4-bit Carry Look Ahead Adder Design

VLSI Design Course Project

Aanchal Amit Mundhada

2023112016

Electronics and Communication Dual Degree

International Institute of Information Technology, Hyderabad

aanchal.mundhada@research.iiit.ac.in

***Abstract* — This This report presents a design of 4-bit carry look ahead adder which has inputs taken from D flip flop and outputs also pass through the D flip flop.**

Keywords—adder, carry look ahead adder, D flip flop, sum block, CLA, Propagate Generate

I. INTRODUCTION

The adder in this project is designed using basic digital logic components like AND gates, OR gates, and inverters, which are combined to form full adders. Each full adder calculates the sum and carry for a pair of input bits. The adder follows a ripple-carry structure, where the carry output from one bit is passed to the next, starting from the least significant bit to the most significant bit. If the numbers to be added are a3a2a1a0 and b3b2b1b0, then the propagate *(pi)* and generate *(gi)* signals for each bit position can be defined as,

$$pi = ai \oplus bi$$

$$gi = ai \,.bi$$

and the carry out (*c(i+1)*) of the i^{th} bit position can be written as,

$$c(i+1) = (pi \,.ci\,) + gi\,,\ i = 0, 1, 2, 3$$

Thus, *c(i+1)* can be expressed entirely in terms of the *pi* and *gi* functions and sum can be represented as follows,

$$sumi = pi \oplus ci$$

TSPC D flip-flops are included to manage timing and hold intermediate values, ensuring the circuit operates reliably in clocked systems. This modular design allows for easy expansion to handle larger bit-widths, making it adaptable for various applications.

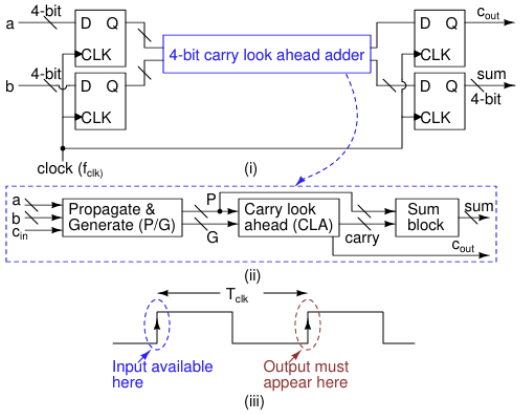


Fig. 1. Adder

II. DESIGN DETAILS

The circuit is made by combining logic gates AND, OR and XOR. It also uses inverters and D flip flops.

The propagate generate block takes input ai and bi from D flip flop and using 2 input AND gate we calculate *gi* as *gi=ai.bi.* On using 2 input XOR gate we calculate *pi* as *pi = ai⊕bi.*

For carry look ahead block the inputs are pi, gi and c0. We calculate c1, c2, c3 and c4.

c1 = g0 + p0c0, here we use 2 input AND gate for p0c0 and then 2 input OR gate to combine g0 and p0c0 to calculate our c1.

c2 = g1 + p1g0 + p1p0c0, here we use 3 input AND gate for p1p0c0 and 2 input AND gate for p1g0, then 3 input OR gate to combine g1, p1g0 and p1p0c0 to calculate our c2.

c3 = g2 + p2g1 + p2p1g0 + p2p1p0c0, here we use 4 input AND gate for p2p1p0c0, 3 input AND gate for p2p1g0 and 2 input AND gate for p2g1, then 4 input OR gate to combine g2, p2g1, p2p1g0 and p2p1p0c0 to calculate our c3.

c4 = g3 + p3g2 + p3p2g1 + p3p2p1g0 + p3p2p1p0c0, here we use 5 input AND gate for p3p2p1p0c0, 4 input AND gate for p3p2p1g0, 3 input AND gate for p3p2g1 and 2 input AND gate for p3g2, then 5 input OR gate to combine g3, p3g2, p3p2g1, p3p2p1g0 and p3p2p1p0c0 to calculate our c4.

(Note: It is given that c0 is zero, so we don't really need 5 input AND and OR gates. I have used % input gates in NGSpice but not in MAGIC)

For sum block, we use 2 input XOR gate with inputs ci and pi calculated from previous blocks as *sumi = pi ⊕ ci.*

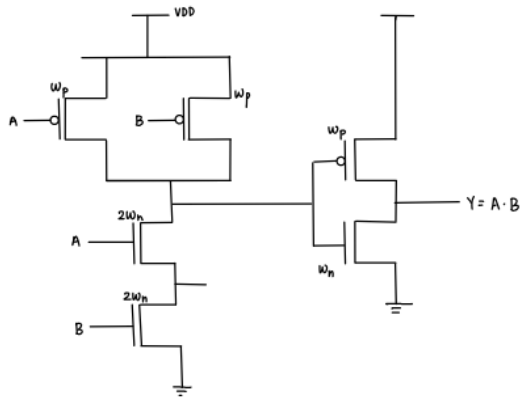
We get outputs s0, s1, s2, s3 and c4 which is passed through D flip flop.

A. Gates

For all the gates we use CMOS Topology. Their circuit diagram along with sizing are given below.

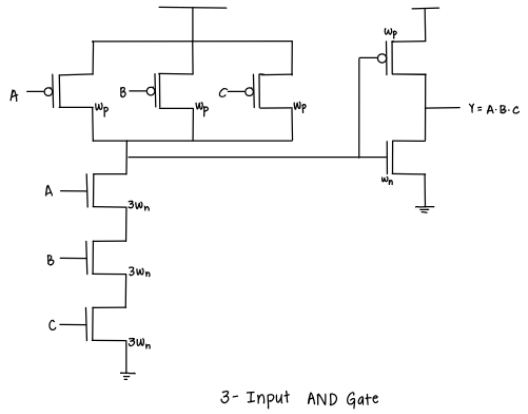
i. AND gates

Sizing: Wp and NWn, N is number of inputs.



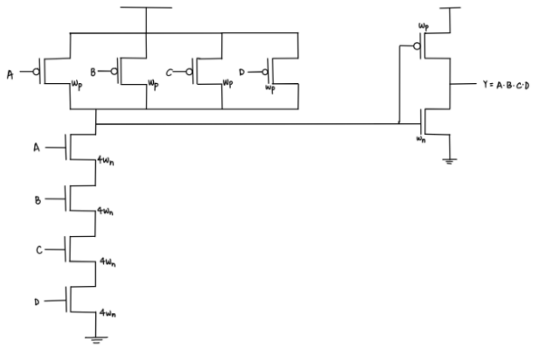
2 – Input AND Gate

(a)



3- Input AND Gate

(b)



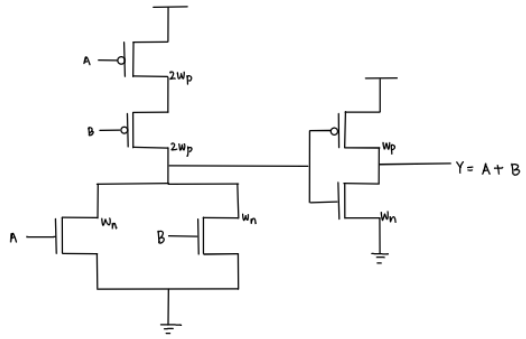
4- Input AND Gate

(c)

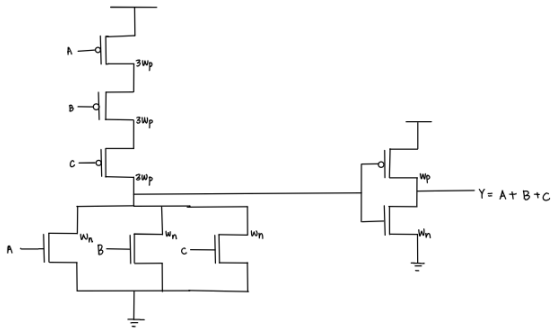
Fig. 2. (a)2 input AND (b) 3 input AND (c)4 input AND

ii. OR gates

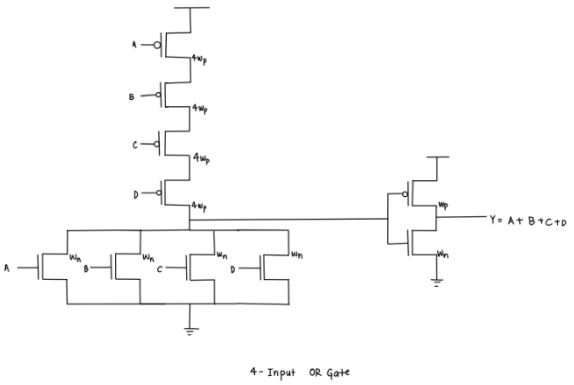
Sizing: NWp and Wn, N is number of inputs.



(a)



(b)



(c)

Fig. 3. (a)2 input OR (b) 3 input OR (c)4 input OR

iii. XOR gate

XOR gate made by using 2 inverters, 2 AND gates and 1 OR gate as shown in Fig. 4

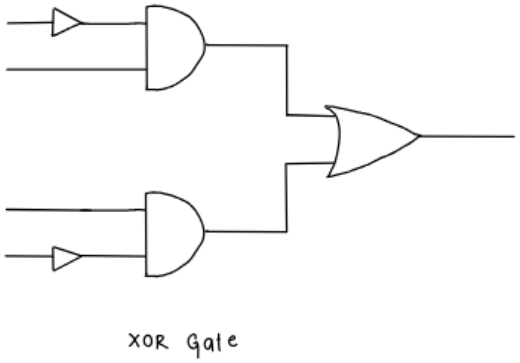


Fig. 4. 2 input XOR gate

B. TSPC flip flop

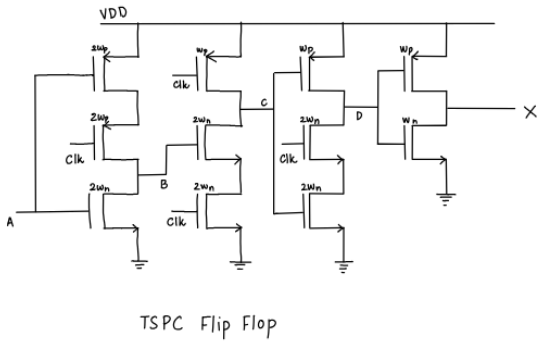


Fig. 5. TSPC flip flop

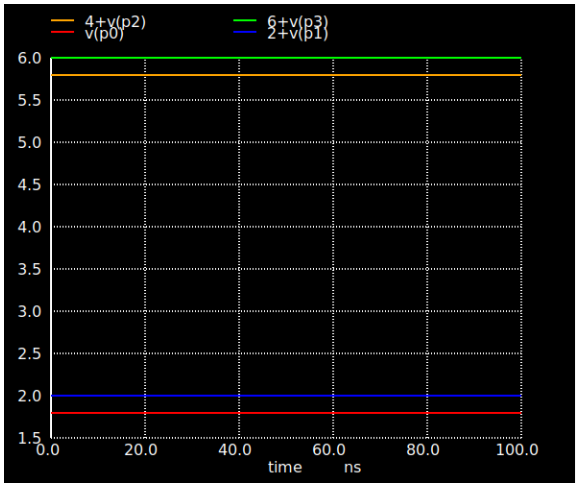
III. PRE-LAYOUT SIMULATION IN NGSPICE

A. CLA adder

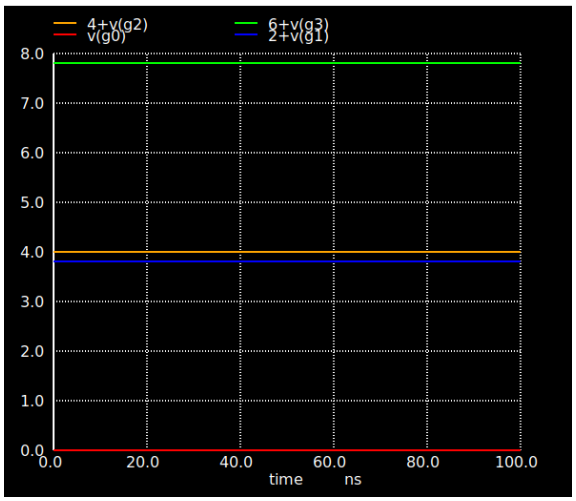
i. Propagate and generate

For, A=1110 and B=1011 we get following outputs,

p0=1, p1=0, p2=1, p3=0 and g0=0, g1=1, g2=0, g3=1



(a)



(b)

Fig. 6. (a) Propagate (b) Generate

ii. Carry look ahead

For, p0=0, p1=1, p2=1, p3=1 and g0=1, g1=1, g2=0, g3=1, c0=0 we get the following outputs,

c1=1, c2=1, c3=1, c4=1

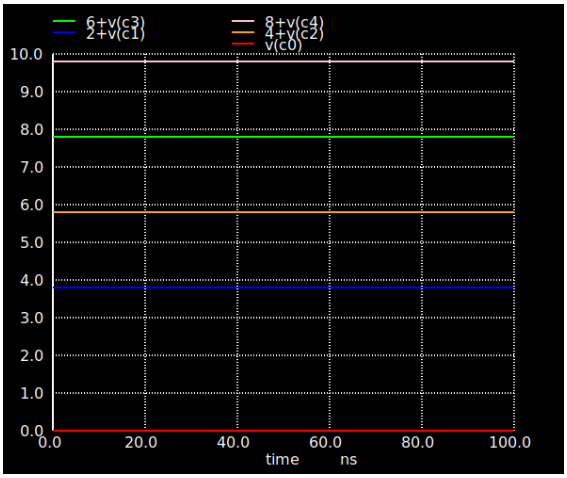


Fig. 7. Carry look ahead block

iii. Sum block

For c0=0, c1=1, c2=1, c3=1 and p0=1, p1=1, p2=0, p3=1 we get following outputs,

s0=1, s1=0, s2=1, s3=0

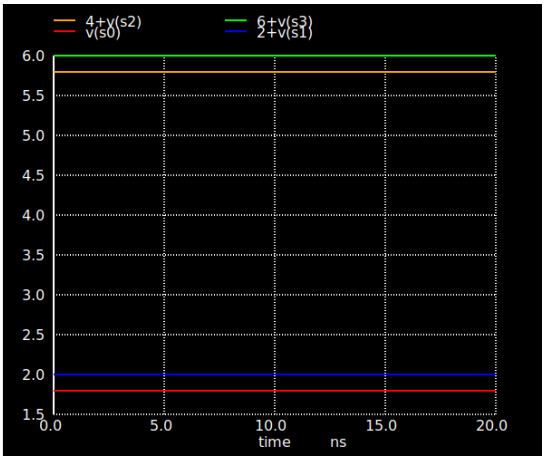


Fig. 8. Sum block

iv. Complete adder

For A=1110 and B=1011 we get sum=1001 and cout=1

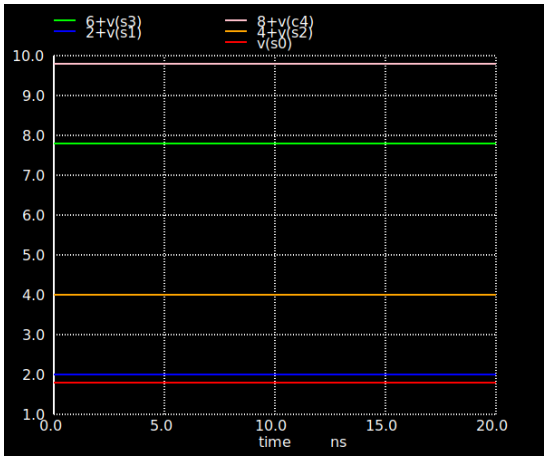


Fig. 9. Adder (pre layout)

We get tpd_max=0.72ns for s3.

```
tpd_rise            =  7.116351e-10 targ=  7.616351e-10 trig=  5.000000e-11
tpd_fall            =  7.343098e-10 targ=  1.088431e-08 trig=  1.015000e-08
tpd_max             =  7.22972e-10
```

Fig. 10. tpd_max (pre layout)

B. D flip flop

As we can see, X is updated to A at positive edge only.

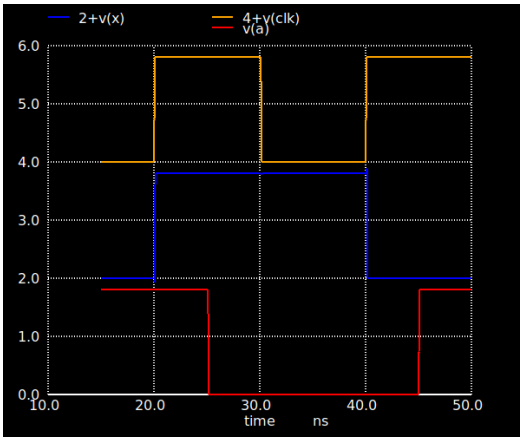


Fig. 11. D flip flop

IV. SETUP TIME, HOLD TIME AND PROPAGATION DELAY FROM CLK TO Q IN FLIPFLOP

We have used TSPC topology for our D-FlipFlop. Since there is no use of clk', we have zero hold time (thold = 0).

For setup time, we did trial and error and found it to be tsetup = 0.1019ns. Any input given less than tsetup before next +ve edge, either input is not registered or output is being corrupted.

Propagation delay from clk to Q will be 0.087ns.

```
tpd_rise            =  5.761607e-11 targ=  2.010762e-08 trig=  2.005000e-08
tpd_fall            =  1.167702e-10 targ=  4.016677e-08 trig=  4.005000e-08
total_prop_delay    =  8.71931e-11
```

Fig. 12. Total propagation delay (pre layout)

V. STICK DIAGRAM OF ALL UNIQUE GATES AND FLIPFLOP

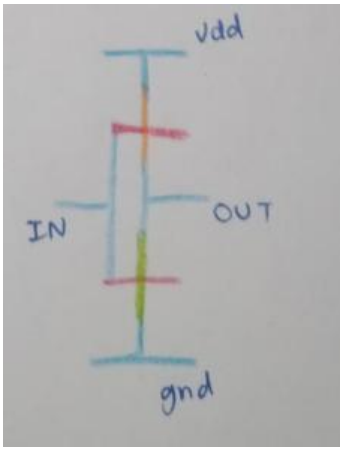


Fig. 13. Inverter stick diagram

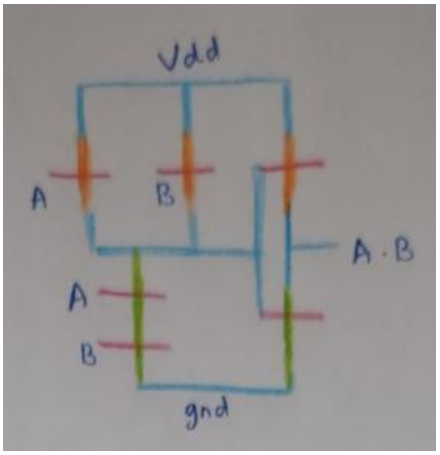


Fig. 14. 2 input AND gate stick diagram

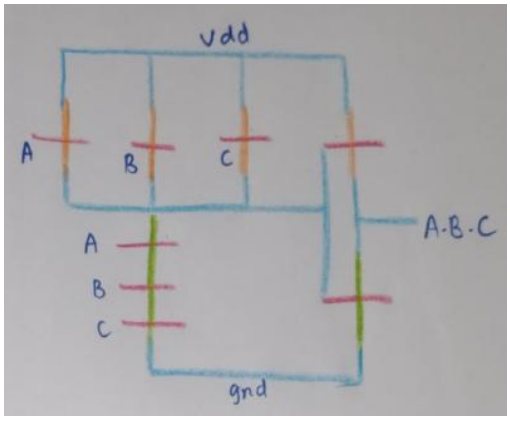


Fig. 15. 3 input AND gate stick diagram

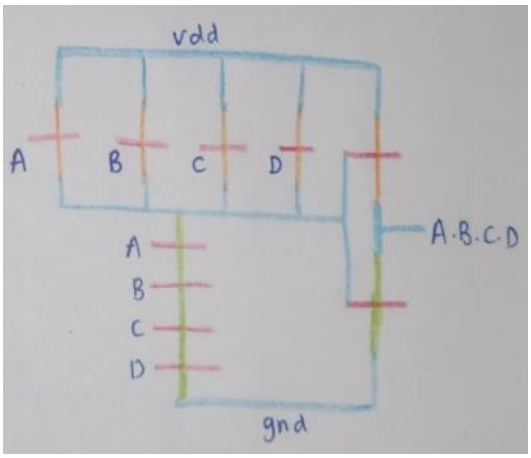


Fig. 16. 4 input AND gate stick diagram

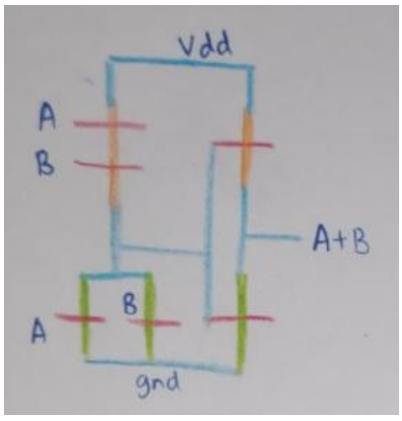


Fig. 17. 2 input OR gate stick diagram

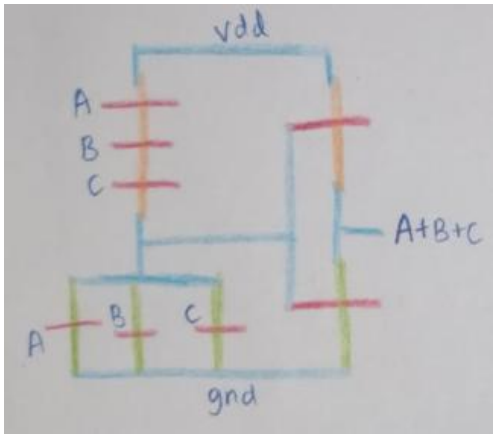


Fig. 18. 3 input OR gate stick diagram

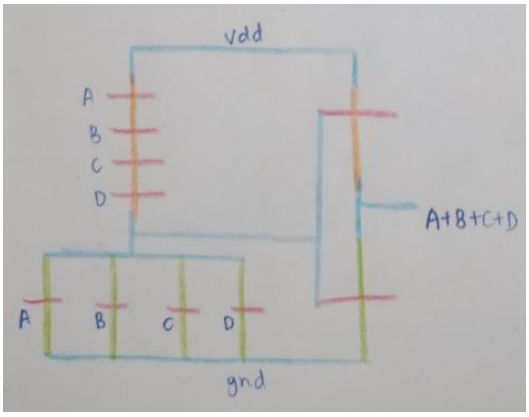


Fig. 19. 4 input OR gate stick diagram

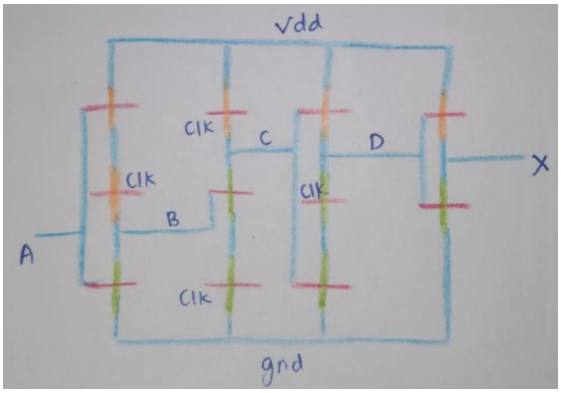


Fig. 20. TSPC flip flop stick diagram

VI. POST LAYOUT SIMULATION FOR MAGIC

A. CLA adder

i. Propagate and generate

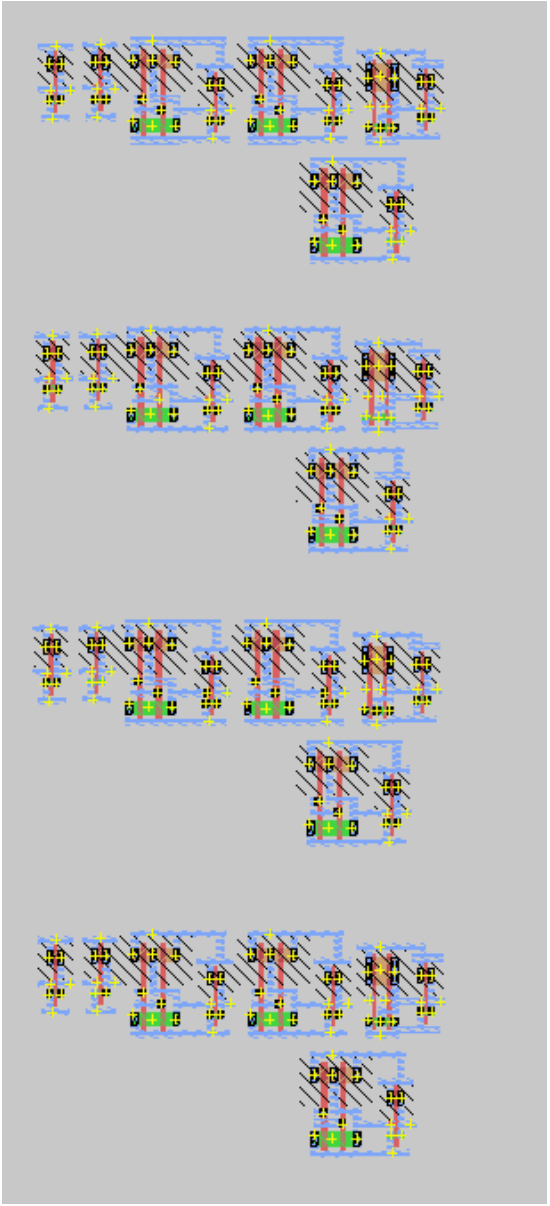
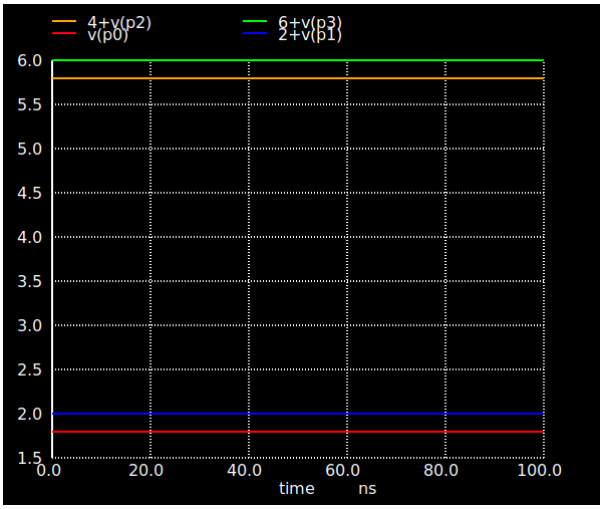


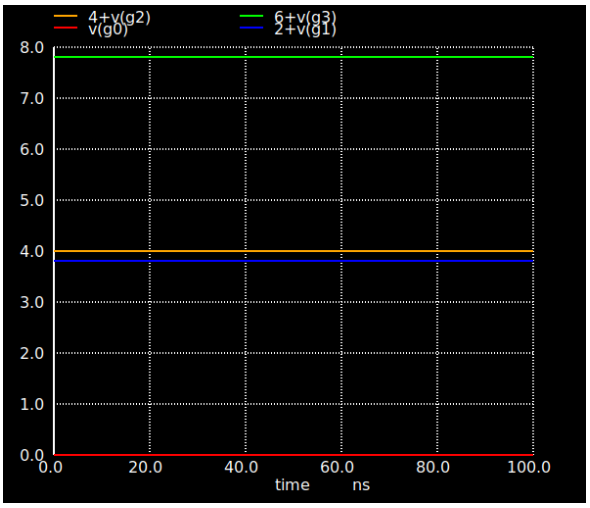
Fig. 21. Propagate Generate MAGIC layout

For, A=1110 and B=1011 we get following outputs,

p0=1, p1=0, p2=1, p3=0 and g0=0, g1=1, g2=0, g3=1



(a)



(b)

Fig. 22. (a)Post layout propagate (b)Post layout generate

ii. Carry look ahead

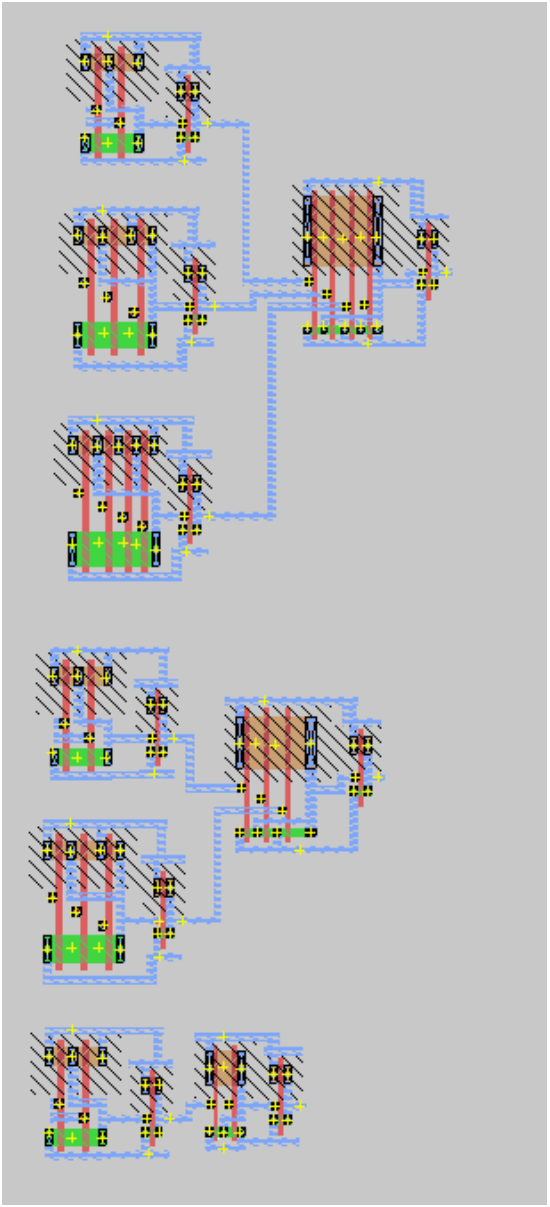


Fig. 23. Carry look ahead MAGIC layout

For, p0=0, p1=1, p2=1, p3=1 and g0=1, g1=1, g2=0, g3=1, c0=0 we get the following outputs,

c1=1, c2=1, c3=1, c4=1

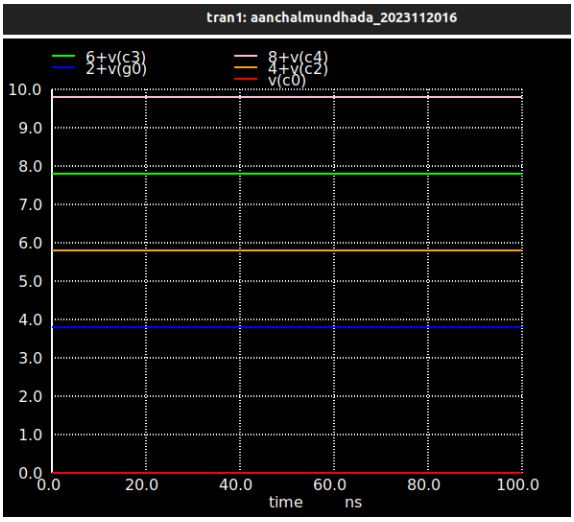


Fig. 24. Post layout carry look ahead block

iii. Sum block

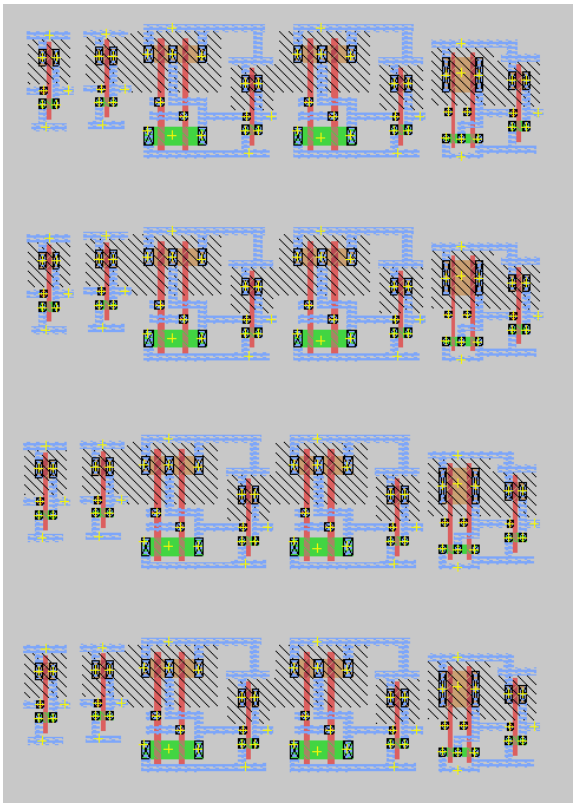


Fig. 25. Sum block MAGIC layout

For c0=0, c1=1, c2=1, c3=1 and p0=1, p1=1, p2=0, p3=1 we get following outputs,

s0=1, s1=0, s2=1, s3=0

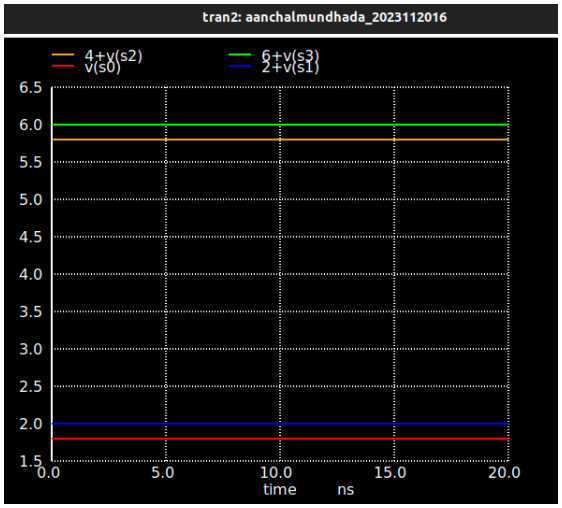


Fig. 26. Post layout sum block

iv. Complete adder

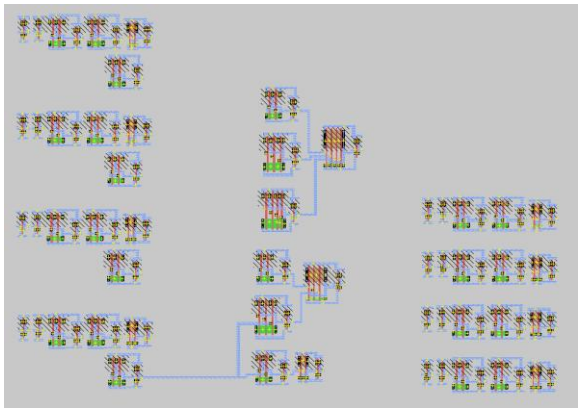


Fig. 27. Adder MAGIC layout

For A=1110 and B=1011 we get sum=1001 and cout=1

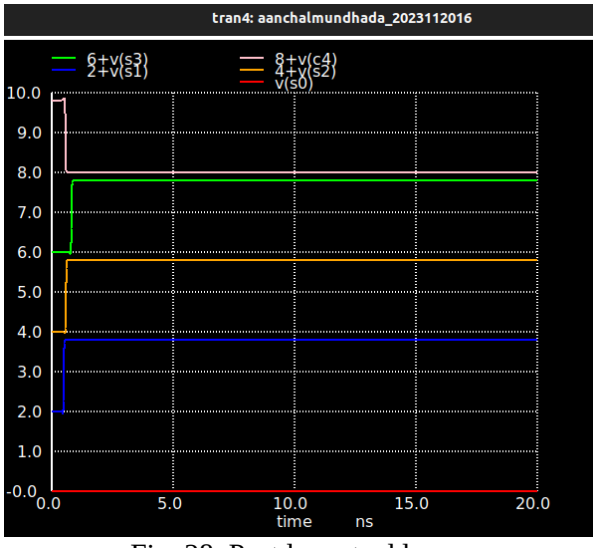


Fig. 28. Post layout adder

We get tpd_max=0.82ns for s3.

```
tpd_rise            =  8.036613e-10 targ=  8.536613e-10 trig=  5.000000e-11
tpd_fall            =  8.453609e-10 targ=  1.099536e-08 trig=  1.015000e-08
tpd_max             =  8.24511e-10
```

Fig. 29. tpd_max (post layout)

B. D flip flop

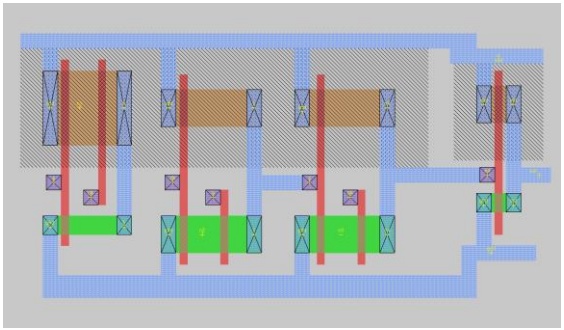


Fig. 30. D flip flop MAGIC layout

As we can see, X is updated to A at positive edge only.

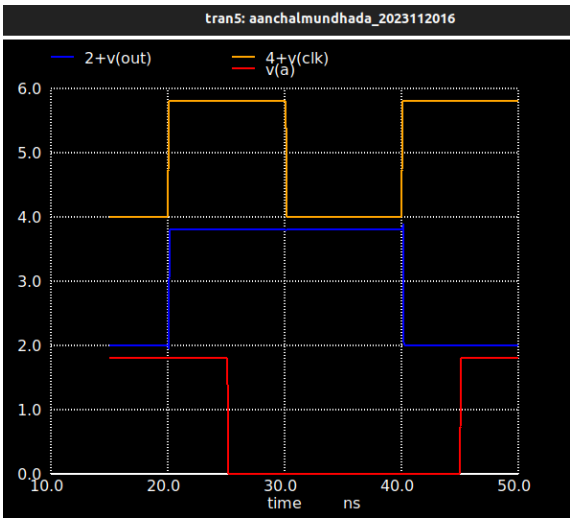


Fig. 31. Post layout D flip flop

Total propagation delay will be 0.10ns.

```
tpd_rise            =  7.060231e-11 targ=  2.012060e-08 trig=  2.005000e-08
tpd_fall            =  1.346496e-10 targ=  4.018465e-08 trig=  4.005000e-08
total_prop_delay    =  1.02626e-10
```

Fig. 32. Total propagation delay (post layout)

	Pre-layout	Post-layout
Propagation delay for D flip flop	0.0872ns	0.1026ns
tpd_max for adder	0.723ns	0.8245ns

Table 1. Propagation delay and tpd_max comparison

VII. SIMULATIONS OF FINAL CIRCUIT IN NGSPICE

We combine the CLA-Adder with D-Flipflops to give input at positive edge and get output at second positive edge.

For A=1110 and B=1011 we get sum as 1001 and cout as 1 at the second positive edge of the clock. This means our circuit is working correctly.

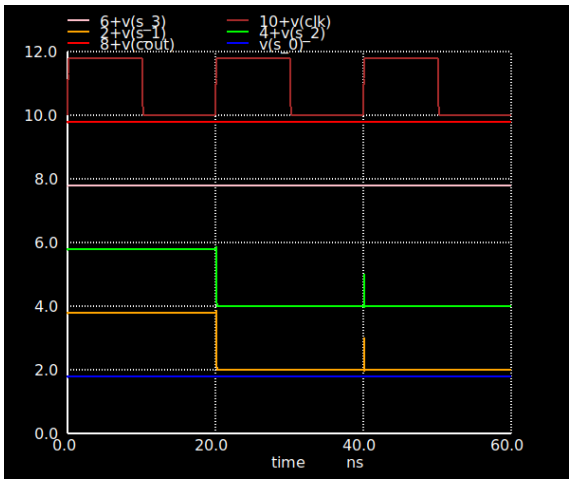


Fig. 33. Final circuit (pre layout)

Worst case delay seen in pre-layout simulation is 0.723ns.

For maximum clock frequency we find minimum clock time.

tclk_min = *thold* + *tpd_max* + *tpcq_max*

tclk_min = 0 + 0.72ns + 0.087ns = 0.807ns

Hence, *fmax* = 1239.16 MHz.

Maximum clock frequency at which our circuit operates correctly is 1239.16 MHz.

VIII. FLOOR PLAN AND MAGIC LAYOUT FOR FINAL CIRCUIT

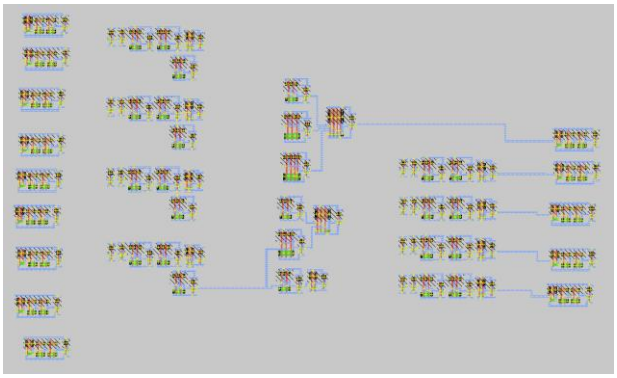


Fig. 34. Final circuit MAGIC layout

This is the complete floor plan of final circuit with Horizontal Pitch 167.13µm and Vertical Pitch 97.65µm with area 16320$(\mu m)^2$.

IX. POST LAYOUT SIMULATIONS OF FINAL CIRCUIT

For A=1110 and B=1011 we get sum as 1001 and cout as 1 at the second positive edge of the clock. It matches with the pre-layout simulation.

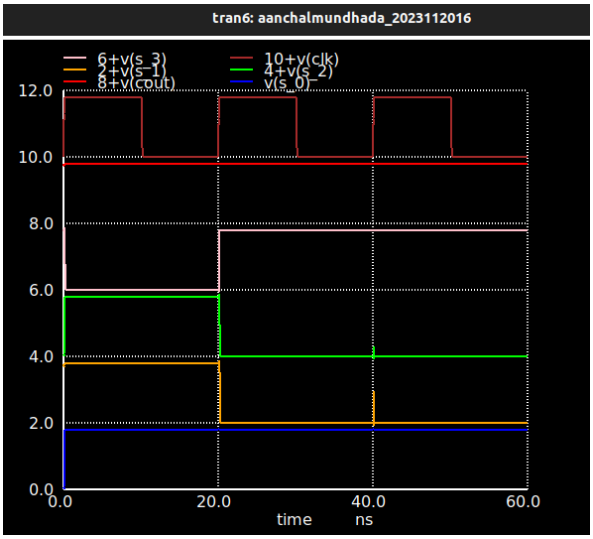


Fig. 35. Final circuit (post layout)

X. DELAY OF CLA ADDER AND MAXIMUM CLOCK FREQUENCY

Worst case delay seen in pre-layout simulation is 0.8245ns.

For maximum clock frequency we find minimum clock time.

tclk_min = thold + tpd_max + tpcq_max

tclk_min = 0 + 0.82ns + 0.103ns = 0.823ns

Hence, *fmax* = 1215.07 MHz.

Maximum clock frequency at which our circuit operates correctly is 1215.07 MHz.

	Pre-layout	Post-layout
tclk_min	0.807ns	0.823ns
fclk_max	1239.16 MHz	1215.07 MHz

Table 2. tclk_min and fclk_max comparison

Post layout time is more than pre layout time.

XI. VERILOG SIMULATIONS AND WAVEFORMS

This 4-bit carry-lookahead adder (CLA) consists of three main components: D flip-flops, a carry-lookahead adder, and a top-level module integrating the two. The D flip-flops are used to store input and output values, ensuring that data is synchronized with the clock signal. Each input bit of the 4-bit numbers (A and B) and the carry-in (Cin) is stored in separate flip-flops before being passed to the adder. Similarly, the output bits of the adder, including the sum (S[3:0]) and carry-out (Cout), are passed through another set of flip-flops to stabilize the output and synchronize it with the clock.

The core arithmetic operation is performed by the carry-lookahead adder. It computes the propagate and generate signals for each bit and uses these to calculate the carry signals hierarchically. The carry-lookahead logic allows faster addition by directly computing carries rather than waiting for sequential propagation. The use of D flip-flops at both input and output ensures proper timing synchronization, making the circuit suitable for clocked or pipelined systems.

A. CLA adder

Fig. 36. Adder verilog simulation

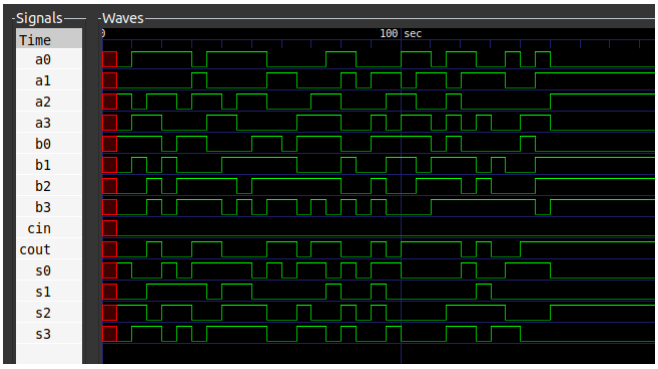


Fig. 37. Adder gtk waveform

B. D flip flop

For verilog simulation we take D flip flop and not TSPC flip flop.

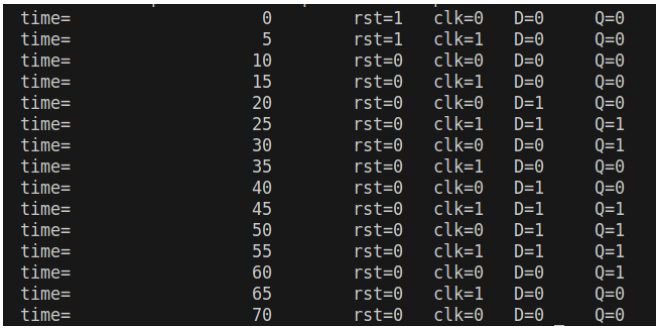


Fig. 38. D flip flop verilog simulation

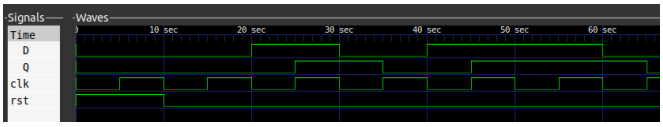


Fig. 39. D flip flop gtk waveform

C. Circuit

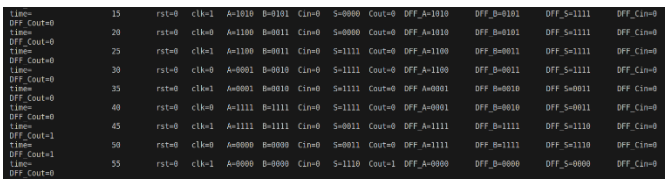


Fig. 40. Circuit verilog simulation

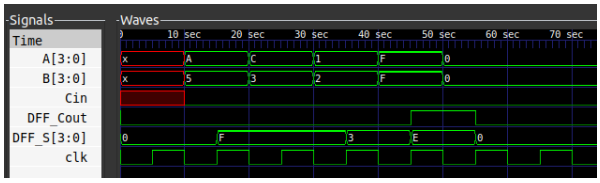


Fig. 41. Circuit gtk waveform

XII. FPGA AND OSCILLOSCOPE WAVEFORM

We take inputs as A=1110 and B=1011 and on using FPGA and oscilloscope we get carry as 1, s3=1, s2=0, s1=0 and s0=1 which is our desired output. Hence our waveforms below are correct.

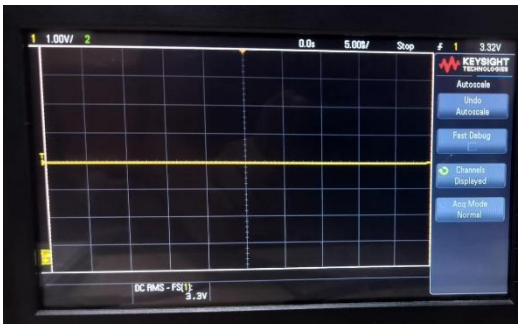


Fig. 42. Carry = 1

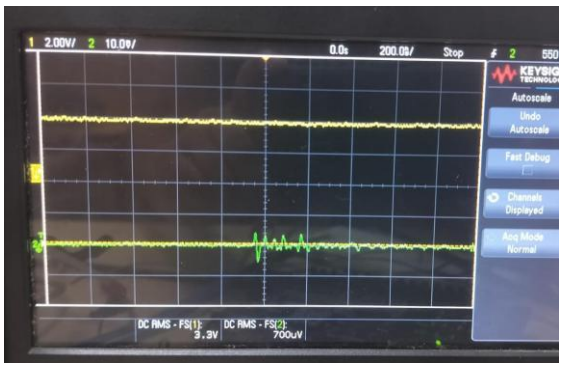


Fig. 43. s3 = 1 (yellow) and s2 = 0 (green)

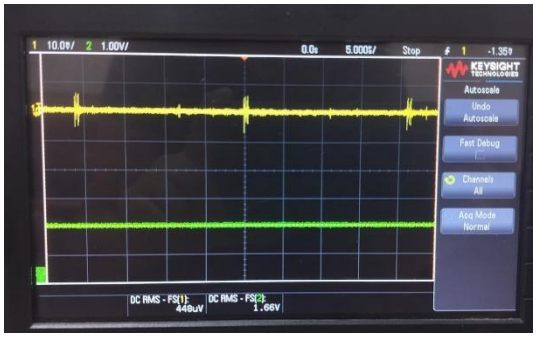


Fig. 44. s1 = 0 (yellow) and s0 = 1 (green)

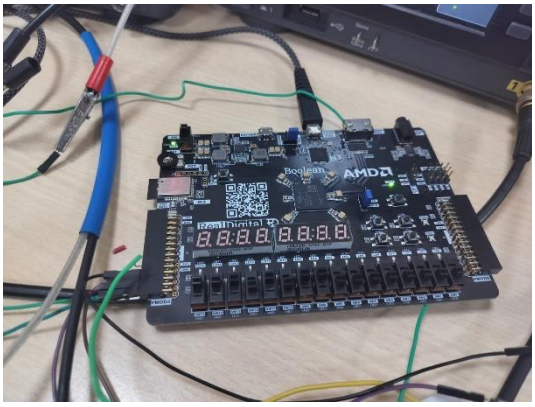


Fig. 45. Boolean board


Fig. 46. Setup of oscilloscope

ACKNOWLEDGMENT

I extend my sincere gratitude Prof. Abhishek Srivastava for entrusting me with this remarkable project, which offered an invaluable opportunity for learning. Furthermore, I appreciate the invaluable guidance provided by the TAs throughout the duration of this project.

REFERENCES

[1] Digital logic and computer design by Morris mano.

[2] Class Notes and Tutorial slides.