

4-Bit CLA Adder

VLSI Course Project

1st Vignesh Vembar

ECE, IIIT Hyderabad

2023102019

vigneshvembar.m@students.iiit.ac.in

Abstract—This document is a report on the Design and Simulation of a 4-bit Carry Look Ahead Adder.

I. INTRODUCTION

The Carry Look Ahead Adder is a digital circuit that is used to add two 4 bit binary numbers. It is faster than the Ripple Carry Adder as it generates the carry signals for all the bits at the same time. We will also be passing the inputs and outputs through D Flip Flops to make the circuit synchronous.

II. PROPOSED DESIGN

A. CLA Adder

[1] From the given inputs signals, we first generate Propagate and Generate signals. These are defined as follows ¹:

$$P_i = A_i \oplus B_i \quad (1)$$

$$G_i = A_i \cdot B_i \quad (2)$$

Then, the carry signals can be generated as follows:

$$C_{i+1} = G_i + P_i \cdot C_i \quad (3)$$

In reality, we use multi-input gates to calculate these carries instead of recursively. This is done using the following equations:

$$C_1 = G_0 + P_0 \cdot C_0 \quad (4)$$

$$C_2 = G_1 + G_0 \cdot P_1 + C_0 \cdot P_0 \cdot P_1 \quad (5)$$

$$C_3 = G_2 + G_1 \cdot P_2 + G_0 \cdot P_1 \cdot P_2 + C_0 \cdot P_0 \cdot P_1 \cdot P_2 \quad (6)$$

$$\begin{aligned} C_4 &= G_3 + G_2 \cdot P_3 + G_1 \cdot P_2 \cdot P_3 \\ &\quad + G_0 \cdot P_1 \cdot P_2 \cdot P_3 + C_0 \cdot P_0 \cdot P_1 \cdot P_2 \cdot P_3 \end{aligned} \quad (7)$$

¹An optimization here could be to use the + operator to calculate Propagate instead of \oplus . Unfortunately, I did not have this insight until too late in the project.

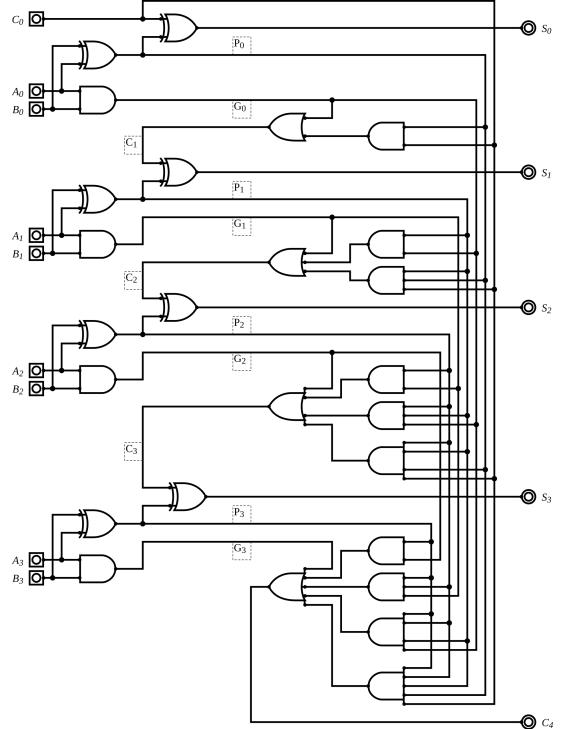


Fig. 1: Circuit Diagram for CLA Adder

B. D Flip Flop

1) *CMOS Implementation*: The obvious and basic choice would be to use CMOS gates to implement the D Flip Flop. This uses two D Latches in a Master-Slave configuration.

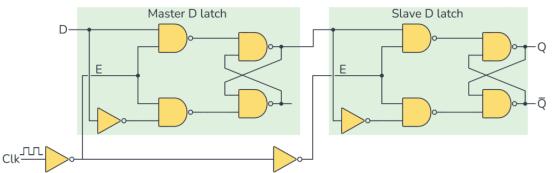


Fig. 2: Circuit Diagram for CMOS D Flip Flop

The issues with this circuit are:

- 1) The circuit is slow as it has a large propagation delay.
- 2) The circuit has large area, as we will see later in the layout.

2) *Optimized Implementation:* This circuit works by storing the value of D after the first inverter when the clock signal is low. In the rising edge of the clock, D does not pass through the first MOSFET, and the stored value is passed to the output Q.

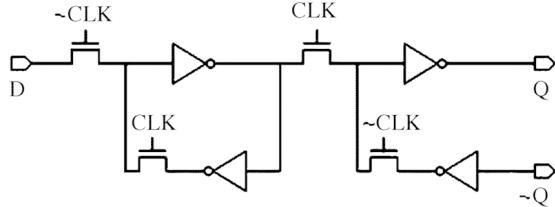


Fig. 3: Circuit Diagram for Optimized D Flip Flop

III. VERILOG SIMULATION

First, we test the digital logic using Verilog simulations. We use GTKWave to plot our results from the testbench. Later, we put this onto an FPGA board to test the circuit in real life.



Fig. 4: GTKWave Plot of CLA Adder Verilog Simulation



Fig. 5: GTKWave Plot of D Flip Flop Verilog Simulation



Fig. 6: GTKWave Plot of Full Circuit Verilog Simulation

IV. NGSPICE SIMULATION

For all my designs, I've chosen to use a CMOS inverter of $W_p/W_n = 2.5$ as basis. The technology files (180nm) used are included in the repo. [2]

A. Inverter

This is the standard CMOS inverter with $W_p/W_n = 2.5$.

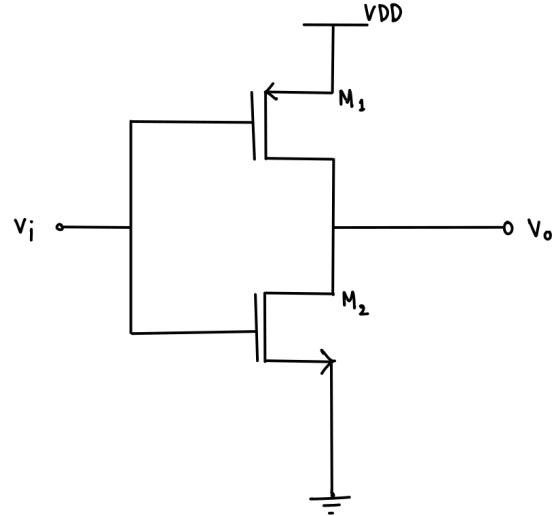


Fig. 7: Circuit Diagram of Inverter

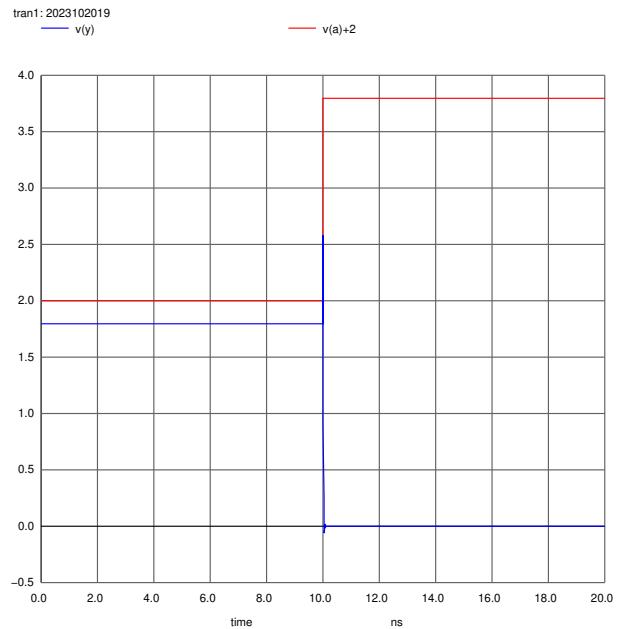


Fig. 8: NGSPICE Plot of Inverter

B. NAND Gate

To size the CMOS NAND gate, we need to equalize the resistance through the Pull-Up network and Pull-Down Network.

For an n input NAND gate, where W_{inv} is the size of the NMOS in the inverter:

$$\frac{n}{W_n} = \frac{1}{W_{inv}} \quad (8)$$

$$W_n = n \cdot W_{inv} \quad (9)$$

W_p stays the same.

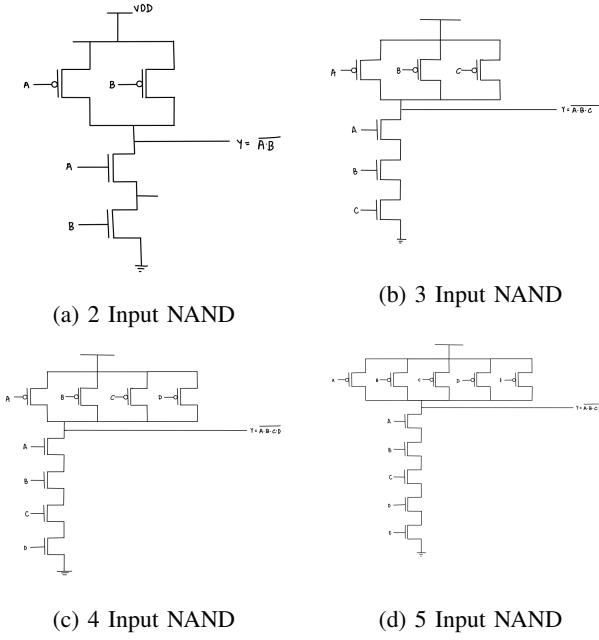


Fig. 9: Circuit Diagram of NAND Gates

$$\frac{n}{W_p} = \frac{1}{2.5 \cdot W_{inv}} \quad (10)$$

$$W_p = 2.5n \cdot W_{inv} \quad (11)$$

W_p stays the same.

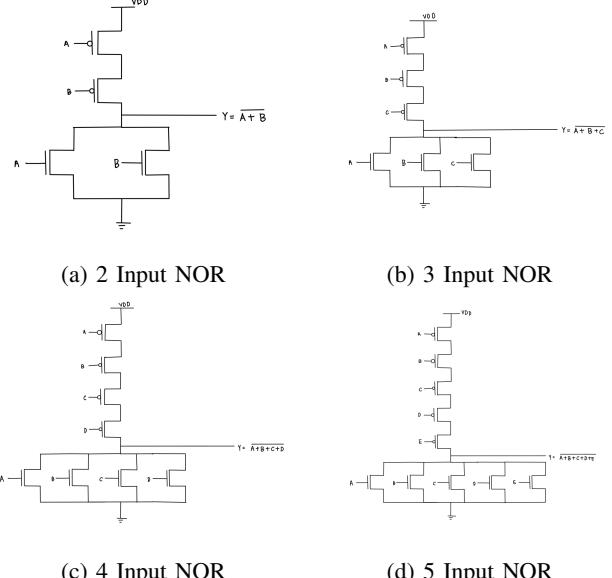


Fig. 11: Circuit Diagram of NOR Gates

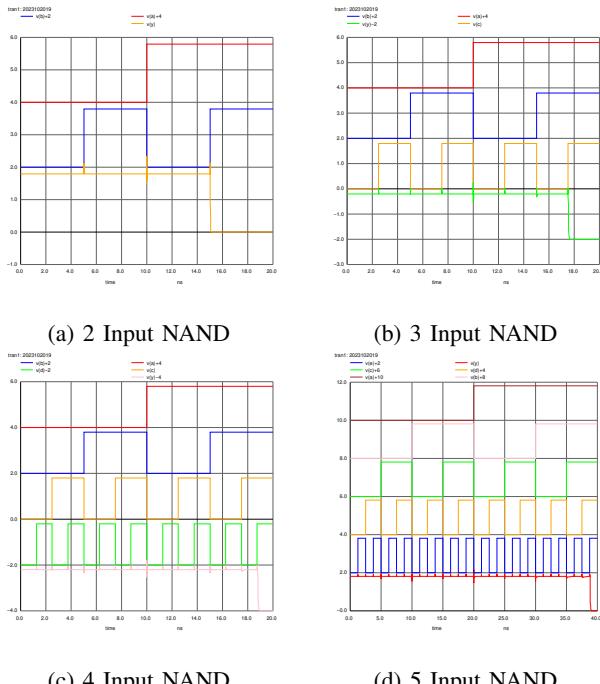


Fig. 10: NGSPICE Plot of NAND Gates

C. NOR Gate

Similarly, for the CMOS NOR gate, we need to equalize the resistance through the Pull-Up network and Pull-Down Network.

For an n input NOR gate, where W_{inv} is the size of the NMOS in the inverter:



Fig. 12: NGSPICE Plot of NOR Gates

D. XOR Gate

1) *CMOS Implementation:* The standard CMOS Implementation works, but we can do better with less gates.

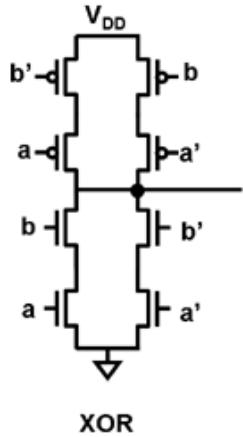


Fig. 13: Circuit Diagram of CMOS XOR Gate

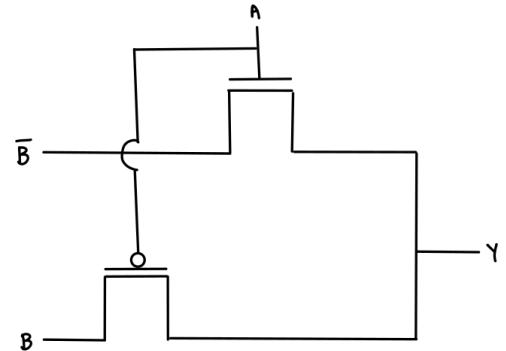


Fig. 15: Circuit Diagram of CPTL XOR Gate

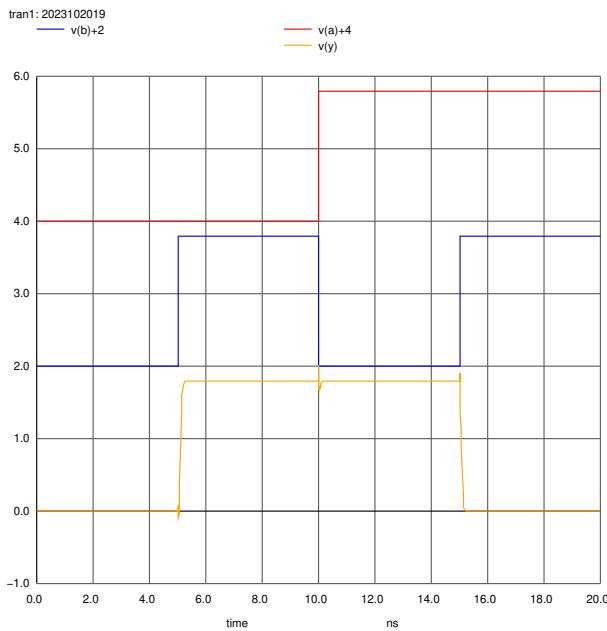


Fig. 14: NGSPICE Plot of CMOS XOR Gate

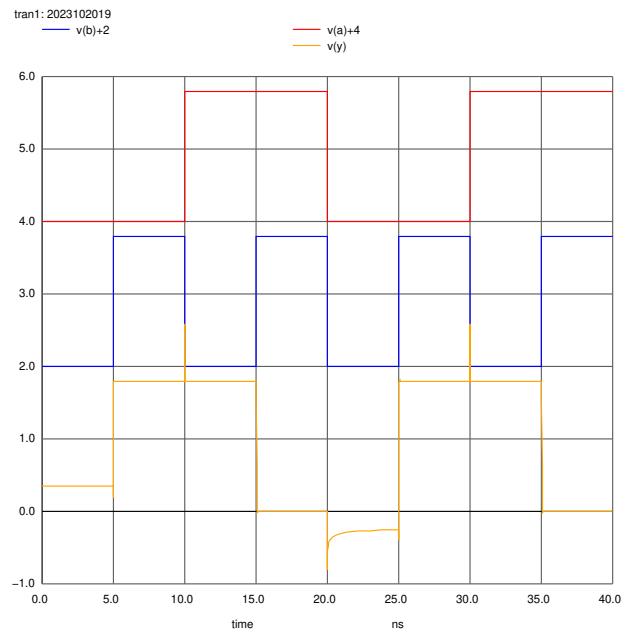


Fig. 16: NGSPICE Plot of CPTL XOR Gate

Because of the nature of NMOS and PMOS and not being able to pass proper values, this does ruin the levels of the outputs, but from testing I have noticed that this does not affect the final result.

E. Propagate/Generate Generator

1) *CMOS Implementation:* Using the circuit diagram in 1, we can implement the Propagate/Generate Generator using the previously defined gates by importing them into our NGSPICE files. First, we test it using the CMOS Implementation of XOR.

2) *Complimentary Pass Transistor Implementation:* We use a CPTL XOR gate to reduce the number of transistors used.

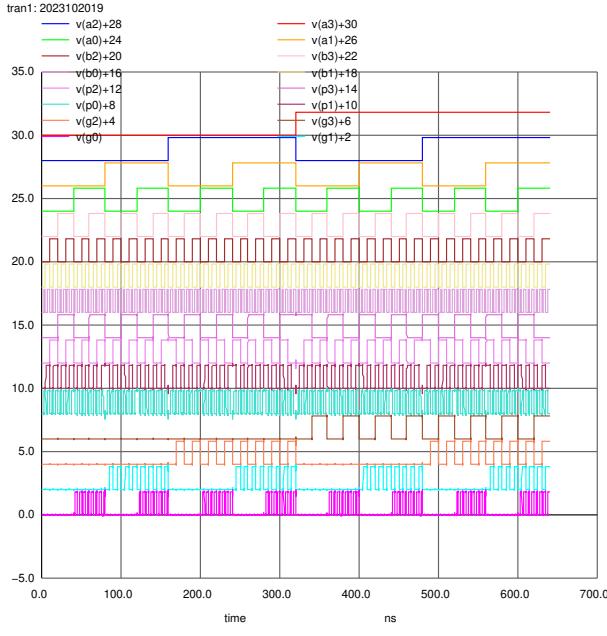


Fig. 17: NGSPICE Plot of CMOS Propagate/Generate Generator

2) *Complimentary Pass Transistor Implementation:* This is a test run using the CPTL XOR gate.

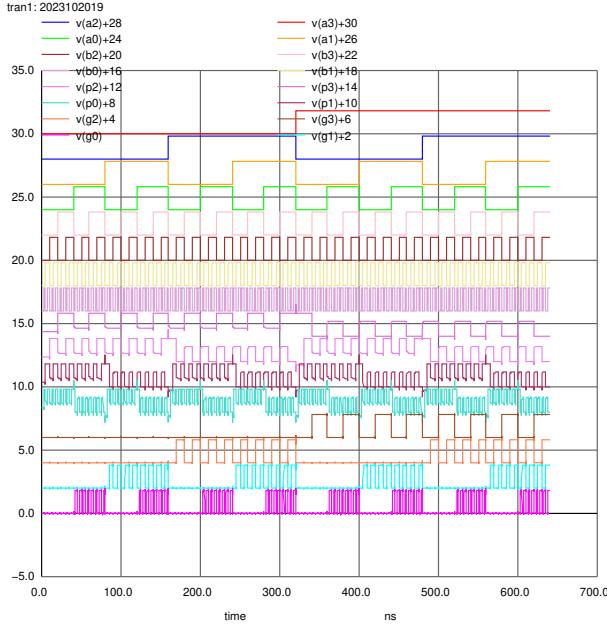


Fig. 18: NGSPICE Plot of CPTL Propagate/Generate Generator

We can already see massive issues in the levels of some of the outputs. This might seem concerning but it all gets fixed in the final circuit due to the nature of the D Flip Flop used.

F. Carry Look Ahead Generator

The Carry Look Ahead Generator is implemented only using CMOS gates.

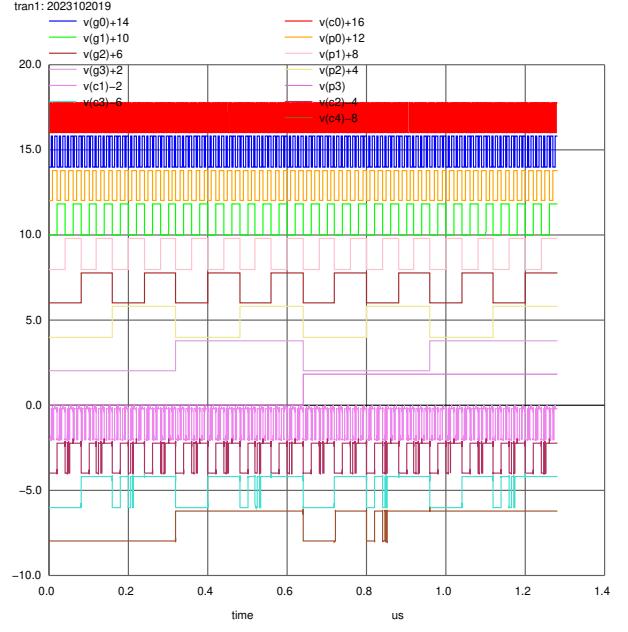


Fig. 19: NGSPICE Plot of CMOS Carry Look Ahead Generator

G. Sum Generator

1) *CMOS Implementation:* We similarly use the CMOS gates to implement the Sum Generator.

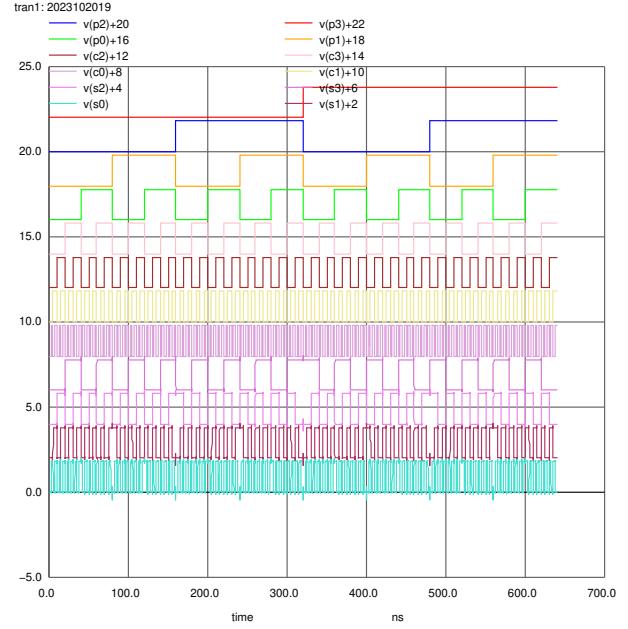


Fig. 20: NGSPICE Plot of CMOS Sum Generator

2) Complimentary Pass Transistor Implementation: We can see that the levels of the outputs are not correct in this circuit either, due to the nature of the CPTL XOR gate.

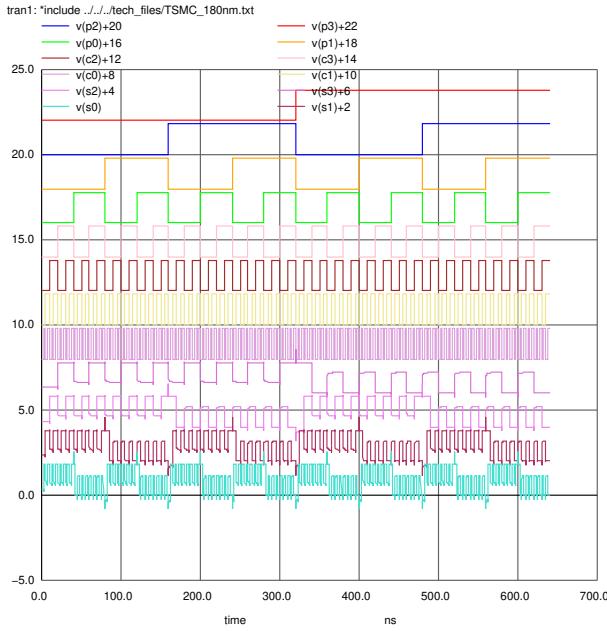


Fig. 21: NGSPICE Plot of CPTL Sum Generator

H. D Flip Flop

1) CMOS Implementation: We make the D Flip Flop using the CMOS gates, but as we noticed before, this is not the best way to implement it.

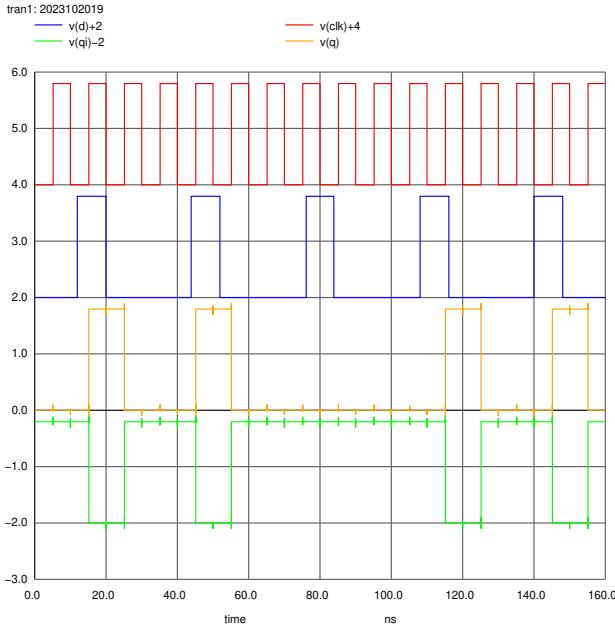


Fig. 22: NGSPICE Plot of CMOS D Flip Flop

2) Optimized Implementation: Making the D Flip Flop using our optimized circuit works as well:

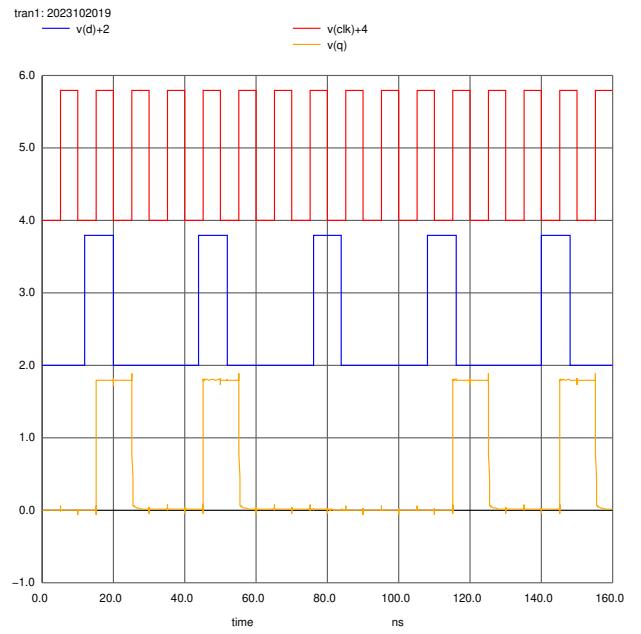


Fig. 23: NGSPICE Plot of Optimized D Flip Flop

I. Full Circuit

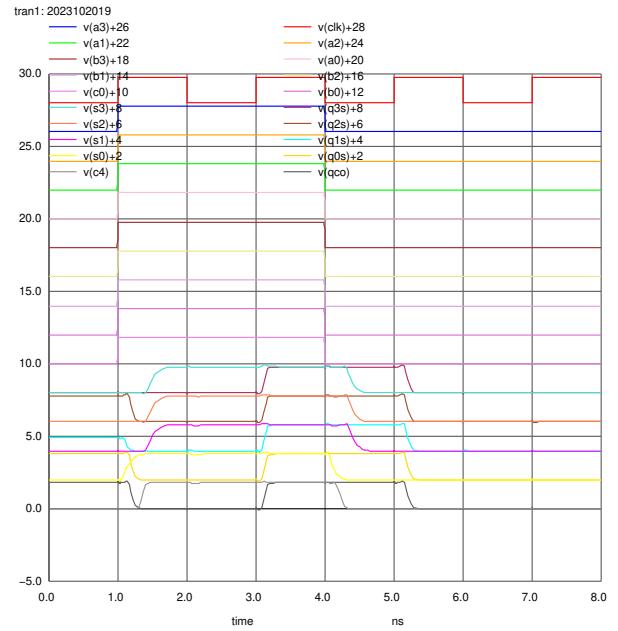


Fig. 24: NGSPICE Plot of CMOS Circuit

I) CMOS Implementation:

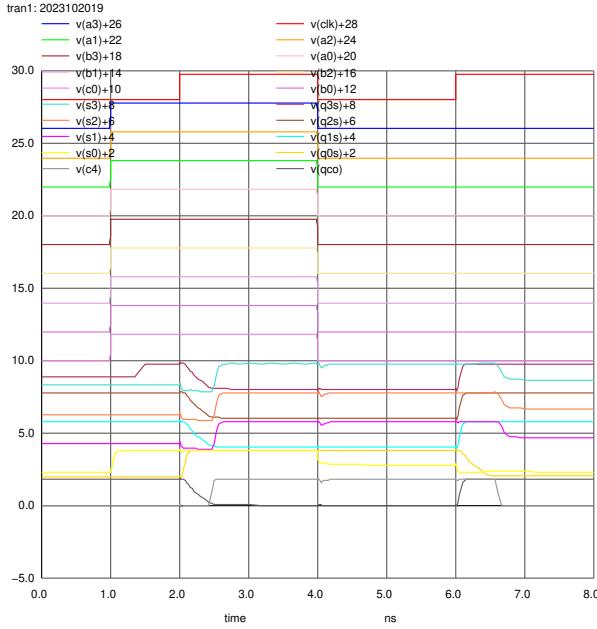


Fig. 25: NGSPICE Plot of Optimized Circuit

2) Optimized Implementation:

V. MAGIC LAYOUT

For layout, we use the free tool MAGIC. All subcircuits are laid out separately and then combined to form the full circuit. Before layout, we draw stick diagrams to have a rough idea of where to place what.

A. Inverter

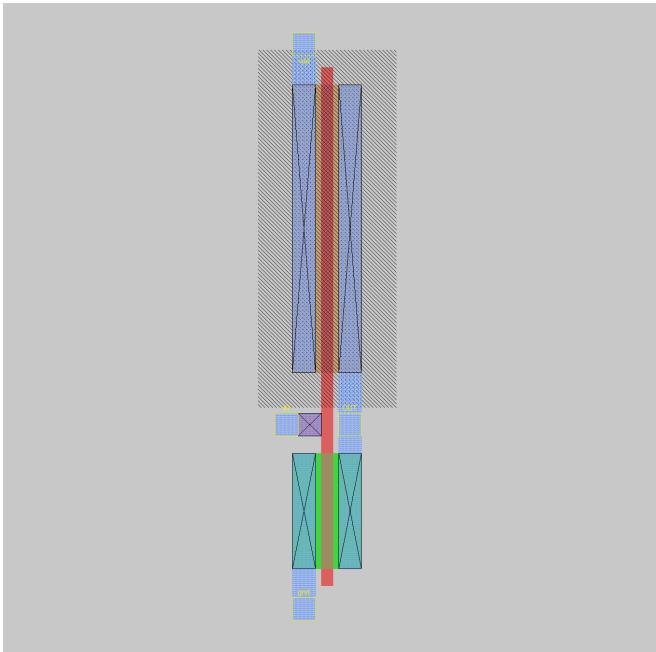


Fig. 27: MAGIC Layout of CMOS Inverter

B. NAND Gate

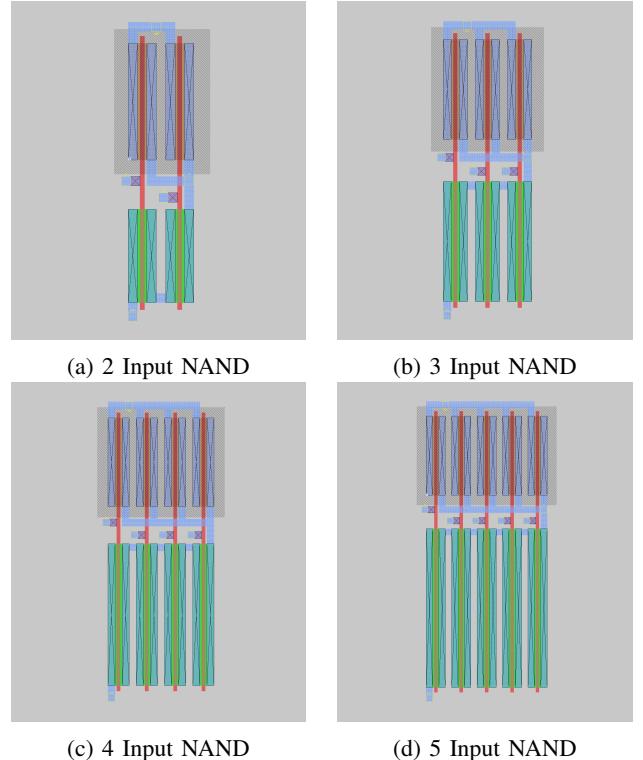


Fig. 28: MAGIC Layout of NAND Gates

C. NOR Gate

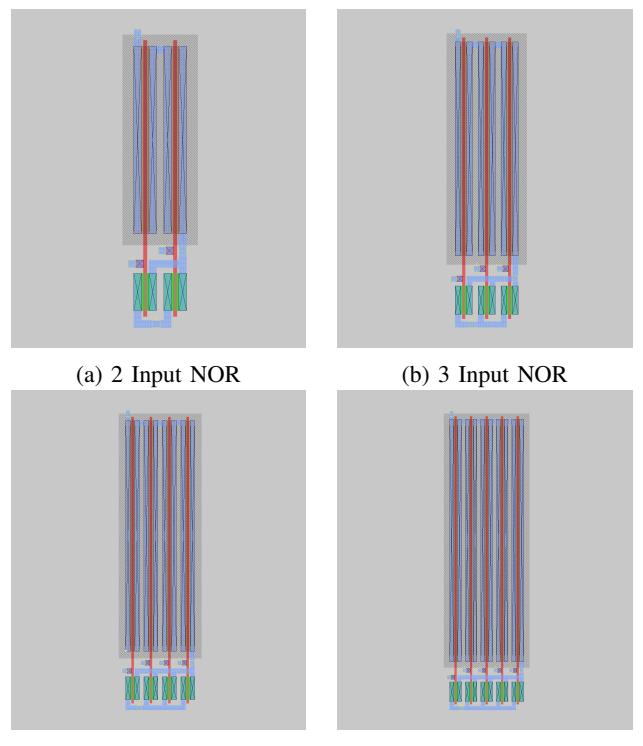


Fig. 29: MAGIC Layout of NOR Gates

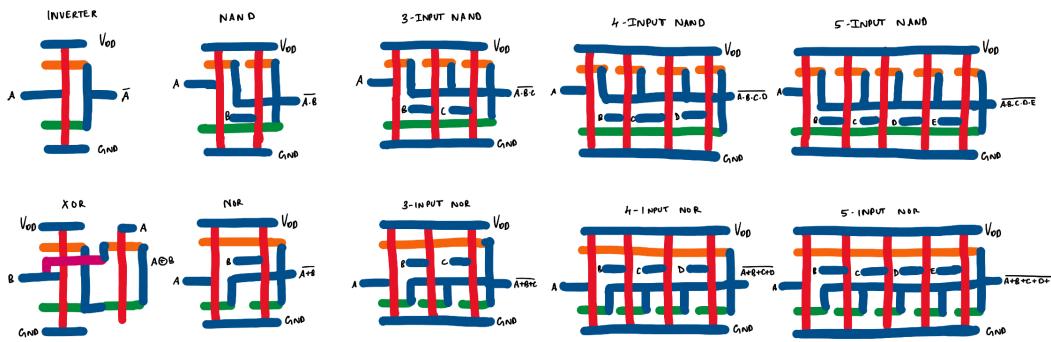


Fig. 26: Stick Diagrams

D. XOR Gate

E. Propagate/Generate Generator

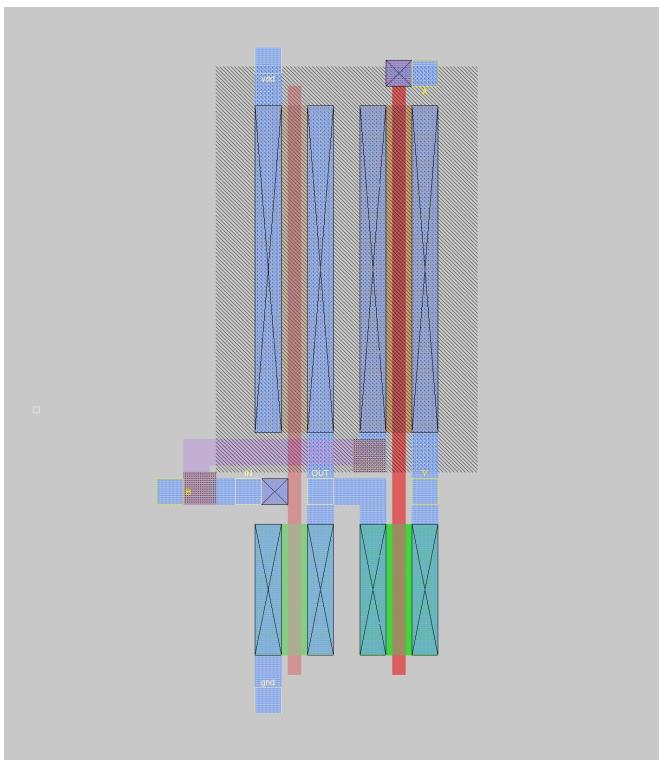


Fig. 30: MAGIC Layout of XOR Gate

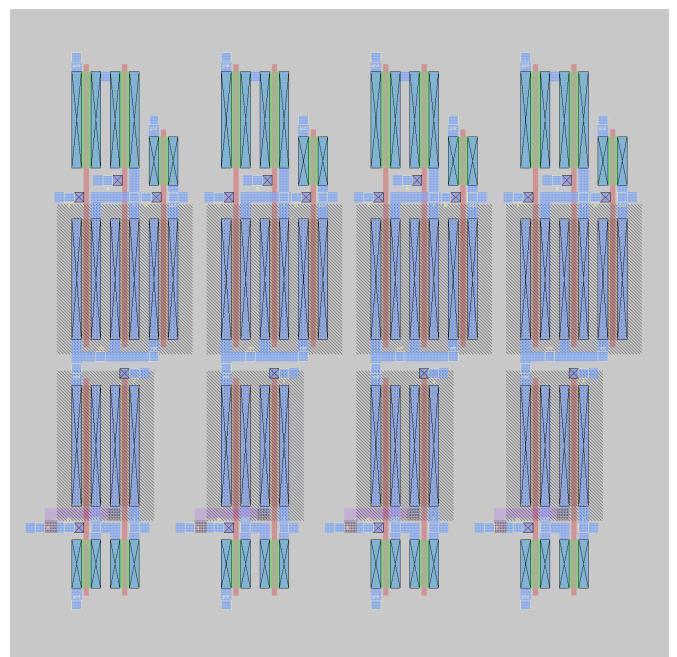


Fig. 31: MAGIC Layout of Propagate/Generate Generator

F. Carry Look Ahead Generator

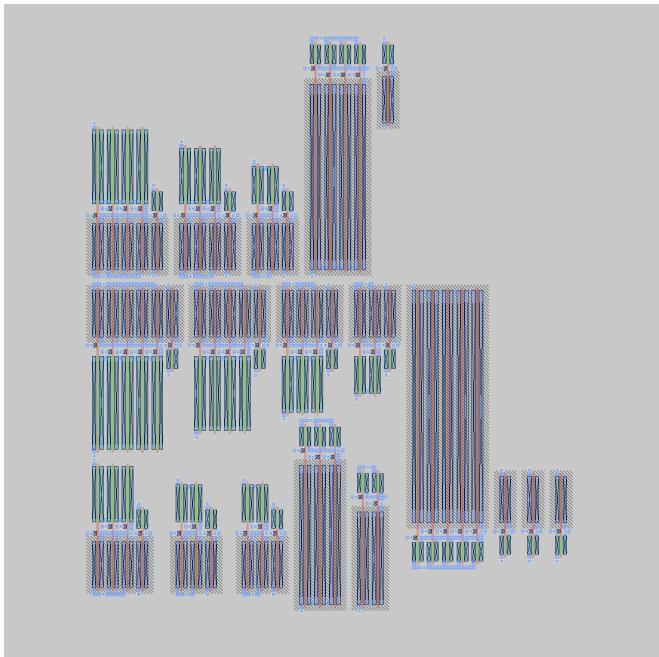


Fig. 32: MAGIC Layout of Carry Look Ahead Generator

G. Sum Generator

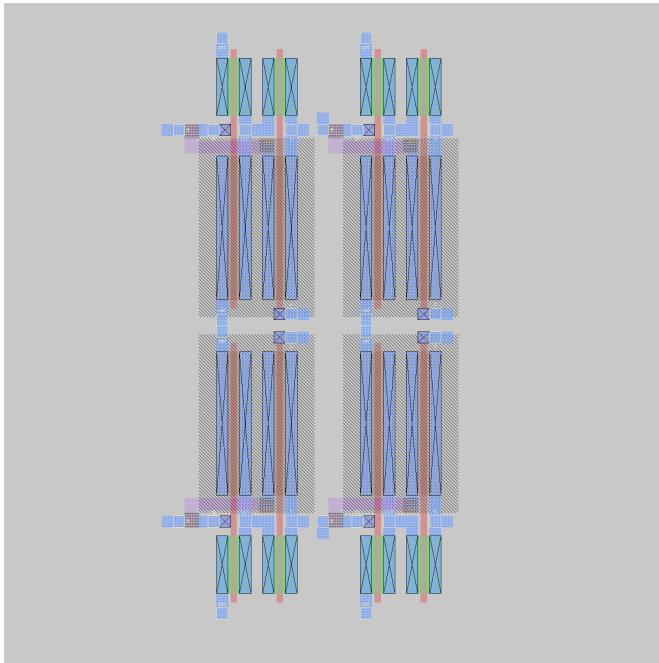


Fig. 33: MAGIC Layout of Sum Generator

H. D Flop Flop

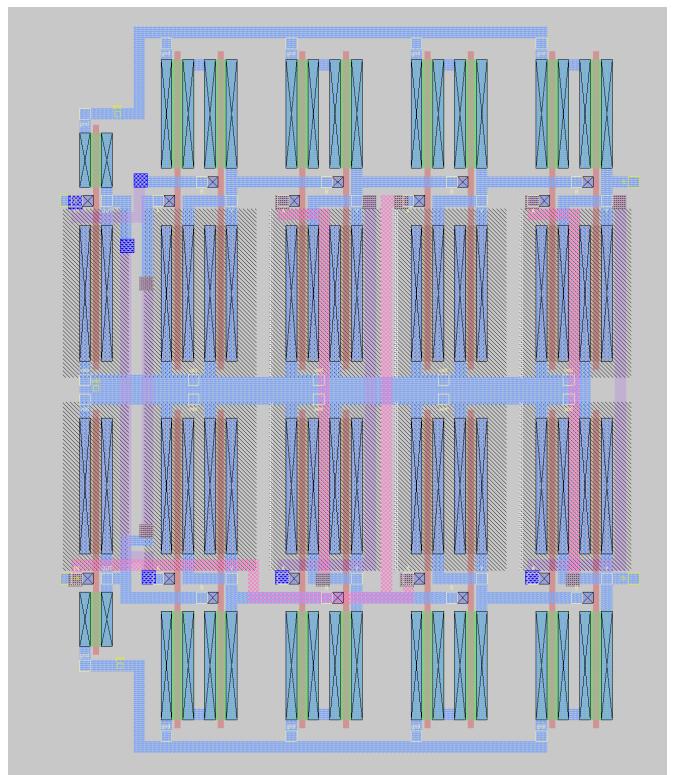


Fig. 34: MAGIC Layout of CMOS D Flip Flop

1) CMOS Implementation:

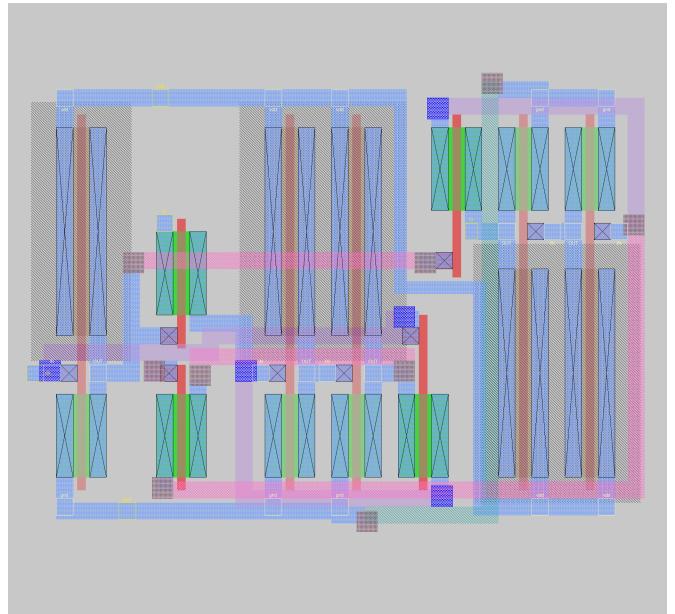


Fig. 35: MAGIC Layout of Optimized D Flip Flop

2) Optimized Implementation:

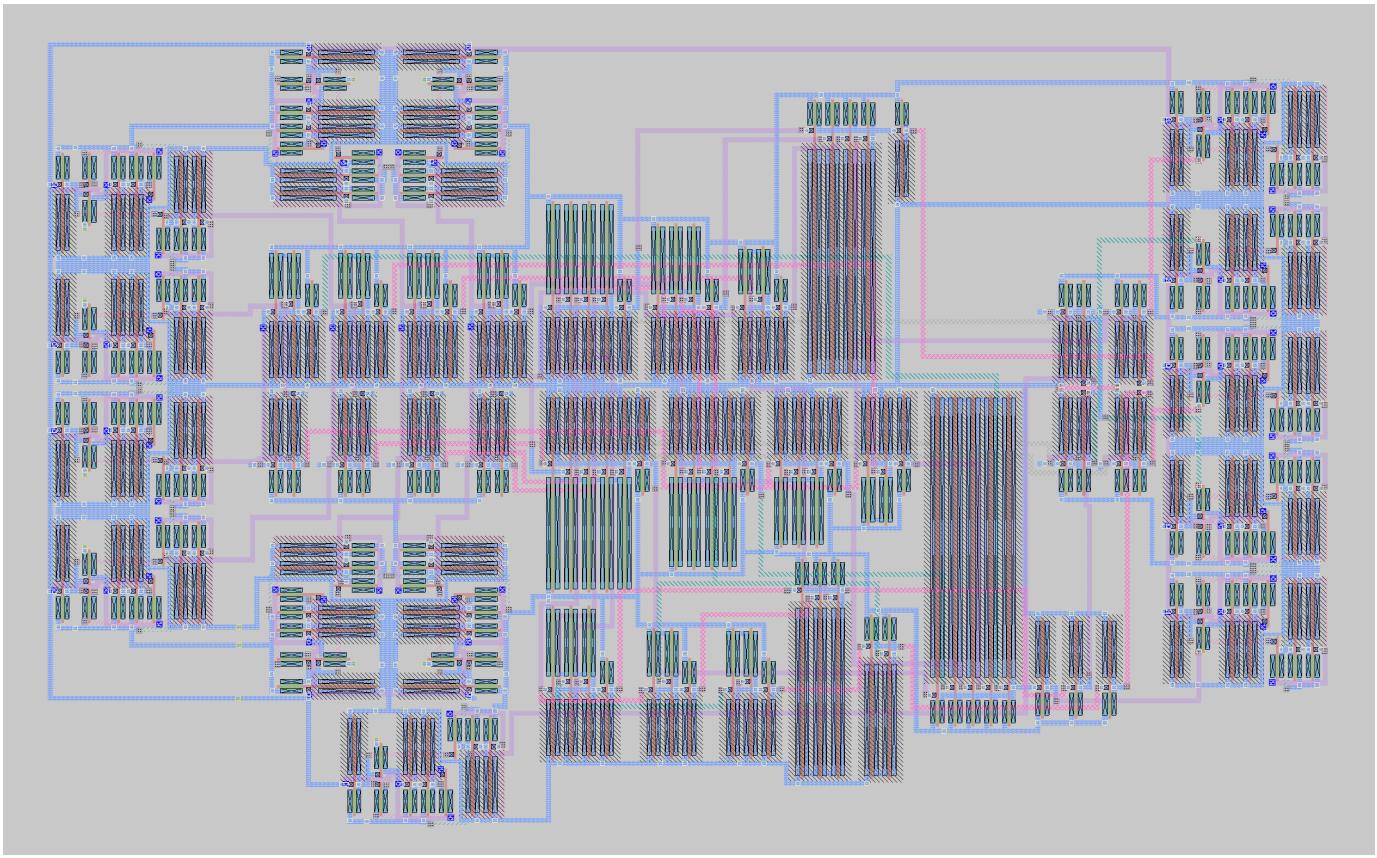


Fig. 36: MAGIC Layout of Full Circuit

I. Full Circuit

VI. POST LAYOUT SIMULATION

A. Inverter

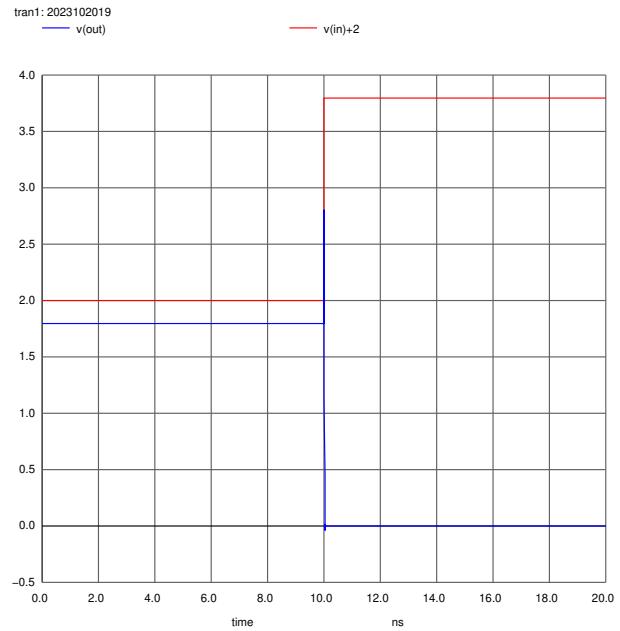


Fig. 37: Post Layout NGSPICE Plot of Inverter

B. NAND Gate

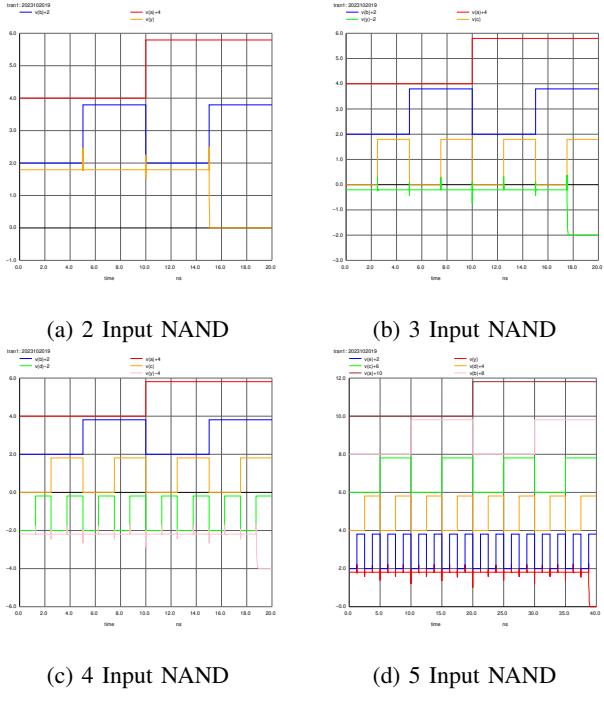


Fig. 38: Post Layout NGSPICE Plot of NAND Gates

C. NOR Gate

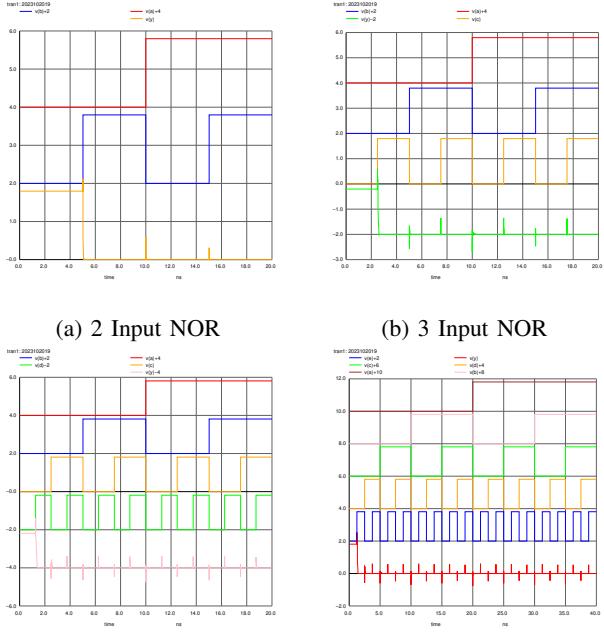


Fig. 39: Post Layout NGSPICE Plot of NOR Gates

D. XOR Gate

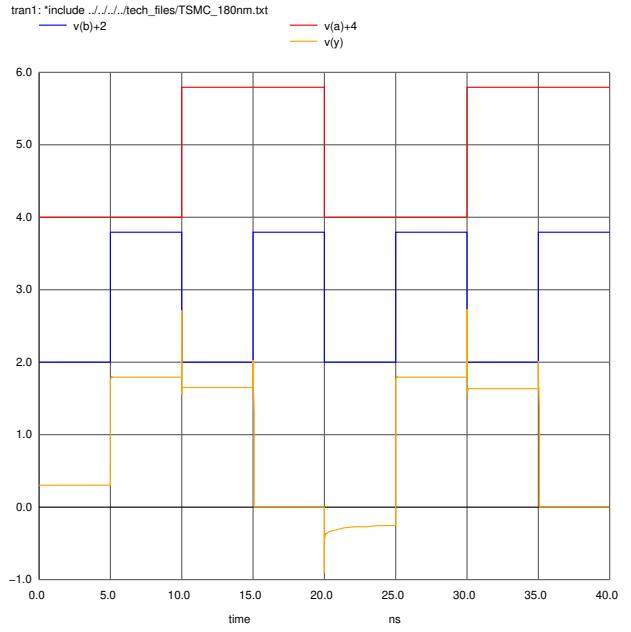


Fig. 40: Post Layout NGSPICE Plot of XOR Gate

E. D Flop Flop

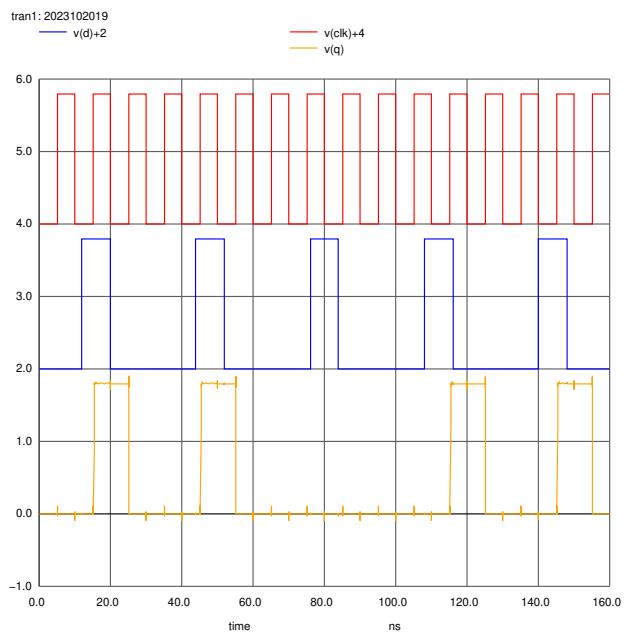


Fig. 41: Post Layout NGSPICE Plot of D Flip Flop

F. Full Circuit (with Load Inverters)

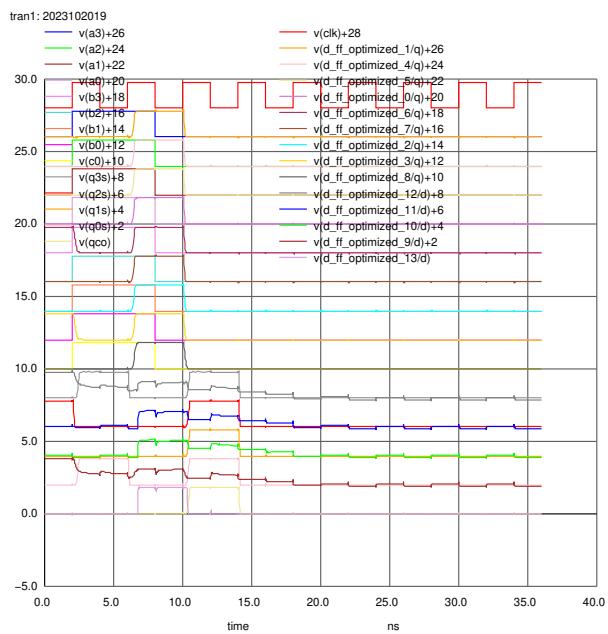
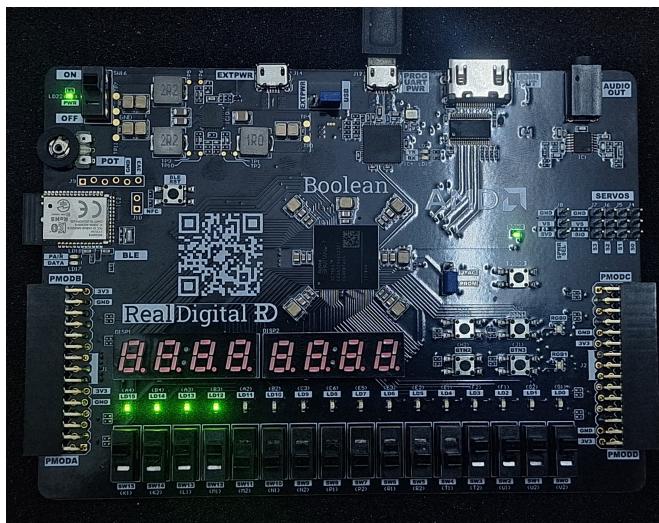


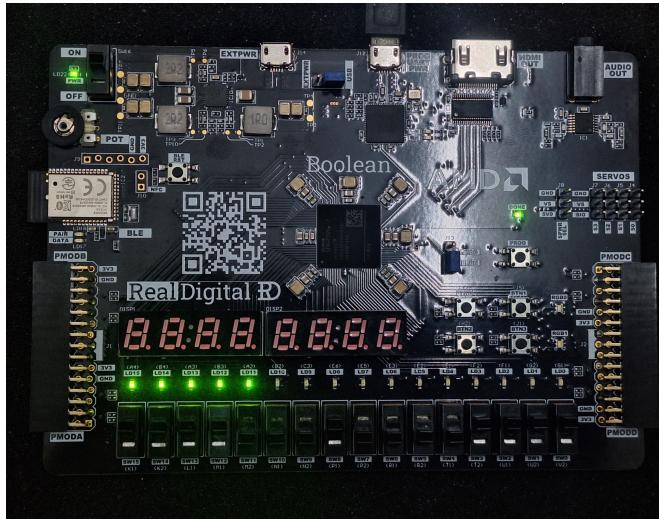
Fig. 42: Post Layout NGSPICE Plot of Full Circuit

VII. FPGA SIMULATION

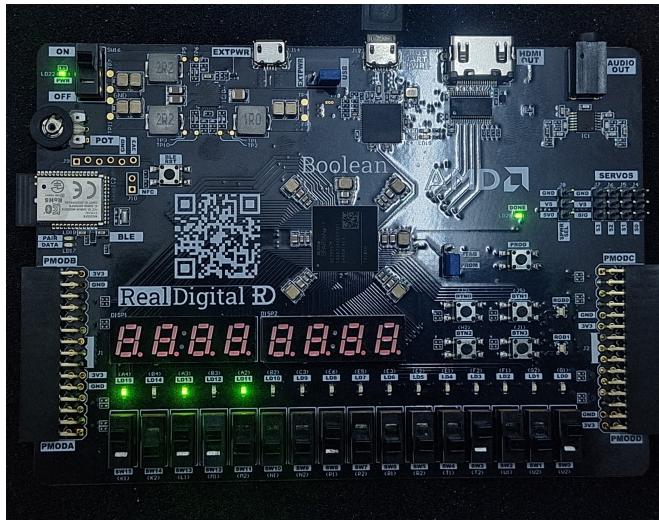
A. On Board LEDs



$$(a) 1111 + 1111 + 0 = 11110$$



$$(b) 1111 + 1111 + 1 = 11111$$



$$(c) 1011 + 1001 + 1 = 10101$$

Fig. 43: FPGA Simulation

B. Waveform Visualization

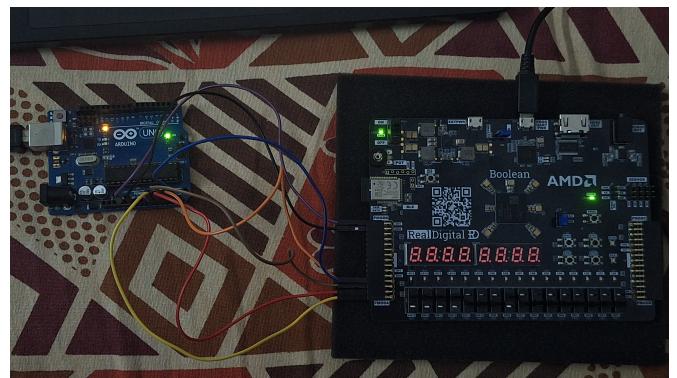


Fig. 44: Viewing Waveform of FPGA Simulation

ACKNOWLEDGMENT

REFERENCES

- [1] Wikipedia, "Carry-lookahead adder." Available at: https://en.wikipedia.org/wiki/Carry-lookahead_adder.
- [2] Github, Repository with all project files Available at: <https://github.com/wig-nesh/vlsi-course-project>.