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. In this Implementation, 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]:

$$P_i = A_i \oplus B_i \tag{1}$$
$$G_i = A_i \cdot B_i \tag{2}$$

Then, the carry signals can be generated as follows:

$$C_{i+1} = G_i + P_i \cdot C_i \tag{3}$$

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

$$C_1 = G_0 + P_0 \cdot C_0 \tag{4}$$
$$C_2 = G_1 + G_0 \cdot P_1 + C_0 \cdot P_0 \cdot P_1 \tag{5}$$
$$C_3 = G_2 + G_1 \cdot P_2 + G_0 \cdot P_1 \cdot P_2 + C_0 \cdot P_0 \cdot P_1 \cdot P_2 \tag{6}$$
$$C_4 = G_3 + G_2 \cdot P_3 + G_1 \cdot P_2 \cdot P_3 + G_0 \cdot P_1 \cdot P_2 \cdot P_3 + C_0 \cdot P_0 \cdot P_1 \cdot P_2 \cdot P_3 \tag{7}$$

[1] An 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.

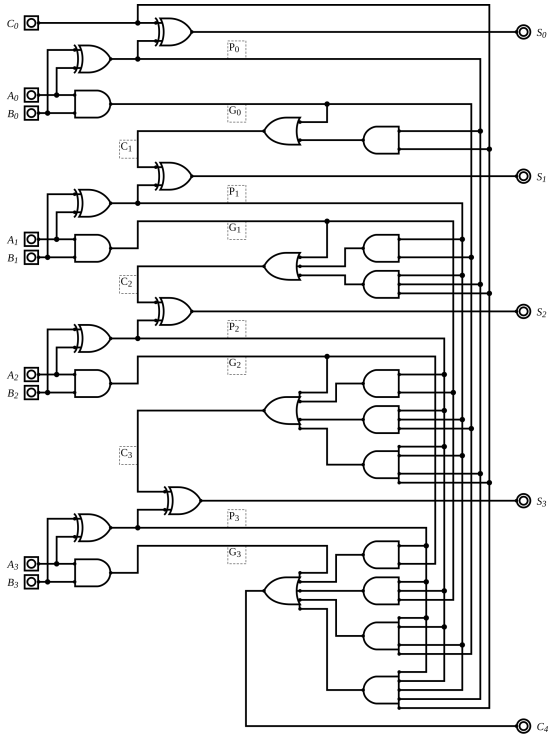


Fig. 1: Circuit Diagram for CLA Adder

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.

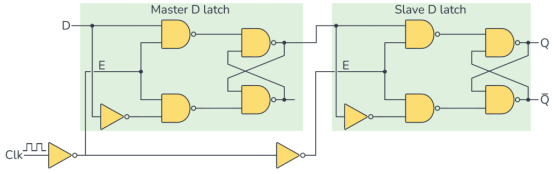


Fig. 2: Circuit Diagram for CMOS D Flip Flop

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.

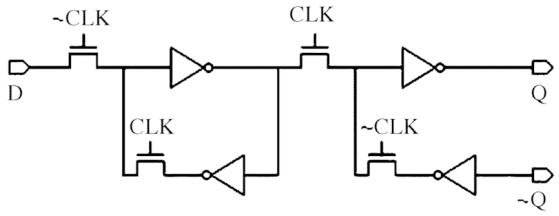


Fig. 3: Circuit Diagram for Optimized D Flip Flop

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 $W_p/W_n = 2.5$.

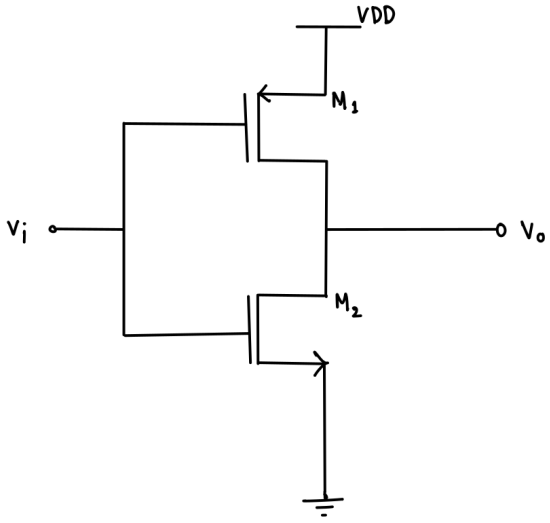


Fig. 7: Circuit Diagram of Inverter

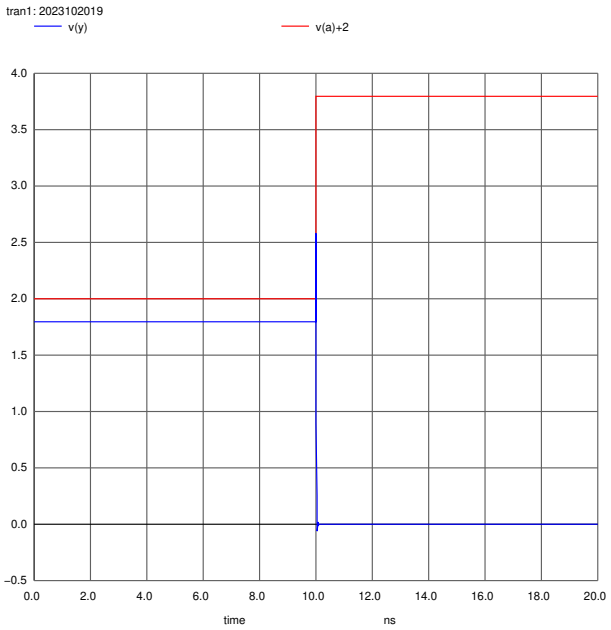


Fig. 8: NGSPICE Plot of Inverter

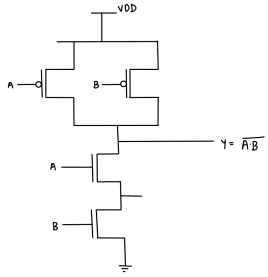
B. NAND Gate

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:

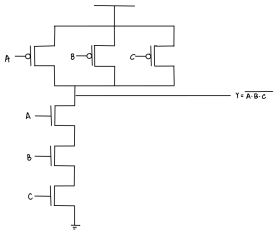
$$\frac{n}{W_n} = \frac{1}{W_{inv}} \tag{8}$$

$$W_n = n \cdot W_{inv} \tag{9}$$

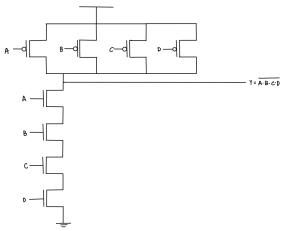
W_p stays the same.



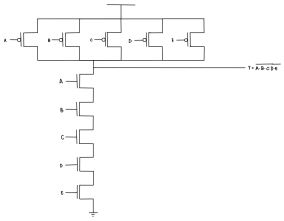
(a) 2 Input NAND



(b) 3 Input NAND

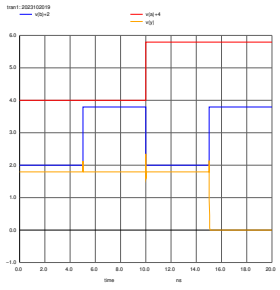


(c) 4 Input NAND

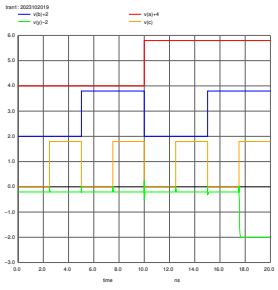


(d) 5 Input NAND

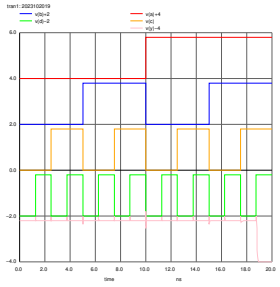
Fig. 9: Circuit Diagram of NAND Gates



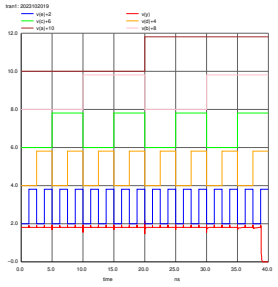
(a) 2 Input NAND



(b) 3 Input NAND



(c) 4 Input NAND



(d) 5 Input NAND

Fig. 10: NGSPICE Plot of NAND Gates

C. NOR Gate

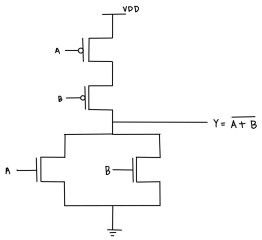
Similarly, for the CMOS NOR gate, we need to equalize the resistance through the Pull-Up network and Pull-Down Network.

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

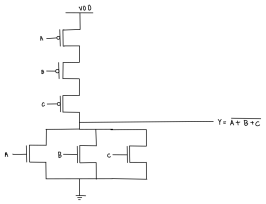
$$\frac{n}{W_p} = \frac{1}{2.5 \cdot W_{inv}} \tag{10}$$

$$W_p = 2.5n \cdot W_{inv} \tag{11}$$

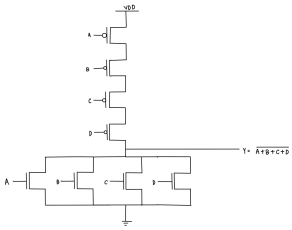
W_p stays the same.



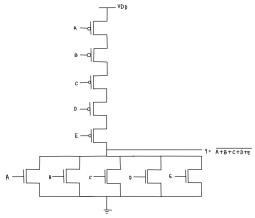
(a) 2 Input NOR



(b) 3 Input NOR

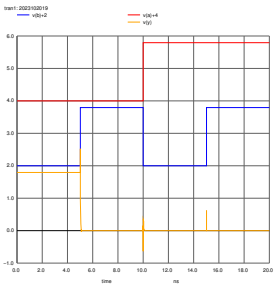


(c) 4 Input NOR

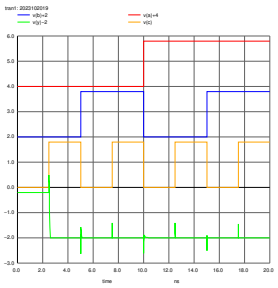


(d) 5 Input NOR

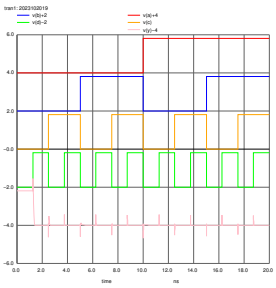
Fig. 11: Circuit Diagram of NOR Gates



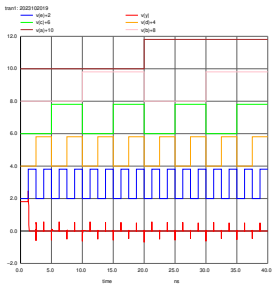
(a) 2 Input NOR



(b) 3 Input NOR



(c) 4 Input NOR



(d) 5 Input NOR

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.

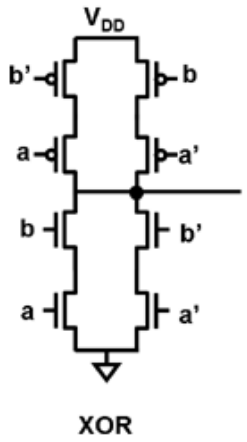


Fig. 13: Circuit Diagram of CMOS XOR Gate

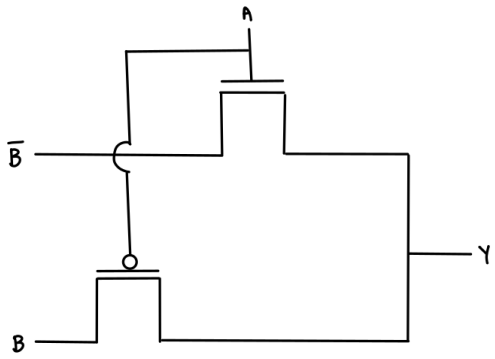


Fig. 15: Circuit Diagram of CPTL XOR Gate

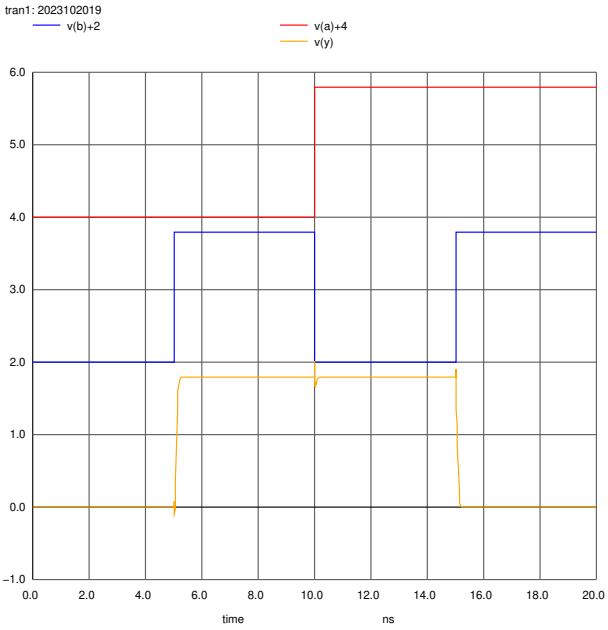


Fig. 14: NGSPICE Plot of CMOS XOR Gate

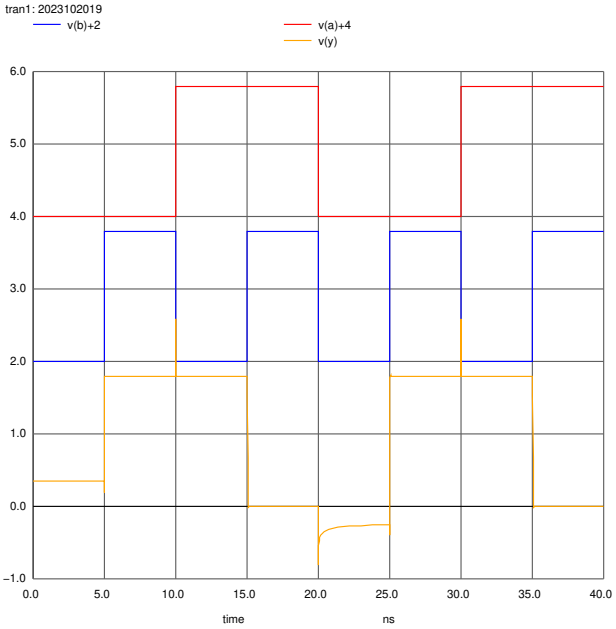


Fig. 16: NGSPICE Plot of CPTL XOR Gate

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

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.

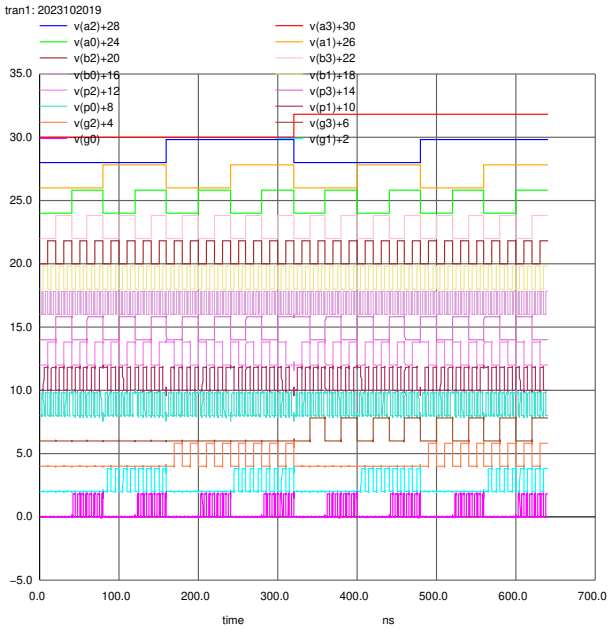


Fig. 17: NGSPICE Plot of CMOS Propagate/Generate Generator

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

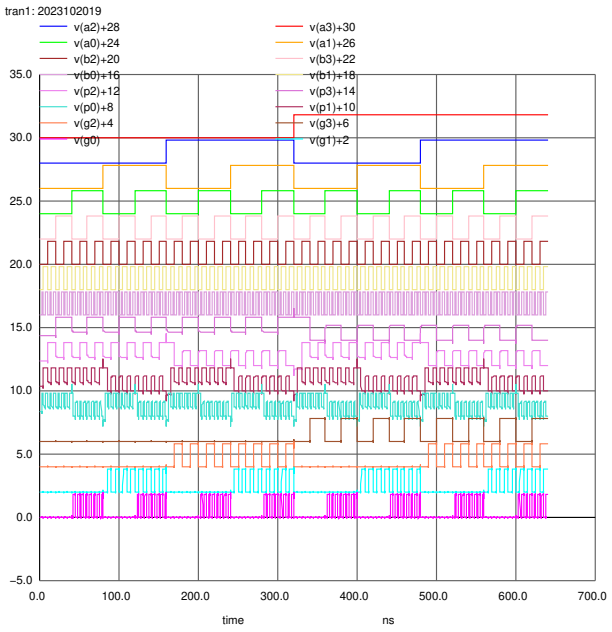


Fig. 18: NGSPICE Plot of CPTL Propagate/Generate Generator

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.

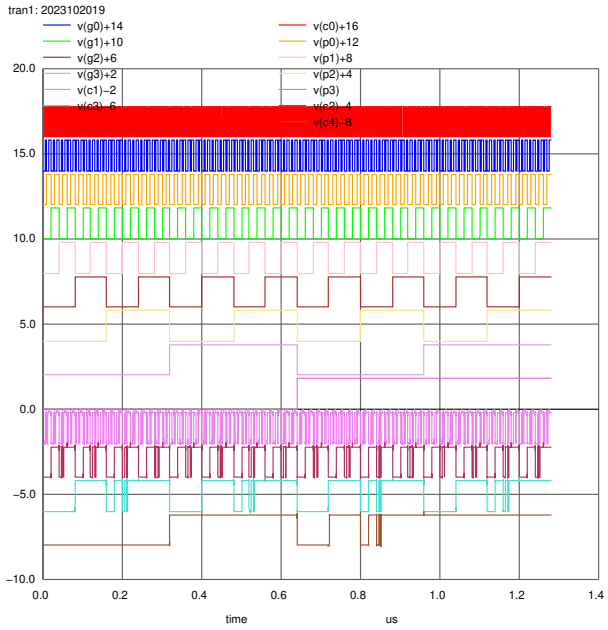


Fig. 19: NGSPICE Plot of CMOS Carry Look Ahead Generator

G. Sum Generator

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

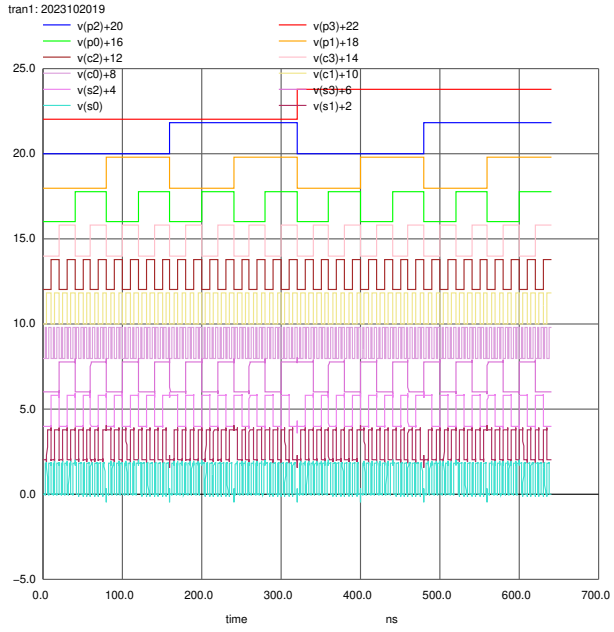


Fig. 20: NGSPICE Plot of CMOS Sum Generator

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.

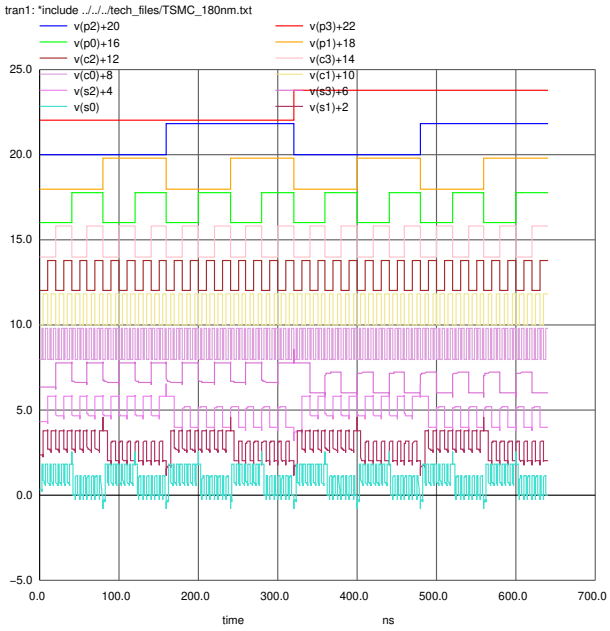


Fig. 21: NGSPICE Plot of CPTL Sum Generator

H. D Flop 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.

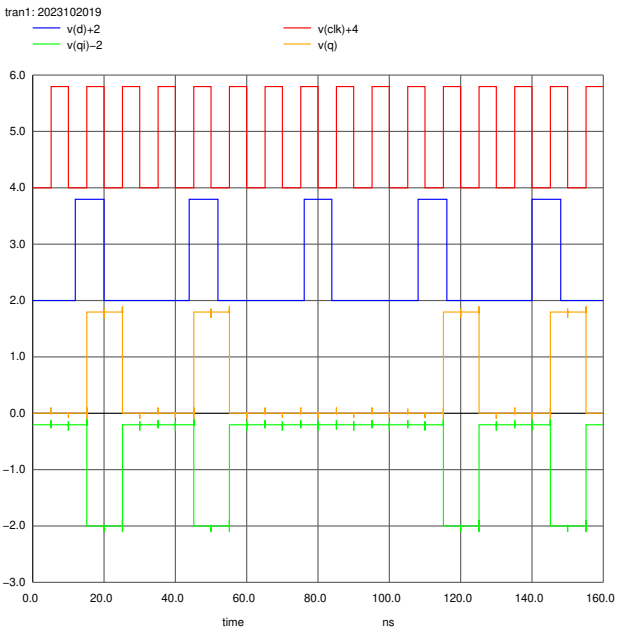


Fig. 22: NGSPICE Plot of CMOS D Flip FLop

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

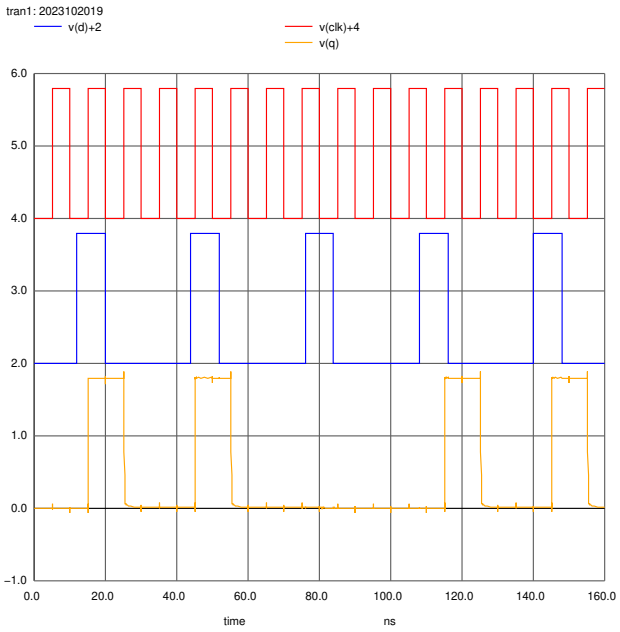


Fig. 23: NGSPICE Plot of Optimized D Flip Flop

We can find the characteristics of the D Flip Flop as follows:

1) t_{cq} is the time taken for the output Q to change after the rising clock edge.
2) t_{su} is the setup time, the time before the rising clock edge that the input D must be stable. We calculate this based on the rise/fall time of the mosfet that passes the input D, and the propagation delay of the inverter that stores the value in the flip flop.
3) t_{h} is the hold time, the time after the rising clock edge that the input D must be stable. We calculate this based on the rise/fall time of the mosfet that passes the input D, and the propagation delay of the inverter that returns the output of the flip flop.

We get these values using:

$$t_{\text{cq}} = t_q - t_{\text{clk}} \quad (\text{at } 0.5 \cdot \text{SUPPLY}) \tag{12}$$

$$t_{\text{su}} = t_{\text{rise}_{M1}} + t_{\text{prop}_{\text{inv1}}} \tag{13}$$

$$t_{\text{h}} = t_{\text{rise}_{M2}} + t_{\text{prop}_{\text{inv2}}} \tag{14}$$

```
t_cq                =  5.24757e-11s
t_su                =  3.80793e-10s
t_h                 =  2.67745e-10s
```

I. Full Circuit

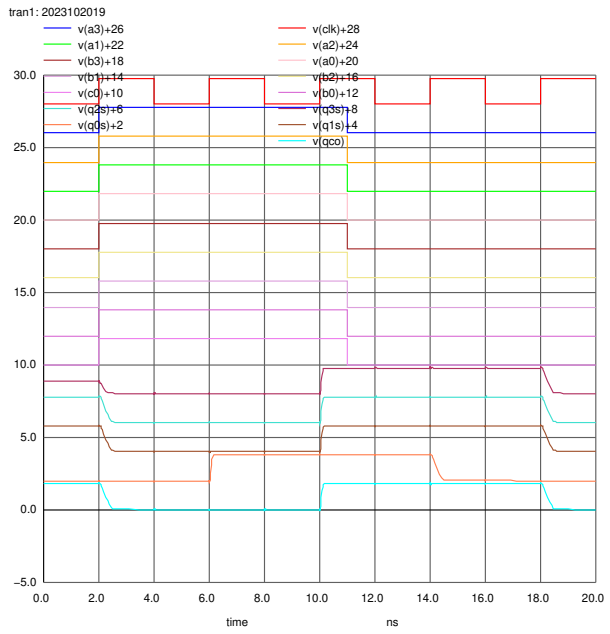


Fig. 24: NGSPICE Plot of Optimized Circuit

We see there are some issues with S_0 getting set and reset before the rest of the signals. This is due to S_0 being relatively the fastest signal to calculate output for. We handle this in the post layout simulation.

We need to find the propagation delay of this circuit without the flip flops. We realize that C_4 must take the most time to calculate, so we find the rise and fall time delay of it.

```
delay_c4_r              =   3.10307e-10s
delay_c4_f              =   2.07737e-10s
```

We know that,

$$T_{clk} > T_{cq} + T_{pd} + T_{su} \tag{15}$$
$$T_{clk} > 7.425 \times 10^{-10}\,\text{s} \approx 0.75\,\text{ns} \tag{16}$$
$$f_{clk} \approx 1.33\,\text{GHz} \tag{17}$$

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

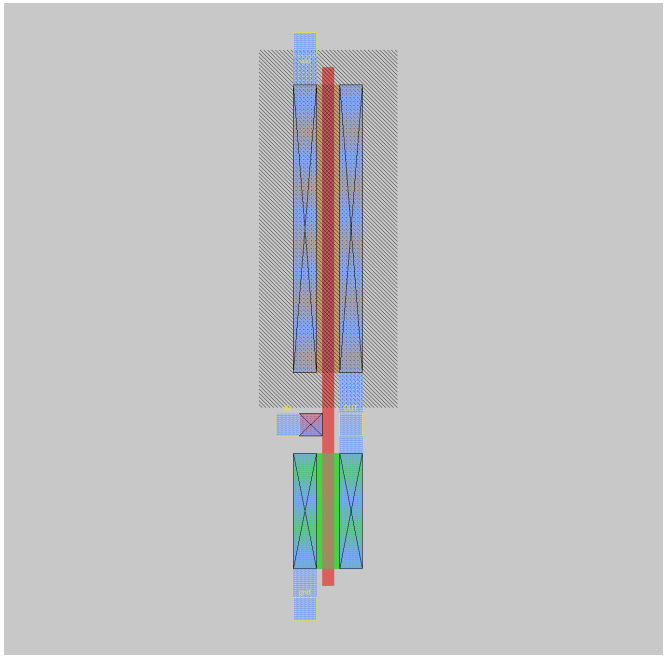
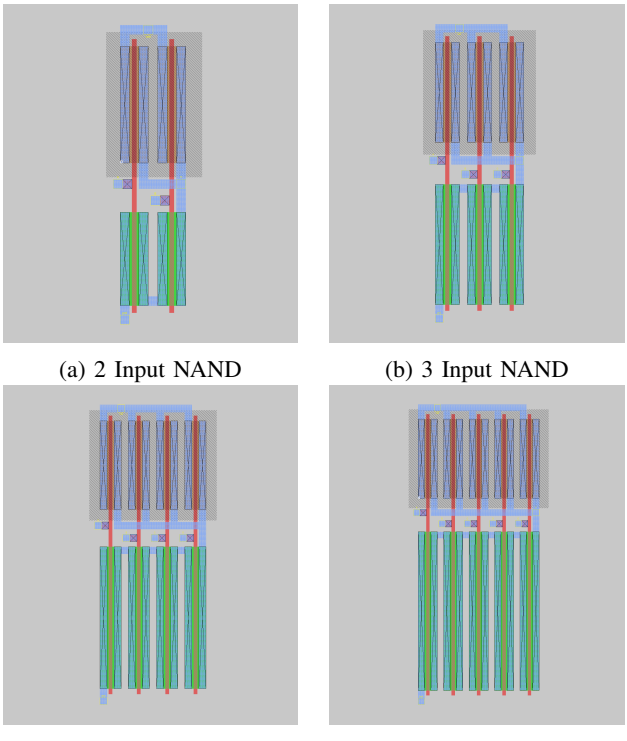


Fig. 26: MAGIC Layout of CMOS Inverter

B. NAND Gate



(a) 2 Input NAND (b) 3 Input NAND

(c) 4 Input NAND (d) 5 Input NAND

Fig. 27: MAGIC Layout of NAND Gates

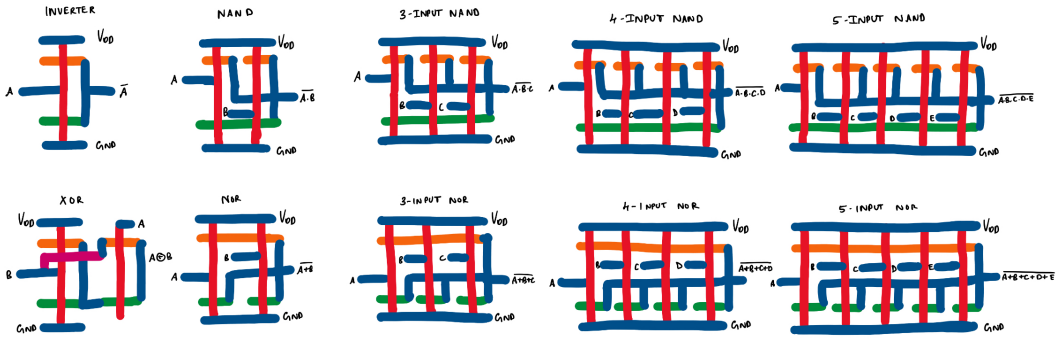
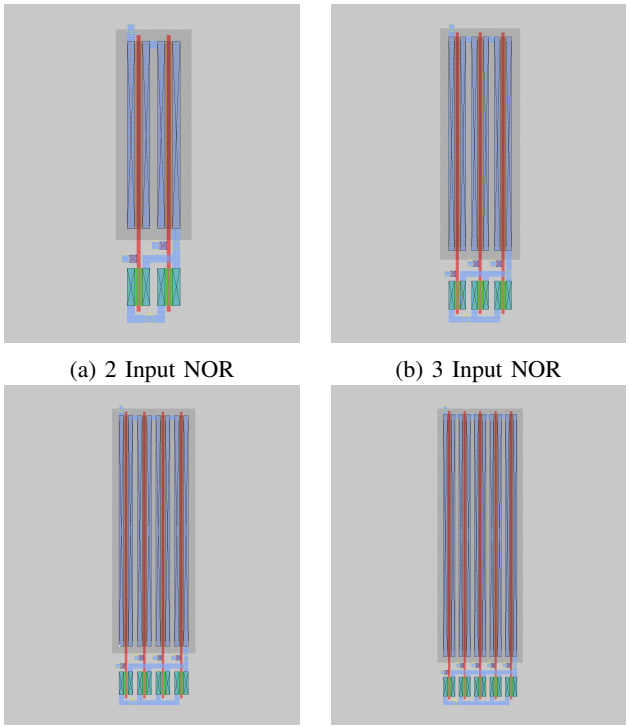


Fig. 25: Stick Diagrams

C. NOR Gate



(a) 2 Input NOR

(b) 3 Input NOR

(c) 4 Input NOR

(d) 5 Input NOR

Fig. 28: MAGIC Layout of NOR Gates

D. XOR Gate

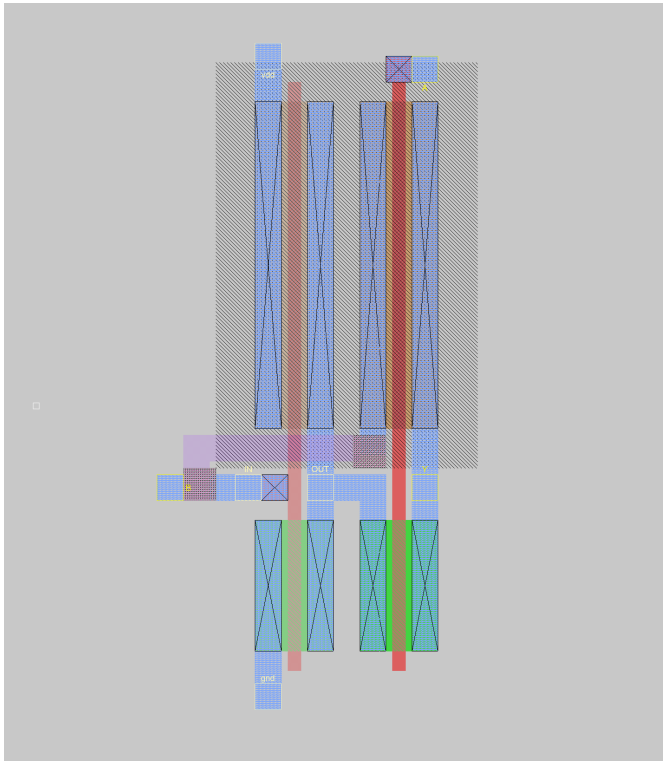


Fig. 29: MAGIC Layout of XOR Gate

E. Propagate/Generate Generator

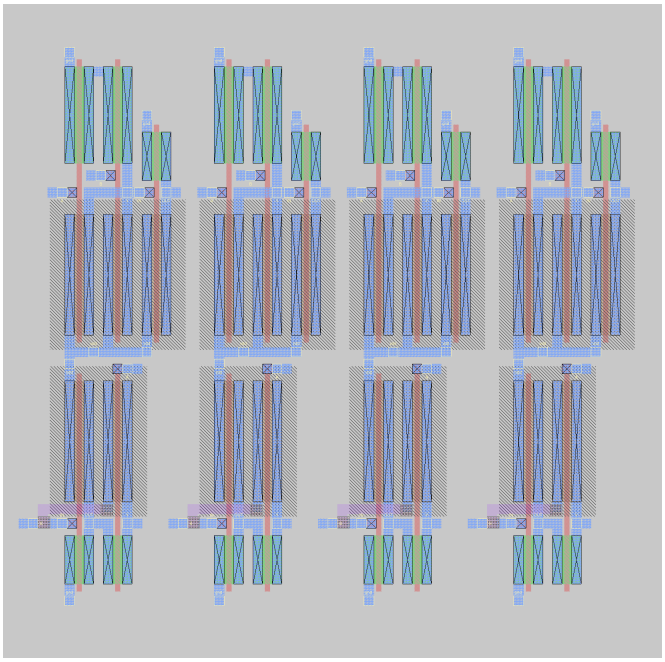


Fig. 30: MAGIC Layout of Propagate/Generate Generator

F. Carry Look Ahead Generator

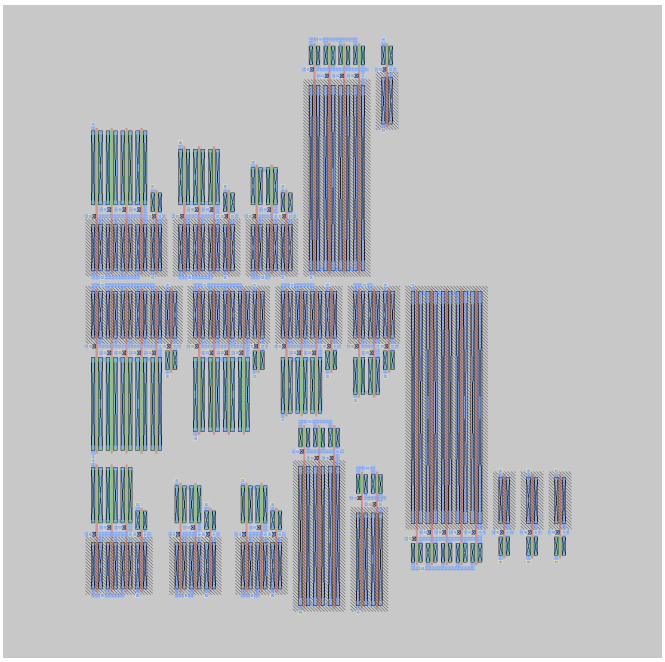


Fig. 31: MAGIC Layout of Carry Look Ahead Generator

G. Sum Generator

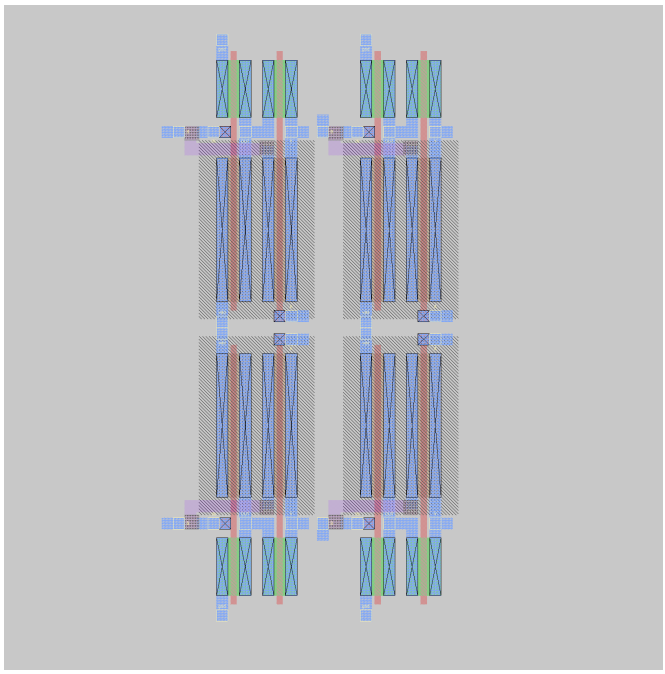


Fig. 32: MAGIC Layout of Sum Generator

H. D Flop Flop

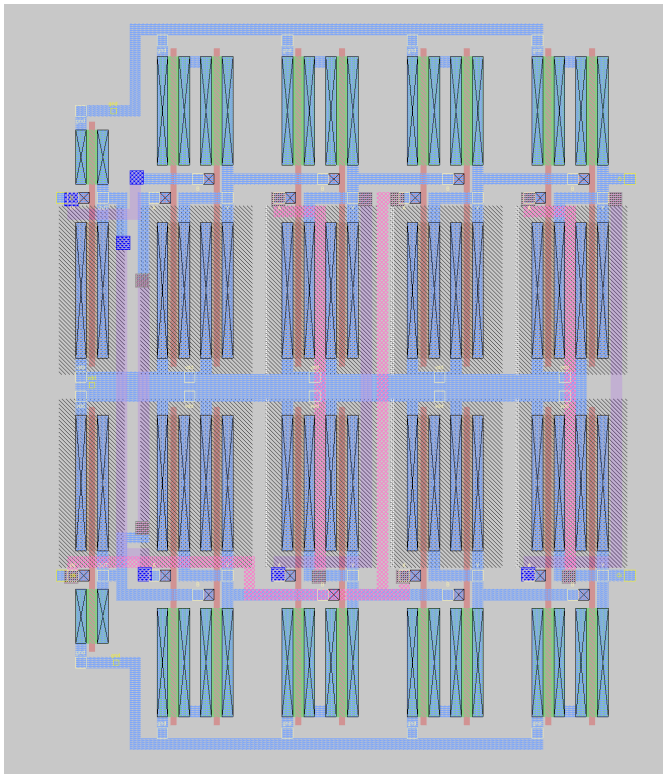


Fig. 33: MAGIC Layout of CMOS D Flip Flop

1) CMOS Implementation:

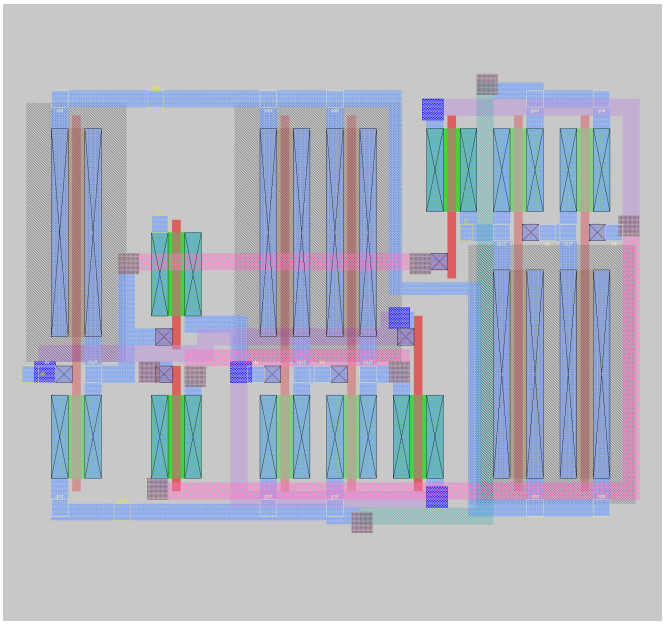


Fig. 34: MAGIC Layout of Optimized D Flip Flop

2) Optimized Implementation:

I. Full Circuit

The size of the final circuit comes out to be $103.5\mu m \times 63.18\mu m$.

VI. Post Layout Simulation

A. Inverter

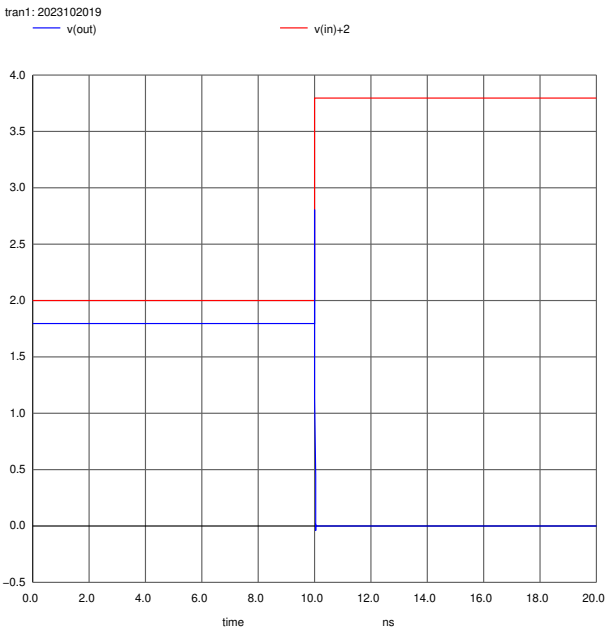
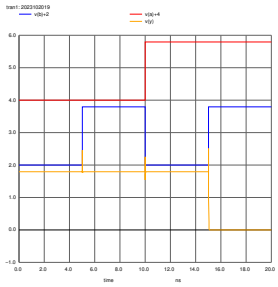
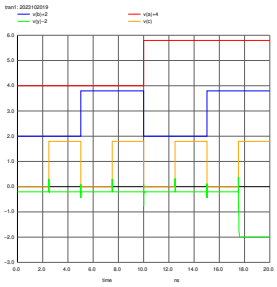


Fig. 36: Post Layout NGSPICE Plot of Inverter

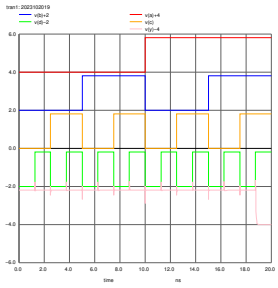
B. NAND Gate



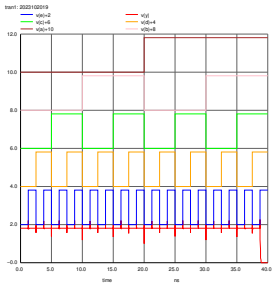
(a) 2 Input NAND



(b) 3 Input NAND



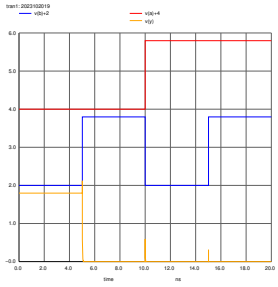
(c) 4 Input NAND



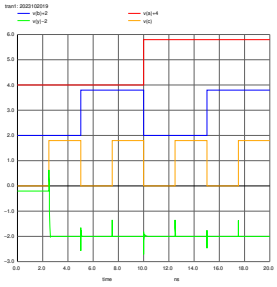
(d) 5 Input NAND

Fig. 37: Post Layout NGSPICE Plot of NAND Gates

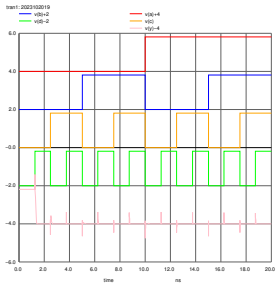
C. NOR Gate



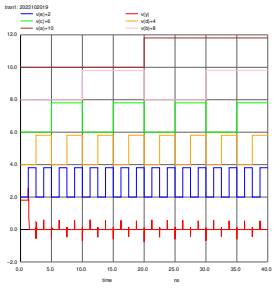
(a) 2 Input NOR



(b) 3 Input NOR



(c) 4 Input NOR



(d) 5 Input NOR

Fig. 38: Post Layout NGSPICE Plot of NOR Gates

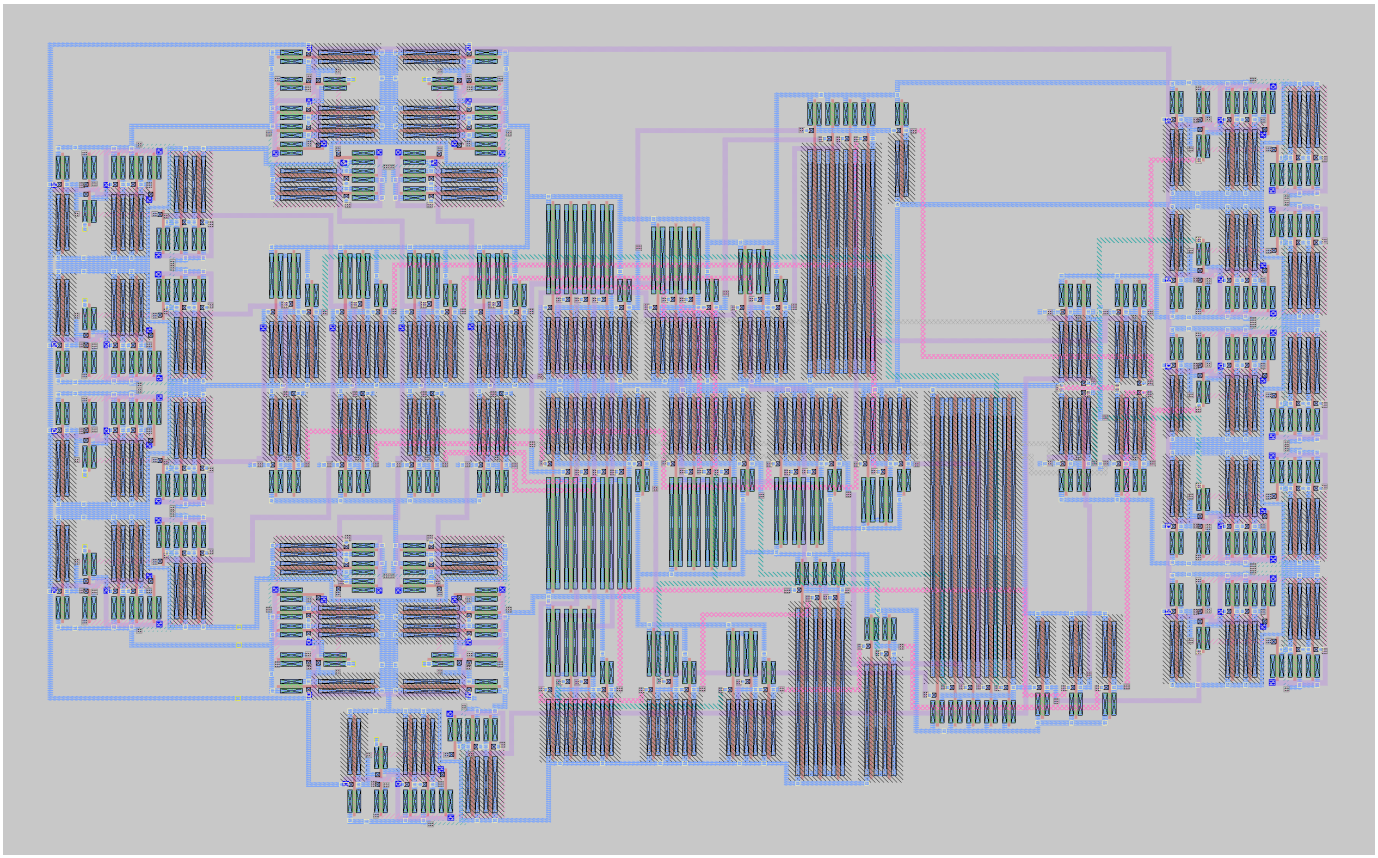


Fig. 35: MAGIC Layout of Full Circuit

D. XOR Gate

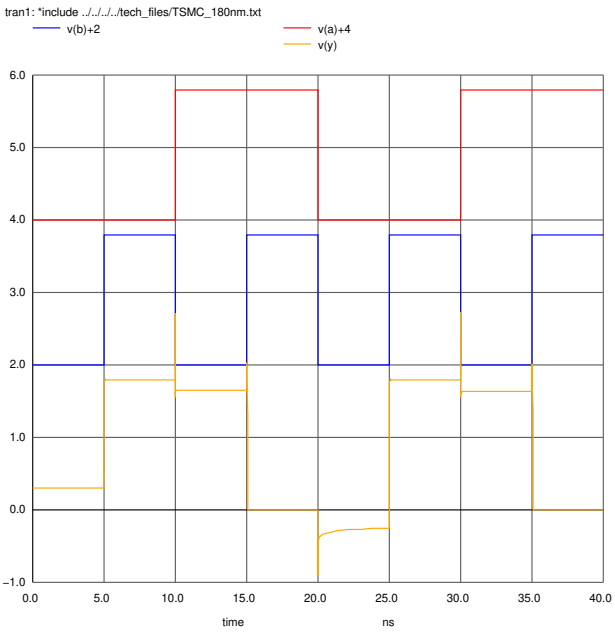


Fig. 39: Post Layout NGSPICE Plot of XOR Gate

E. D Flop Flop

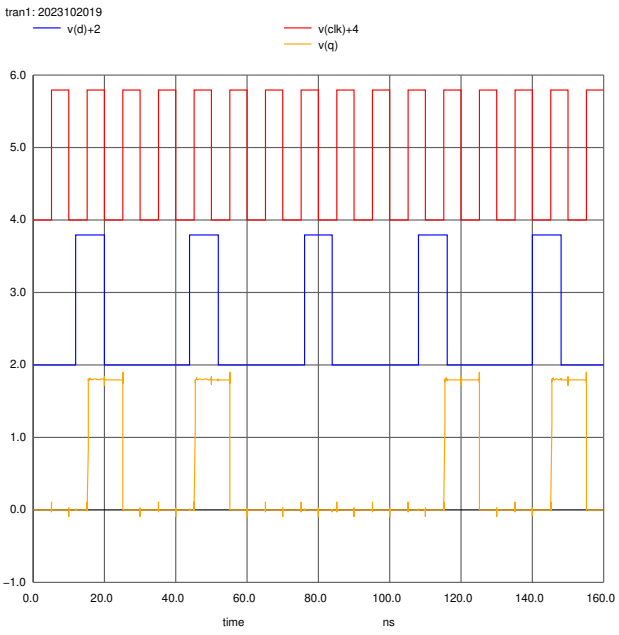


Fig. 40: Post Layout NGSPICE Plot of D Flip Flop

```
t_cq                    =   3.04861e-10s
t_su                    =   4.72107e-10s
t_h                     =   2.88159e-10s
```

Post layout the D flip flop has a slower t_{cq} but the setup and hold times are still at low ranges.

F. Full Circuit (with Load Inverters)

Load inverters are added to the outputs of the circuit to make sure the outputs are strong enough to drive the next stage.

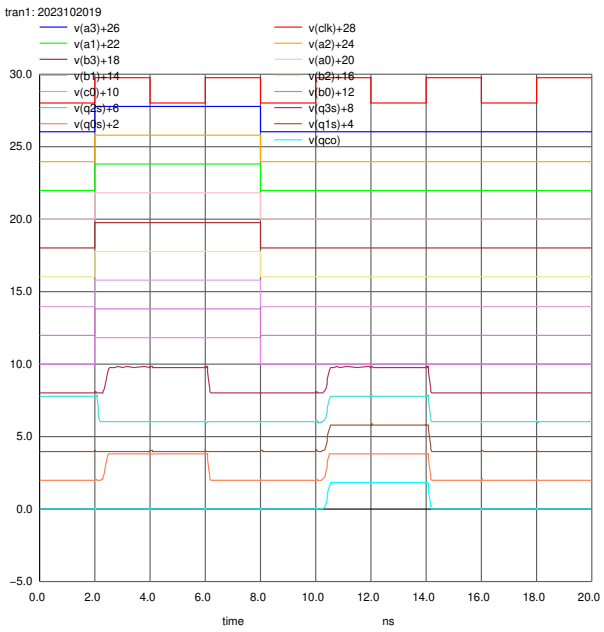


Fig. 41: Post Layout NGSPICE Plot of Full Circuit

Again here we notice that C_4 must take the most time to calculate, so we find the rise and fall time delay of it.

```
delay_c4_r              =   2.71282e-10s
delay_c4_f              =   2.11945e-10s
```

$$T_{clk} > T_{cq} + T_{pd} + T_{su} \tag{18}$$
$$T_{clk} > 10.48 \times 10^{-10}\,\text{s} \approx 1.05\,\text{ns} \tag{19}$$
$$f_{clk} \approx 0.952\,\text{GHz} \tag{20}$$

Parameter	Pre	Post
t_{cq}	$5.247 \cdot 10^{-11}$ s	$3.048 \cdot 10^{-10}$ s
t_{su}	$3.8 \cdot 10^{-10}$ s	$4.72 \cdot 10^{-10}$ s
t_h	$2.67 \cdot 10^{-10}$ s	$2.88 \cdot 10^{-10}$ s
t_{pd}	$3.1 \cdot 10^{-10}$ s	$2.712 \cdot 10^{-10}$ s
T_{clk}	$7.425 \cdot 10^{-10} \approx 0.74ns$	$10.48 \cdot 10^{-10} \approx 1.05ns$
f_{clk}	1.33 GHz	0.952 GHz

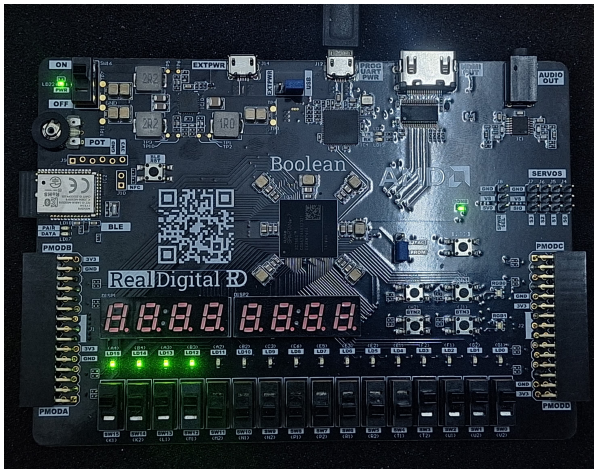
TABLE I: Comparison of parameters for pre and post layout.

VII. FPGA Simulation

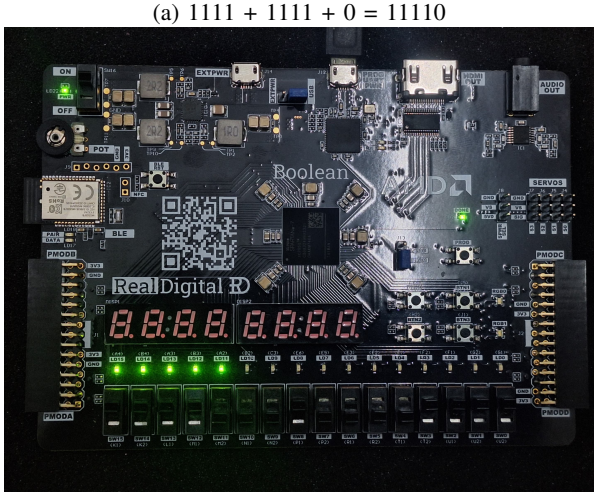
We now simulate the circuit on an FPGA board. We use the Xilinx Vivado software and Spartan-7 XC7S50-CSGA324 FPGA.

A. On Board LEDs

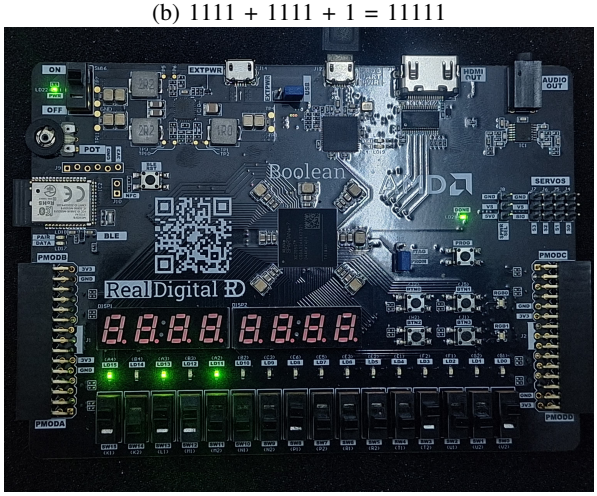
Reusing the previous verilog file, we also create a constraint file to map the inputs and outputs to the on board LEDs.



(a) 1111 + 1111 + 0 = 11110



(b) 1111 + 1111 + 1 = 11111



(c) 1011 + 1001 + 1 = 10101

Fig. 42: FPGA Simulation

B. *Waveform Visualization*

For this we make an arduino oscilloscope, and change the constraint files to map the outputs accordingly.

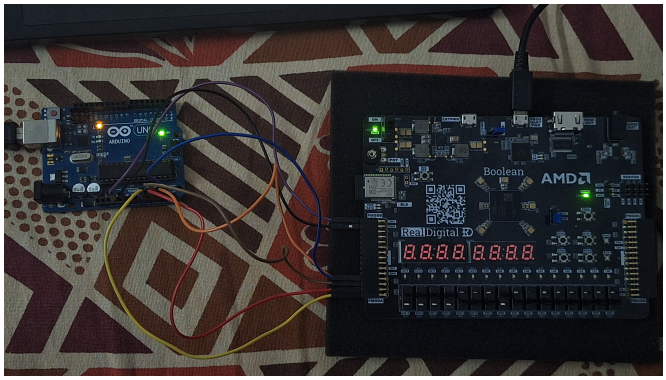


Fig. 43: Viewing Waveform of FPGA Simulation

VIII. Conclusion

We have successfully designed and simulated a 4-bit Carry Look Ahead Adder with D Flip Flops. We have also tested the circuit on an FPGA board. All my project files are available on my Github repository. [2]

Acknowledgment

I would like to express my gratitude to Prof. Abhishek Srivastava for the opportunity to work on this project and for his guidance throughout the course. I would also like to thank my Teaching Assistants for their support and help.

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.