VLSI COURSE PROJECT

Garima Mittal
INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD, INDIA
garima.mittal@students.iiit.ac.in

***Abstract*—This paper presents the design and implementation of a 4-bit carry-lookahead adder (CLA) in 180nm CMOS technology. Optimized for reduced delay, the CLA adder is designed using static CMOS logic and validated through NGSPICE simulations. Key modules include propagate and generate blocks, a carry-lookahead circuit, and an output sum buffer, all verified post-layout in MAGIC to meet timing requirements. The design was also implemented on FPGA to confirm functionality, with results demonstrating reliable operation at the targeted clock frequency.**

***Index Terms*—carry-lookahead adder, propagate, generate, NGSPICE simulation, MAGIC layout, FPGA implementation, Verilog**

I. Introduction

High-speed adders are essential in digital systems, with the carry lookahead adder (CLA) being a widely used design for its efficient carry propagation mechanism. This project implements a 4-bit CLA in 180nm CMOS technology, aiming to minimize delay by carefully evaluating various VLSI logic styles, including static CMOS, dynamic, and mixed logic, to determine the optimal approach. The CLA design includes propagate and generate logic to precompute carry signals, a lookahead circuit for parallel carry generation, and a sum block buffered to meet drive requirements. Each module was simulated in NGSPICE to verify timing and functionality, followed by layout and post-layout verification in MAGIC to assess parasitic effects. The design was further validated through FPGA implementation, confirming reliable operation at high clock speeds. This systematic approach demonstrates the CLA's efficiency in reducing delay, meeting timing targets essential for high-performance digital circuits.

II. D-FLIPFLOP

The designed **D flip-flop** uses ***True Single-Phase Clock (TSPC)*** logic with 11 transistors, optimized for ***compactness*** and ***high speed***. It operates with three inputs: ***D (data input)***, ***clk (clock signal)***, and ***vdd (power supply)***, producing an output ***Q***. The circuit works dynamically, where the ***clock signal*** governs data flow and storage. During the clock's ***high phase***, the input data propagates through intermediate nodes (***X, Y, Z***), while during the ***low phase***, the stored charge maintains the output state through a ***static CMOS inverter***. This ***edge-triggered design*** ensures synchronized data storage with minimal delay and transistor count.

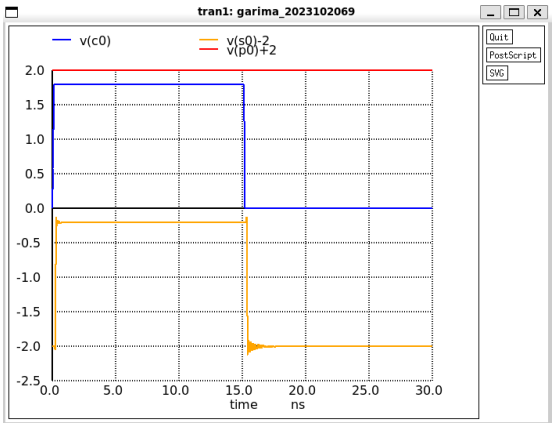


Fig. 1. DFF.

III. STRUCTURE OF ADDER

4-bit Carry Look-Ahead Adder Design

The 4-bit Carry Look-Ahead Adder is implemented using basic CMOS logic gates such as AND, OR, and XOR. This design improves the delay observed in a conventional carry ripple adder, which requires the carry output from the previous bit to compute the sum of the next bit. In contrast, the carry look-ahead adder leverages pre-defined sub-circuits, namely **Propagate** and **Generate**, which depend solely on the input bits A and B.

This circuit is based on usage of different gates like NAND, NOR, INVERTER and XOR, AND, OR made using these basic gates. Following are the CMOS implementation of the gates used in the circuit:

AND GATE

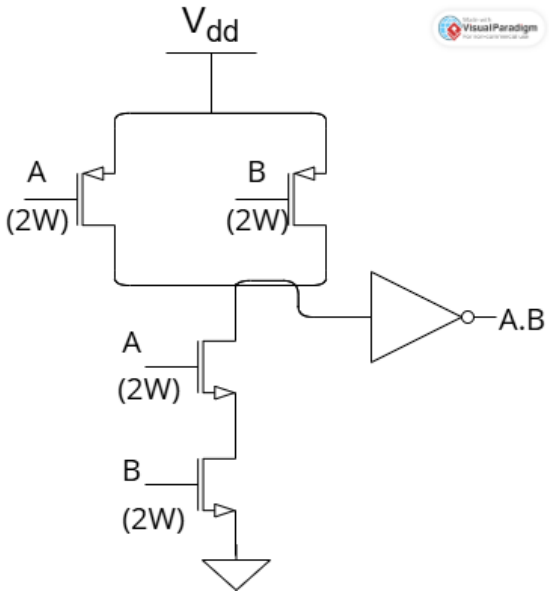


Fig. 2. AND.

OR GATE

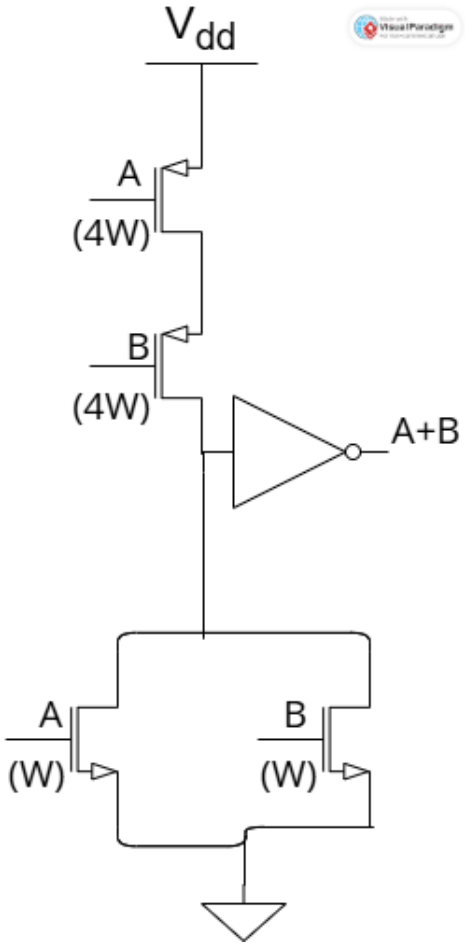


Fig. 3. OR.

INVERTER

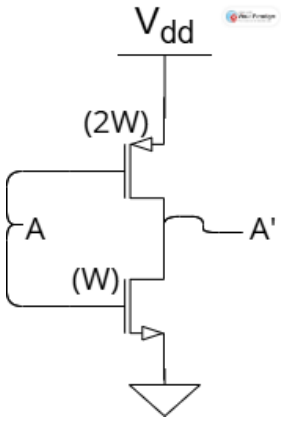


Fig. 4. INVERTER.

XOR GATE

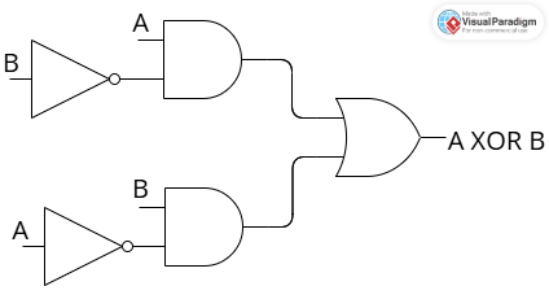


Fig. 5. XOR.

Propagate and Generate Sub-Circuits

The sub-circuits are defined as follows:

- **Propagate**: The propagate signal determines whether the carry will propagate to the next stage. It is calculated as the XOR of the input bits:

$$p_i = a_i \oplus b_i$$

- **Generate**: The generate signal determines whether a carry will be generated in the current stage. It is calculated as the AND of the input bits:

$$g_i = a_i \cdot b_i$$

Carry Computation

Using the propagate and generate sub-circuits, the carry for the next bit can be computed directly from the input bits, eliminating the dependency on the carry from the previous stage. This results in a significant improvement in delay compared to the carry ripple adder.

The carry required to compute the sum of the subsequent bits can be efficiently determined using the formula below:

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

Thus, leveraging the carry look-ahead methodology, the proposed adder structure computes all bits simultaneously. Following are the circuits for individual carry circuits:

CARRY 1

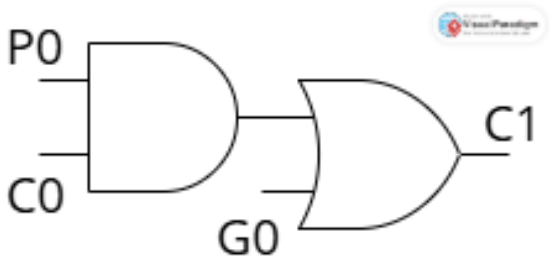


Fig. 6. CARRY-1.

CARRY 2

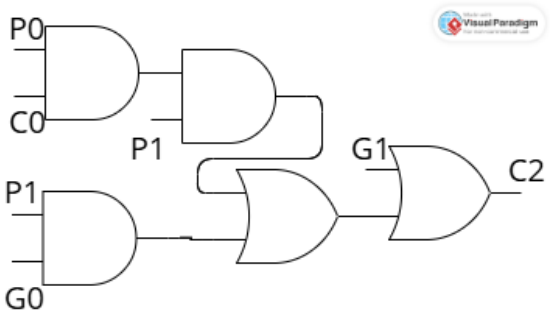


Fig. 7. CARRY-2.

CARRY 3

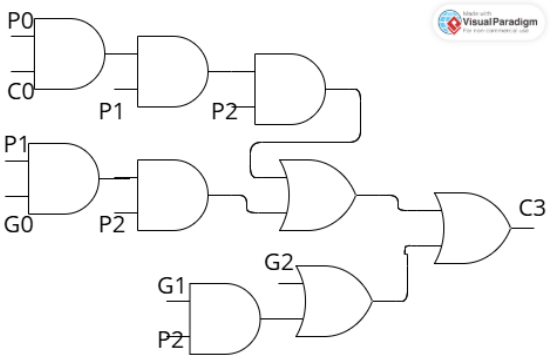


Fig. 8. CARRY-3.

CARRY 4

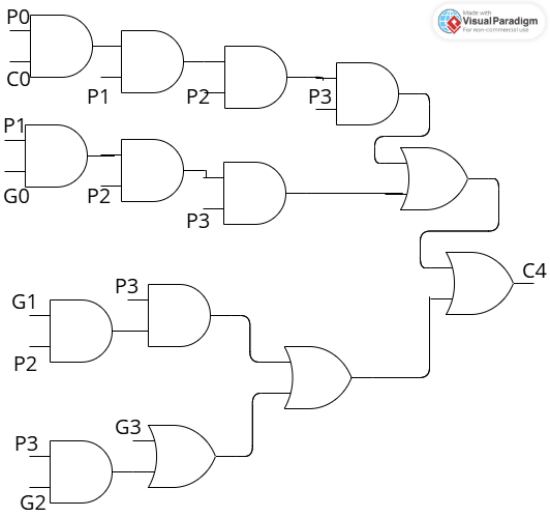


Fig. 9. CARRY-4.

SUM COMPUTATION

After all the carry are computed, the sum is the xor of Pi and Ci ie

$$S_i = P_i \oplus C_i$$

IV. TOPOLOGY AND SIZING

GATES	PMOS	NMOS
AND	2W	2W
OR	4W	W
INVERTER	2W	W
DFF	4W, 2W	W, 2W

TABLE I
SIZING

Above are the sizings of different gates:
INVERTER is minimum sized- 2W, W
In AND gate, PMOS=2W, NMOS = 2W
In OR gate, Pmos = 4W, NMOS = W
In DFF the topology is different, there are 3 cascaded levels, for the first level 2 PMOS are in series with one NMOS so the sizing is 4W,W while in other two levels there is one PMOS and two NMOS in series so sizing is 2W,2W.

V. NGSPICE RESULTS:

A. D FLIP FLOP

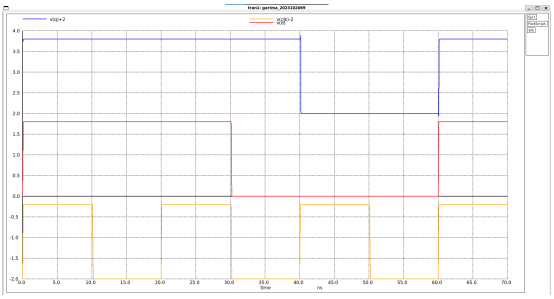


Fig. 10. dff waveform.

B. PROPAGATE AND GENERATE BLOCK

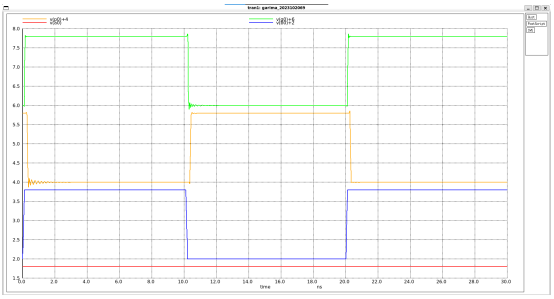


Fig. 11. waveform.

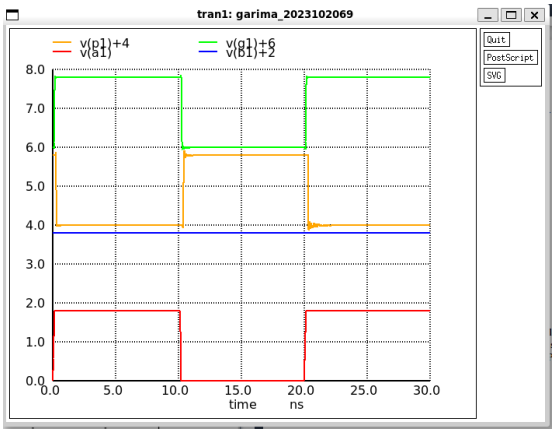


Fig. 12. waveform.

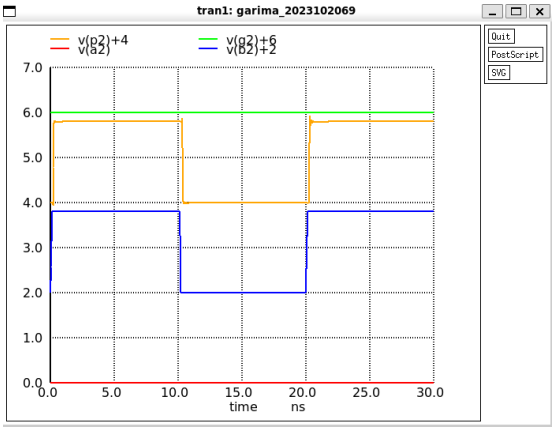


Fig. 13. waveform.

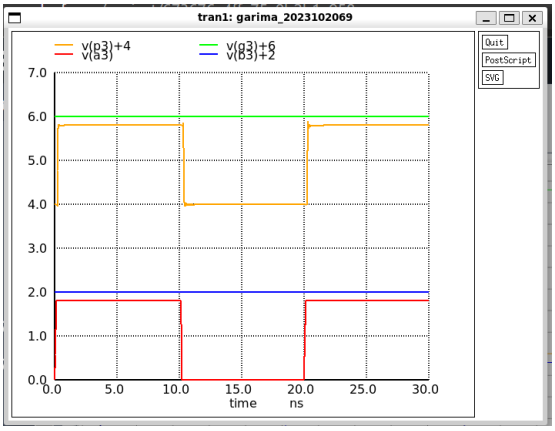


Fig. 14. waveform.

C. CARRY BLOCK

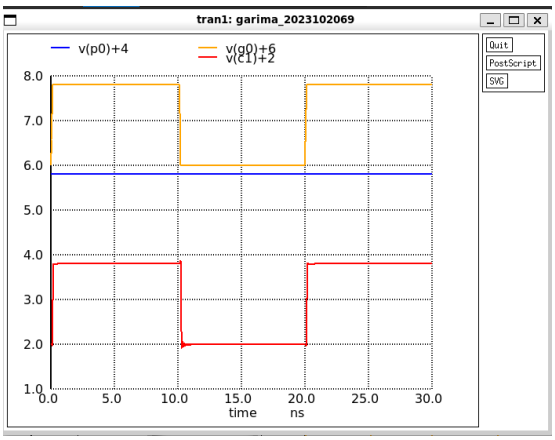


Fig. 15. waveform.

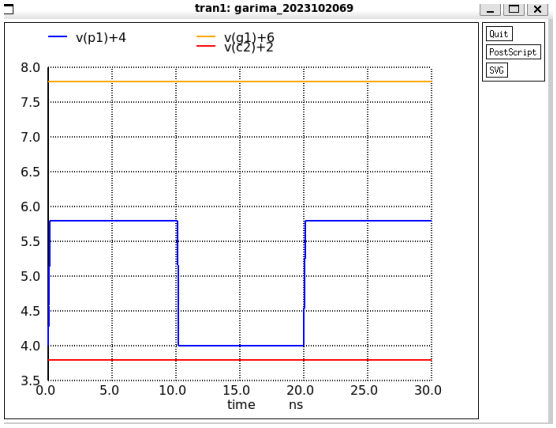


Fig. 16. waveform.

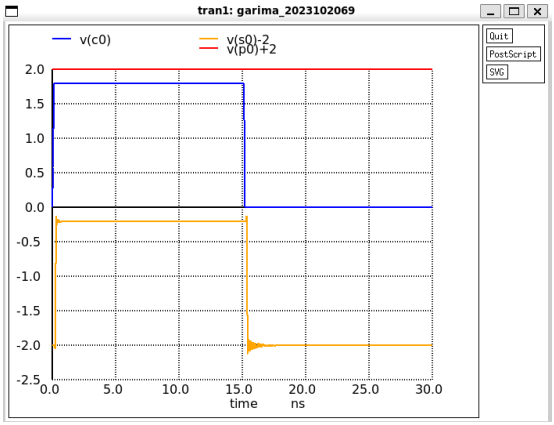


Fig. 19. waveform.

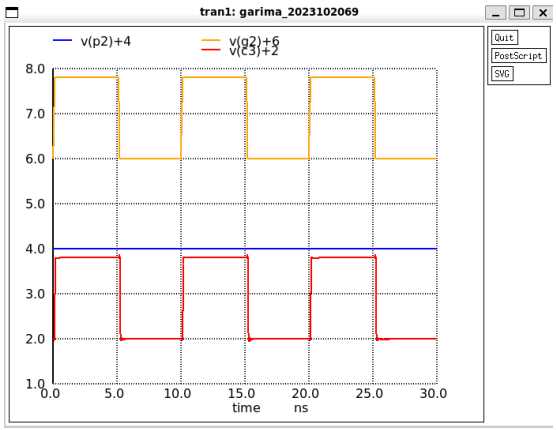


Fig. 17. waveform.

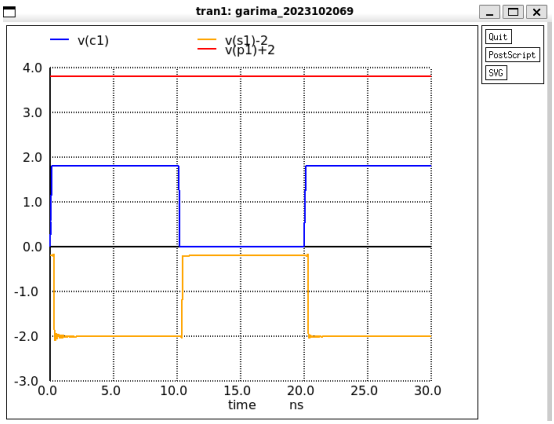


Fig. 20. waveform

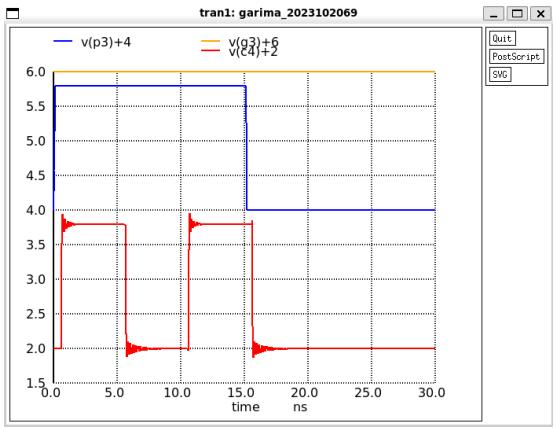


Fig. 18. waveform.

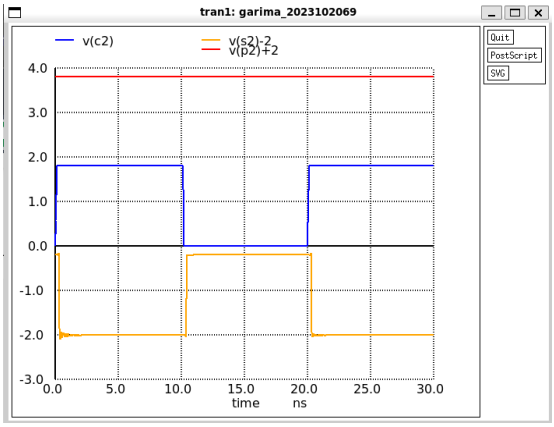


Fig. 21. waveform

D. SUM BLOCK

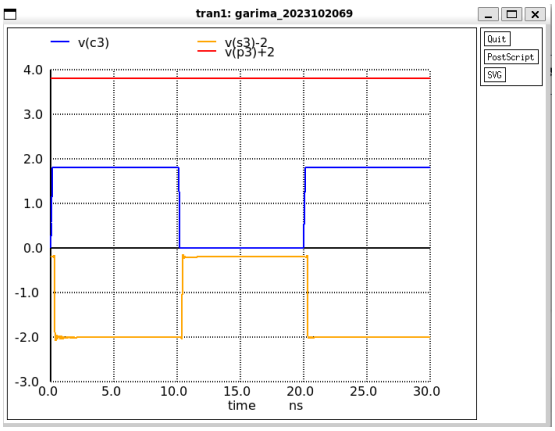


Fig. 22. waveform

E. ADDER

The adder circuit's functionality is analyzed below.

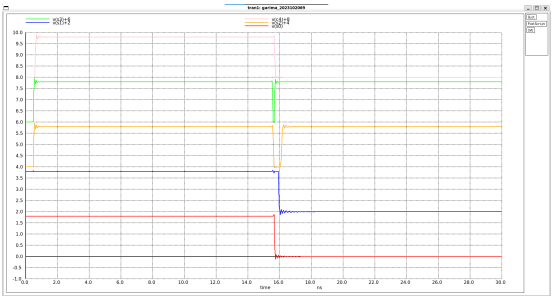


Fig. 23. Waveform for adder circuit.

VI. DELAY TIME IN D FLIP FLOP

For setup time, I brought the rise of input *D* closer and closer to the clock's rising edge to observe where the output *Q* changes. When the rising clock edge is at 5ns and the rising input edge is at 4.98ns, the output shows a slight incorrect value on the lowering edge, as shown below.

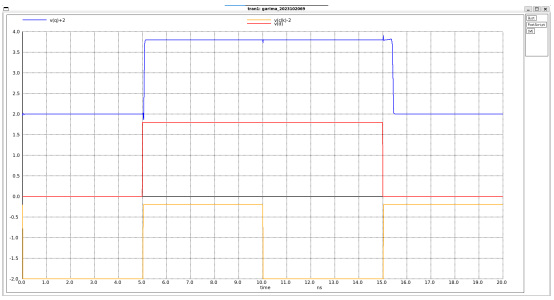


Fig. 24. Setup time for the D flip-flop.

Thus, the setup time of the D flip-flop is 0.02ns. Since the TSPC logic is used, the hold time ideally should be 0. But during my calculations I got a hold time of 15ps. As after this time my output comes out to be correct but in this time window is is coming the opposite.

As for the clock-to-Q delay, the value was found to be 75 picoseconds, as shown in the following plot.

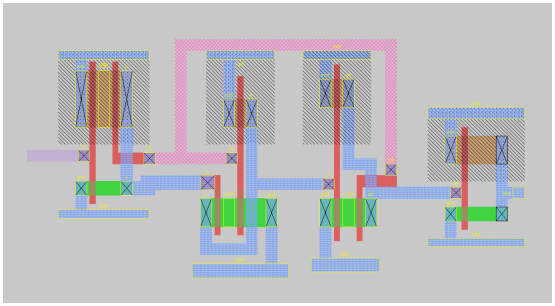
```
Measurements for Transient Analysis

tpd_rise            =  4.461176e-11 targ=  5.059612e-09 trig=  5.015000e-09
tpd_fall            =  1.063610e-10 targ=  1.512136e-08 trig=  1.501500e-08
total_prop_delay    =  7.54864e-11
```

Fig. 25. Clock-to-Q delay measurement.

VII. MAGIC LAYOUT

A. D FLIP FLOP



B. PROPAGATE AND GENERATE

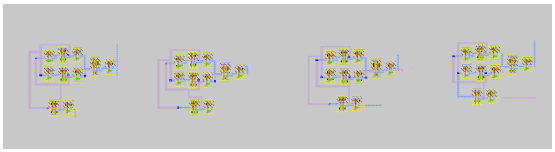


Fig. 26. Propagate and Generate blocks

C. CARRY BLOCKS

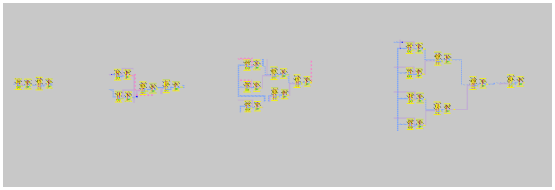


Fig. 27. Different carry blocks

D. SUM BLOCKS

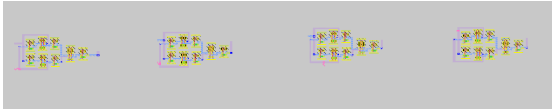


Fig. 28. Different sum blocks

E. ADDER

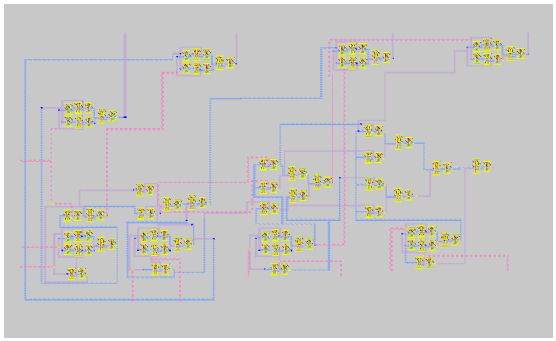


Fig. 29. adder

F. FULL ADDER

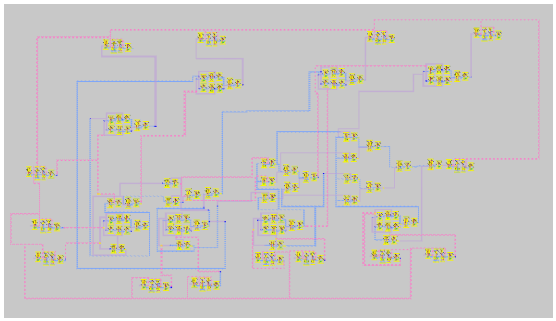


Fig. 30. full adder

VIII. POST NGSPICE SIMULATIONS

A. D FLIPFLOP

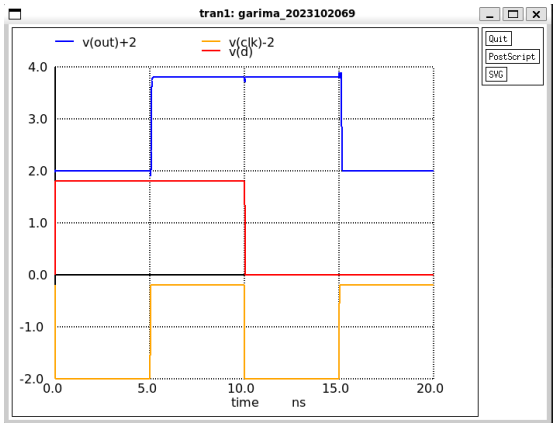


Fig. 31. dff

B. PROPAGATE GENERATE

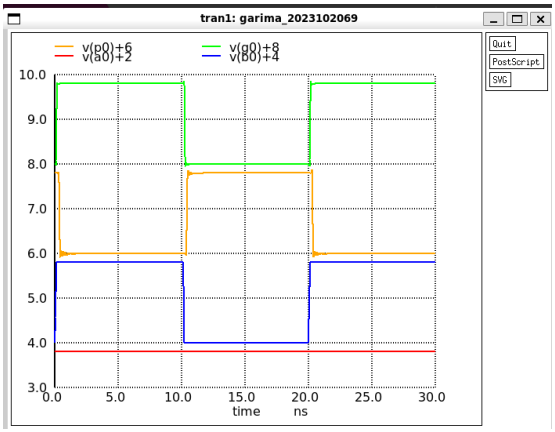


Fig. 32. Different propgen blocks

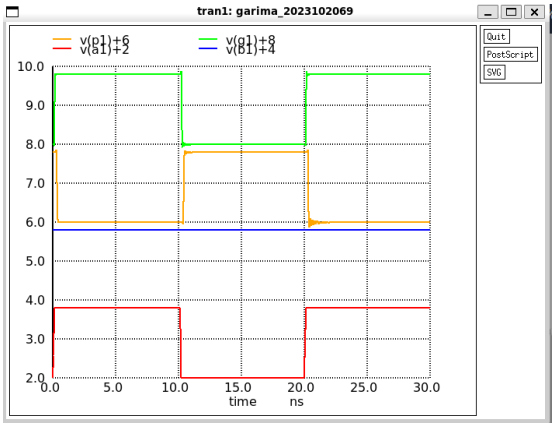


Fig. 33. Different propgen blocks

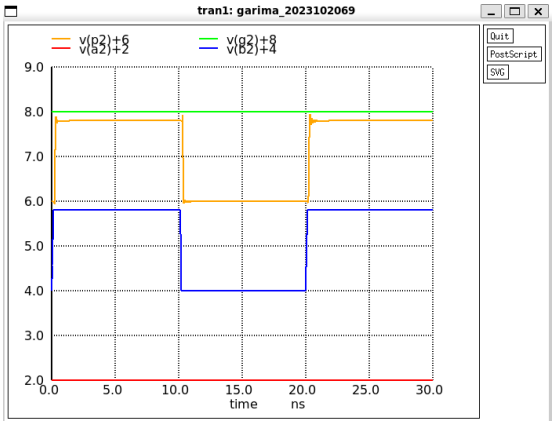


Fig. 34. Different propgen blocks

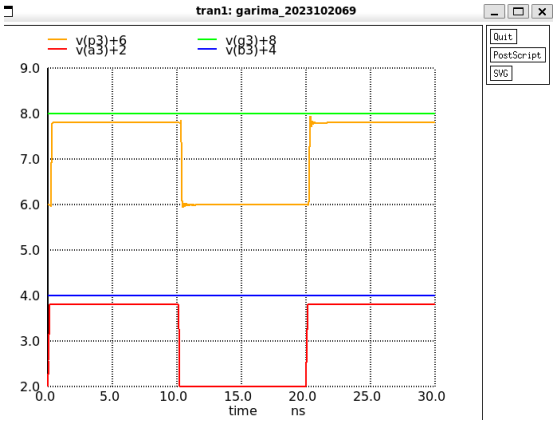


Fig. 35. Different propgen blocks

C. CARRY BLOCKS

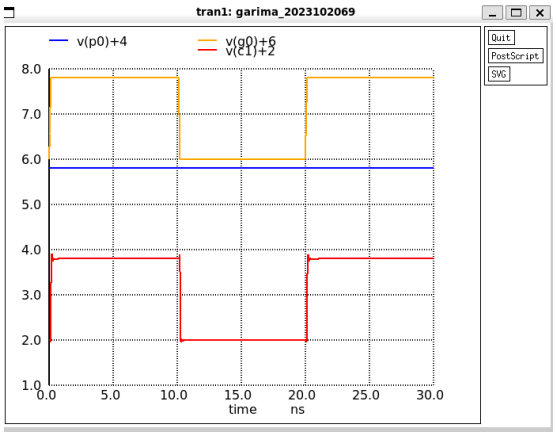


Fig. 36. Different carry blocks

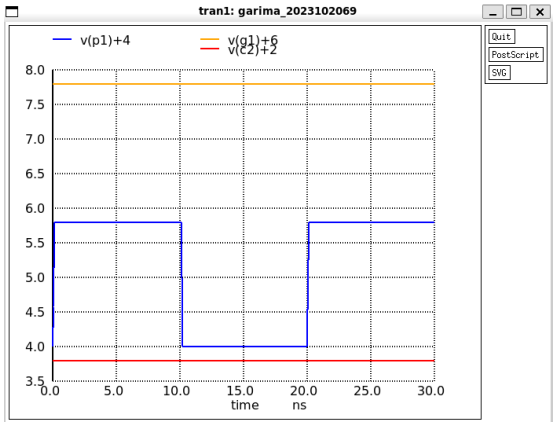


Fig. 37. Different carry blocks

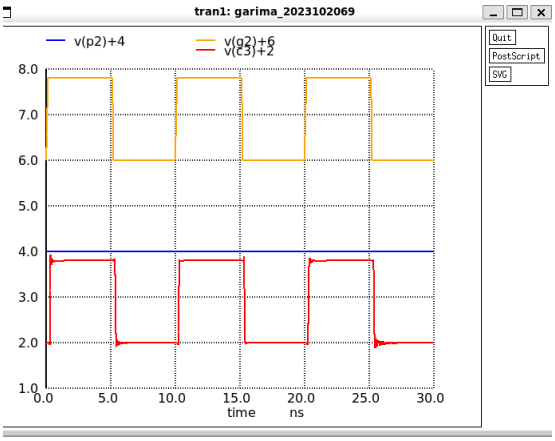


Fig. 38. Different carry blocks

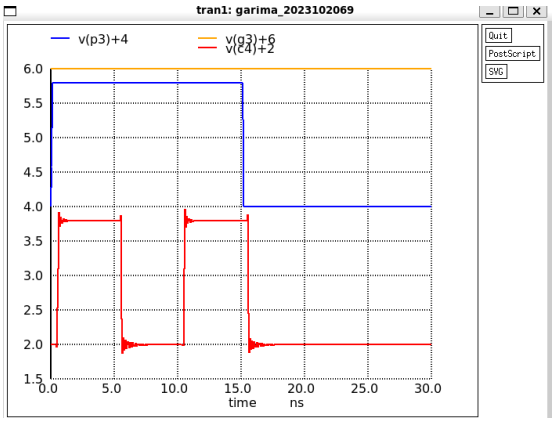


Fig. 39. Different carry blocks

D. SUM BLOCKS

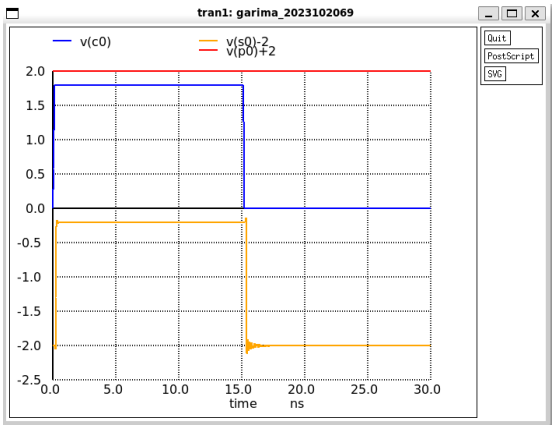


Fig. 40. Different sum blocks

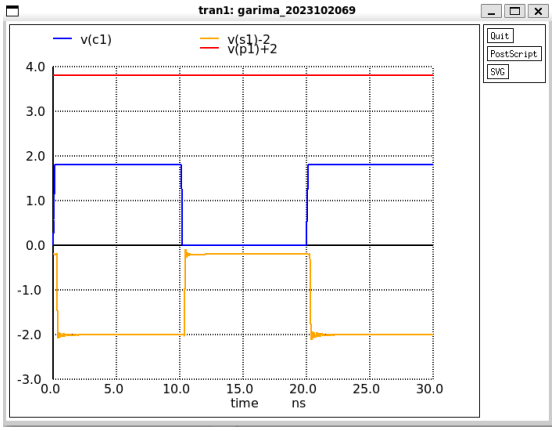


Fig. 41. Different sum blocks

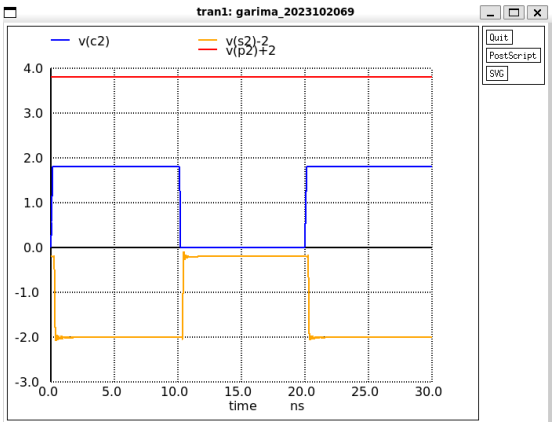


Fig. 42. Different sum blocks

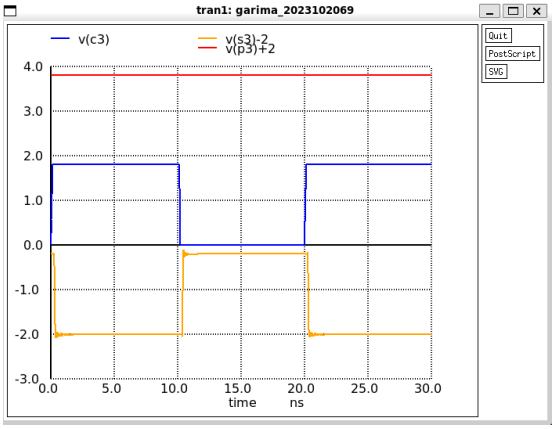


Fig. 43. Different sum blocks

E. ADDER

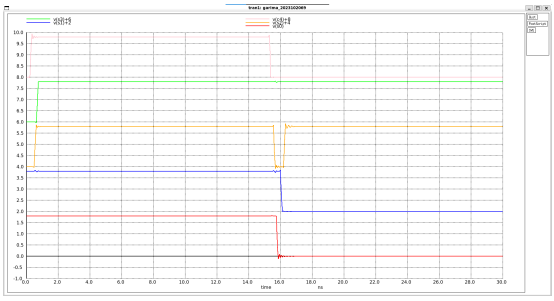


Fig. 44. adder

F. ADDER WITH DFF

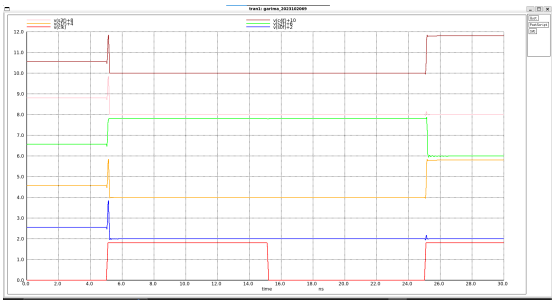


Fig. 45. full adder

IX. STICK DIAGRAMS

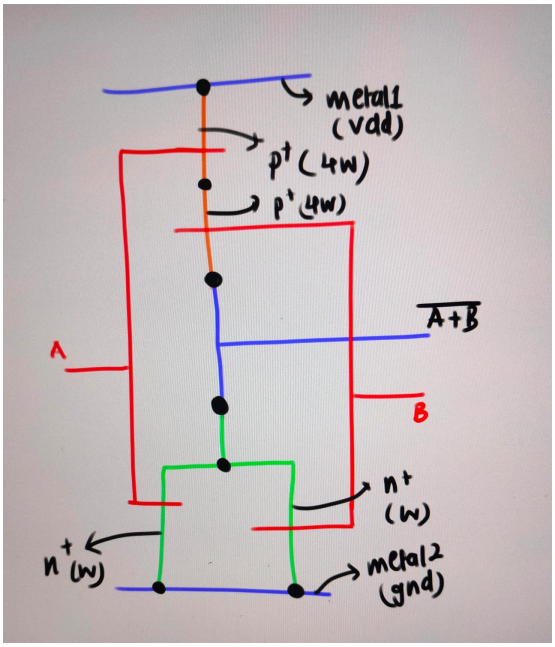


Fig. 46. NAND

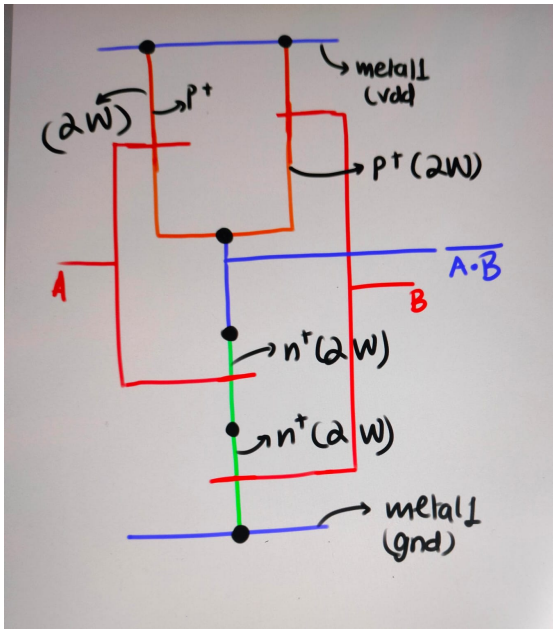


Fig. 47. NOR

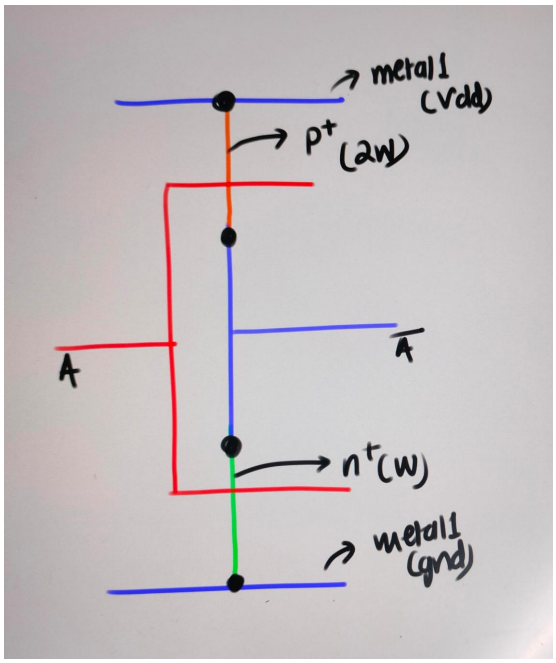


Fig. 48. INVERTER

These are the 3 unique gates I used for my simulation NAND, NOR, INVERTER. OR, AND are made attaching the inverter to NOR, NAND respectively and XOR gate is the combination of all the three unique gates as it is shown in the circuit diagram given before.

X. DELAY OF ADDER

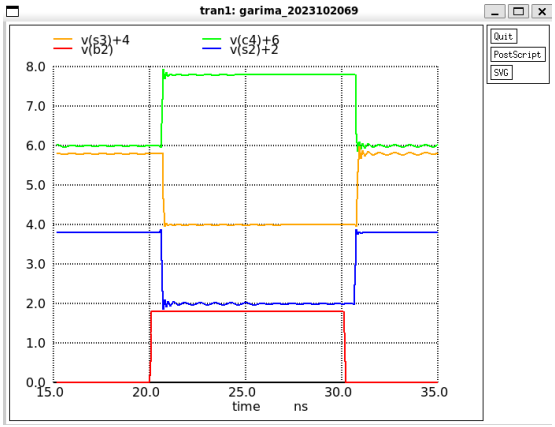


Fig. 49. waveform

The above figure is worst case of delay in post adder. My worst case delay came out to be 0.646ns.

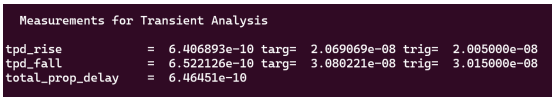


Fig. 50. waveform

XI. COMPARING PRE AND POST LAYOUT

QUANTITY	PRE LAYOUT	POST LAYOUT
SETUP TIME	20ps	20ps
HOLD TIME	14ps	15ps
CLK TO Q DELAY	75ps	89ps
ADDER DELAY	646ps	646ps
MAX CLOCK FREQUENCY	1.35×10^9 Hz	1.32×10^9 Hz

TABLE II
COMPARISON

XII. FLOOR PLAN

Following numbers are number of boxes where each box is of length 0.09 micro*0.09 micro.
Propagate generate block - 1250*315
Carry block - 1594*350
Sum block - 1202*209
DFF = 173*89
total layout - 1856*974

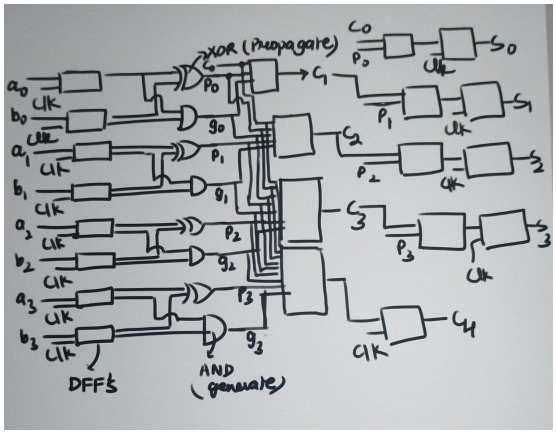


Fig. 51. floor plan

XIII. VERILOG

A. D FLIP FLOP

```
VCD info: dumpfile dff.vcd opened for output.
At time                    0, d = 0, q = x
At time                    5, d = 0, q = 0
At time                   10, d = 1, q = 0
At time                   15, d = 1, q = 1
At time                   25, d = 0, q = 0
At time                   35, d = 1, q = 1
At time                   45, d = 0, q = 0
At time                   55, d = 1, q = 1
```

Fig. 52. terminal

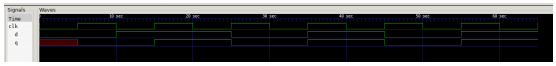


Fig. 53. dff

B. ADDER

```
Time = 0, A = 0000, B = 0000, Cin = 0, Sum = 0000, Cout = 0
Time = 10, A = 0101, B = 0011, Cin = 0, Sum = 1000, Cout = 0
Time = 20, A = 1111, B = 0001, Cin = 1, Sum = 0001, Cout = 1
Time = 30, A = 1010, B = 0101, Cin = 0, Sum = 1111, Cout = 0
Time = 40, A = 1111, B = 1111, Cin = 1, Sum = 1111, Cout = 1
```

Fig. 54. terminal

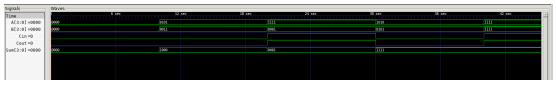


Fig. 55. adder

C. ADDER WITH DFF

```
Time =     5 | A = 0000 | B = 0000 | Cin = 0 | Sum = xxxx | Cout = x
Time =    10 | A = 0001 | B = 0001 | Cin = 0 | Sum = xxxx | Cout = x
Time =    15 | A = 0001 | B = 0001 | Cin = 0 | Sum = 0000 | Cout = 0
Time =    20 | A = 0010 | B = 0010 | Cin = 0 | Sum = 0000 | Cout = 0
Time =    25 | A = 0010 | B = 0010 | Cin = 0 | Sum = 0010 | Cout = 0
Time =    30 | A = 0100 | B = 0100 | Cin = 0 | Sum = 0010 | Cout = 0
Time =    35 | A = 0100 | B = 0100 | Cin = 0 | Sum = 0100 | Cout = 0
Time =    40 | A = 1000 | B = 1000 | Cin = 0 | Sum = 0100 | Cout = 0
Time =    45 | A = 1000 | B = 1000 | Cin = 0 | Sum = 1000 | Cout = 0
Time =    50 | A = 1111 | B = 1111 | Cin = 1 | Sum = 1000 | Cout = 0
Time =    55 | A = 1111 | B = 1111 | Cin = 1 | Sum = 0001 | Cout = 1
Time =    60 | A = 1010 | B = 0101 | Cin = 1 | Sum = 0001 | Cout = 1
Time =    65 | A = 1010 | B = 0101 | Cin = 1 | Sum = 1111 | Cout = 1
Time =    70 | A = 0110 | B = 0011 | Cin = 0 | Sum = 1111 | Cout = 1
Time =    75 | A = 0110 | B = 0011 | Cin = 0 | Sum = 1111 | Cout = 0
Time =    80 | A = 1100 | B = 0011 | Cin = 1 | Sum = 1111 | Cout = 0
Time =    85 | A = 1100 | B = 0011 | Cin = 1 | Sum = 1010 | Cout = 0
```

Fig. 56. terminal

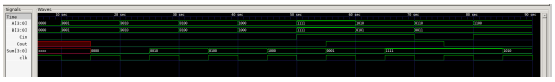


Fig. 57. full adder

XIV. FPGA

For the working of FPGA, we require a Verilog HDL code which when put in Vivado and constraint file gives output on a boolean board and can give an oscilloscope output when connected through wires to the DSO.

Following are through an board and oscilloscope outcomes:

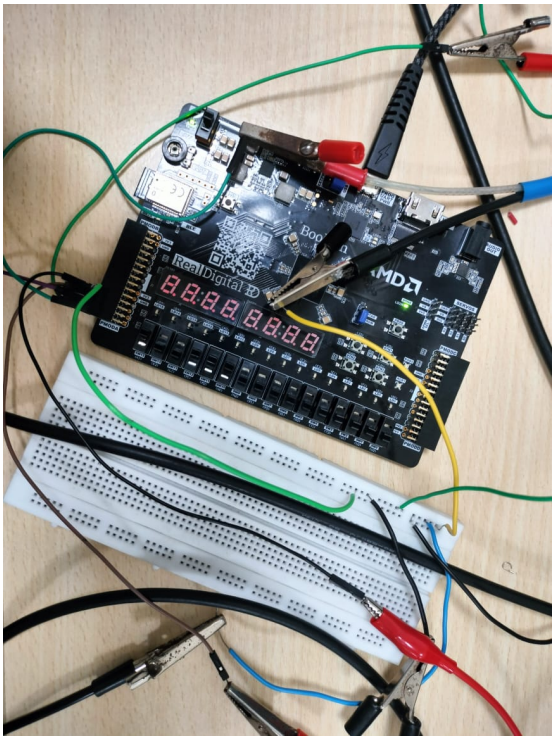


Fig. 58. setup of DSO

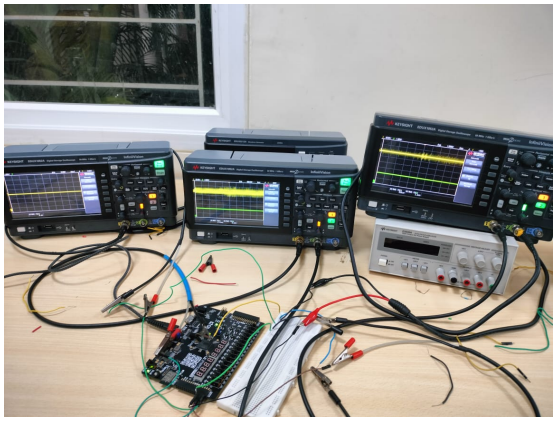


Fig. 59. DSO output

The above picture shows the output of do in the order of c4,s3,s2,s1, and s0 from left to right in the dso. The input for this result is 1001, 1100, and the output should be 10101, as seen in the picture.

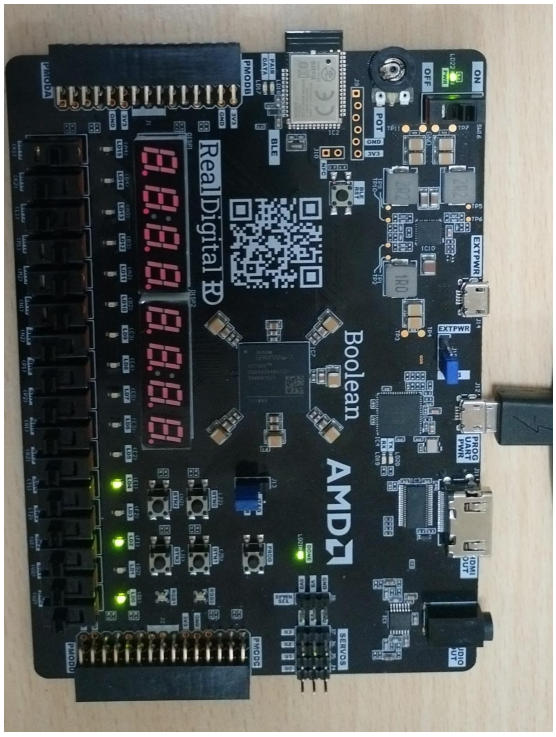


Fig. 60. BOARD output

The inputs to the above output are 1001, and 1100 and the output should be 10101.

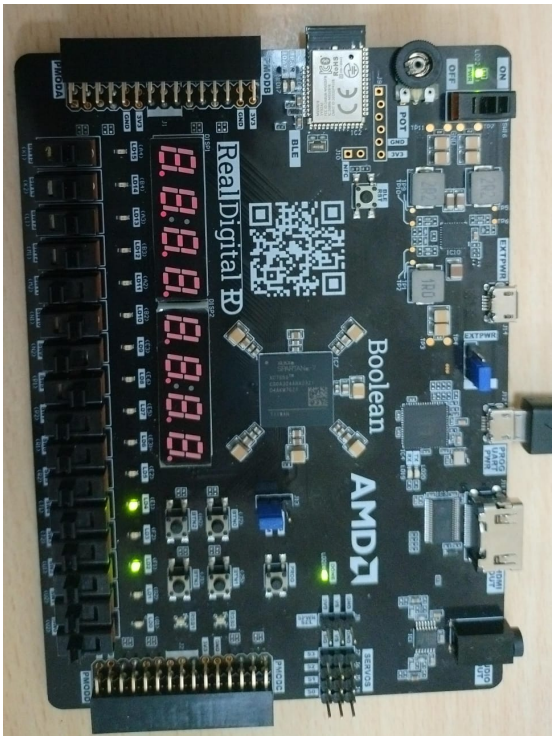


Fig. 61. Board output

The inputs for the above output are 1111, and 0101 and the output should be 10100 as shown.

XV. ACKNOWLEDGEMENT

I sincerely express my gratitude to all the teaching assistants of the VLSID course and Dr. Abhishek Srivastav for their guidance and support throughout this project. Designing a 4-bit Carry Look-Ahead Adder has enriched my understanding of digital circuits and VLSI design. I appreciate the resources and tools provided, which were instrumental in completing this project.

XVI. REFERENCES

REFERENCES

[1] International Journal of Recent Technology and Engineering (IJRTE), ISSN: 2277-3878 (Online), Volume-8, Issue-1, May 2019.
[2] *Computer Architecture and Organisation.*
[3] *Digital Logic and Computer Design.*
[4] International Journal of Engineering Trends and Technology (IJETT), Volume 25, Number 3, July 2015, ISSN: 2231-5381.
[5] IJSRD - International Journal for Scientific Research & Development, Vol. 3, Issue 06, 2015, ISSN (online): 2321-0613.
[6] IEEE Journal of Solid-State Circuits, Vol. 24, No. 1, February 1989.
[7] IEEE Journal of Solid-State Circuits, Vol. 29, No. 6, June 1994.
[8] YouTube video, https://www.youtube.com/watch?v=3kygidqfXyE.
[9] *Carry Lookahead Adder*, Elprocus. Available at: https://www.elprocus.com/carry-look-ahead-adder/.
[10] *Carry Lookahead Adder Propagation Delay Calculation*, Electronics Stack Exchange. Available at: https://electronics.stackexchange.com/questions/495242/carry-look-ahead-adder-propagation-delay-calculation.
[11] *Carry-lookahead Adder*, Wikipedia. Available at: https://en.wikipedia.org/wiki/Carry-lookahead_adder.

[12] *How to Choose Device Sizing for a TSPC Edge Triggered DFF*, Edaboard Forum. Available at: https://www.edaboard.com/threads/how-to-choose-device-sizing-for-a-tspc-edge-triggered-dff.151010/.