Implementation of 4-bit Carry Look Ahead Adder

Harsh Kapoor
Electronics and Communication Engineering
International Institute of Information Technology
Hyderabad, India
harsh.kapoor@research.iiit.ac.in
2023112004

***Abstract*—This paper serves as a submission to the course project of VLSI Design, a course for second year electronics and communication undergraduates. This provides a summary and overview of the work done by me to implement a 4-bit Carry Look Ahead Adder, and the various simulations and layouts done by me to achieve it. It also should provide an overview of the thought process by which I came to the design choices in this project.**

I. Introduction

In modern digital systems, the efficient execution of arithmetic operations plays a critical role in enhancing computational speed and performance. Among these operations, addition is one of the most fundamental. Adders are essential components in various hardware units such as arithmetic logic units (ALUs), processors, and digital signal processors (DSPs). Over the years, several adder architectures have been developed to improve speed, power efficiency, and area utilization. Among these, the Carry Look-Ahead Adder (CLA) has emerged as a vital design for achieving faster addition by reducing the delay associated with carry propagation.

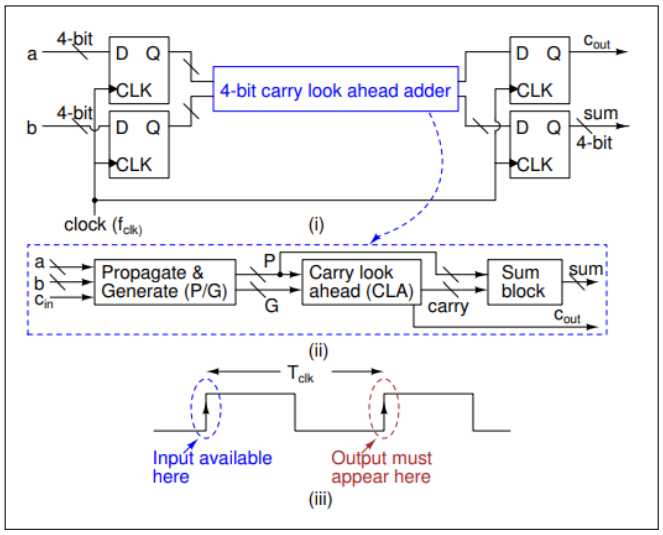
The CLA adder is a highly optimized version of the basic ripple carry adder (RCA), which suffers from significant delay due to the sequential nature of carry propagation. The CLA, on the other hand, employs a parallel processing approach to generate the carry signals, allowing for much faster addition, especially in large-bit-width systems. This improvement in speed is achieved by predicting carry bits ahead of time, rather than waiting for them to ripple through each bit position.

In an RCA, the carry-out of each full adder is connected to the carry-in of the next higher-order bit adder. While simple in design, the RCA suffers from a significant speed disadvantage because the carry must ripple through each bit, causing a delay proportional to the number of bits being added. For large numbers, this ripple delay becomes a bottleneck.

To overcome this limitation, the Carry Look-Ahead Adder (CLA) was developed. The CLA uses a more sophisticated carry generation mechanism based on the concepts of carry generate (G) and carry propagate (P) functions. These functions are used to determine whether a carry will be generated or propagated through each bit position without waiting for the carry to propagate sequentially. By computing these signals in parallel, the CLA significantly reduces the overall delay associated with carry propagation.

II. High Level Overview and Adder Structure

We are asked to design a 4-bit carry look ahead (CLA) adder as shown in Fig. below. Different modules of the CLA-adder are shown in the other Fig.. Each output sum bit needs to drive an inverter of size Wp/Wn = $20\lambda/10\lambda$, where λ = 0.09μm. Consider that input bits are available before the rising edge of the clock and the output should be computed and present at the next rising edge of the clock. We can choose any logic style (static, dynamic, mix) to implement the circuit.



If the numbers to be added are $a_3a_2a_1a_0$ 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_1 = 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:

$sum_i = p_i \oplus c_i$

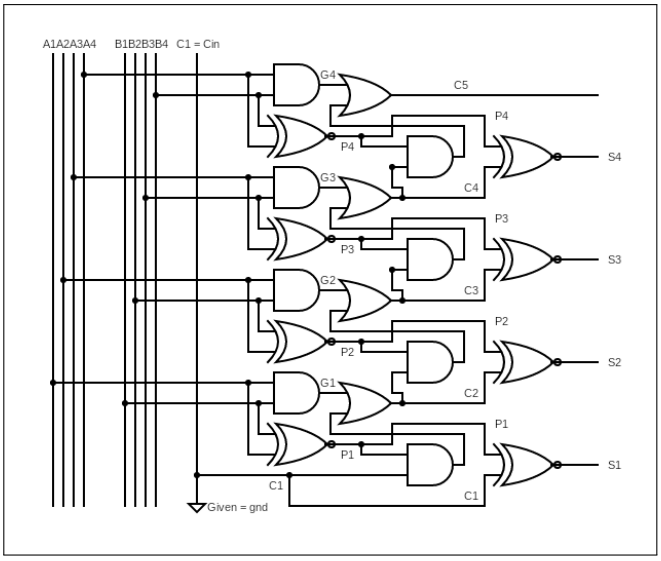
From this, we can write the expanded form of c_i as follows:

- $c_1 = c_{in} = 0$ .....{given}
- $c_2 = g_1 + p_1.c_1 = g_1$
- $c_3 = g_2 + p_2.c_2 = g_2 + p_2.g_1$
- $c_4 = g_3 + p_3.c_3 = g_3 + p_3.(g_2 + p_2.g_1)$
- $c_5 = g_4 + p_4.c_4 = g_4 + p_4.(g_3 + p_3.(g_2 + p_2.g_1))$

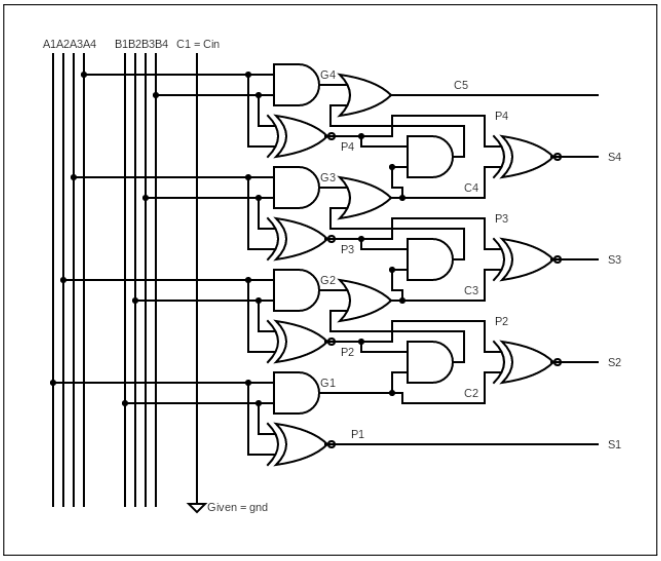
And from this we can conclude that the final carry, c_out will be c_5, and c_5 only depends on the variables $a_1a_2a_3a_4$ and $b_1b_2b_3b_4$. Let's take the following example, with 4th subscript being the MSB and 1st being the LSB.

X_i	X_5	X_4	X_3	X_2	X_1
A_i	-	1	0	1	1
B_i	-	1	1	1	0
G_i	-	1	0	1	0
P_i	-	0	1	0	1
C_i	1	1	1	0	0
S_i	-	1	0	0	1

From this table we can say that according to the above model, the Sum and Carry bits give the correct outputs for this example, and hence the model works. A Basic circuit diagram of this circuit can be built as follows -



But upon closer inspection one could find that the given circuit could be modified by reducing the $C_1 = gnd$ as per the given condition. Then the following circuit can be used -



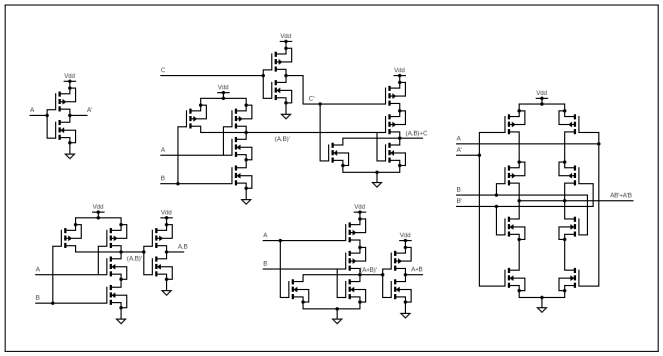
The following changes are made -

- since $S_1 = P_1 \oplus C_1$ and $0 \oplus P_1 = P_1$, the xor gate is removed and $S_1 = P_1$
- since and gate to $P_1.C_1$ would just yield gnd and $G1 + 0$ would wield G_1, the OR and the AND gate were removed.

Please note that these modifications were done according to the given parameter that $C_in = C_1 = 0/gnd$. The author is well aware that due to this the circuit will not be able to be cascaded but this will reduce the power usage of the circuit and also reduce the chip area utilized. This circuit is however not used in this paper and implementation, such that cascading of the circuits is possible. The following link contains all the circuit diagrams used in this paper, uploaded on onedrive.

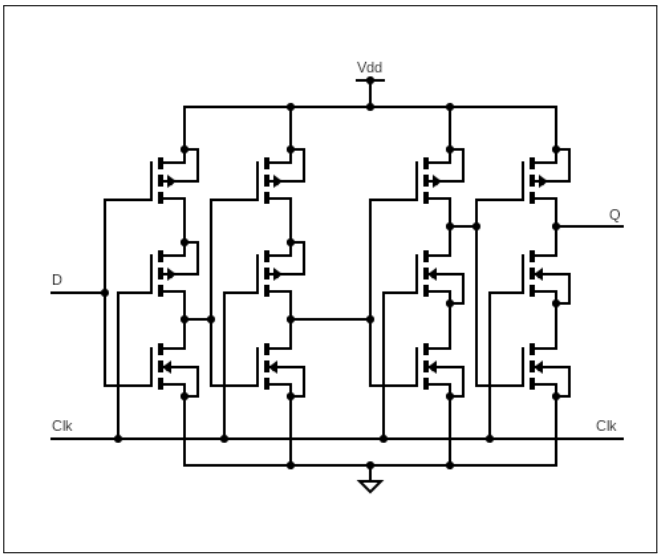
III. Design Details for Circuit Elements

The main circuit elements used in the adder structure are - XOR gate, AND gate and OR gate. Other elements maybe used are - NOT gates as repeaters to restore logic levels, delay managements and also for driving the next stages. Given was the constrainsts - Each output sum bit needs to drive an inverter of size W_p/W_n = 20λ/10λ, where λ= 0.09μm. The decided topology for the circuit is CMOS, due to its ease and simplicity and no loss of logic levels. Due to CMOS being used, AND gate and OR gate are not directly implementable but need NAND and NOR gate connected to a CMOS Inverter. A common pattern used here is that of $(A.B) + C$, which if implemented as $\overline{\overline{A.B}.\overline{C}} = \overline{\overline{A.B}} + \overline{\overline{C}} = A.B + C$ will save the additional inverter being used. Following are images of the common structures being used i.e. $A.B$, $A + B$, $\overline{A}$, $A \oplus B$ and $(A.B) + C$.



The main idea is to have a ratio-less configuration, so that our circuit logic still works and our main effort is in driving the next stage.

Next element is the Flip-flops before and after the circuit. I will be using the TSPC topology as it offers a 0 hold time, which just simplifies alot of things and offers High-speed-switching logic. Moreover, the fact that no inverters are used either in the clock or in the inputs just adds another level of ease in managing the timinigs of the circuit. Although, each flip-flop uses 12 transistors the cost is justified due to the benefits mentioned. A transistor level circuit is as follows -



Next step will be simulate individual elements in NG-Spice and then create a complete circuit.

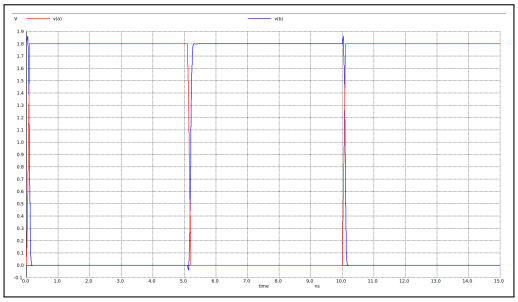
IV. Individual Spice Simulations

- ***INVERTER (NOT GATE)***
  Following is the spice code for the not gate. This sub-circuit inv will be called whenever we want to use the inverter logic in the circuit.

NOT GATE

```
.subckt inv y x vdd gnd Wn={Wn} Wp={Wp}
M1 y x gnd gnd CMOSN W={Wn} L={2*LAMBDA}
+ AS={5*Wn*LAMBDA} PS={10*LAMBDA+2*Wn}
+ AD={5*Wn*LAMBDA} PD={10*LAMBDA+2*Wn}

M2 y x vdd vdd CMOSP W={Wp} L={2*LAMBDA}
+ AS={5*Wp*LAMBDA} PS={10*LAMBDA+2*Wp}
+ AD={5*Wp*LAMBDA} PD={10*LAMBDA+2*Wp}
.ends inv
```

These are the waveforms for the code, note that 'a' is the input to the inverter and 'b' is the output driving the given inverter in problem statement.



Following are the timing analysis for the Not gate -

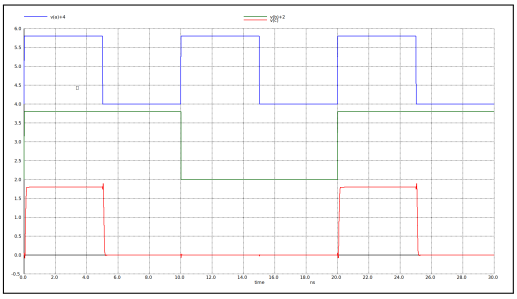
```
rise_time_b          =  7.792620e-11
fall_time_b          =  5.714753e-11
tpd                  =  6.75369e-11
delay_a_to_b         =  3.992033e-11
```

- ***AND GATE***
  Following is the spice code for the and gate. This sub-circuit and_with_inv will be called whenever we want to use the and logic in the circuit. Sizing is done to compare it with CMOS inverter (i.e. $W_{n_{Nand}} = 2 \times W_{n_{inv}}$ and similarly $W_{p_{Nand}} = W_{p_{inv}}$).

AND GATE

```
.subckt and_with_inv y x1 x2 vdd gnd Wn={Wn} Wp={Wp}
* NAND section
M1 n1 x1 vdd vdd CMOSP W={Wp} L={2*LAMBDA}
+ AS={5*Wp*LAMBDA} PS={10*LAMBDA+2*Wp}
+ AD={5*Wp*LAMBDA} PD={10*LAMBDA+2*Wp}

M2 n1 x2 vdd vdd CMOSP W={Wp} L={2*LAMBDA}
+ AS={5*Wp*LAMBDA} PS={10*LAMBDA+2*Wp}
+ AD={5*Wp*LAMBDA} PD={10*LAMBDA+2*Wp}

M3 n1 x1 n2 gnd CMOSN W={2*Wn} L={2*LAMBDA}
+ AS={10*Wn*LAMBDA} PS={10*LAMBDA+4*Wn}
+ AD={10*Wn*LAMBDA} PD={10*LAMBDA+4*Wn}

M4 n2 x2 gnd gnd CMOSN W={2*Wn} L={2*LAMBDA}
+ AS={10*Wn*LAMBDA} PS={10*LAMBDA+4*Wn}
+ AD={10*Wn*LAMBDA} PD={10*LAMBDA+4*Wn}

* Inverter section
M5 y n1 gnd gnd CMOSN W={Wn} L={2*LAMBDA}
+ AS={5*Wn*LAMBDA} PS={10*LAMBDA+2*Wn}
+ AD={5*Wn*LAMBDA} PD={10*LAMBDA+2*Wn}

M6 y n1 vdd vdd CMOSP W={Wp} L={2*LAMBDA}
+ AS={5*Wp*LAMBDA} PS={10*LAMBDA+2*Wp}
+ AD={5*Wp*LAMBDA} PD={10*LAMBDA+2*Wp}
.ends and_with_inv
```

These are the waveforms for the code, note that 'a' and 'b' are the inputs to the AND gate and 'c' is the output driving the given inverter in problem statement. 'a' and 'b' are shifted by 4 and 2 volts respectively for clearer plots.



Following are the timing analysis for the And gate, a to c delay is calculated on the first simultaneously falling edge which occurs at t = 5ns calculated for both equaling vdd/2 in the plot attached above -

```
rise_time_c          =  6.938390e-11
fall_time_c          =  6.220276e-11
tpd                  =  6.57933e-11
delay_a_to_c         =  8.585314e-11
```

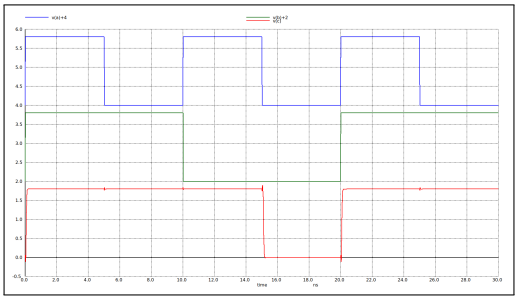
- ***OR GATE***

Following is the spice code for the or gate. This sub-circuit or_with_inv will be called whenever we want to use the or logic in the circuit. Sizing is done to compare it with CMOS inverter (i.e. $W_{p_{Nor}} = 2 \times W_{p_{inv}}$ and similarly $W_{n_{Nor}} = W_{n_{inv}}$).

OR GATE

```
.subckt or_with_inv y x1 x2 vdd gnd Wn={Wn} Wp={Wp}
* NOR section
M1 n1 x1 n2 vdd CMOSP W={2*Wp} L={2*LAMBDA}
+ AS={10*Wp*LAMBDA} PS={10*LAMBDA+4*Wp}
+ AD={10*Wp*LAMBDA} PD={10*LAMBDA+4*Wp}

M2 n2 x2 vdd vdd CMOSP W={2*Wp} L={2*LAMBDA}
+ AS={10*Wp*LAMBDA} PS={10*LAMBDA+4*Wp}
+ AD={10*Wp*LAMBDA} PD={10*LAMBDA+4*Wp}

M3 n1 x1 gnd gnd CMOSN W={Wn} L={2*LAMBDA}
+ AS={5*Wn*LAMBDA} PS={10*LAMBDA+2*Wn}
+ AD={5*Wn*LAMBDA} PD={10*LAMBDA+2*Wn}

M4 n1 x2 gnd gnd CMOSN W={Wn} L={2*LAMBDA}
+ AS={5*Wn*LAMBDA} PS={10*LAMBDA+2*Wn}
+ AD={5*Wn*LAMBDA} PD={10*LAMBDA+2*Wn}

* Inverter section
M5 y n1 gnd gnd CMOSN W={Wn} L={2*LAMBDA}
+ AS={5*Wn*LAMBDA} PS={10*LAMBDA+2*Wn}
+ AD={5*Wn*LAMBDA} PD={10*LAMBDA+2*Wn}

M6 y n1 vdd vdd CMOSP W={Wp} L={2*LAMBDA}
+ AS={5*Wp*LAMBDA} PS={10*LAMBDA+2*Wp}
+ AD={5*Wp*LAMBDA} PD={10*LAMBDA+2*Wp}
.ends or_with_inv
```

These are the waveforms for the code, note that 'a' and 'b' are the inputs to the OR gate and 'c' is the output driving the given inverter in problem statement. 'a' and 'b' are shifted by 4 and 2 volts respectively for clearer plots.



Following are the timing analysis for the Or gate, a to c delay is calculated on the first simultaneously falling edge which occurs at t = 15ns calculated for both equaling vdd/2 in the plot attached above -

```
rise_time_c          =  6.198511e-11
fall_time_c          =  6.401276e-11
tpd                  =  6.29989e-11
delay_a_to_c         =  8.750264e-11
```

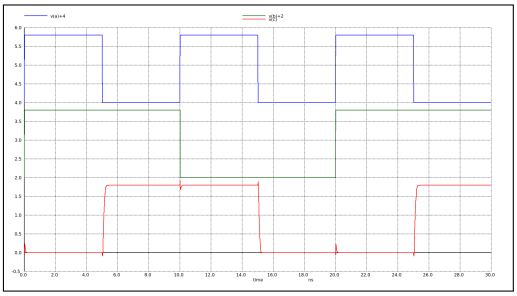
- ***XOR GATE***

Following is the spice code for the xor gate. This sub-circuit xor will be called whenever we want to use the xor logic in the circuit. Sizing is done to compare it with CMOS inverter (i.e. $W_{p_{Xor}} = 2 \times W_{p_{inv}}$ and similarly $W_{n_{Xor}} = 2 \times W_{n_{inv}}$).

xor

```
.subckt xor y x1 x1_bar x2 x2_bar vdd gnd Wn={Wn} Wp={Wp}
M1 n1 x1_bar vdd vdd CMOSP W={2*Wp} L={2*LAMBDA}
+ AS={10*Wp*LAMBDA} PS={10*LAMBDA+4*Wp}
+ AD={10*Wp*LAMBDA} PD={10*LAMBDA+4*Wp}

M2 y x2 n1 vdd CMOSP W={2*Wp} L={2*LAMBDA}
+ AS={10*Wp*LAMBDA} PS={10*LAMBDA+4*Wp}
+ AD={10*Wp*LAMBDA} PD={10*LAMBDA+4*Wp}

M3 n2 x1 vdd vdd CMOSP W={2*Wp} L={2*LAMBDA}
+ AS={10*Wp*LAMBDA} PS={10*LAMBDA+4*Wp}
+ AD={10*Wp*LAMBDA} PD={10*LAMBDA+4*Wp}

M4 y x2_bar n2 vdd CMOSP W={2*Wp} L={2*LAMBDA}
+ AS={10*Wp*LAMBDA} PS={10*LAMBDA+4*Wp}
+ AD={10*Wp*LAMBDA} PD={10*LAMBDA+4*Wp}

M5 y x2_bar n3 gnd CMOSN W={2*Wn} L={2*LAMBDA}
+ AS={10*Wn*LAMBDA} PS={10*LAMBDA+4*Wn}
+ AD={10*Wn*LAMBDA} PD={10*LAMBDA+4*Wn}

M6 n3 x1_bar gnd gnd CMOSN W={2*Wn} L={2*LAMBDA}
+ AS={10*Wn*LAMBDA} PS={10*LAMBDA+4*Wn}
+ AD={10*Wn*LAMBDA} PD={10*LAMBDA+4*Wn}

M7 y x2 n4 gnd CMOSN W={2*Wn} L={2*LAMBDA}
+ AS={10*Wn*LAMBDA} PS={10*LAMBDA+4*Wn}
+ AD={10*Wn*LAMBDA} PD={10*LAMBDA+4*Wn}

M8 n4 x1 gnd gnd CMOSN W={2*Wn} L={2*LAMBDA}
+ AS={10*Wn*LAMBDA} PS={10*LAMBDA+4*Wn}
+ AD={10*Wn*LAMBDA} PD={10*LAMBDA+4*Wn}
.ends xor
```

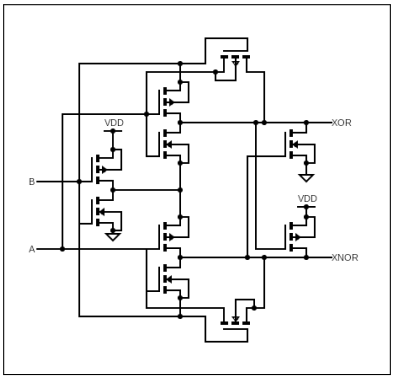
These are the waveforms for the code, note that 'a' and 'b' are the inputs to the XOR gate and 'c' is the output driving the given inverter in problem statement. 'a' and 'b' are shifted by 4 and 2 volts respectively for clearer plots.



Following are the timing analysis for the Xor gate, a to c delay is calculated on the first simultaneously falling edge which occurs at t = 15ns calculated for both equaling vdd/2 in the plot attached above -

```
rise_time_c          =  1.378215e-10
fall_time_c          =  1.091577e-10
tpd                  =  1.23490e-10
delay_a_to_c         =  7.524655e-11
```

It is clear that this implementation of XOR gate is not the most ideal approach as the total number of transistors required (including for inverting the signals) is 12. This however can be prevented by using an alternative logic proposed by Amini Valashani, Mehdi & Mirzakuchaki, in their 2018 paper "Design and analysis of a novel low-power and energy-efficient 18T hybrid full adder", published in Microelectronics Journal. The structure proposed is as follows -



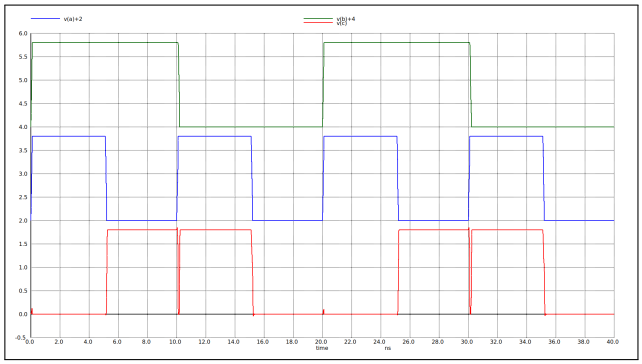
It uses 10 transistors and produces both XOR and XNOR logic. Following is the spice code for the new xor topology.

xor_new

```
.subckt xor_new n2 x1 x2 vdd gnd
M1 n1 x2 vdd vdd CMOSP W={Wp} L={LAMBDA}
+ AS={5*Wp*LAMBDA} PS={10*LAMBDA+2*Wp}
+ AD={5*Wp*LAMBDA} PD={10*LAMBDA+2*Wp}

M2 n1 x2 gnd gnd CMOSN W={Wn} L={LAMBDA}
+ AS={5*Wp*LAMBDA} PS={10*LAMBDA+2*Wp}
+ AD={5*Wp*LAMBDA} PD={10*LAMBDA+2*Wp}

M3 n2 x1 x2 vdd CMOSP W={Wp} L={LAMBDA}
+ AS={5*Wp*LAMBDA} PS={10*LAMBDA+2*Wp}
+ AD={5*Wp*LAMBDA} PD={10*LAMBDA+2*Wp}

M4 n2 x1 n1 gnd CMOSN W={Wn} L={LAMBDA}
+ AS={5*Wp*LAMBDA} PS={10*LAMBDA+2*Wp}
+ AD={5*Wp*LAMBDA} PD={10*LAMBDA+2*Wp}

M5 n3 x1 n1 vdd CMOSP W={Wp} L={LAMBDA}
+ AS={5*Wp*LAMBDA} PS={10*LAMBDA+2*Wp}
+ AD={5*Wp*LAMBDA} PD={10*LAMBDA+2*Wp}

M6 n3 x1 x2 gnd CMOSN W={Wn} L={LAMBDA}
+ AS={5*Wp*LAMBDA} PS={10*LAMBDA+2*Wp}
+ AD={5*Wp*LAMBDA} PD={10*LAMBDA+2*Wp}

M7 x1 x2 n2 vdd CMOSP W={Wp} L={LAMBDA}
+ AS={5*Wp*LAMBDA} PS={10*LAMBDA+2*Wp}
+ AD={5*Wp*LAMBDA} PD={10*LAMBDA+2*Wp}

M8 n3 x2 x1 gnd CMOSN W={Wn} L={LAMBDA}
+ AS={5*Wp*LAMBDA} PS={10*LAMBDA+2*Wp}
+ AD={5*Wp*LAMBDA} PD={10*LAMBDA+2*Wp}

M9 n3 n2 vdd vdd CMOSP W={Wp} L={LAMBDA}
+ AS={5*Wp*LAMBDA} PS={10*LAMBDA+2*Wp}
+ AD={5*Wp*LAMBDA} PD={10*LAMBDA+2*Wp}

M10 n2 n3 gnd gnd CMOSN W={Wn} L={LAMBDA}
+ AS={5*Wp*LAMBDA} PS={10*LAMBDA+2*Wp}
+ AD={5*Wp*LAMBDA} PD={10*LAMBDA+2*Wp}
```

.ends xor_new

These are the waveforms for the code, note that 'a' and 'b' are the inputs to the new XOR topology and 'c' is the output driving the given inverter in problem statement. 'a' and 'b' are shifted by 4 and 2 volts respectively for clearer plots.



Following are the timing analysis for the new Xor topology, a to c delay is calculated on the first falling edge of a and rising edge of c which occurs at t = 5ns calculated for both equaling vdd/2 in the plot attached above -

```
rise_time_c          =  5.498745e-11
fall_time_c          =  4.912215e-11
tpd                  =  5.20548e-11
delay_a_to_c         =  5.490771e-11
```

Hence, we find that this topology works better for our case as it not only reduces the transistor count but also it makes the circuit faster as the rise and fall times are greatly reduced. therefore going forward we will be using this topology.

- ***FLIP-FLOP***

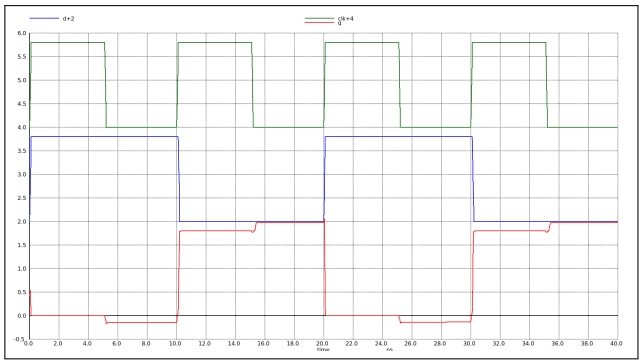
Following is the spice code for the D-FLIPFLIP. Sizing is done to compare it with CMOS inverter (i.e. $W_{p_{master}} = 2 \times W_{p_{inv}}$, $W_{n_{master}} = W_{n_{inv}}$ and similarly $W_{p_{slave}} = W_{p_{inv}}$, $W_{n_{slave}} = 2 \times W_{n_{inv}}$). Here clock signal is taken arbitarily to be 10ns and the time period for the input signal D is 20 ns. These values can be calculated using the SET-UP TIME VIOLATION constraint -

$$T_{Clk} >= t_{Clk\_to\_Q} + t_{ckt\_delay} + t_{setup}$$

This can only be calculated once we have made the entire circuit and have the delay measurement of the entire circuit and the clock to Q time of the individual flip-flop. Another option which was considered was to use a 6-T cell based array for input and output, but was not implemented after discussion with TA's and their advise.

```
flip-flop

.subckt lllatch d clk lout vdd gnd
M1 vdd d n1 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}

M2 n1 clk n2 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}

M3 n2 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}

M4 vdd n2 n3 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}

M5 n3 clk lout 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}

M6 lout n2 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}
.ends lllatch

.subckt hllatch in clk q vdd gnd
M1 vdd in n1 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}

M2 n1 clk n2 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 n2 in 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}

M4 vdd n1 q 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}

M5 q clk n3 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}

M6 n3 n1 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}
.ends hllatch

.subckt dflipflop d clk q vdd gnd
x1 d clk lout vdd gnd lllatch
x2 lout clk q vdd gnd hllatch
.ends dflipflop
```

Here, we have broken down the flip-flop in two sub-circuits, High-Level-Latch (hllatch) and Low-Level-Latch (lllatch). This has been done to increase modularity of the circuit. Following are the waveforms for input (d), output (q) and clk. Here, clk and input are shifted up by 4 and 2 units respectively for better plotting.



The overshoot and undershoot both measure to be about 0.2V and in general should not affect the circuit. The following is the timing analysis of the flipflop. Rise and fall times are measured for the first rise and fall which occurs at 10 and 20 ns respectively and are measured between 0.9Vdd and 0.1Vdd. Clock to Q time is measured for simulataneous rise of clk and q signal at 10 ns, measured at Vdd/2 volt.

```
rise_time_q             =  6.125147e-11
fall_time_q             =  4.104450e-11
tpd                     =  5.11480e-11
delay_clk_to_q          =  8.849728e-11
```

Here we find that the Clk to Q delay is $8.9 \times 10^{-11} s$. For setup time, the value of input is changed (delayed) by various amount till the output varies by almost 10%. This happens at $3.75 \times 10^{-11} s$. The following is the clk to q delay for this.

```
delay_clk_to_q      =  9.044632e-11 targ=  2.014045e-08 trig=  2.005000e-08
```

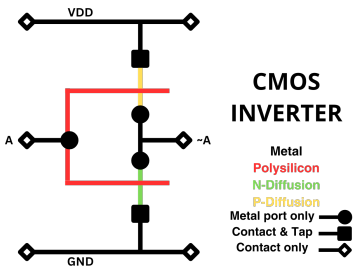
Therefore, our maximum clock time is currently set by the following inequality:

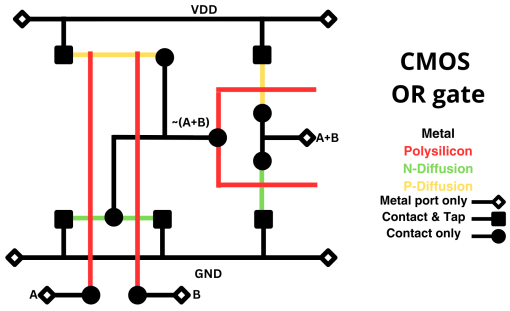
$$T_{clk} >= t_{max\_circuit} + 12.1 \times 10^{-11} s$$

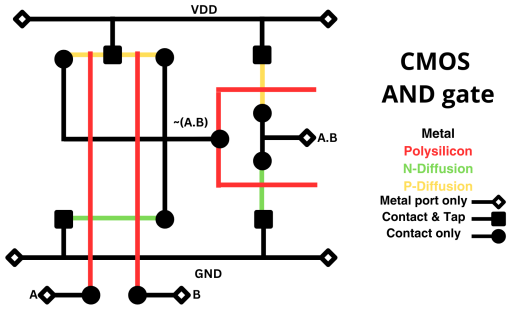
With this all the individual spice simulations are done. The corresponding .sp files can be found at the following onedrive link. It also has a folder for all the images of the waveforms for deeper inspection. The next steps require us to plan and layout the given circuit.

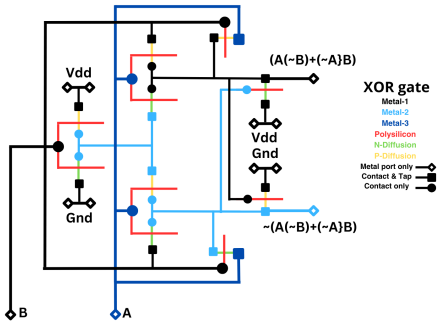
V. Stick Diagrams for Gates

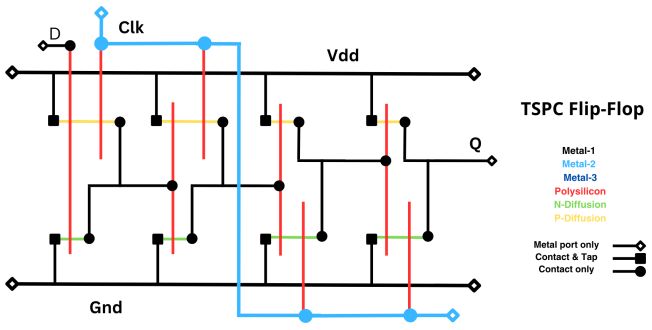
At its core, a stick diagram is a simple, schematic representation of the layout of transistors and their connections in a digital circuit. It employs a minimalist approach, using lines and rectangles to depict transistors and their connections, respectively. Unlike detailed layout diagrams, stick diagrams focus on the topology of the circuit rather than its physical dimensions, making them an invaluable tool for quickly iterating through design concepts and assessing their feasibility. Following are the stick diagrams for the individual gates (NOT gate, OR gate, AND gate, XOR gate and the D-FLIP-FLOP) -









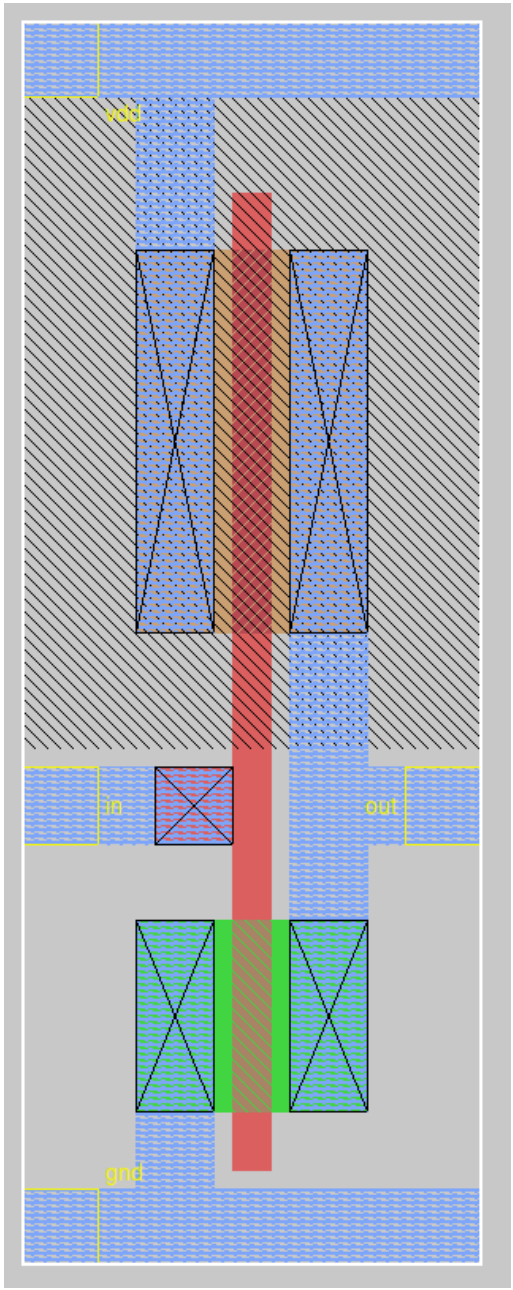


The color scheme and the type of connections are mentioned in each one of the stick diagrams there is an error in the stick diagram of the inverter where the metal port and the contact only connections are swapped and editing it again would have been a troublesome job, apologies in advance. These diagrams were made on CANVA and the following link contains the images in .png format uploaded on onedrive.

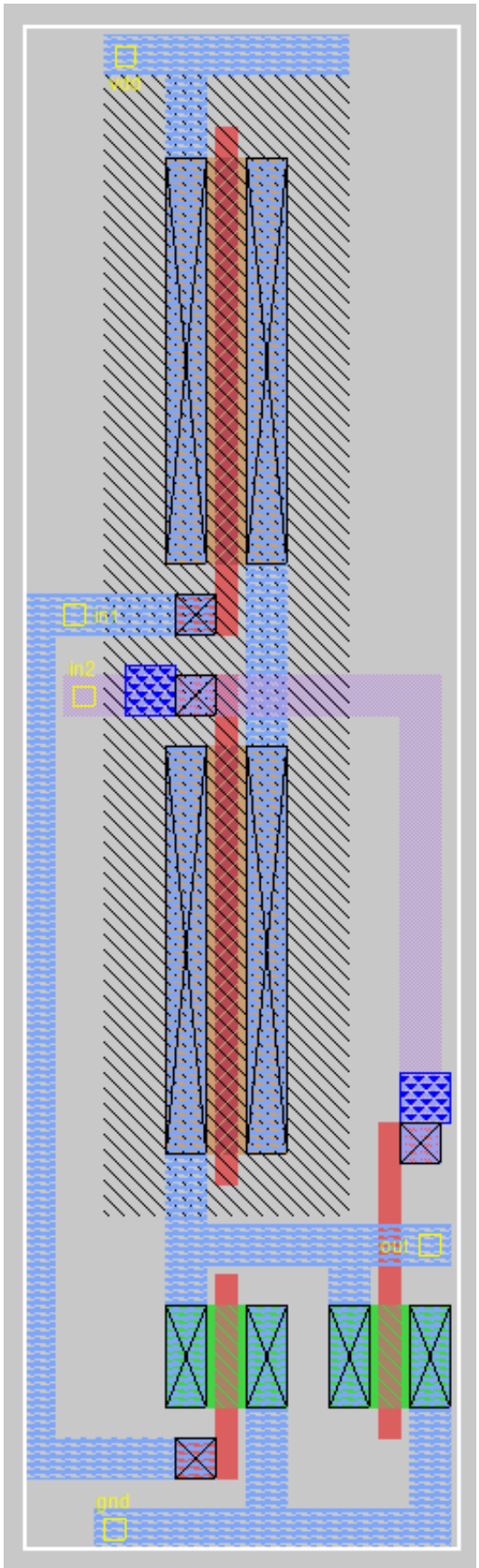
VI. Magic Layout of Individual Gates

The following images are of the individual Layouts done on the software MAGIC. The circuits are NOT Gate, AND Gate, OR Gate, XOR Gate and the D-Flip-Flop. The Images, .mag files, .ext files and the .spice files can be found in this onedrive link.

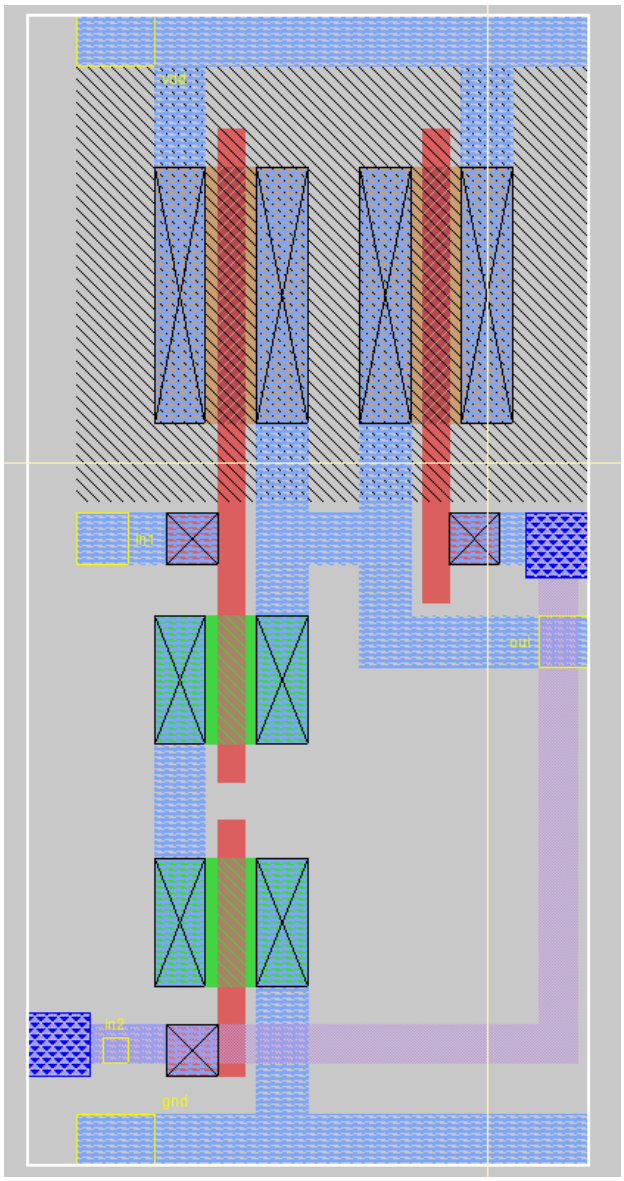
- ***NOT GATE***



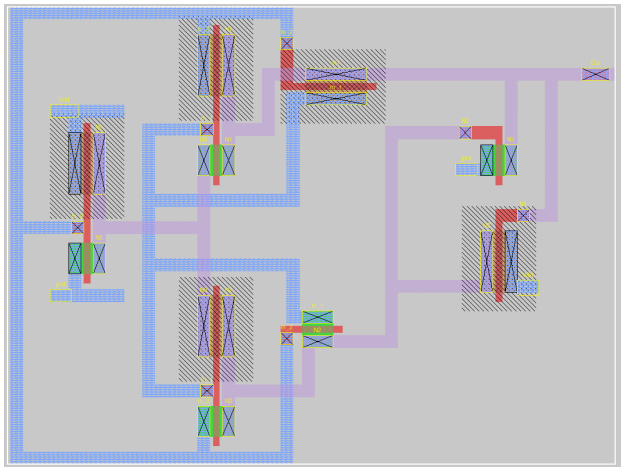
- *NAND GATE*



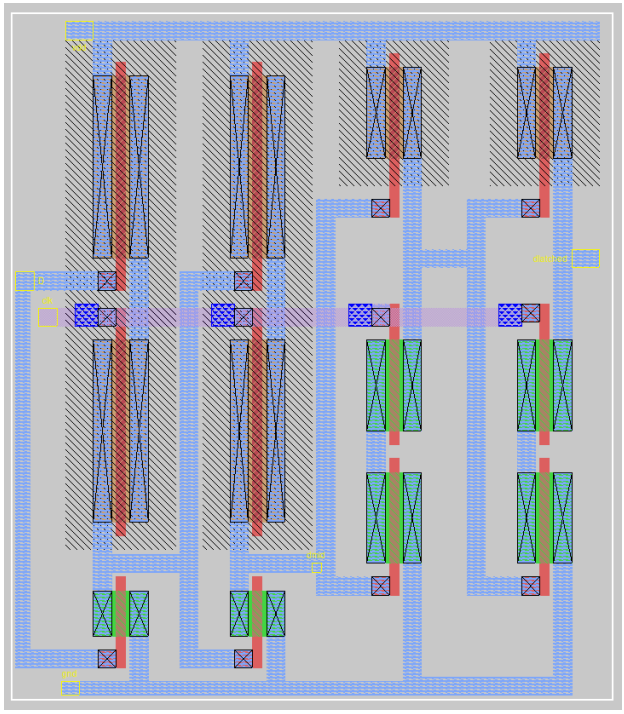
- *NOR GATE*



- *XOR GATE*



- ***D-FLIP-FLOP***



Here kindly do note that the xor gate is of the new topology and I have not implemented the old xor topology due to it being inefficient. Next steps were to extract the parameters (parasitic capacitance) for the layout in a .ext format and then in a .spice format, both of which are uploaded on the onedrive link provided earlier. The post layout analysis of the .cir files was made a lot simpler by running a python script made by a batch-mate Krish Pandya. The script used is as follows -

convert.py

```python
import sys

def convert_spice_to_cir(input_file, output_file):
    with open(input_file, 'r') as infile, open(output_file, 'w') as outfile:
        outfile.write(".include TSMC_180nm.txt\n")
        outfile.write(".param SUPPLY=1.8\n")
        outfile.write(".global gnd vdd\n\n")
        outfile.write("* Supply\n")
        outfile.write("Vdd vdd gnd 'SUPPLY'\n\n")

        for line in infile:
            line = line.replace('nfet', 'CMOSN')
            line = line.replace('pfet', 'CMOSP')
            line = line.replace('**FLOATING', ' ')
            if '.option scale=' in line:
                scale_value = line.split('=')[1].strip()
                if scale_value not in ['90n', '0.09u']:
                    line = '.option scale=0.09u\n'

            outfile.write(line)

if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("Usage: python convert.py <input_file> <output_file> ")
        sys.exit(1)

    input_file = sys.argv[1]
    output_file = sys.argv[2]

    convert_spice_to_cir(input_file, output_file)
    print(f"Conversion complete. {output_file}")
```

The link to the post layout files are here, and it also contains the python sript used. With this the post analysis was done. For reference the XOR post layout code is as follows -

```
.option scale=0.09u

M1000 a_25_n84# In_1 In_2 Gnd CMOSN w=10 l=2
+  ad=50 pd=30 as=50 ps=30
M1001 a_53_31# In_2 In_1 w_45_18# CMOSP w=20 l=2
+  ad=100 pd=50 as=100 ps=50
M1002 a_25_1# In_1 a_18_1# Gnd CMOSN w=10 l=2
+  ad=50 pd=30 as=50 ps=30
M1003 a_117_n37# a_115_n40# a_110_n37# w_104_n43# CMOSP w=20 l=2
+  ad=100 pd=50 as=100 ps=50
M1004 a_117_1# a_103_13# gnd Gnd CMOSN w=10 l=2
+  ad=50 pd=30 as=100 ps=60
M1005 a_25_n58# In_1 a_18_n58# w_12_n66# CMOSP w=20 l=2
+  ad=100 pd=50 as=100 ps=50
M1006 a_25_27# In_1 In_2 w_12_19# CMOSP w=20 l=2
+  ad=100 pd=50 as=100 ps=50
M1007 a_n17_n31# In_2 gnd Gnd CMOSN w=10 l=2
+  ad=50 pd=30 as=0 ps=0
M1008 a_n17_n5# In_2 Vdd w_n30_n13# CMOSP w=20 l=2
+  ad=100 pd=50 as=100 ps=50
M1009 In_1 In_2 a_52_n55# Gnd CMOSN w=10 l=2
+  ad=50 pd=30 as=50 ps=30
C0 w_n30_n13# In_2 0.08fF
C1 w_12_19# In_1 0.08fF
C2 a_115_n40# N1 0.03fF
C3 a_18_1# N0 0.04fF
C4 In_2 N2 0.09fF
C5 a_25_27# N1 0.05fF
C6 w_12_n66# N2 0.21fF
C7 gnd a_n17_n31# 0.14fF
C8 In_2 a_52_n55# 0.05fF
C9 w_104_n43# N1 0.06fF
C10 a_117_1# gnd 0.10fF
C11 w_12_19# N1 0.21fF
C12 w_104_n43# a_110_n37# 0.03fF
C13 In_2 gnd 0.05fF
C14 w_104_n43# a_115_n40# 0.15fF
C15 In_1 In_2 0.08fF
C16 w_n30_n13# a_n17_n5# 0.03fF
C17 a_52_n55# N2 0.06fF
C18 w_12_19# a_25_27# 0.03fF
C19 w_12_n66# In_1 0.08fF
C20 w_45_18# In_2 0.13fF
C21 a_18_n58# N0 0.05fF
C22 a_n17_n31# N0 0.04fF
C23 a_18_n58# a_25_n58# 0.21fF
C24 a_117_1# N1 0.04fF
C25 a_143_32# N1 0.05fF
C26 a_103_13# N2 0.03fF
C27 In_2 N1 0.01fF
C28 w_12_n66# N0 0.19fF
C29 In_1 a_52_n55# 0.10fF
C30 w_n30_n13# N0 0.21fF
C31 w_12_n66# a_25_n58# 0.03fF
C32 In_1 gnd 0.03fF
C33 N0 N2 0.05fF
C34 In_1 a_25_1# 0.03fF
C35 In_1 a_53_31# 0.21fF
C36 In_2 a_25_27# 0.24fF
C37 w_n30_n13# Vdd 0.07fF
C38 w_45_18# a_53_31# 0.03fF
C39 a_25_n58# N2 0.05fF
C40 w_45_18# In_1 0.06fF
C41 w_12_19# In_2 0.07fF
C42 a_110_n37# N2 0.05fF
C43 a_n17_n5# N0 0.05fF
C44 a_110_n37# a_117_n37# 0.21fF
C45 a_25_1# N1 0.04fF
C46 a_53_31# N1 0.07fF
C47 w_104_n43# N2 0.19fF
C48 In_1 N0 0.26fF
C49 In_2 a_25_n84# 0.14fF
C50 w_45_18# N1 0.21fF
C51 Vdd a_n17_n5# 0.24fF
C52 w_12_n66# a_18_n58# 0.03fF
C53 w_104_n43# a_117_n37# 0.04fF
C54 a_18_1# a_25_1# 0.10fF
C55 In_1 a_18_1# 0.08fF
C56 N1 N0 0.03fF
C57 a_25_n84# N2 0.04fF
C58 N2 Gnd 0.54fF
C59 N0 Gnd 0.71fF
C60 a_25_n84# Gnd 0.02fF
C61 a_52_n55# Gnd 0.01fF
C62 a_117_n37# Gnd 0.01fF
C63 a_n17_n31# Gnd 0.02fF
C64 gnd Gnd 0.11fF
C65 a_103_13# Gnd 0.14fF
C66 a_143_32# Gnd 0.11fF
C67 In_2 Gnd 0.14fF
C68 In_1 Gnd 0.74fF
C69 w_104_n43# Gnd 0.82fF
C70 w_12_n66# Gnd 0.82fF
C71 w_45_18# Gnd 0.29fF
```

```
  Measurements for Transient Analysis

delay_a_to_c        =  2.689435e-11 targ=  7.689435e-11 trig=  5.000000e-11
```

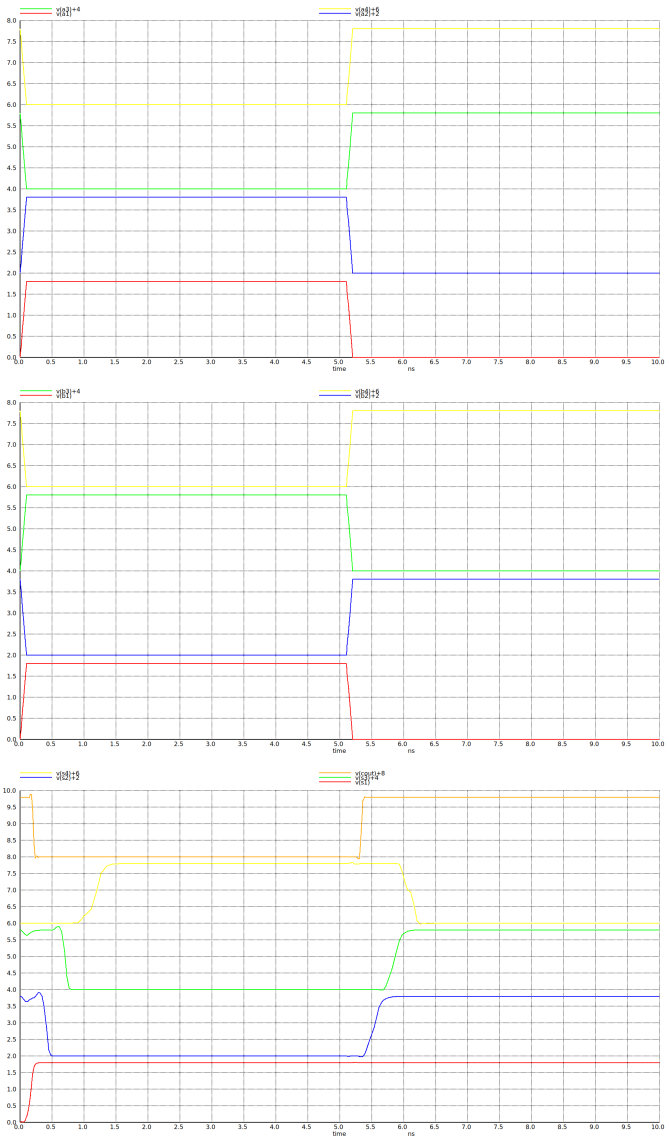
We leave the images of codes and timing analysis of other circuits for the report but you can find them in the MAGICs one drive link. We find that there is a change of 51%. Similar changes can be observed in other circuits as well.

VII. Integrated NG Spice code

Here we have made the complete circuit as illustrated in the circuit design given before. The screenshots of the code are avoided here due to the sheer length of it, but it can be found in the NG-SPICE onedrive link attached before. This avoids the usage of flipflops initially to calculate the maximum delay in propagation we have. This allows us to find a clock frequency of for my flipflops and they are used later.

```
xa1 g1 a1 b1 vdd gnd and_with_inv
xa2 g2 a2 b2 vdd gnd and_with_inv
xa3 g3 a3 b3 vdd gnd and_with_inv
xa4 g4 a4 b4 vdd gnd and_with_inv
xx1 p1 a1 b1 vdd gnd xor_new
xx2 p2 a2 b2 vdd gnd xor_new
xx3 p3 a3 b3 vdd gnd xor_new
xx4 p4 a4 b4 vdd gnd xor_new
xa5 n1 p1 c1 vdd gnd and_with_inv
xx5 s1 p1 c1 vdd gnd xor_new
xo1 c2 g1 n1 vdd gnd or_with_inv
xa6 n2 p2 c2 vdd gnd and_with_inv
xx6 s2 p2 c2 vdd gnd xor_new
xo2 c3 g2 n2 vdd gnd or_with_inv
xa7 n3 p3 c3 vdd gnd and_with_inv
xx7 s3 p3 c3 vdd gnd xor_new
xo3 c4 g3 n3 vdd gnd or_with_inv
xa8 n4 p4 c4 vdd gnd and_with_inv
xx8 s4 p4 c4 vdd gnd xor_new
xo4 cout g4 n4 vdd gnd or_with_inv

.tran 0.1n 10n
.measure tran delay_a1_to_s1
+ TRIG v(a1) VAL='SUPPLY/2' RISE=1
+ TARG v(s1) VAL='SUPPLY/2' RISE=1
.measure tran delay_a2_to_s2
+ TRIG v(a2) VAL='SUPPLY/2' RISE=1
+ TARG v(s2) VAL='SUPPLY/2' FALL=1
.measure tran delay_a3_to_s3
+ TRIG v(a3) VAL='SUPPLY/2' RISE=1
+ TARG v(s3) VAL='SUPPLY/2' RISE=1
.measure tran delay_a4_to_s4
+ TRIG v(a4) VAL='SUPPLY/2' RISE=1
+ TARG v(s4) VAL='SUPPLY/2' FALL=1
```

The Waveforms and the timing analysis of the circuit without the flipflop are as follows, kindly do note that the input of signal A ($A_4A_3A_2A_1$) is plotted in the first plot and the input of signal B ($B_4B_3B_2B_1$) is plotted in the next plot. These are shifted by 2 units for a better viewing. The third plot has the Output Sum Signals ($S_4S_3S_2S_1$) and the Carry out signal (C_{OUT})-



These plots are also included in the onedrive link provided here. The timing analysis for this circuit is as follows -

```
  Measurements for Transient Analysis

delay_a1_to_s1      =  1.133381e-10 targ=  1.633381e-10 trig=  5.000000e-11
delay_a2_to_s2      =  3.574642e-10 targ=  4.074642e-10 trig=  5.000000e-11
delay_a3_to_s3      =  7.012267e-10 targ=  5.851227e-09 trig=  5.150000e-09
delay_a4_to_s4      =  9.619201e-10 targ=  6.111920e-09 trig=  5.150000e-09
```

Using the inequality obtained earlier the clock time can be calculated as -

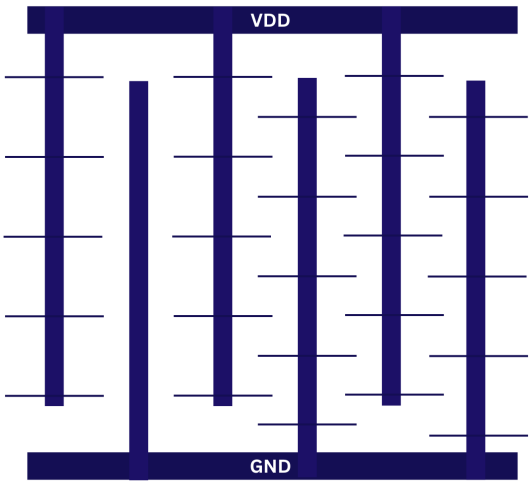
$$T_{clk} >= t_{max\_circuit} + 12.1 \times 10^{-11} s$$

$$T_{clk} >= 10 \times 10^{-10} + 12.1 \times 10^{-11} s = 1.2 \times 10^{-9} s$$

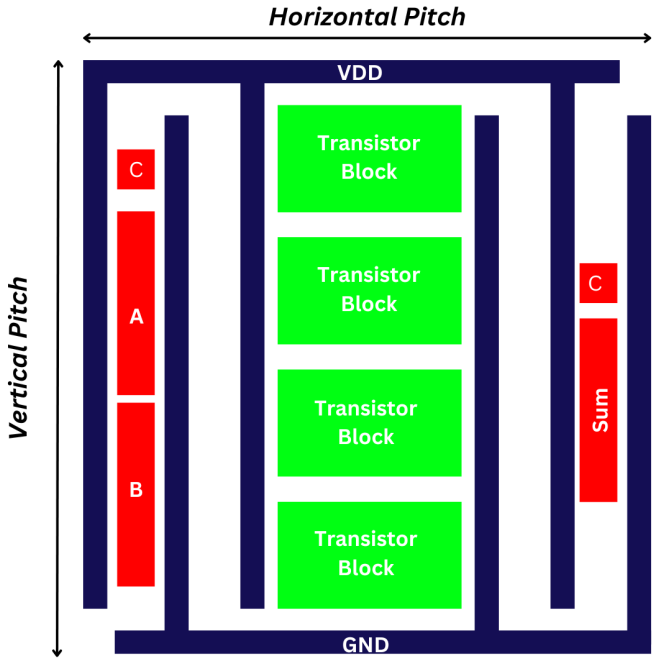
Here we set our clock to $1.2 \times 10^{-9}s = 0.853GHz$ for our flip flop analysis. This has been done so that the circuit has enough time to drive the output to the desired level.

VIII. Floorplan of the Layout

The first step in floor plan is the routing of Supply and clock rails. Power and ground must be available to all systems, and should only cross when absolutely necessary. One idea is to have an inter-digitated structure for atleast the Vdd and Gnd wires. An image of the same is attached as follows -

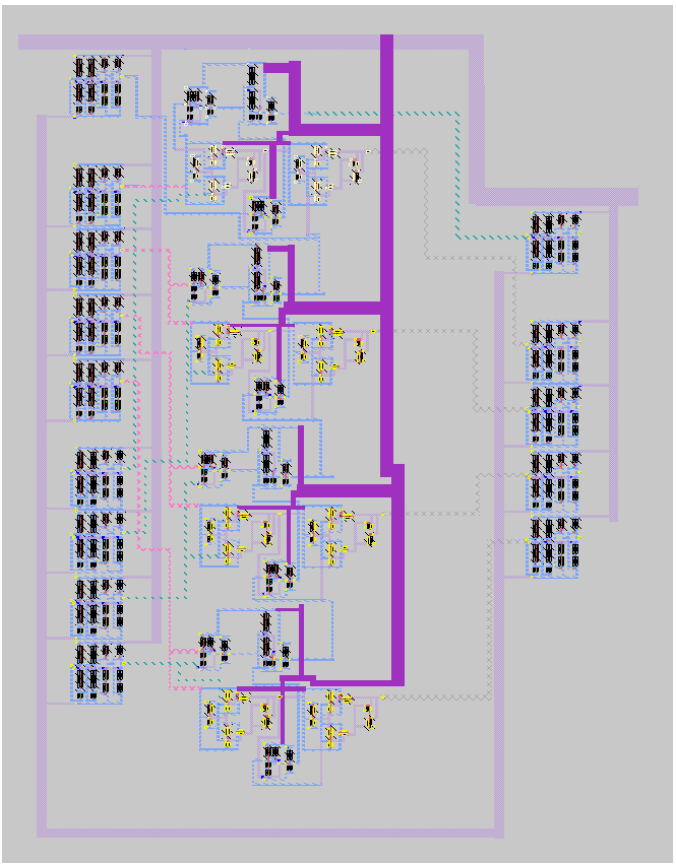


Polysilicon can be used for medium routing lines due to its lower resistivity. Our main strategy is to split the routing into a global routing and a local routing problem. We also aim to keep the Input and Output Registers on the edges for ease of access during tapeout. The following image gives my plan for the combined layout.



IX. Complete Magic Layout

After all this effort, now comes the big deal, the combined layout of the circuit. It follows a general trend of the planned layout explained in the previous section. This ensures that we follow a structured manner and the debugging is easier. The Completed Layout is as follows -



The onedrive link to this layout is this. It will have all the magic files, .ext, .spice, .cir extracted version and other relevant files. For the sake of simplicity, I will not show the plots and the timing analysis for the circuit (it can be found in the onedrive folder shared above but here is a table comparing the results of the pre and the post layout circuits-

Parameter	Pre-Layout	Post-Layout
Clock Period	$1.2 \times 10^{-9}s$	$1.9 \times 10^{-8}s$
Max Delay in O/p (without FF)	$9.6 \times 10^{-10}s$	$5.8 \times 10^{-9}s$

Here we find a massive change in the value of delay which affects our performance of the circuit. With this our SPICE-MAGIC part of the project comes to an end. It was a great learning experience for me going through the various books and papers to make this circuit. Now the Hardware-Verilog part of the project begins. Here we will make a Synthesizable HDL in Verilog and test it while also using FPGA boards to test our logic. Oscilloscopes were used to plot the waveforms from the FPGA board.

X. Verilog Code

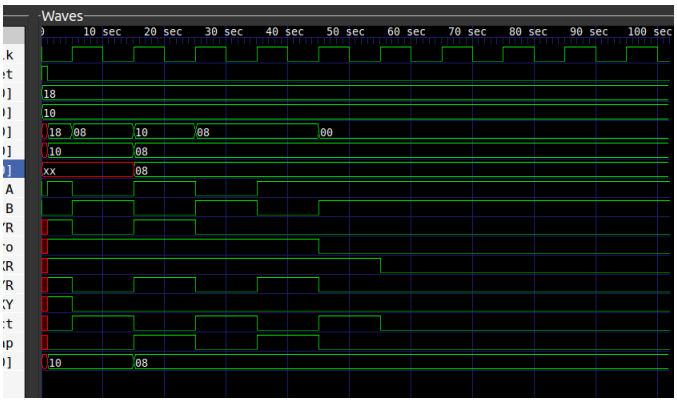
The following code was used in Verilog to implement this circuit -

```
module CLA_4bit (
    input [3:0] a, b,       // 4-bit inputs
    input cin,              // Carry-in
    output [3:0] sum,       // 4-bit sum output
    output cout             // Carry-out
);
    wire [3:0] p, g;         // Propagate and Generate signals
    wire [3:0] c;            // Carry signals

    // Propagate and Generate signals
    assign p = a ^ b;        // Propagate: Pi = Ai XOR Bi
    assign g = a & b;        // Generate: Gi = Ai AND Bi

    // Carry signals
    assign c[0] = cin;
    assign c[1] = g[0] | (p[0] & c[0]);
    assign c[2] = g[1] | (p[1] & c[1]);
    assign c[3] = g[2] | (p[2] & c[2]);
    assign cout = g[3] | (p[3] & c[3]);

    // Sum calculation
    assign sum = p ^ c[3:0]; // Si = Pi XOR Ci
endmodule
```

With this, the following testbench file was used -

```
`timescale 1ns/1ps
module CLA_4bit_tb;
    reg [3:0] a, b;         // 4-bit inputs
    reg cin;                // Carry-in
    wire [3:0] sum;         // 4-bit sum output
    wire cout;              // Carry-out

    // Instantiate the CLA module
    CLA_4bit uut (
        .a(a),
        .b(b),
        .cin(cin),
        .sum(sum),
        .cout(cout)
    );

    initial begin
        // Initialize inputs
        a = 4'b0000; b = 4'b0000; cin = 1'b0;
        #10 a = 4'b1100; b = 4'b1010; cin = 1'b0; // Test case 1
        #10 a = 4'b0101; b = 4'b0111; cin = 1'b1; // Test case 2
        #10 a = 4'b1111; b = 4'b1111; cin = 1'b1; // Test case 3
        #10 a = 4'b0001; b = 4'b0010; cin = 1'b0; // Test case 4
        #10 $finish;
    end

    initial begin
        $dumpfile("CLA_4bit.vcd"); // Dump file for GTKWave
        $dumpvars(0, CLA_4bit_tb); // Dump variables
    end
endmodule
```

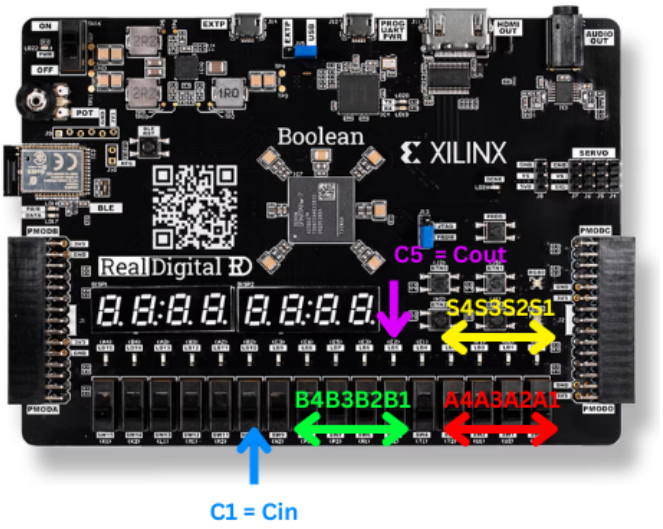
The plots are uploaded on the onedrive for the sake of keeing this report short. One of the plots is as follows -



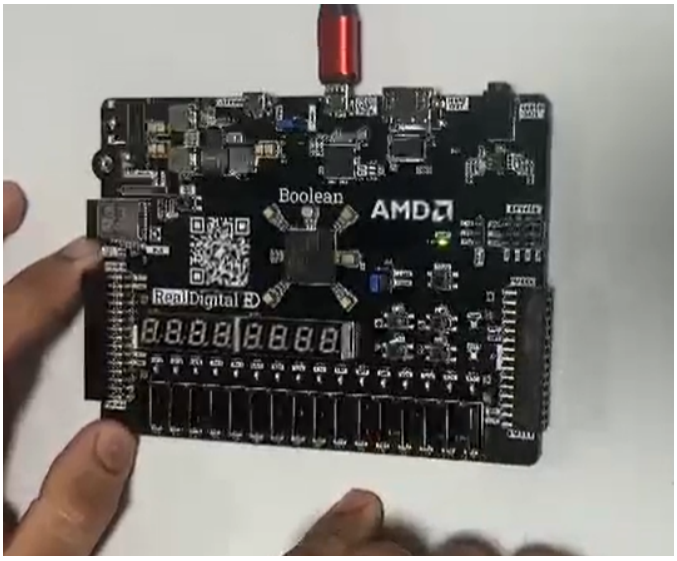
The link of onerive containing the codes and the plots is here

XI. FPGA Implemenntation

The following is a FPGA IO diagram used in the project -



The plots, Oscilloscope readings, videos of the output detection, etc are uploaded to the onedrive folder for ease of viewing. One of the photos of the testing is as follows -



ACKNOWLEDGMENT

A sincere thanks goes to all the people who helped and motivated me to the completion of this project. It could not have been possible without the constant support of the TAs and my family and friend. A special thanks goes out to our professor Dr. A. Srivastava, who gave us the opportunity to do this project. It was a golden opportunity for us to implement the theory we learned in class to use and see the circuit we made come to life.

This project was the most genuine learning experience we could have asked for in the domain of VLSI Designing. This comes straight from heart that the project was one of the best ones i have ever attempted. Thanks all for giving me this opportunity to do this project.