VLSI COURSE PROJECT 2024

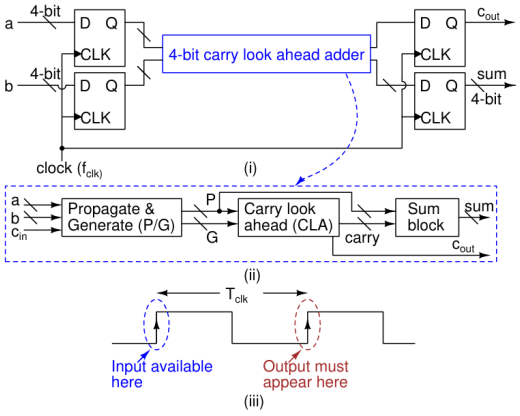
Ritama Sanyal
International Institute of Information Technology Hyderabad
2023112027
ritama.sanyal@research.iiit.ac.in

***Abstract*—This project entails designing a 4-bit Carry Look-Ahead (CLA) adder using 180 nm technology. The process includes defining the adder's structure, designing functional blocks (D-flip-flops, adder modules), and verifying functionality via NGSPICE simulations. MAGIC layout editor is used for layout creation, post-layout extraction, and schematic comparisons. The complete circuit's netlist is validated to determine delay and max clock speed. Additionally, Verilog HDL description, FPGA implementation, and oscilloscope-based hardware validation are required.**

I. Introduction

The 4-bit Carry Look-Ahead (CLA) adder is a high-speed adder design that minimizes delay by generating carry signals in advance, based on propagate and generate logic. This project aims to implement and evaluate a CLA adder using 180 nm technology, focusing on layout design, performance optimization, and functionality verification through simulations and hardware testing.

II. Provided Information



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

$$p_i = a_i \oplus b_i$$

$$g_i = a_i.b_i$$

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

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

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

$$sum_i = p_i \oplus c_i$$

Fig. 1. Provided Theory

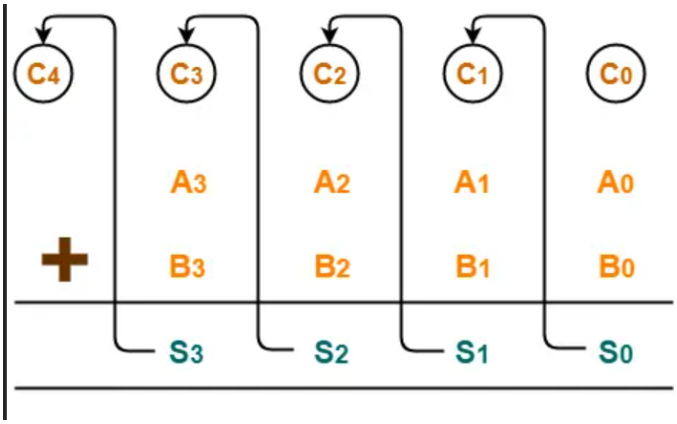


Fig. 2. Adding 2 4-bit numbers

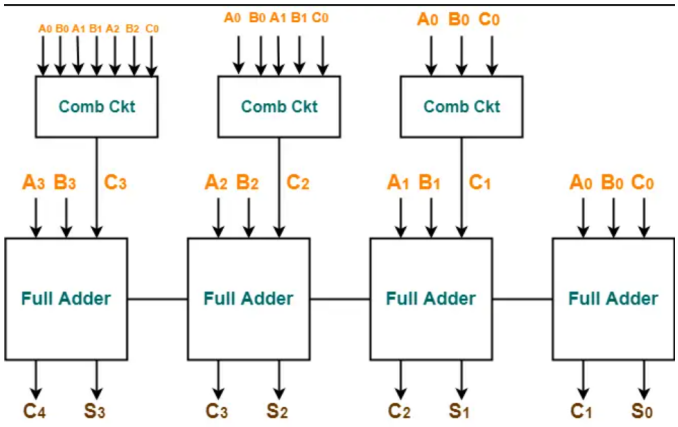


Fig. 3. Logic Diagram

III. Proposed Structure of Adder

From here, we have:

$$C_1 = C_0(A_0 \oplus B_0) + A_0B_0$$

$$C_2 = C_1(A_1 \oplus B_1) + A_1B_1$$

$$C_3 = C_2(A_2 \oplus B_2) + A_2B_2$$

$$C_4 = C_3(A_3 \oplus B_3) + A_3B_3$$

For simplicity, let:

- $G_i = A_iB_i$ where G is called the carry generator
- $P_i = A_i \oplus B_i$ where P is called the carry propagator

Then, rewriting the above equations, we have:

$$C_1 = C_0P_0 + G_0 \quad \ldots \quad (1)$$

$$C_2 = C_1P_1 + G_1 \quad \ldots \quad (2)$$

$$C_3 = C_2P_2 + G_2 \quad \ldots \quad (3)$$

$$C_4 = C_3P_3 + G_3 \quad \ldots \quad (4)$$

Now,

- Clearly, C_1, C_2, and C_3 are intermediate carry bits.
- Let us remove C_1, C_2, and C_3 from the right-hand side of every equation.
- Substituting (1) in (2), we get C_2 in terms of C_0.
- Then, substituting (2) in (3), we get C_3 in terms of C_0 and so on.

Finally, we have the following equations:

$$C_1 = C_0P_0 + G_0$$

$$C_2 = C_0P_0P_1 + G_0P_1 + G_1$$

$$C_3 = C_0P_0P_1P_2 + G_0P_1P_2 + G_1P_2 + G_2$$

$$C_4 = C_0P_0P_1P_2P_3 + G_0P_1P_2P_3 + G_1P_2P_3 + G_2P_3 + G_3$$

These equations show that the carry-in of any stage full adder depends only on:

- Bits being added in the previous stages
- Carry bit which was provided in the beginning

We have the carries of the form

$$C_{i+1} = P_i \cdot C_i + G_i = (A_i \oplus B_i) \cdot C_i + A_i \cdot B_i$$

The truth table will be as follows:

C_i	A_i	B_i	$A_i \oplus B_i$	$A_i + B_i$	$A_i \cdot B_i$	C_{i+1}	Type of carry
0	0	0	0	0	0	0	None
0	1	0	1	1	0	0	None
0	0	1	1	1	0	0	None
0	1	1	0	1	1	1	Generate
1	0	0	0	0	0	0	None
1	1	0	1	1	0	1	Propagate
1	0	1	1	1	0	1	Propagate
1	1	1	0	1	1	1	Generate/Propagate

It is seen that XOR and OR differ only when both A and B are 1, but in this case, the $i+1$-th carry will be independent of P as G is AND, which will be 1. Therefore, the $i+1$-th carry is given by OR of G and AND of P and i-th carry. Hence, we can replace the XOR operations in P with an OR operation in the carry lookahead calculation segment of the circuit.

$$C_{i+1} = P_i \cdot C_i + G_i = (A_i + B_i) \cdot C_i + A_i \cdot B_i$$

We can see that this carry lookahead design would require the use of AND, OR, and XOR functions. We know that complementary logic gates are inverting for CMOS static logic. Hence, it will be easier and more efficient to make NAND and NOR gates in static CMOS logic than AND and OR because they would require inverting the outputs of NAND and NOR gates, respectively.

We thereby modify the logic functions using DeMorgan's theorem to replace AND and OR with NAND and NOR wherever possible.

We take

$$P_i' = \overline{A_i + B_i}$$

and

$$G_i' = \overline{A_i \cdot B_i}$$

Now we have

$$C_1 = \overline{G_0' \cdot (P_0' + \overline{C_0})}$$

$$C_2 = \overline{G_1' \cdot (P_1' + G_0')} + (\overline{P_1' + P_0'} \cdot C_0)$$

$$C_3 = \overline{G_2' \cdot (P_2' + G_1')} + (\overline{P_2' + P_1'} \cdot \overline{G_0'(P_0' + \overline{C_0})})$$

$$C_4 = \overline{P_3' + P_2' \cdot P_1' + P_0'} + C_0 + \overline{G_3' \cdot (P_3' + G_2')} + (\overline{P_3' + P_2'} \cdot \overline{G_1'(P_1' + G_0')})$$

Using these equations, we can arrive at the following structure for the circuit.

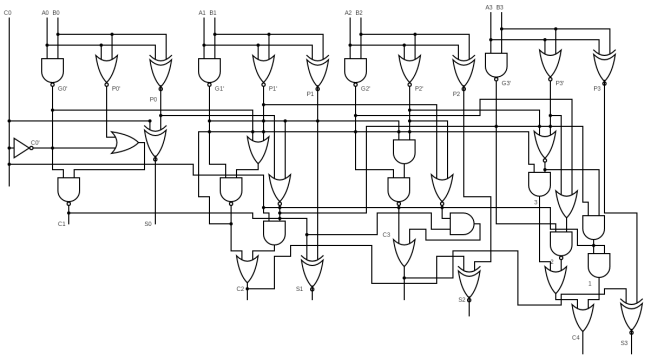


Fig. 4. Adder Structure (made using www.circuit-diagram.org)

IV. Topology and Sizing of TSPC D-Flip Flop

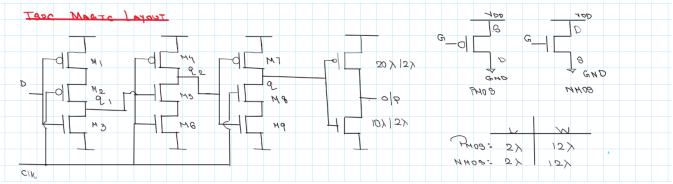
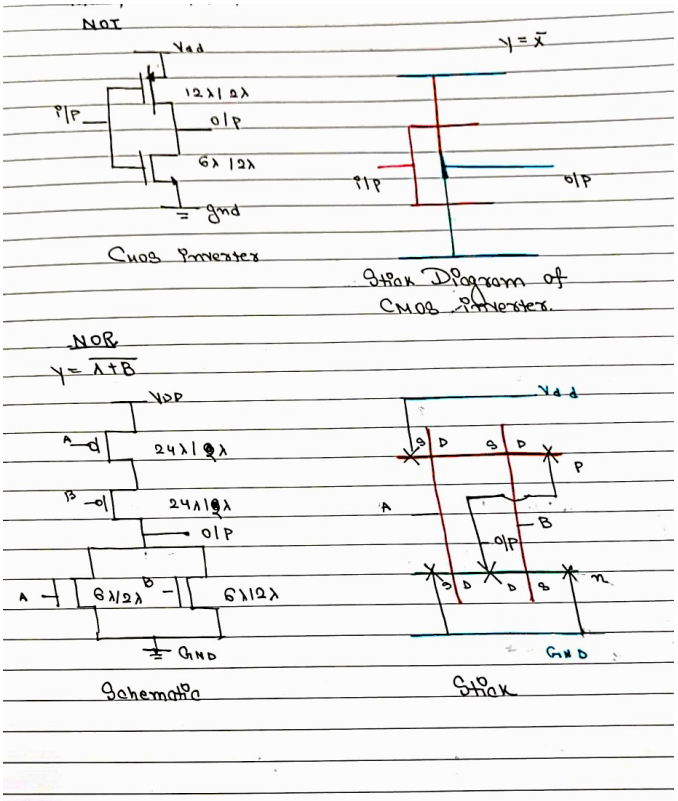


Fig. 5. D Flip Flop structure and Sizing

V. Stick Diagram and Sizing of Unique Gates

The sizing of unique gates, such as NAND, NOR, and inverters, involves selecting appropriate transistor widths to balance performance and power consumption while meeting specific design constraints like switching speed and output drive capability. Proper sizing ensures the gate operates efficiently within the desired parameters for delay, noise margin, and energy efficiency.Stick diagrams help in planning the arrangement of transistors, wires, and contacts using simple lines, often with color codes to denote different materials like polysilicon, metal layers, and diffusion regions.

Gates used in Adder and D Flip Flop:



VI. NGSPICE Simulation of Adder

A Carry Look-Ahead Adder (CLA) is an advanced type of adder circuit designed to enhance the speed of binary addition by addressing the delay problems inherent in simpler adders, such as the Ripple Carry Adder (RCA). In an RCA, each bit's sum and carry-out depend on the carry-in from the previous stage, causing a cumulative delay as the carry propagates sequentially through all bit positions. In contrast, the CLA adder significantly reduces this delay by computing the carry signals in parallel, leveraging a more complex but efficient carry generation mechanism. Instead of waiting for each bit's carry to propagate sequentially, the CLA uses generate (G) and propagate (P) signals to determine all carry bits in parallel, enabling faster addition. This parallel computation significantly enhances performance compared to a ripple carry adder, especially for large bit-widths. The adder used is combinational.

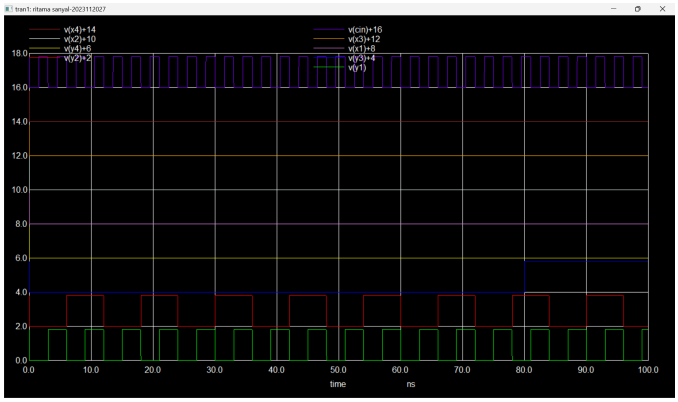


Fig. 6. Waveform of Input of Adder

VII. NGSPICE Simulation of D Flip Flop

A flip-flop is a fundamental digital memory circuit used to store one bit of data. It has two stable states, representing binary values 0 and 1, and can change states based on input signals. Flip-flops are used to store and maintain a binary state until directed to change by a control signal, such as a clock pulse. They are crucial components in various digital systems, including registers, counters, and memory devices, enabling the sequential storage and synchronization of data.

The True Single-Phase Clock (TSPC) D flip-flop is a high-speed, efficient flip-flop design that operates using a single clock phase, reducing clock distribution complexity and power consumption. It uses dynamic logic principles, where the storage elements are controlled by clocked transistors, allowing for reduced transistor count and faster switching. TSPC D flip-flops are widely used in high-performance and low-power applications, as they provide significant advantages in terms of speed and energy efficiency compared to traditional static flip-flop designs.

Setup time is the minimum time before the clock edge during which the data input must remain stable for proper capturing by a flip-flop. Violating this can lead to metastability and unreliable outputs. **Setup time of D flip-flop is 0.3**

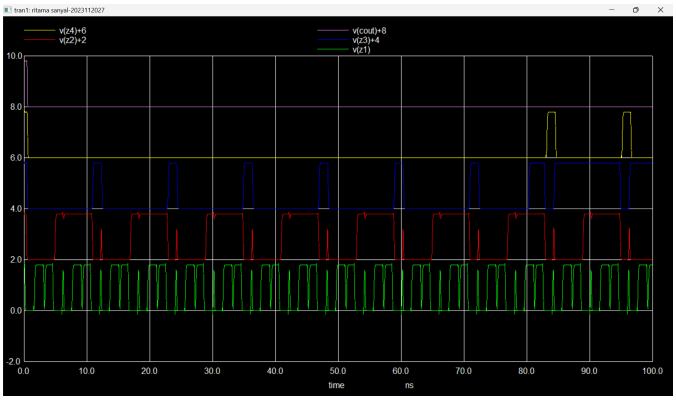


Fig. 7. Waveform of Output of Adder

```
  Measurements for Transient Analysis

tpdr1                =  7.718883e-08 targ=  8.320483e-08 trig=  6.016000e-09
tpdf1                =  5.266701e-10 targ=  5.306701e-10 trig=  4.000000e-12
tpd1                 =  3.88578e-08
```

Fig. 8. Delay Measurements of Adder

picoseconds.

Hold time is the minimum time after the clock edge during which the data input must remain stable to ensure it is properly captured by a flip-flop. Violating this can cause incorrect or unstable outputs. **Hold time is 0 seconds.**

VIII. MAGIC Layout of Structures used

Magic enables the design and visualization of the physical layout of ICs, representing how transistors, interconnects, and other components are fabricated on a chip.

IX. Layout of Entire Circuit

Module	Pre Layout	Post Layout
TSPC Flip Flop	641ns	810ns
Adder	0.388ns	5.1ns
Full Circuit	0.900ns	4.08ns

TABLE I
PERFORMANCE COMPARISON OF MODULES BEFORE AND AFTER LAYOUT.

The clock period is calculated using the formula:

$$t_{\text{clock}} = t_{\text{setup}} + t_{\text{clock-to-Q}} + t_{\text{propagation delay}}$$

The clock frequency can be calculated as:

$$f_{\text{clock}} = \frac{1}{t_{\text{clock}}}$$

Given:

$$t_{\text{clock}} = 815.1\,\text{ns} = 815.1 \times 10^{-9}\,\text{s}$$

Substituting the value:

$$f_{\text{clock}} = \frac{1}{815.1 \times 10^{-9}} \approx 1.226\,\text{MHz}$$

Therefore, the clock frequency is approximately **1.226 MHz**.

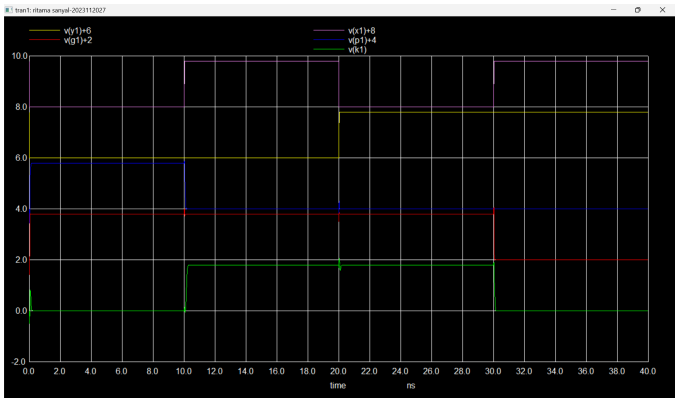


Fig. 9. Simulation of Adder module(Propagate and Generate Block)

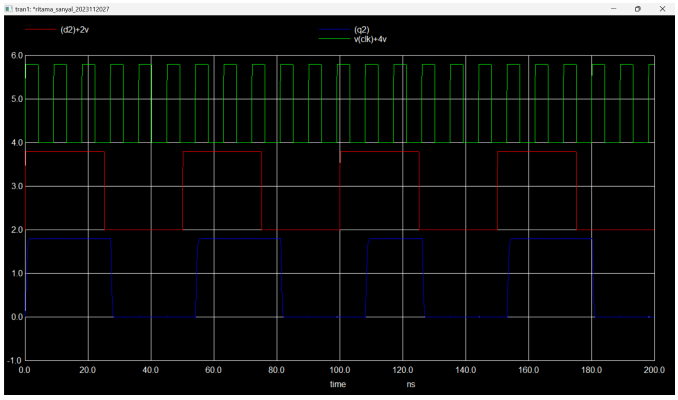


Fig. 10. Waveform of TSPC Flip Flop

X. VERILOG

XI. FPGA Implementation of Carry Lookahead Adder (CLA)

I have implemented a Carry Lookahead Adder (CLA) on an FPGA. The functionality of the adder was verified by performing the addition of two 4-bit binary numbers:

$$0101 + 1011$$

The result of the operation was:

$$\text{Sum} = 0000, \quad \text{Carry} = 1$$

This demonstrates the correct operation of the CLA design and validates the FPGA implementation.

XII. Citations

References

[1] Neil H. E. Weste and David Money Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, Pearson Education, 2010.

[2] Ivan E. Sutherland, Bob F. Sproull, and David L. Harris, *Designing Fast CMOS Circuits*, Addison-Wesley, 1999.

[3] Jan M. Rabaey, *Digital Integrated Circuits*, Prentice Hall, 1996.

[4] Samir Palnitkar, *Verilog HDL: A Guide to Digital Design and Synthesis, Second Edition*, Prentice Hall, 2003.

[5] Stephen Brown and Zvonko Vranesic, *Fundamentals of Digital Logic with Verilog Design*, McGraw-Hill, 2005.

```
  Measurements for Transient Analysis
delay                =  3.264524e-10 targ=  3.764524e-10 trig=  5.000000e-11
```

Fig. 11. Delay Measurements of TSPC Flip Flop

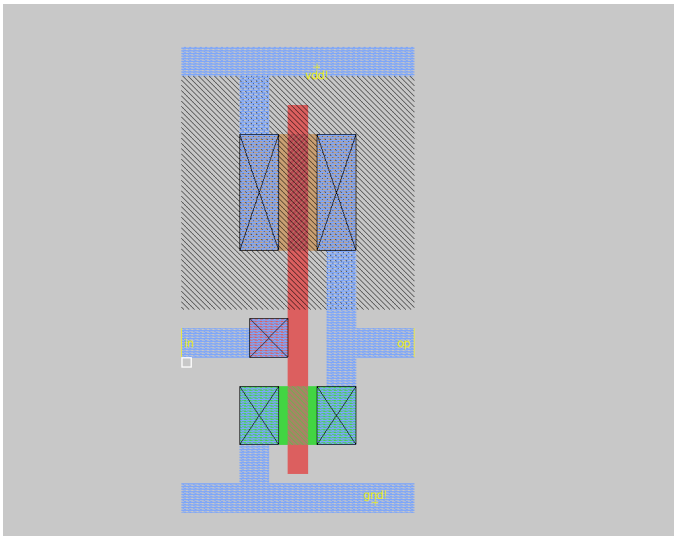


Fig. 12. Inverter

[6] J. Bhasker, *Verilog HDL Synthesis: A Practical Primer*, Star Galaxy Publishing, 2004.

[7] Patrik Larsson and Christer Svensson, "Impact of Clock Slope on True Single Phase Clocked (TSPC) CMOS Circuits," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 5, pp. 757–764, 1996.

[8] Jiren Yuan and Christer Svensson, "High-speed CMOS Circuit Technique," *IEEE Journal of Solid-State Circuits*, vol. 24, no. 1, pp. 62–70, 1997.

[9] Priyanka Sharma and Rajesh Mehra, "True Single Phase Clocking Based Flip-Flop Design Using Different Foundries," *International Journal of Electronics and Communication*, vol. 10, no. 3, pp. 50–55, 2021.

[10] Abhishek Shrivastav, Class Notes, 2024.

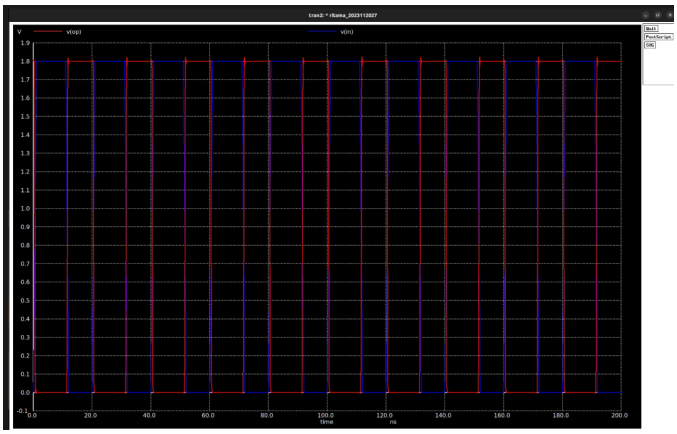


Fig. 13. Post layout of Inverter

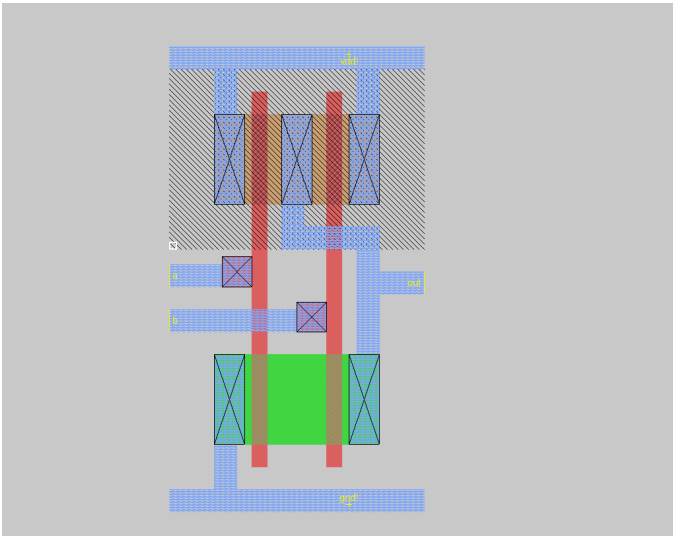


Fig. 14. NAND Gate

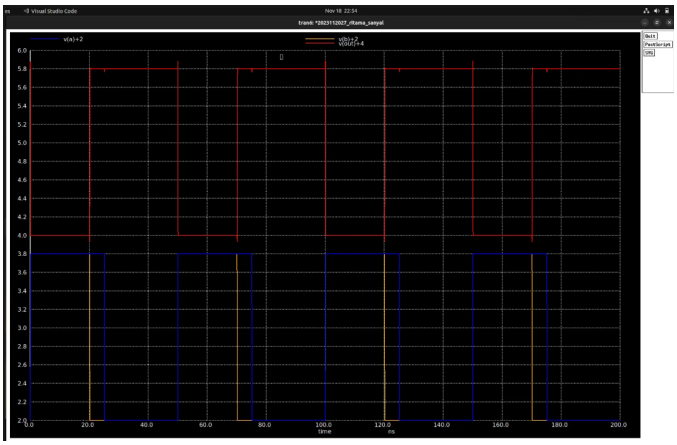


Fig. 15. Post Layout of NAND Gate

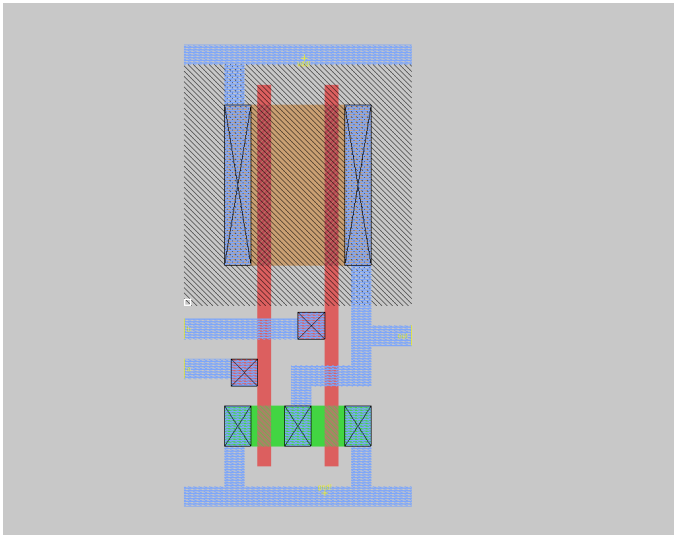


Fig. 16. NOR Gate

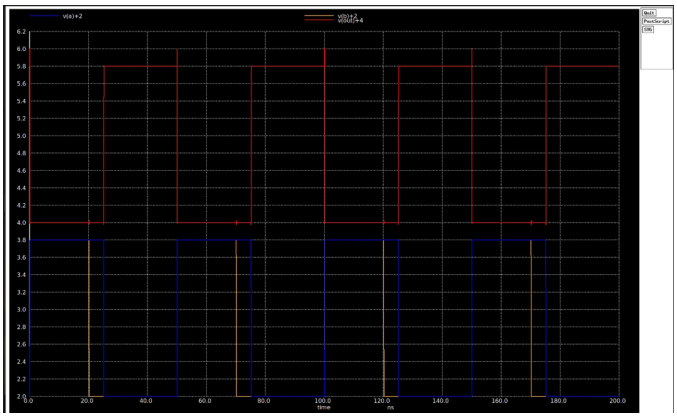


Fig. 17. Post Layout of NOR Gate

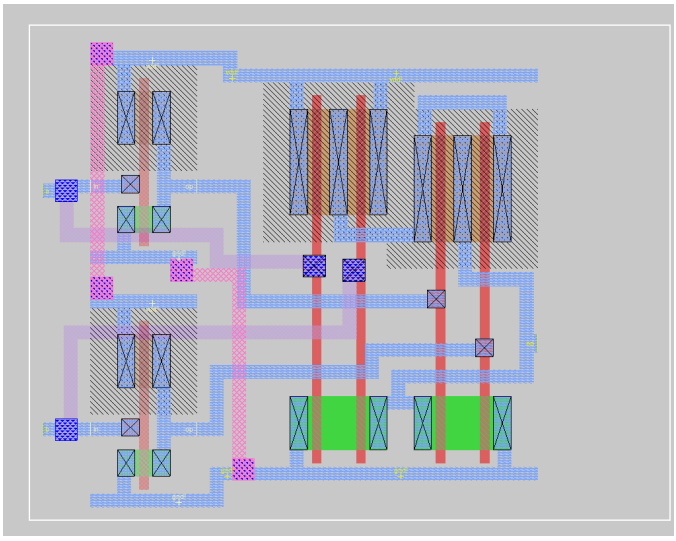


Fig. 18. XOR Gate

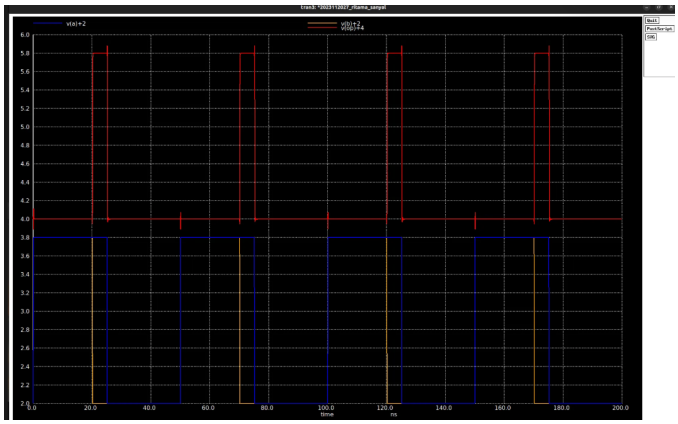


Fig. 19. Post Layout of XOR Gate

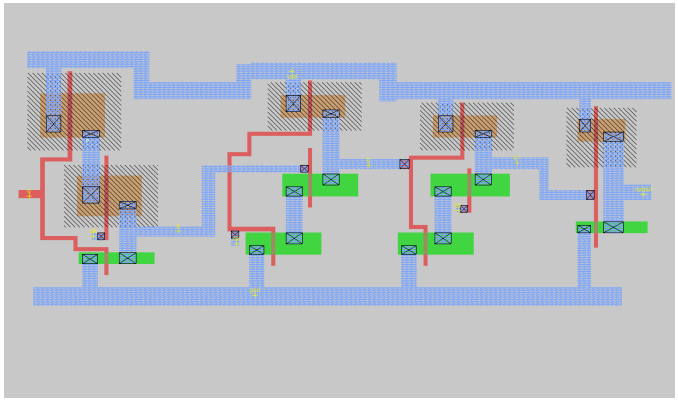


Fig. 20. TSPC FlipFlop

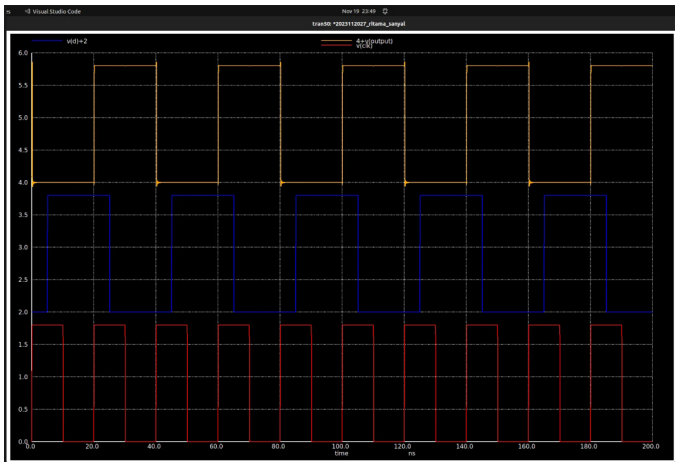


Fig. 21. Post layout of TSPC

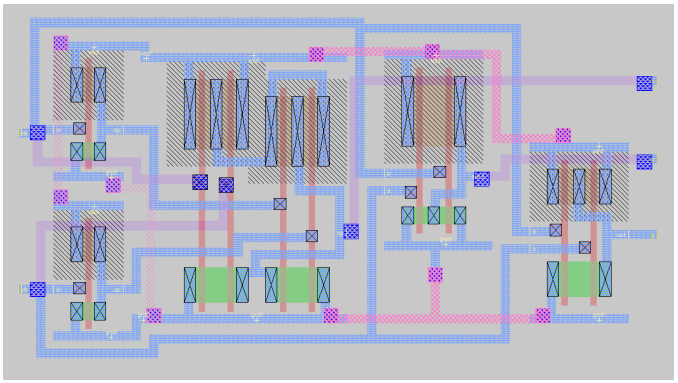


Fig. 22. Propagate and Generate Block

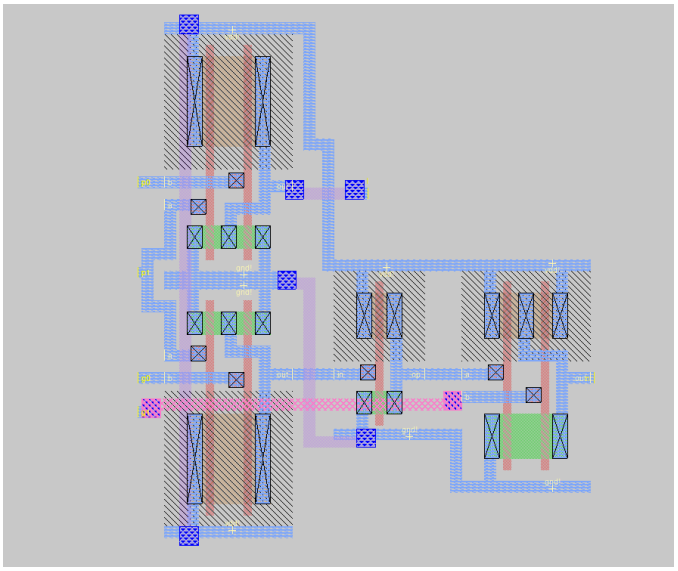


Fig. 23. Carry Look Ahead Adder

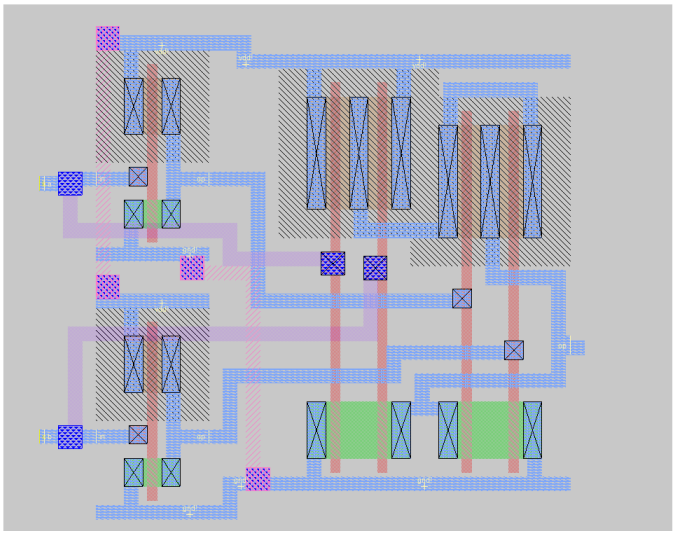


Fig. 24. Sum block

Fig. 25. Adder

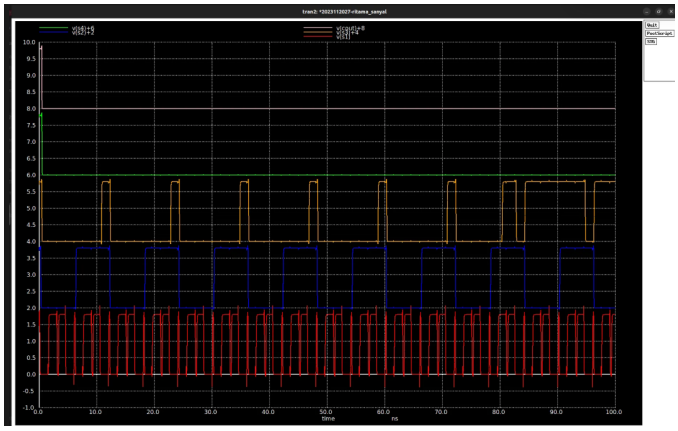


Fig. 26. Post Layout of adder

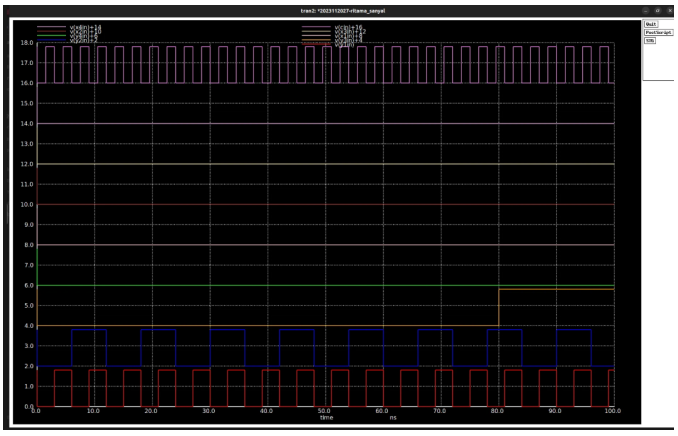


Fig. 27. Post layout of Adder

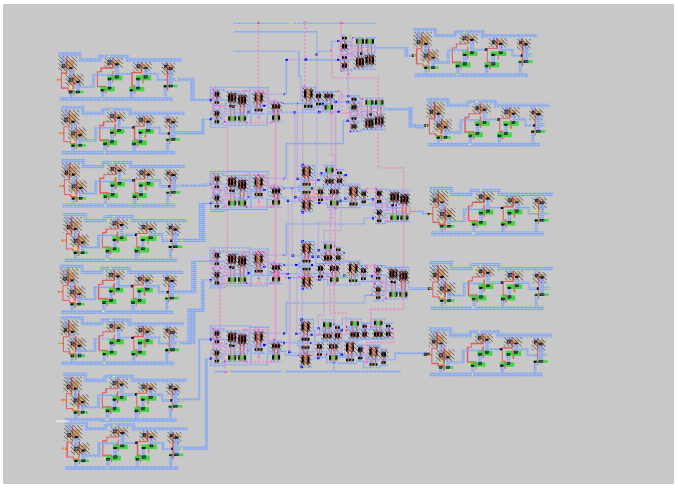


Fig. 28. Full Circuit

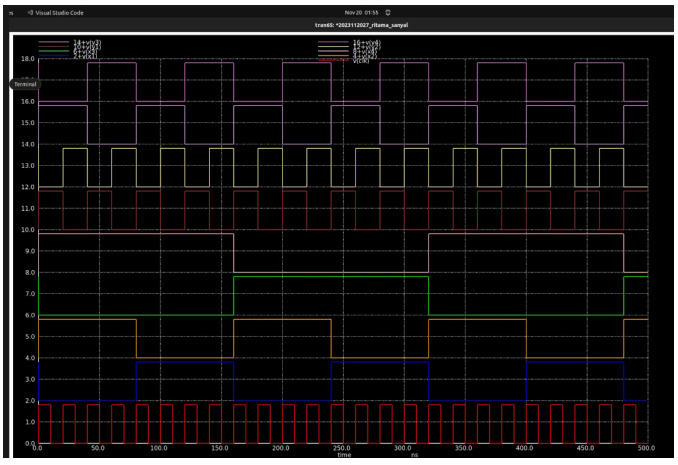


Fig. 29. Post Layout of Full Circuit

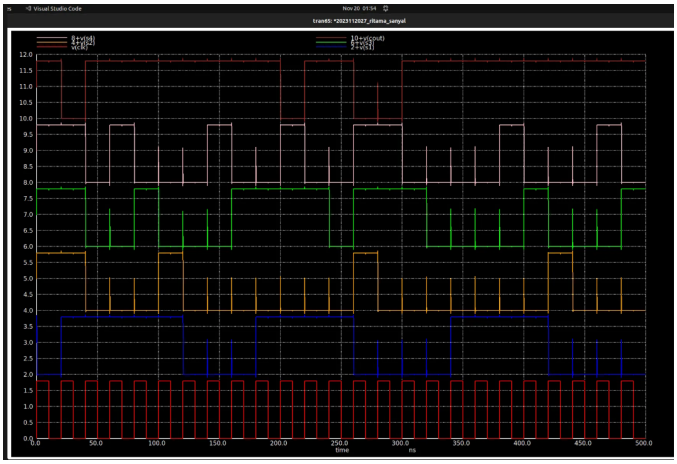


Fig. 30. Post Layout of Full Circuit

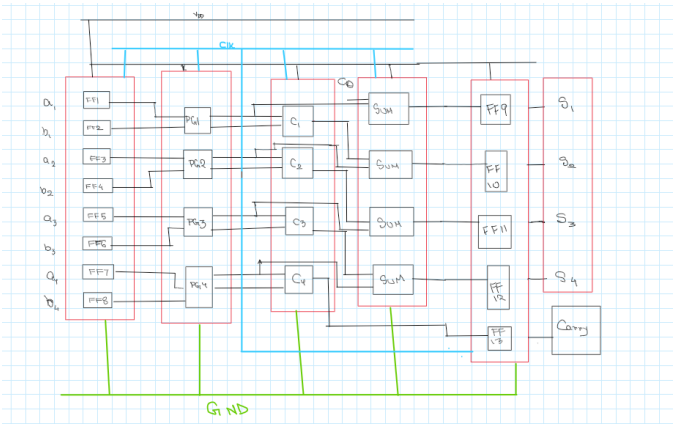


Fig. 31. Layout of Circuit

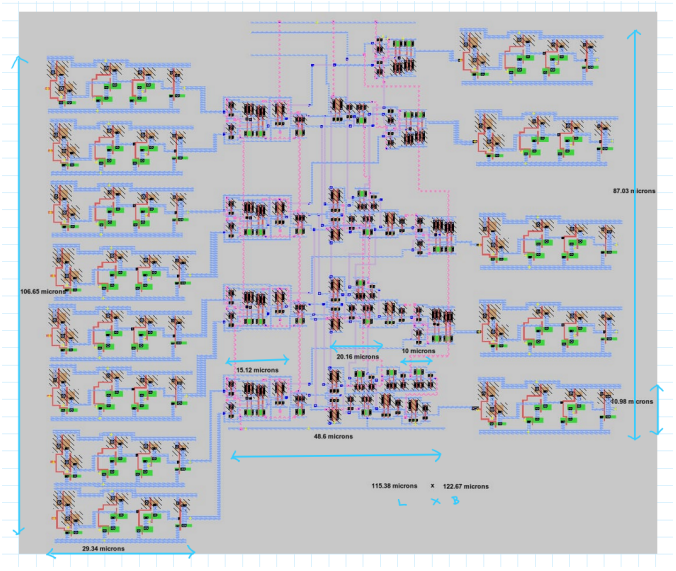


Fig. 32. Horizontal and vertical pitches

```
D=0 clk=0 Q1=x Qbar1=x Q2=x Qbar2=x
D=1 clk=1 Q1=x Qbar1=x Q2=0 Qbar2=1
D=1 clk=0 Q1=1 Qbar1=0 Q2=0 Qbar2=1
D=0 clk=1 Q1=1 Qbar1=0 Q2=1 Qbar2=0
D=0 clk=0 Q1=0 Qbar1=1 Q2=1 Qbar2=0
D=1 clk=1 Q1=0 Qbar1=1 Q2=0 Qbar2=1
D=1 clk=0 Q1=1 Qbar1=0 Q2=0 Qbar2=1
D=1 clk=1 Q1=1 Qbar1=0 Q2=1 Qbar2=0
D=0 clk=0 Q1=1 Qbar1=0 Q2=1 Qbar2=0
D=1 clk=1 Q1=1 Qbar1=0 Q2=0 Qbar2=1
D=1 clk=0 Q1=1 Qbar1=0 Q2=0 Qbar2=1
D=1 clk=1 Q1=1 Qbar1=0 Q2=1 Qbar2=0
D=1 clk=0 Q1=1 Qbar1=0 Q2=1 Qbar2=0
D=1 clk=1 Q1=1 Qbar1=0 Q2=1 Qbar2=0
(base) ritama@ritama:~/Downloads/Ver
```

Fig. 33. TSPC Verilog Output

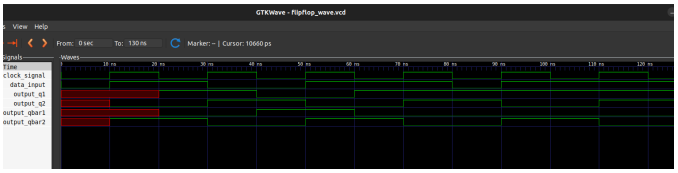


Fig. 34. GTKWaveform for TSPC Flipflop

```
VCD info: dumpfile test.vcd opened for o
clk=x a=xxxx b=xxxx cin=x o=xxxx cout=x
clk=1 a=xxxx b=xxxx cin=x o=xxxx cout=x
clk=0 a=0000 b=0000 cin=0 o=0000 cout=0
clk=1 a=0000 b=0000 cin=0 o=0000 cout=0
clk=0 a=0011 b=0001 cin=0 o=0100 cout=0
clk=1 a=0011 b=0001 cin=0 o=0100 cout=0
clk=0 a=0011 b=0001 cin=0 o=0100 cout=0
clk=1 a=0011 b=0001 cin=0 o=0100 cout=0
clk=0 a=0011 b=0001 cin=0 o=0100 cout=0
(base) ritama@ritama:~/Downloads/Veri/Co
```

Fig. 35. Carry Look Ahead Adder Verilog Output

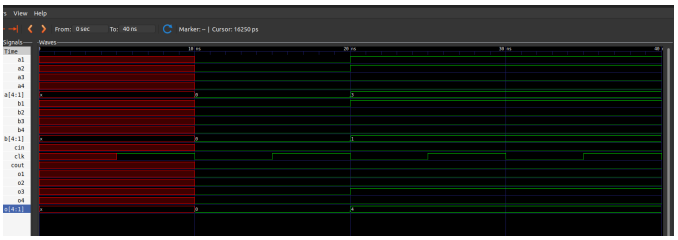


Fig. 36. GTKWave output for CLA

```
VCD info: dumpfile test.vcd opened for output.
clk=0 a=xxxx b=xxxx ci=x co=x s=xxxx
clk=1 a=xxxx b=xxxx ci=x co=x s=xxxx
clk=0 a=0000 b=0000 ci=0 co=x s=xxxx
clk=1 a=0000 b=0000 ci=0 co=x s=xxxx
clk=0 a=0001 b=0001 ci=0 co=0 s=0000
clk=1 a=0001 b=0001 ci=0 co=0 s=0000
clk=0 a=0010 b=0011 ci=1 co=0 s=0010
clk=1 a=0010 b=0011 ci=1 co=0 s=0010
clk=0 a=1111 b=0000 ci=0 co=0 s=0110
clk=1 a=1111 b=0000 ci=0 co=0 s=0110
clk=0 a=0000 b=1111 ci=0 co=0 s=1111
clk=1 a=0000 b=1111 ci=0 co=0 s=1111
clk=0 a=1111 b=1111 ci=0 co=0 s=1111
clk=1 a=1111 b=1111 ci=0 co=0 s=1111
clk=0 a=1111 b=1111 ci=1 co=1 s=1110
clk=1 a=1111 b=1111 ci=1 co=1 s=1110
clk=0 a=1000 b=1000 ci=0 co=1 s=1111
clk=1 a=1000 b=1000 ci=0 co=1 s=1111
clk=0 a=1000 b=1000 ci=1 co=1 s=0000
clk=1 a=1000 b=1000 ci=1 co=1 s=0000
clk=0 a=0101 b=0011 ci=0 co=1 s=0001
clk=1 a=0101 b=0011 ci=0 co=1 s=0001
clk=0 a=0101 b=0011 ci=1 co=0 s=1000
clk=1 a=0101 b=0011 ci=1 co=0 s=1000
clk=0 a=0110 b=1001 ci=0 co=0 s=1001
clk=1 a=0110 b=1001 ci=0 co=0 s=1001
clk=0 a=0110 b=1001 ci=1 co=0 s=1111
clk=1 a=0110 b=1001 ci=1 co=0 s=1111
clk=0 a=0111 b=0111 ci=0 co=1 s=0000
clk=1 a=0111 b=0111 ci=0 co=1 s=0000
clk=0 a=0111 b=0111 ci=1 co=0 s=1110
clk=1 a=0111 b=0111 ci=1 co=0 s=1110
clk=0 a=0111 b=0111 ci=1 co=0 s=1111
clk=1 a=0111 b=0111 ci=1 co=0 s=1111
clk=0 a=0111 b=0111 ci=1 co=0 s=1111
clk=1 a=0111 b=0111 ci=1 co=0 s=1111
clk=0 a=0111 b=0111 ci=1 co=0 s=1111
clk=1 a=0111 b=0111 ci=1 co=0 s=1111
```

Fig. 37. Circuit Verilog Output

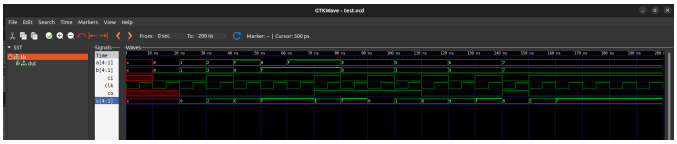


Fig. 38. GTKWaveform of final output

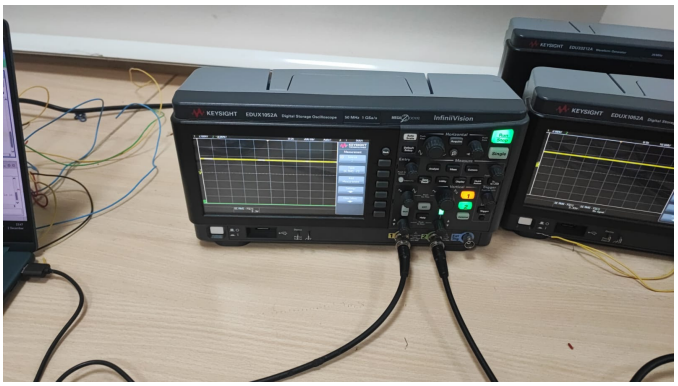


Fig. 39. FPGA

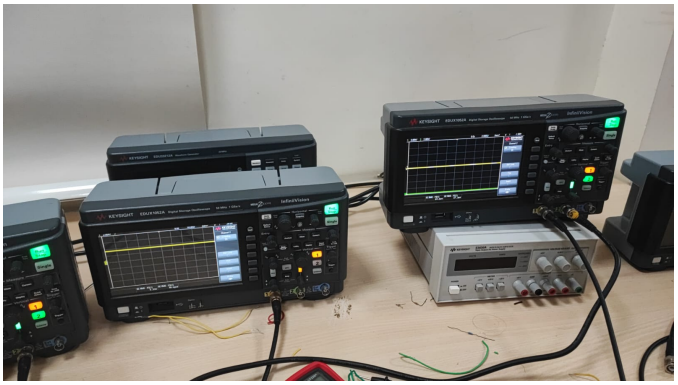


Fig. 40. FPGA

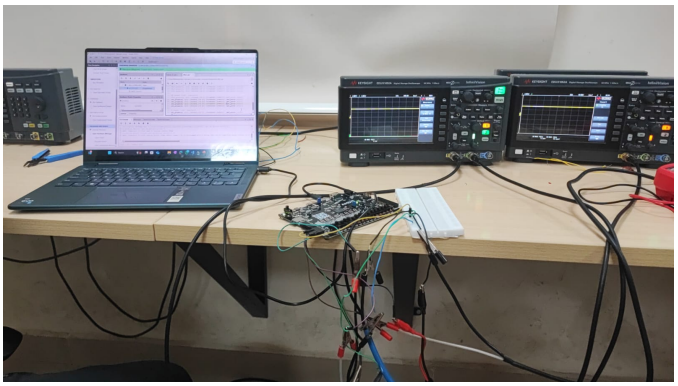


Fig. 41. Osciloscope implementation