# Optimised 4-Bit CLA Adder

Sai Rithvik
*2023012060*

***Abstract*—Addition is a basic arithmetical operation in almost all of the equipment, and optimizing the efficiency of addition is a constantly attractive research topic. Traditional CLA is constructed by XOR, AND, and OR gates. The proposed circuit uses NAND gates to replace the AND and NOT gates in CLA, it can decrease the cost of CLA and increase the speed of CLA.**

## I. General construction of Adder

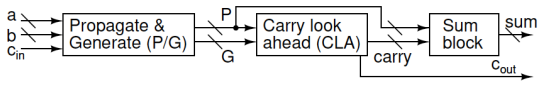


Fig. 1. Adder in a nutshell

The adder consists of three major blocks:

- **Propagate and Generate block**: generates propagate and generate bits.

$$P_i = A_i \oplus B_i$$

$$G_i = A_i \cdot B_i$$

- **CLA block**: generates carry bits using the propagate and generate bits.
- **SUM block**: generates sum bits.

$$S_i = P_i \oplus C_{i-1}$$

By focusing on optimizing each of these blocks will lead to a better functioning and a faster adder. I will explain my approach in the coming sections.

## II. Propagate and Generate Blocks

For implementation of any successful adder, the propagate and generate block must be reliable and robust.

### A. Generate Block

I have modified the 'generate block' to generate $\overline{G_i} = \overline{A_i \cdot B_i}$. This is to avoid the more transistor costly AND gate.

*1) NAND gate:* NAND gate used in this block and other subsequent blocks are made with static CMOS logic.

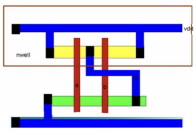

Fig. 2. Stick Diagram of 2-NAND gate

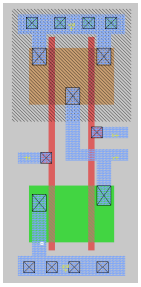

Fig. 3. layout of 2-NAND gate

### B. Propagate Block

Propagate block employs solely one gate .i.e XOR gate.

*1) XOR gate:* Finding a suitable XOR gate is critical for the proper functioning of the CLA adder. Various XOR gates have been tested, some of which are given below:

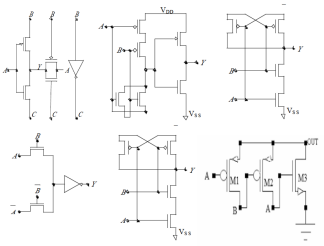

Fig. 4. Various XOR Gates

All these styles have several research papers written on them that discusses the efficacy of the implementation. But with my own experience limited by the software I used, I found the 3T style XOR gate more attractive.

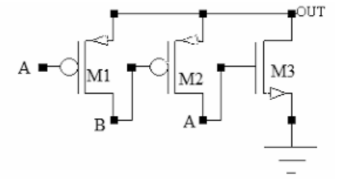


Fig. 5. XOR gate used

But they are still few drawbacks:

- The circuit is not symmetrical with respect to inputs given, leading to few inconsistencies in non ideal conditions.
- There is no $V_{dd}$ in this circuit, this would mean that this gate is heavily reliant on the inputs being 'strong' 1s and 0s.

To counteract these drawbacks, I included an inverter buffer after every XOR gate to make sure that the outputs have no discrepancies.

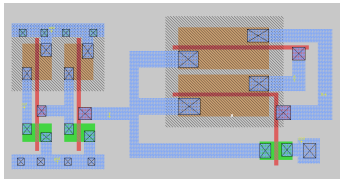

Fig. 6. XOR layout

For sizing:
$W_p = 50\lambda$, $W_n = 10\lambda$

## III. CLA Block

It is imperative to keep the transistor count in mind before designing the CLA block, as it is the most resource intensive block. A typical CLA block is made using AND and OR gates. This design disregards the fact that an AND gate is more transistor costly than a NAND gate.

Therefore, a modification is needed. A CLA that only uses NAND gates and reduces the number of transistors needed significantly.

To implement this design, we represent the carry bits as:

$$C_1 = G_0$$

$$C_2 = \overline{\overline{G_1} \cdot \overline{P_1 G_0}}$$

$$C_3 = \overline{\overline{G_2} \cdot \overline{P_2 G_1} \cdot \overline{P_2 P_1 G_0}}$$

$$C_4 = \overline{\overline{G_3} \cdot \overline{P_3 G_2} \cdot \overline{P_3 P_2 G_1} \cdot \overline{P_3 P_2 P_1 G_0}}$$

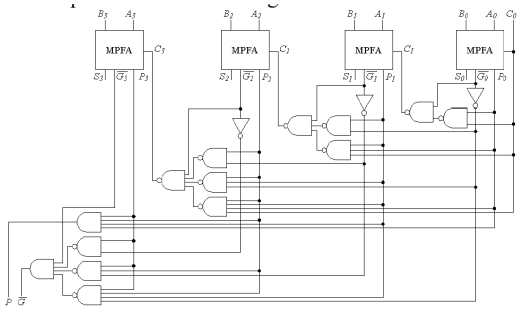


Fig. 7. Modified CLA block

### A. NAND gates

For this implementation we would require a variety of NAND gates namely:

- 2-NAND gate
- 3-NAND gate
- 4-NAND gate

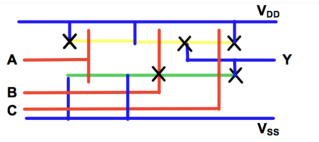


Fig. 8. 3-NAND gate stick diagram

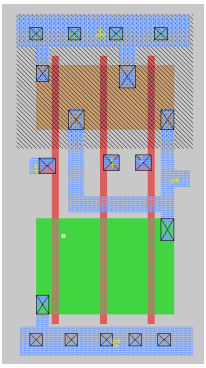

Fig. 9. 3-NAND gate layout

For sizing 3-NAND:
$W_p = 20\lambda$, $W_n = 30\lambda$.

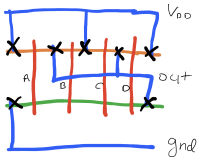

Fig. 10. 4-NAND gate stick diagram

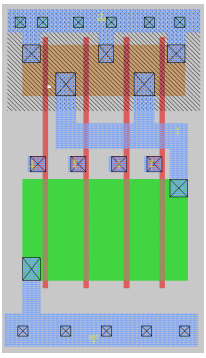

Fig. 11. 4-NAND gate layout

For sizing 4-NAND:
$W_p = 20\lambda$, $W_n = 40\lambda$.

## IV. Sum Block

Finding the Sum after finding the carry and propagate bits is easy. We only need to XOR both these bits to obtain our Sum.

$$S_i = P_i \oplus C_{i-1}$$

Note that $S_0 = P_0$ and $S_4 = C_4$.

## V. 4-Bit Adder

By joining these 3 blocks we create a 4-Bit combinational adder.

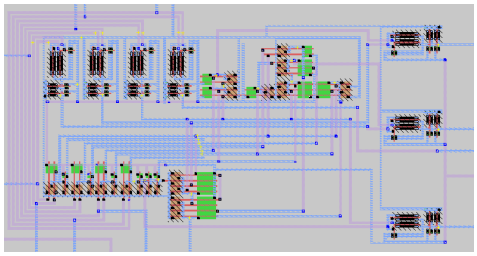

Fig. 12. 4-bit adder

### A. Performance

To characterize the maximum delay, i made use of following inputs:
A = 0 0 0 pulse
B = 1 1 1 1

S = 1 0 0 0 0 / 0 1 1 1 1 (pulsing)
and then measured $t_{pd}$ from $A_1$ from $C_4$.

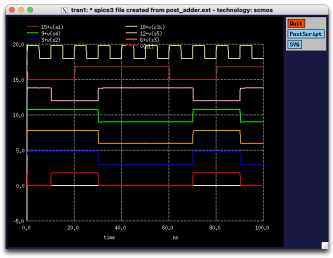

Fig. 13. Output plots

```
tpdr            =  5.760610e-10 targ=  2.084529e-08 trig=  2.026923e-08
tpdf            =  5.823147e-10 targ=  6.090734e-08 trig=  6.032502e-08
tpd             =  5.79188e-10
```

Fig. 14. pre-layout delay

```
tpdr            =  4.565199e-10 targ=  2.072223e-08 trig=  2.026571e-08
tpdf            =  4.954823e-10 targ=  6.081163e-08 trig=  6.031615e-08
tpd             =  4.76001e-10
```

Fig. 15. post-layout delay

Interestingly, the post layout delay is lesser than pre layout delay.

## VI. Adding D flip flops

The project does not end at the adder however. We must be able to sequentialise the circuit with the help of D flip flops.

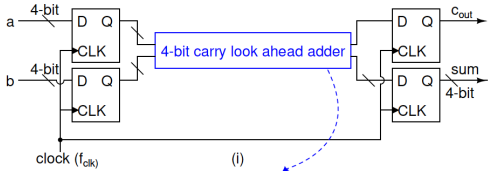


Fig. 16. The total circuit

In this circuit we are giving an additional input 'clock'. This clock will determine the time period in which outputs are expected. The outputs must appear at the next rising edge of the edge where inputs are given.

## VII. D flip flop

There are many logic styles with which a D flip flop can be built. Some which are given below.

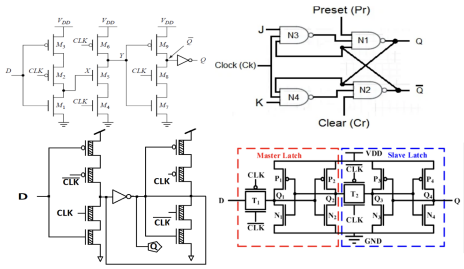


Fig. 17. Some styles for D flip flop

I chose TSPC style, for the convenience of having only one clock phase.

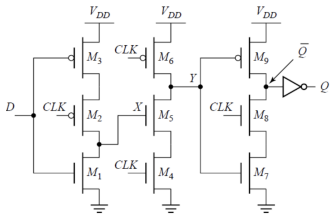


Fig. 18. TSPC style

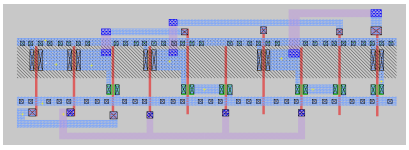

Fig. 19. Dff layout

For sizing:
$W_p = 20\lambda$, $W_n = 10\lambda$

### A. *Performance*

- $t_{pcQ}$:
  I measured $t_{pcQ}$ by manipulating clk and one of the input bits and then measuring the delay between clk and output.

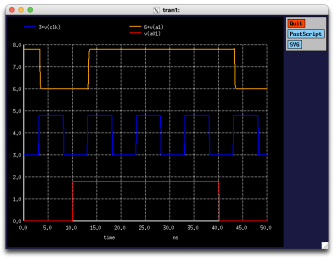

Fig. 20. $t_pcQ$ measurement

```
tpdf              =  2.731593e-10 targ=  3.323159e-09 trig=  3.050000e-09
tpdr              =  2.221970e-10 targ=  1.327220e-08 trig=  1.305000e-08
tpcq              =  2.47678e-10
```

Fig. 21. pre layout $t_{pcQ}$

```
tpdf              =  2.519472e-10 targ=  3.301947e-09 trig=  3.050000e-09
tpdr              =  2.174605e-10 targ=  1.326746e-08 trig=  1.305000e-08
tpcq              =  2.34704e-10
```

Fig. 22. post layout $t_{pcQ}$

Here we again see that the post layout delay is lesser than the pre layout delay.

- $t_{su}$:
  I found the $t_su$ by manipulating clk and one of input bits. I shifted the selected input bit to the left of clk till I found the expected output.

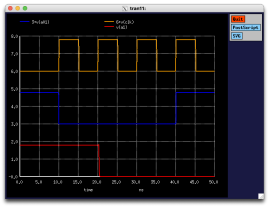

Fig. 23. When diff between clk and input is less than $t_{su}$

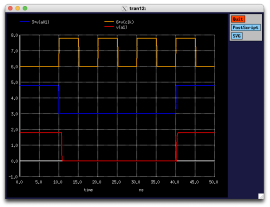

Fig. 24. When diff between clk and input is more than $t_{su}$

pre-layout $t_{su}$:

$$t_{su} \approx 0.07ns$$

post-layout $t_{su}$:

$$t_{su} \approx 0.06ns$$

- $t_h$: since we are dealing with a TSPC style D flip flop, the $t_h$ is very very less and near zero.

$$t_h \approx 0$$

## VIII. Clock Speed Constraints

We can find the constraints of $T_{clk}$ by observing the delays we have already calculated, i.e. $t_{pcQ}$, $t_{pd}$, $t_{su}$ and $t_h$.

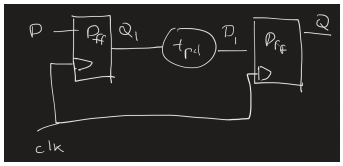


Fig. 25. Circuit diagram

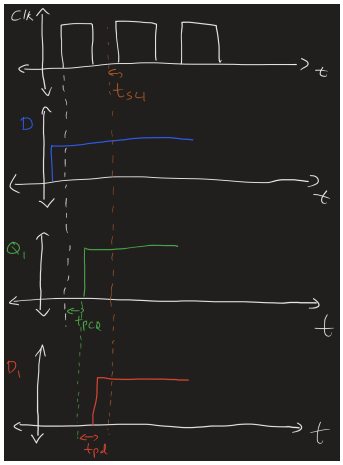


Fig. 26. Timing Diagram for $t_{su}$ constraint

Therefore,

$$t_{pcQ} + t_{pd} + t_{su} < T_{clk}$$

implies,

$$f_{clk} < \frac{1}{t_{pcQ} + t_{pd} + t_{su}}$$

| | pre layout | post layout |
|---|---|---|
| $t_{pd}$ | 2.477e-10 | 2.347e-10 |
| $t_{pcQ}$ | 5.792e-10 | 4.76e-10 |
| $t_{su}$ | 7e-11 | 6e-11 |
| $f_{max}$ | 1.11e09 | 1.29e09 |

TABLE I
COMPARISON BETWEEN PRE AND POST LAYOUT

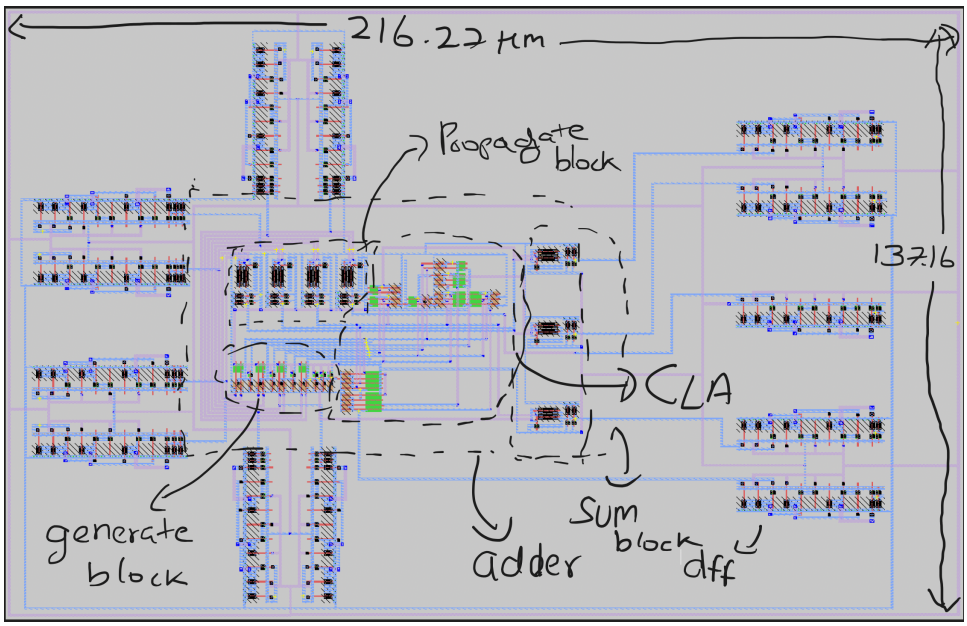


Fig. 27. Total layout

## IX. Total Layout

My total layout area is $\approx 2.966e-08m^2$.
The area can be more optimized, by placing the D flip flops in a better fashion, but I couldn't pursue it due to time constraint.

## X. FPGA Implementation

To test the running, I have given these inputs:
A = 1 0 0 1
B = 0 1 0 1

S = 0 1 1 1 0

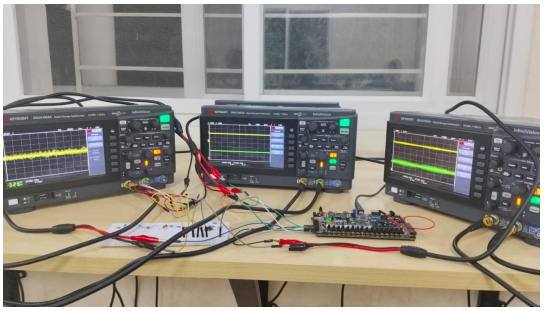

Fig. 28. Setup

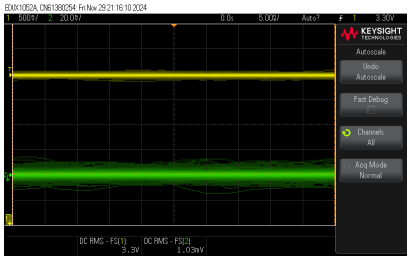


Fig. 29. $S_0$ and $S_1$ bits

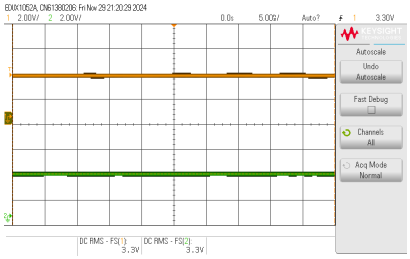


Fig. 30. $S_2$ and $S_3$ bits

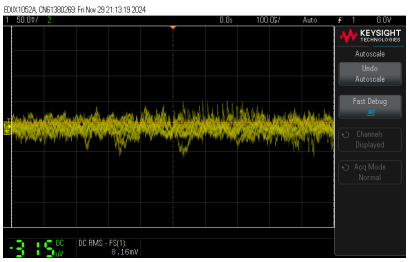


Fig. 31. $C_{out}$ bit

## XI. Some test inputs

- A = 1 0 1 0
  B = 0 0 1 1

  S = 0 1 1 0 1

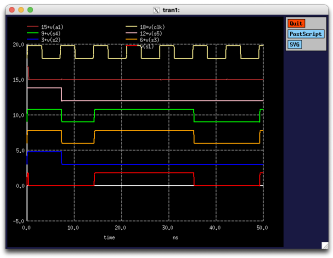

Fig. 32. Output pre layout

- A = 1 1 0 0
  B = 0 1 1 0

  S = 1 0 0 1 0

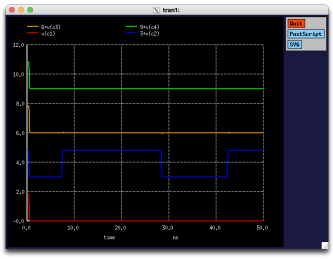

Fig. 33. CLA output pre layout

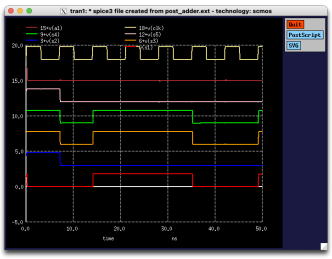

Fig. 34. Output post layout

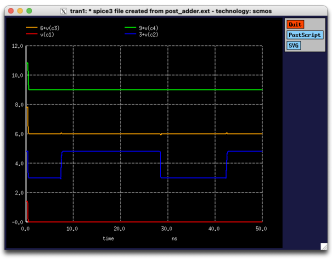

Fig. 35. CLA output post layout