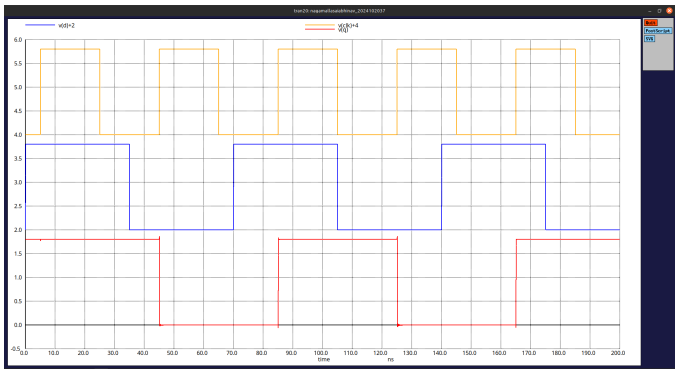


Fig. 10: TSPC D Flip Flop Plot : Pre-Layout

H. PG Block

This block just generates $\overline{p_i}$ and $\overline{g_i}$ signals using NOR and NAND gates respectively.

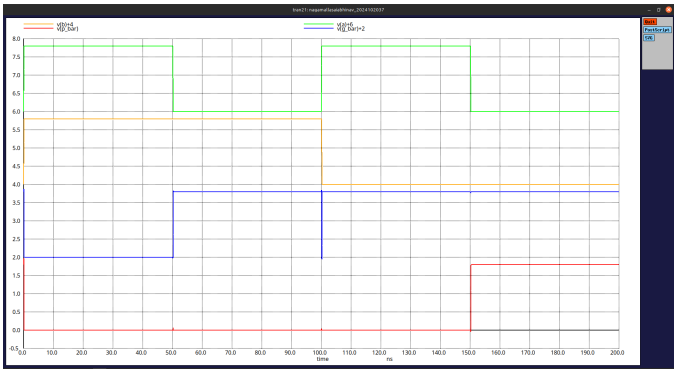


Fig. 11: PG Block Plot : Pre-Layout

I. Sum Block

This block generates the sum of a, b, and carry i.e. it outputs $a \oplus b \oplus c$ as the output sum (s).

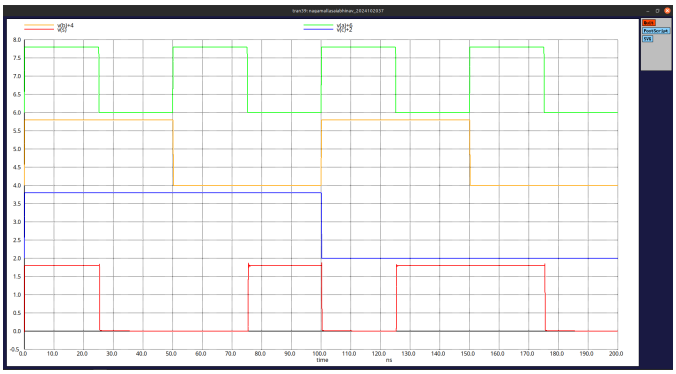


Fig. 12: Sum Block Plot : Pre-Layout

V. D Flip Flop Characterization from NGSPISE

A. TSPC Flip Flop

The TSPC D-Flip-Flop operates with a single clock phase, avoiding the need for complementary clock signals. There is one implementation which uses master-slave connection requiring 12 transistors. The one which is implemented takes 1 lesser transistor. The sizing is explained in the previous section.

B. Clock to Q Delay t_{cq}

It is the time it takes for the output (Q) to reflect a change in the input (D) after the clock edge.

Parameter	Time (in ps)
t_{pcq0} (delay for fall of Q)	122.23
t_{pcq1} (delay for rise of Q)	64.19
t_{cq} (average of the two)	93.22

```
tpcq0                =  1.222396e-10 targ=  4.517224e-08 trig=  4.505000e-08
tpcq1                =  6.419710e-11 targ=  8.511420e-08 trig=  8.505000e-08
tcq                  =  9.32183e-11
```

Fig. 13: Relevant NGSPICE Outputs for t_{cq} measurement

C. Set Up Time t_{su}

It is the minimum time the input data (D) must be stable before the clock edge. For TSPC D Flip Flop, it is simply the delay of the 1^{st} stage [2] i.e. delay between nodes D and X.

Parameter	Time (in ps)
t_{su0} (delay for fall of X)	72.69
t_{su1} (delay for rise of X)	66.69
t_{su} (average of the two)	69.69

```
tsu1                 =  6.669900e-11 targ=  3.511670e-08 trig=  3.505000e-08
tsu0                 =  7.268943e-11 targ=  7.012269e-08 trig=  7.005000e-08
tsu                  =  6.96942e-11
```

Fig. 14: Relevant NGSPICE Outputs for t_{su} measurement

D. Hold Time t_{hold}

It is the minimum time the input data (D) must remain stable after the clock edge. For TSPC D Flip Flop, it is simply the delay of the 2^{nd} stage [2] i.e. delay between nodes X and Y when clock goes from 0 to 1 i.e. at the positive edge. We get t_{hold} = 39.03 ps.

```
thold                =  3.903399e-11 targ=  4.508903e-08 trig=  4.505000e-08
```

Fig. 15: Relevant NGSPICE Outputs for t_{hold} measurement

VI. Stick Diagrams

- Red Line - Polysilicon
- Green Line - N diffusion layer
- Yellow Line - P diffusion layer
- Black dot - Connection
- Blue line - Metal 1
- Purple line - Metal2

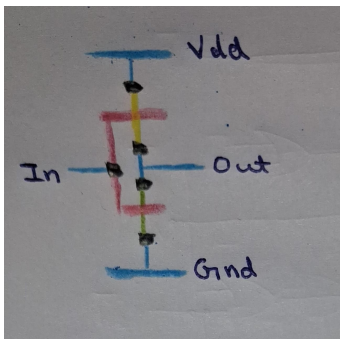


Fig. 16: Inverter

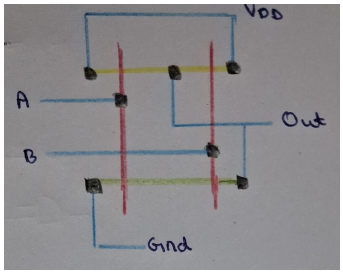


Fig. 17: NAND Gate

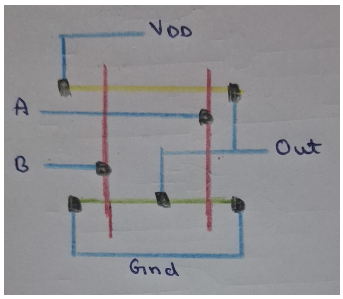


Fig. 18: NOR Gate

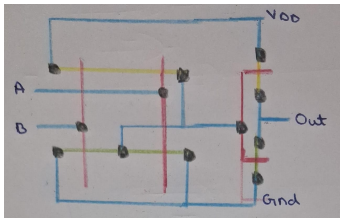


Fig. 19: OR Gate

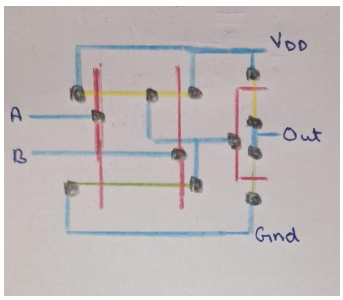


Fig. 20: AND Gate

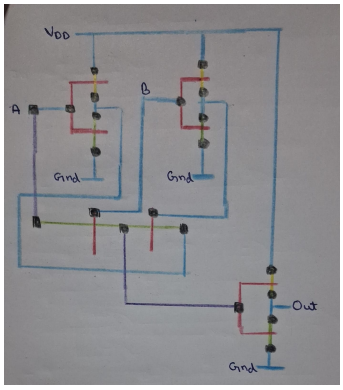


Fig. 21: XOR Gate

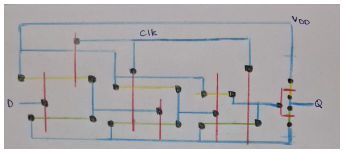


Fig. 22: D Flip Flop

VII. MAGIC Layouts

A. Inverter

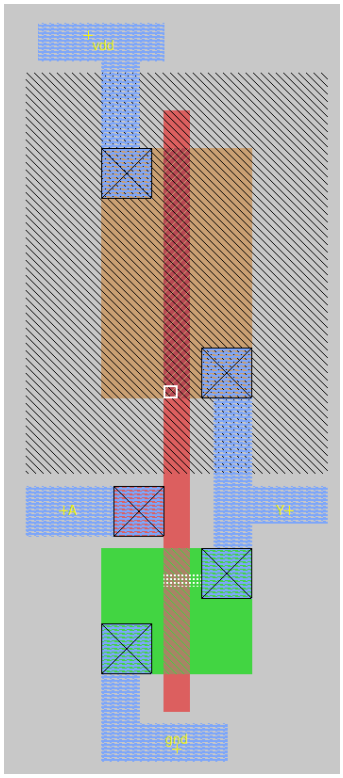


Fig. 23: MAGIC Layout of Inverter

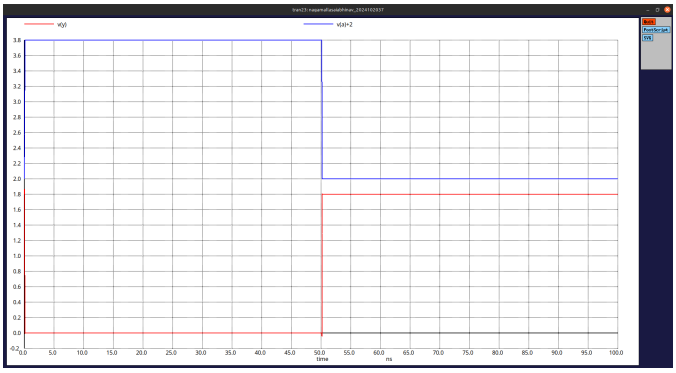


Fig. 24: Inverter Plot : Post-Layout

B. NAND Gate

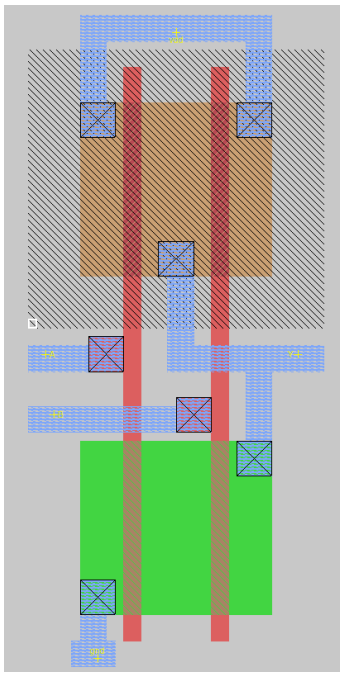


Fig. 25: MAGIC Layout of NAND Gate

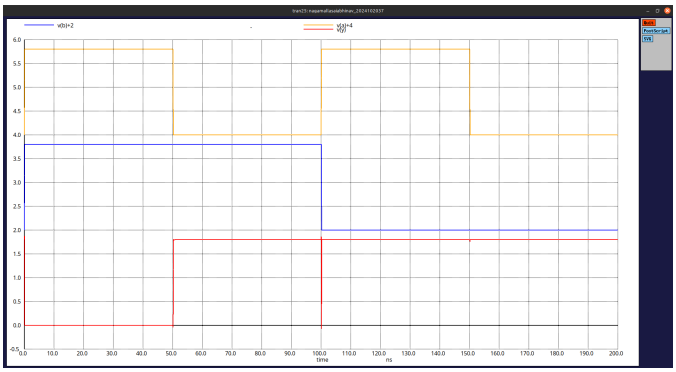


Fig. 26: 2 input NAND Plot : Post-Layout

C. NOR Gate

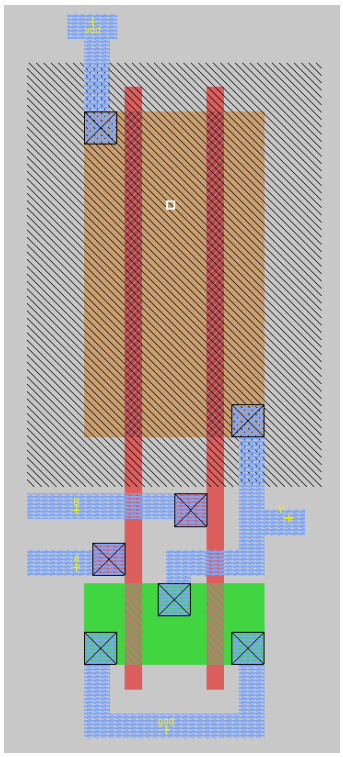


Fig. 27: MAGIC Layout of NOR Gate

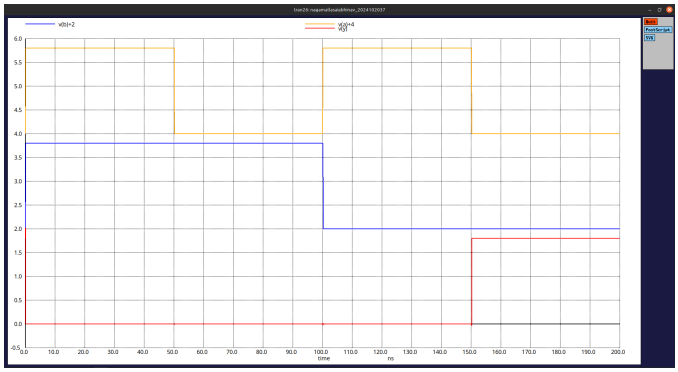


Fig. 28: 2 input NOR Plot : Post Layout

D. AND Gate

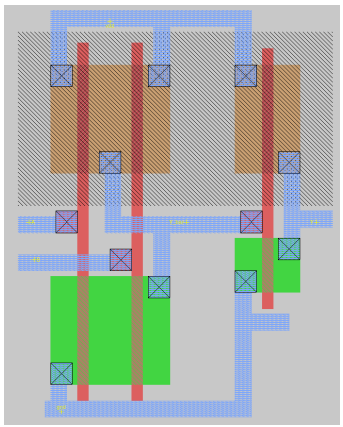


Fig. 29: MAGIC Layout of AND Gate

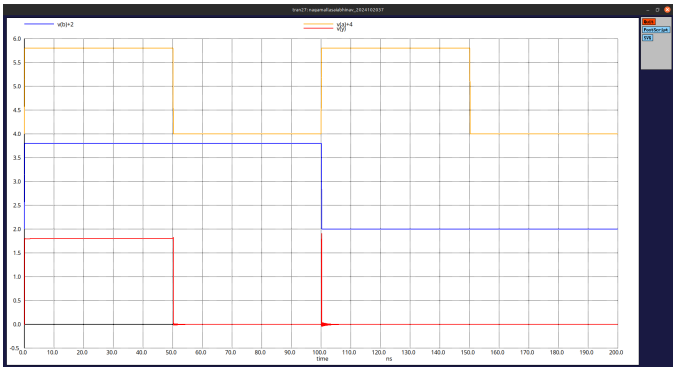


Fig. 30: 2 input AND Plot : Post-Layout

E. OR Gate

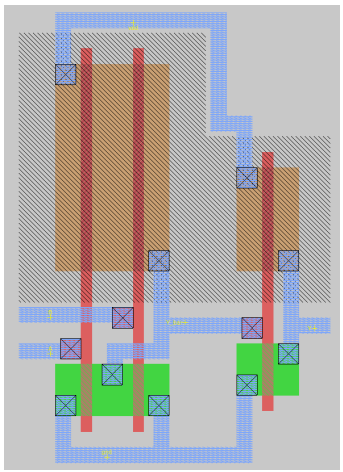


Fig. 31: MAGIC Layout of OR Gate

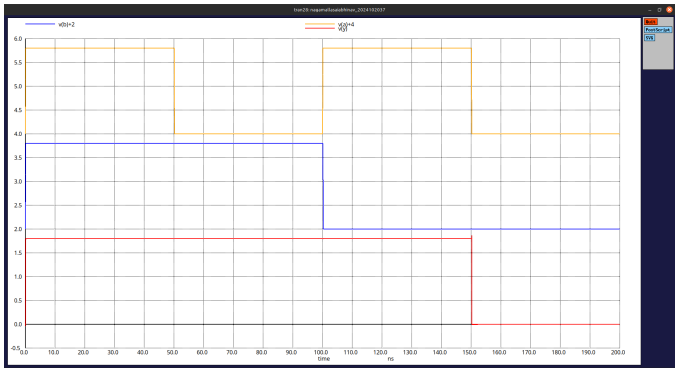


Fig. 32: 2 input OR Plot : Post-Layout

F. XOR Gate

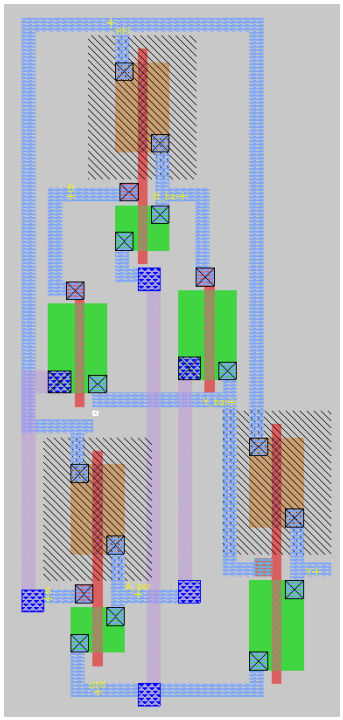


Fig. 33: MAGIC Layout of XOR Gate

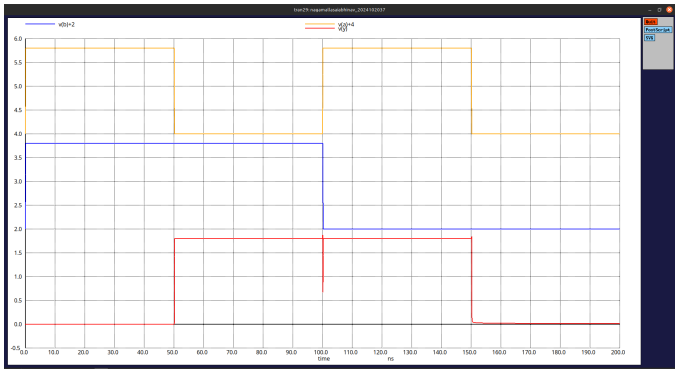


Fig. 34: 2 input XOR Plot : Post-Layout

G. D Flip Flop

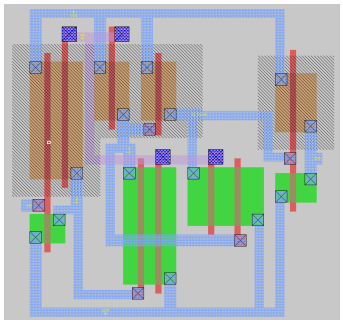


Fig. 35: MAGIC Layout of D Flip Flop

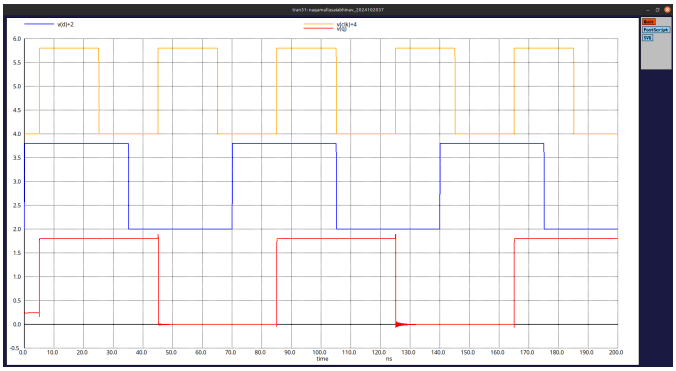


Fig. 36: TSPC D Flip Flop Plot : Post-Layout

H. Clock to Q Delay t_{cq} of the Flip Flop

Parameter	Time (in ps)
t_{pcq0} (delay for fall of Q)	85.11
t_{pcq1} (delay for rise of Q)	40.46
t_{cq} (average of the two)	62.78

I. Set Up Time t_{su} of the Flip Flop

Parameter	Time (in ps)
t_{su0} (delay for fall of X)	52.61
t_{su1} (delay for rise of X)	44.90
t_{su} (average of the two)	48.79

J. Hold Time t_{hold} of the Flip Flop

We get t_{hold} = 30.17 ps.

```
tpcq0           =  8.511517e-11 targ=  4.513512e-08 trig=  4.505000e-08
tpcq1           =  4.046082e-11 targ=  8.509046e-08 trig=  8.505000e-08
tcq             =  6.27880e-11
tsu1            =  4.490465e-11 targ=  3.519490e-08 trig=  3.515000e-08
tsu0            =  5.260862e-11 targ=  7.010261e-08 trig=  7.005000e-08
tsu             =  4.87566e-11
thold           =  3.017675e-11 targ=  4.508018e-08 trig=  4.505000e-08
```

Fig. 37: Relacant NGSPICE Outputs for D Flip Flop Characterization

K. PG Block

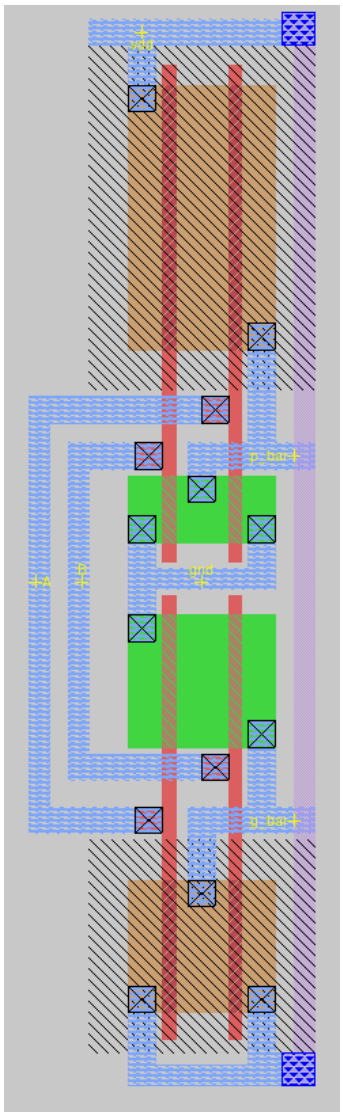


Fig. 38: MAGIC Layout of PG Block

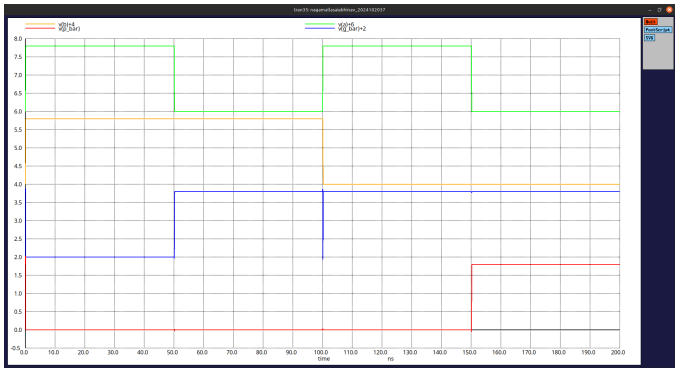


Fig. 39: PG Block Plot : Post-Layout

L. Carry Block

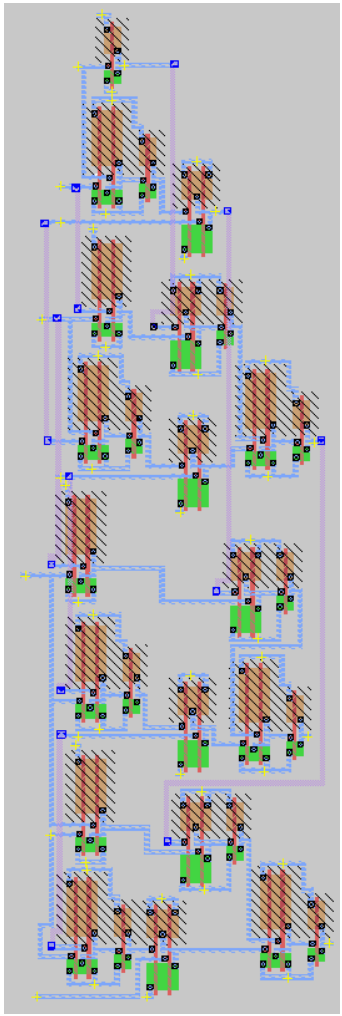


Fig. 40: MAGIC Layout of Carry Block

The following are the outputs for the DC inputs of $\overline{p_i}$ and $\overline{g_i}$.

```
Va g0_bar gnd 0
Vb g1_bar gnd 1.8
Vc p1_bar gnd 1.8
Vd p2_bar gnd 0
Ve g2_bar gnd 1.8
Vf p3_bar gnd 0
Vg g3_bar gnd 1.8
Vh p4_bar gnd 0
Vi g4_bar gnd 1.8
```

Fig. 41: Input combination

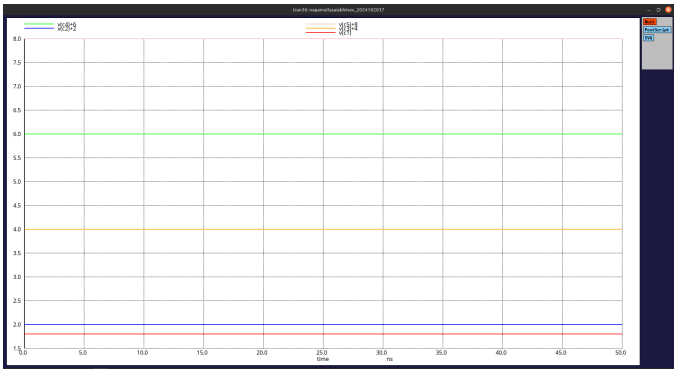


Fig. 42: Carry Block Plot : Post-Layout

M. Sum Block

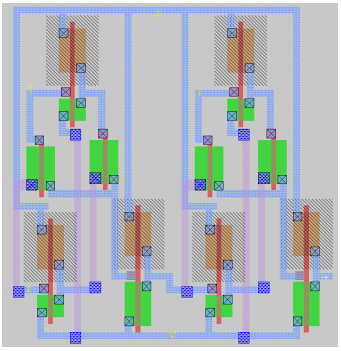


Fig. 43: MAGIC Layout of Sum Block

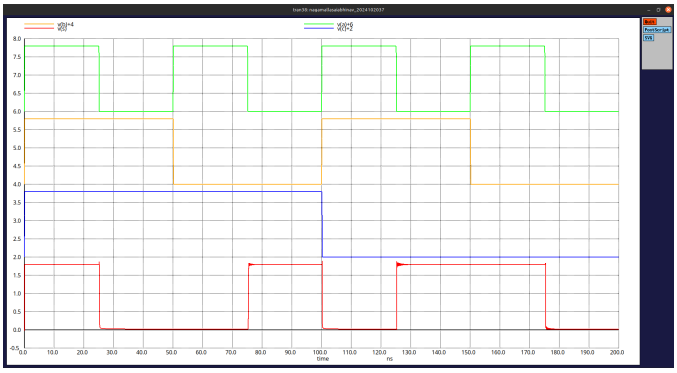


Fig. 44: Sum Block Plot : Post-Layout

We see that both the pre-layout (NGSPICE) and post-layout (MAGIC extracted file) plots match for all the gates and blocks. We also see the decrease in t_{su}, t_{cq} and t_{hold} of the Flip Flop post layout. This is due to the diffusion sharing implemented in the layout, which reduces capacitance compared to the schematic topology simulated in NGSPICE.

VIII. NGSPICE Circuit (Pre Layout)

For the input 10110 and 01101, the following is the NGSPICE output waveforms. All the inputs are given as DC values. They are loaded into the combinational circuit on the first positive edge and are shown at the output on second

positive edge. The output of carry = 1 and sum = 00011 does indeed match the theoretical sum.

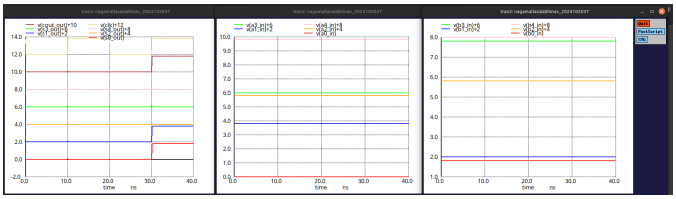


Fig. 45: Pre Layout Plot

Worst case delay is obtained when the longest path (critical path) is taken from inputs to the outputs. We are trying to get the worst case delay of the combinational block here. There are 2 critical paths i.e. from the input bits a_0, b_0 to the output c_5 and from the input bits a_0, b_0 to the output s_4. Note that all these are the nodes of the combi part. One of the inputs which tests the former path is $a = 01111$ and $b = 10111$. The latter part can be tested by inputs like $a = 11011$ and $b = 10110$.

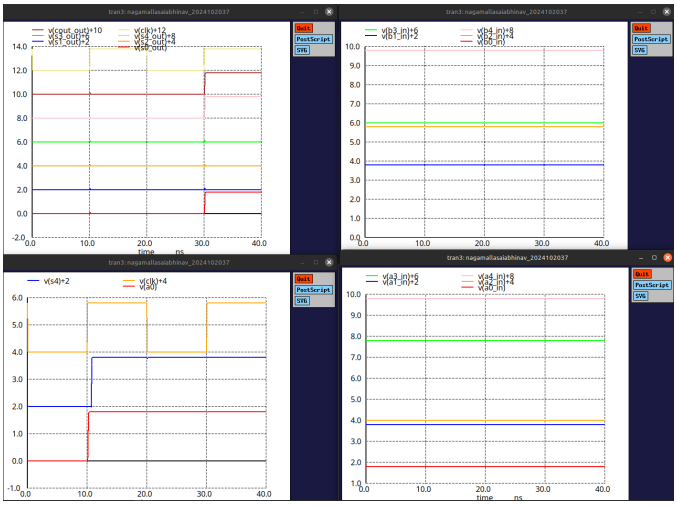


Fig. 46: Testing Path a_0 to s_4

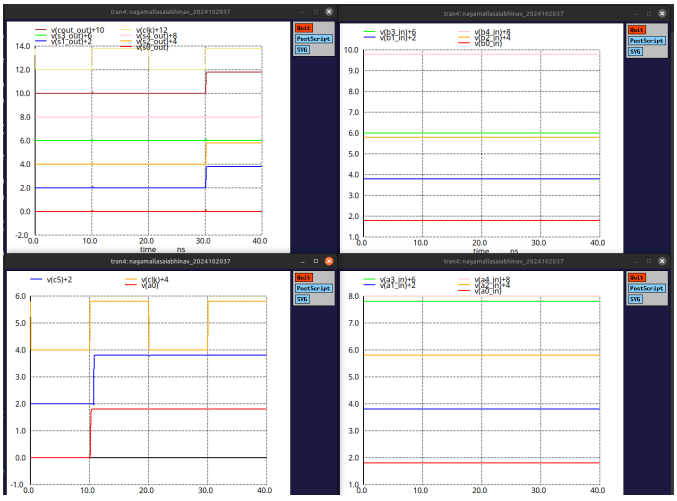


Fig. 47: Testing path a_0 to c_5

Path	Delay (in ps)
a_0 to c_5	585.23
a_0 to s_4	592.53

On measuring delays, we get that the latter is slightly slower than the former one. For the best case delay i.e. the contamination for the D flip flop (to meet hold time constraint), the inputs $a = 00001$ and $b = 00000$ will perfectly work and the minimum time is from the inputs to the output sum s_0. The best case delay is 99.27 ps.

On performing STA, we see that $t_{hold} \leq t_{cq} + t_{logicmin}$ which is comfortably satisfied here, indicating that there is not hold time violation anywhere. To get the maximum clock speed i.e. minimum T_{clk} for operation, the set up time constraint should be met which is $T_{clk} \geq t_{sumax} + t_{logicmax} + t_{cqmax}$. For this to be minimum, just taking the equality will give $T_{clk}(min) = 787.45ps$ giving us $f_{max} = 1/T_{min} \approx 1.27GHz$. At this clock frequency, we can see that the circuit works for all the cases (it is supposed to because this is calculated from taking the worst case scenario). We can test it with the previous critical path inputs(the worst case) and verify.

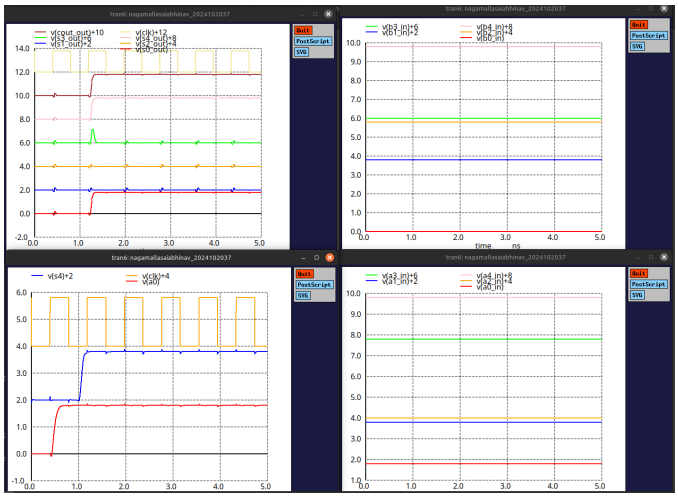


Fig. 48: Verifying operation at f_{max}

So, it works! Therefore, this analysis will give the theoretical bound on the f_{clk} which is equal to 1.27 GHz.

The critical path can be found out by finding the longest path which is of 5 gates. Then, we back track along these gates to get the required input.

IX. MAGIC Layout Plan

The MAGIC Layout floor-plans is given below. The final circuit has $(20\lambda/10\lambda)$ sized inverter at the outputs as loads.

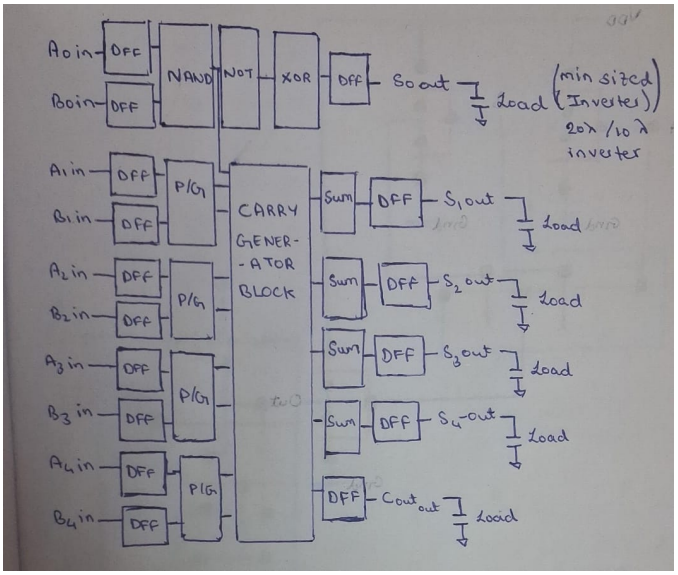


Fig. 49: MAGIC Layout Floor Plan

X. Final MAGIC Layout

The following is the final MAGIC Layout with load inverters at the output nodes.

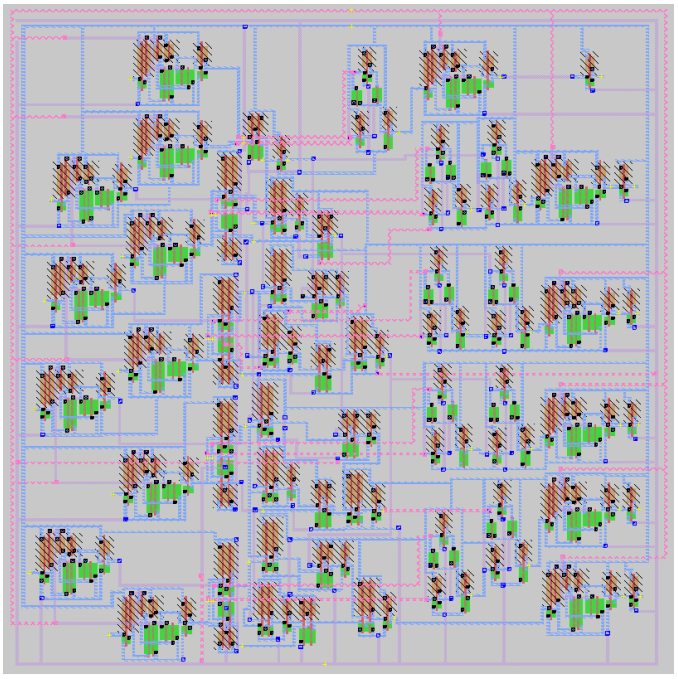


Fig. 50: Full Circuit MAGIC Layout

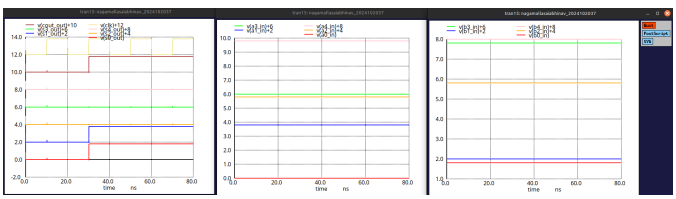


Fig. 51: Post Layout Plot

We can see that the output plot for the same inputs as given in pre layout, we get the same output indicating the functionality of the post layout circuit.

Vertical pitch is the vertical distance between the two consecutive power rails. Horizontal pitch is the horizontal distance between the two consecutive power rails. The layout I made is a square cell which is surrounded by the power rails. So, these pitches will be just the dimensions of this square cell. It has 386 Transistors.

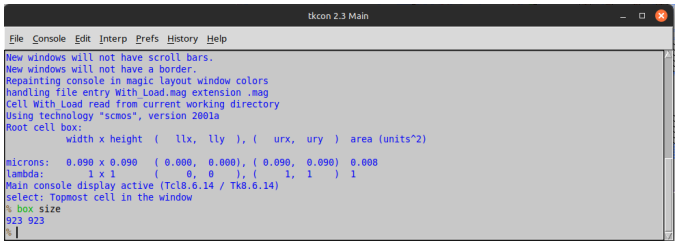


Fig. 52: MAGIC Console displaying the circuit dimensions

Pitch	Value (in μm)
Horizontal	83.07
Vertical	83.07

XI. Post Layout Simulations and Delays

For the same best and worst case delay cases as discussed in the pre-layout section, the following are the delay values:

Path	Delay (in ps)
a_0 to c_5 (worst case 1)	433.57
a_0 to s_4(worst case 2)	412.98
a_0 to s_0 (best case)	65.1

So, we see that the delays are reduced in post layout because of the shared diffusion layers in the layout which will decrease capacitance. Also, the worst case path did change in the post layout analysis.

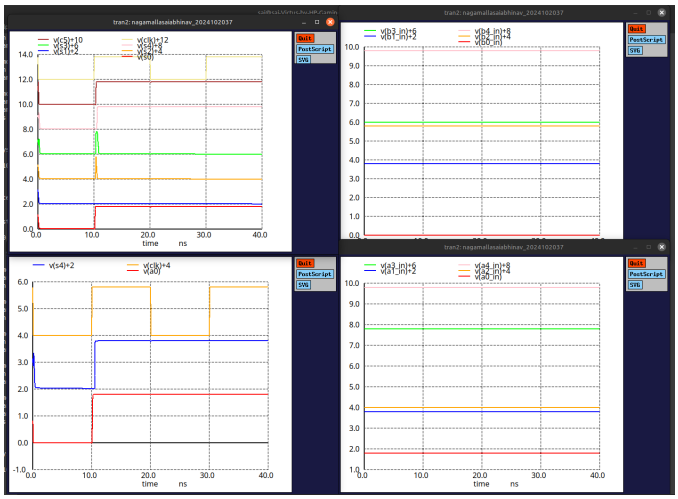


Fig. 53: Testing Path a_0 to s_4

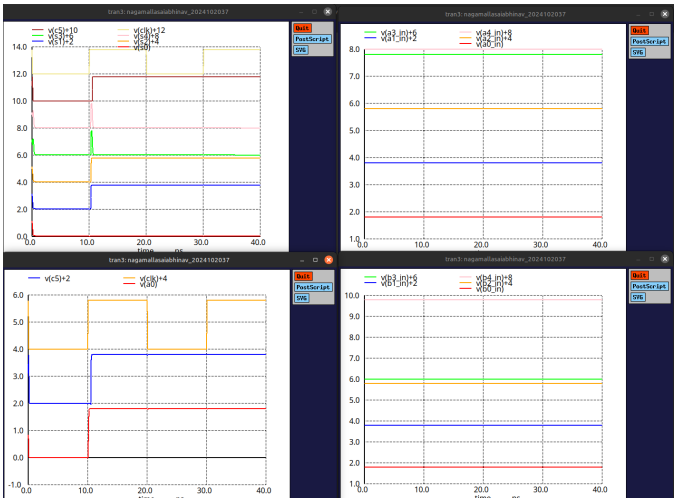


Fig. 54: Testing Path a_0 to c_5

So, doing STA again, we get $t_{cq}(min)$ + $t_{logic}(min) \geq t_{hold}$ satisfying here, indicating that there is no hold time violation.

To find the minimum clock frequency at which the circuit operates, we can use set up time constraint which gives $T_{clk} \geq t_{su} + t_{cq} + t_{logic}$. To find the minimum clock time period, taking equality, we get $T_{clk}(min) = t_{su}(max) + t_{cq}(max) + t_{logic}(max)$. This will give us $T_{clk} = 571.29ps$, giving us $f_{max} = 1/T_{min} \approx 1.75GHz$. At this clock frequency, we can see that the circuit works for all the cases (it is supposed to because this is calculated from taking the worst case scenario). We can test it with the previous critical path inputs(the worst case) and verify.

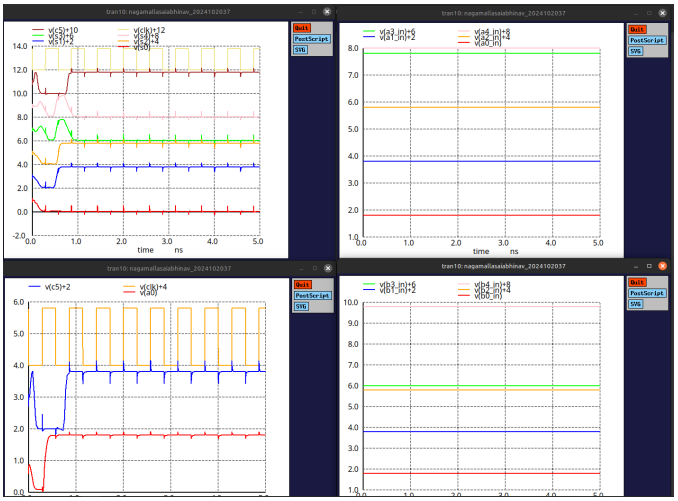


Fig. 55: Verifying operation at f_{max}

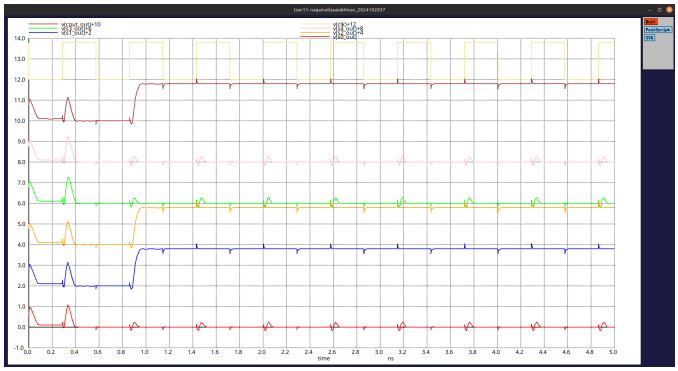


Fig. 56: Verifying operation at f_{max}

So, it works! Therefore, this analysis will give the theoretical bound on the f_{clk} which is equal to 1.75 GHz.

Parameter	Pre-Layout	Post-Layout
t_{su}	69.69 ps	48.79 ps
t_{cq}	93.22 ps	62.78 ps
t_{hold}	39.09 ps	30.17 ps
$T_{clk}(min)$	787.45 ps	571.29 ps
$f_{clk}(max)$	1.27 GHz	1.75 GHz

TABLE I: Comparision of Pre and Post Layout

XII. Verilog HDL

The following circuit is implemented on Verilog HDL. The following is the timing diagram plot.

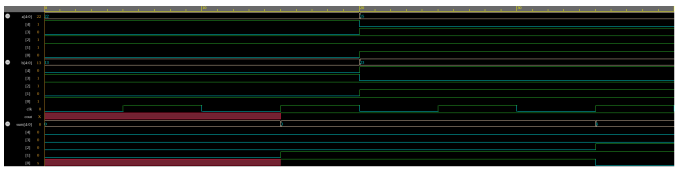


Fig. 57: Verilog Timing Diagram Plot

XIII. FPGA Implementation

Now we implement the HDL on actual hardware using FPGA. The inputs are given from the slide switches of the FPGA (the first 5 are input bits of A and the second 5 are input bits of B). The rightmost 5 LEDs are the sum bits and the 6^{th} LED from the right in Carry. The below image is for A = 01111(15) and B = 10111(23). The sum we expect is 00110(6) and the carry is 1 (overflow bit) which is what we see.

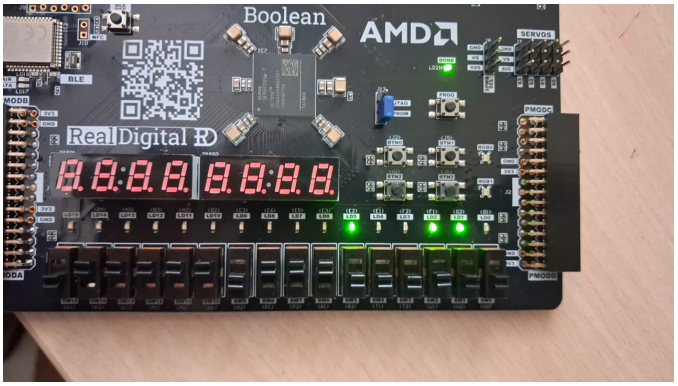
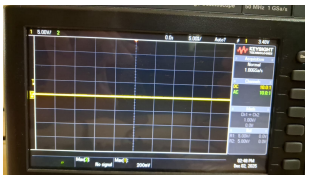


Fig. 58: FPGA output

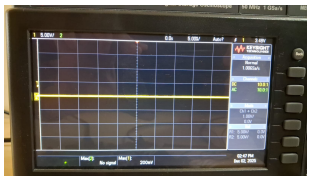
Now, using the PMOD pins, the output bits have been plotted on an Oscilloscope. Again, using the same inputs of A = 01111(15) and B = 10111(23), the following outputs are seen with $V_{high} = 3.3V$ and $V_{low} = 0V$. Due to probe noise, there will be some difference from these expected values.



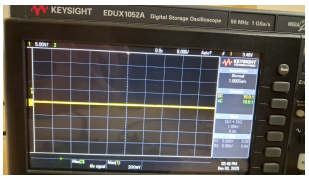
(a) S0

(b) S1

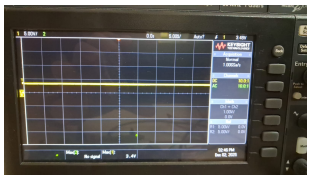
(c) S2



(d) S3



(e) S4



(f) Carry

Fig. 59: Oscilloscope Outputs

ACKNOWLEDGMENT

I express my sincere gratitude to Professor Abhishek Srivastava for giving me the opportunity to work on this project. His lectures have been a constant source of motivation and greatly encouraged me to explore the circuit design in greater depth.

I would also like to thank the Teaching Assistants for their invaluable guidance and support throughout the project. I am equally grateful to my peers for their insightful discussions and continual encouragement.

REFERENCES

[1] Jia Miao and Shuguo Li, "A Novel Implementation of 4-bit Carry Look-ahead Adder" in *2017 International Conference on Electron Devices and Solid-State Circuits (EDSSC)*, 2017.

[2] Jan M.Rabaey, Anantha Chandrakasan, Borivoje Nikolic, "Digital Integrated Circuits" Textbook

[3] Morris Mano, Michael D. Ciletti, " Digital Design" Textbook

[4] M. Hasan, P. Biswas, M. S. Alam, H. U. Zaman, M. Hossain and S. Islam, "High Speed and Ultra Low Power Design of Carry-Out Bit of 4-Bit Carry Look-Ahead Adder," in *Proc. 2019 10th Int. Conf. Comput., Commun. Netw. Technol. (ICCCNT)*, Kanpur, India, 2019.