Design and Implementation of a 5-bit Carry Lookahead Adder in 180 nm CMOS Technology

DEVANG BORDOLOI

International Institute of Information Technology Hyderabad, India
devang.bordoloi@research.iiit.ac.in

2025122003

INTRODUCTION

Design of a 5-bit Carry Look-Ahead (CLA) Adder using static CMOS and TSPC logic has been presented. Performance parameters of the proposed 5-bit CLA architecture have been simulated validated by designing a layout and extracting parameters to conduct spice simulations. MAGIC layout design tool and NGSpice spice circuit simulator engine were used to extract de sign parameters and performs simulations for 180 nm technology. The design was analysed in terms of average power consumption, propagation delay and power delay product (PDP).

PROPOSED DESIGN

The proposed 5-bit carry look-ahead adder (CLA) is designed to meet high-speed performance requirements while ensuring accurate functionality in synchronous operation. The primary objective of the design is to compute the output sum and carry within one clock cycle, ensuring that the output is valid at the next rising edge of the clock after the inputs are latched. The detailed analysis and the exact design of each block are presented in the design methodology subsection.

A. Architecture Overview

The CLA adder is designed using a modular approach, consisting of the following blocks:

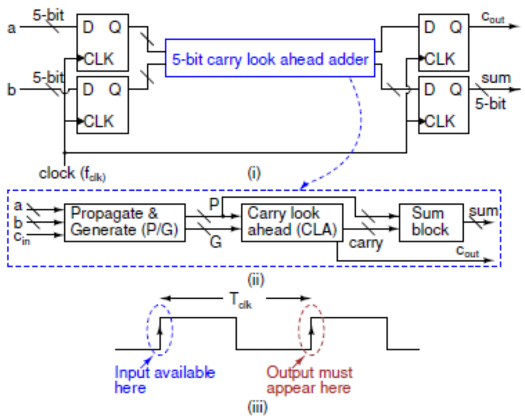
• D-Flip-Flops: To latch inputs and intermediate signals, ensuring synchronization with the clock.

• Propagate and Generate Logic: For computing propagate (pi = ai ⨁bi) and generate (gi = ai · bi) signals.

• Carry Computation Logic: To compute carry signals in parallel, minimizing the delay associated with sequential carry propagation.

• SumLogic: To compute the sum bits using the propagate and carry signals.

Each block is implemented with carefully chosen logic styles and transistor sizing to optimize performance in terms of speed, power, and area.



1.D FLIP FLOP

A. Functionality :

A positive edge-triggered D flip-flop is a sequential digital circuit that transfers the data input (D) to its output (Q) only during the rising edge of the clock signal. It is commonly used in synchronous systems for data storage, synchronization, and timing control.

• **Edge-Sensitivity**: The flip-flop responds only during the positive edge (rising edge) of the clock signal, i.e., the transition from 0 to 1.

• **Data Storage**: The value at the input D is sampled at the rising edge of the clock and stored internally. This value appears at the output Q and remains stable until the next rising edge.

• **Output Behavior**:– If D = 1 at the positive edge, Q becomes 1.– If D = 0 at the positive edge, Q becomes 0

clock	D	Q(n+1)
1	0	0
1	1	1
0	X	Q(n)

B. D-Flip-Flop Design

Static D flip-flop is very slow when it has to be used in a MHz frequency range [1], so to avoid that, a TSPC D flip f lop in [13] is selected. TSPC flip-flops operate with a single clock phase, which eliminates the need for complementary clock signals typically required in other designs like master slave flip-flops. This reduction in clocking circuitry minimises dynamic power dissipation. Unlike other styles, TSPC flip flops require only a single-phase clock. This simplifies the clock distribution network and reduces clock skew, leading to more robust timing performance. TSPC flip-flops utilise a simplified architecture that requires fewer transistors compared to traditional master-slave flip-flops. This not only reduces the area but also decreases parasitic capacitances, further improving speed and power efficiency.

TABLE I
COMPARISON OF TSPC FLIP-FLOP WITH OTHER STYLES

Feature	TSPC FF	Master-Slave FF	Dynamic FF
Clock Phases	Single	Two	Single
Power Efficiency	High	Moderate	Moderate
Speed	High	Moderate	High[1]
Transistor Count	Low	High	Moderate
Area	Low	High	Moderate
Complexity	Low	High	High[2]

[1] With potential glitches
[2] Due to precharge/discharge

However there are numerous glitches in the intermediate nodes, due to that the overall performance of the circuit gets degraded. Thus a modified version of the TSPC D flip flop which is presented in [] is used in this project.

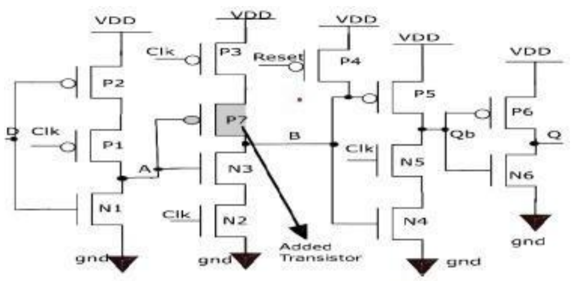


Fig.3. Positive edge triggered MTSPC DFF.

Transient Response

Transistor sizing is done with respect to the minimum-sized 2W/W inverter. The width of the PMOS and NMOS of the minimum-sized inverter is W = 20λ, where λ = 0.09µm. Thus, the sizes are:

• P1, P2, P3, P7 have a sizing of 4W each

• N1 and N6 have a sizing of W each

• N2, N3, N4, N5, P5, P6 have a sizing of 2W each

C. NGSPICE Stimulation :

Inputs are provided to Flip flop as follows

A Clock signal of period 10ηs with Rise time and Fall time of 0s A Input signal is delayed version of Clock signal. The delay is changed for requirement.

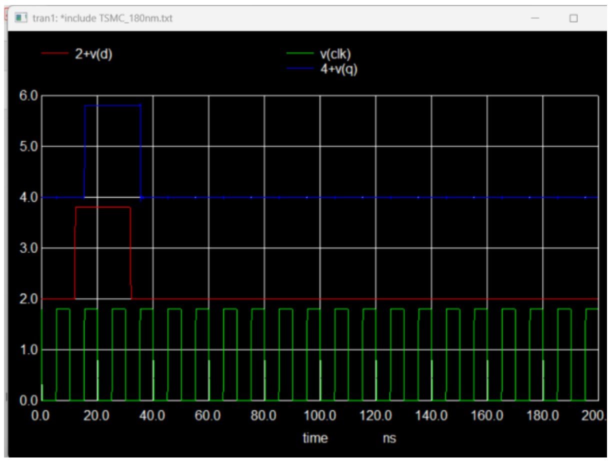


FIG-NGSPICE D Flip-FLop O/P Waveform

Performance parameter	TSPC D-flipflop
T pcq	36.84ps
Setup time	89.69ps
Hold time	16ns(approx)

D. Post Stimulation :

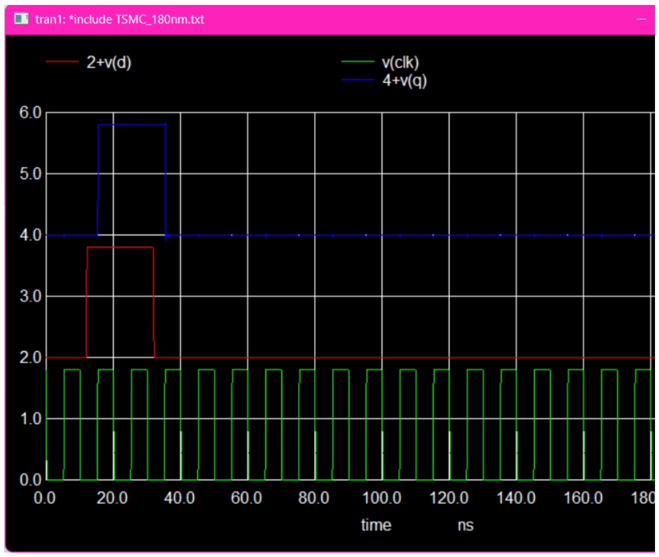


FIG-Post Layout D Flip-FLop O/P Waveform

Performance parameter	TSPC D-flipflop
Tpcq	141.77ps
Setup time	23.3ns
Hold time	16.75ns (approx)

Upon comparing the schematic (pre-layout) and extracted (post-layout) simulations of the D Flip-Flop, it is observed that all timing parameters—Propagation Delay, Setup Time, and Hold Time have increased. This increase is attributed to the introduction of parasitic elements in the layout view which were absent in the ideal schematic view.

II. 2INPUT NANDGATE

A. Functionality:

The following truth table shows the NAND for all possible combinations of inputs a and b.

a	b	Y=a.b’
0	0	1
0	1	1
1	0	1
1	1	0

B Designing :

A2-Input NAND Gate can be designed using Static CMOS Logic . It is designed to built using Static CMOS 2-input NAND gate. 2-input NAND gate to have delay close to Delay of Minimum Sized Inverter it is sized as follow:

Size of PMOS of NAND Gate: (20∗λ)

Size of NMOS of NAND Gate: (20∗λ)

III. 3 INPUT NAND GATE

A. Functionality: The truth table displays the NAND operation for all possible combinations of inputs a, b and c.

a	b	c	Y=a.b.c’
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

:

B. D esigning:

3-Input NAND Gate can be designed using Static CMOS Logic, implemented with a Static CMOS 3-input NAND gate. To achieve delay comparable to a Minimum Sized Inverter, the components are sized as:

- Size of PMOS of NAND Gate: (20 * λ)
- Size of NMOS of NAND Gate: (30 * λ)

IV.4 INPUT NAND GATE

A. Functionality: The following truth table shows the NAND for all possible combinations of inputs a, b , c and d

a	b	c	d	Y=a.b.c.d’
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

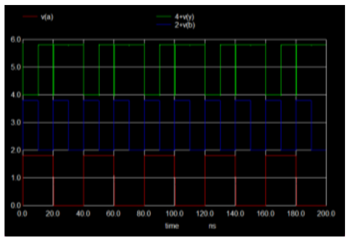
B. Designing:

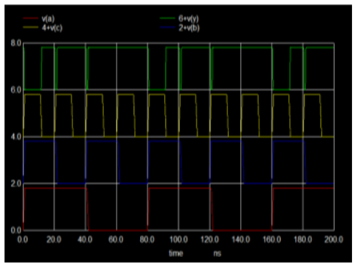
A 4 Input NAND Gate can be designed using Static CMOS Logic . It is designed to built using Static CMOS4-input NAND gate. 4-input NAND gate to have delay close to Delay of Minimum Sized Inverter it is sized as follow:

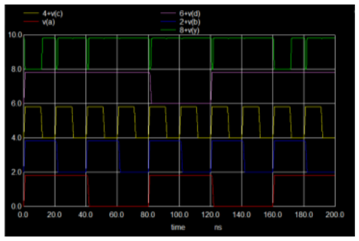
Size of PMOS of NAND Gate: (20∗λ) Size of

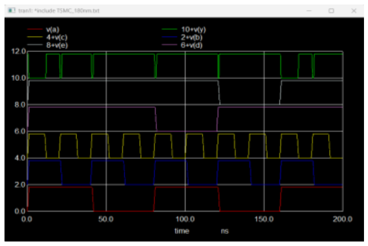
NMOS of NAND Gate: (40∗λ)

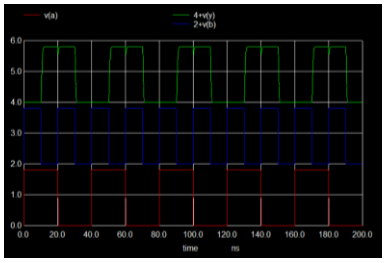
TEST PLOTS FOR MODULES











V. 5 INPUT NAND GATE

A. Functionality: The 5 input NAND is 0 only when all the 5 inputs are zero simultaneously

B. Designing: A 5 input NAND gate can be designed using Static CMOS Logic. It is designed to build using It is designed to built using Static CMOS4-input NAND gate. 4-input NAND gate to have delay close to Delay of Minimum Sized Inverter it is sized as follow:

Size of PMOS of NAND Gate: (20∗λ)

Size of NMOS of NAND Gate: (50∗λ)

VI. 2 INPUT XOR GATE

A.Functionality:

The following truth table shows the XOR for all possible combinations of inputs a and b.

a	b	Y=a.b’+b’a
0	0	0
0	1	1
1	0	1
1	1	0

B Designing :

A2-Input XOR Gate can be designed using Static CMOS logic. . It is designed to built using Static CMOS logic with A, B, A’ and B’ as inputs. 2-input XOR gate to have delay close to Delay of Minimum Sized Inverter it is sized as follow:

Size of PMOS of XOR Gate: (λ)

Size of NMOS of XOR Gate: (λ)

VII. Propagate and Generate BLOCK

A. Functionality:

The Propagation(p) and Generate(g) signal are defined as follows:

The Propagation signal indicates whether the Carry-in (cin) will propagate to the Carry-out (cout): p =a ⊕ b

• When p=1 ,any incoming Carry(cin) will propagate to the next stage. •When p=0, the Carry-in is blocked from propagating. The generate signal indicates whether a Carry is generated by the addition of the inputs a and b, irrespective of the Carry-in: g=a · b •When g=1, a carry is generated in the current stage

. •When g=0 ,no carry is generated by the current stage. These signals are widely used in carry look-ahead adders (CLA) for efficient carry computation, reducing the delay caused by the Ripple-carry effect in traditional adders. The following truth table shows the values of propagation(p) and generate(g) signals for all possible combinations of inputs a and b

a	b	P= a ⊕ b	G= a . b
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

VII. CLA BLOCK

A. Functionality: Carry Look-Ahead Adder (CLA) Characteristics:

Propagate (Pi) and Generate (Gi) Signals:

Pi = ai ⊕ bi (Carry propagates if inputs differ) Gi=ai ·bi (Carry is generated if both inputs are1)

Carry Computation (parallel):

C1 =G0+P0C0

C2 =G1+P1G0+P1P0C0

C3 =G2+P2G1+P2P1G0+P2P1P0C0

C4 =G3+P3G2+P3P2G1+P3P2P1G0+P3P2P1P0

C5=G5+P5G4+P5P4G3+P5P4P3G2+P5P4P3P2G1

Delay Analysis: Proportional to log(n) where n = number of bits

Advantages:

1 Faster than RCA for large bit-widths (parallel computation)

2 Suitable for high-speed designs

Disadvantages:

1 More complex circuit design

2 Higher power consumption and area

Ripple Carry Adder (RCA) Characteristics:

1 Carry Propagation: Ci+1 = Gi + PiCi

2 Sequential ripple through n stages

Delay Analysis: Proportional to n bits For 4-bit adder: Totaldelay = 4 × tcarry-propagation +tsum-generation

Advantages:

1 Simple, straightforward design

2 Area and power efficient for small bit-widths

- **Disadvantages:**

Slower than CLA for large bit-widths due to sequential carry propagation.

Delay increases linearly with the number of bits.

Since we are designing a Adder so C0 = 0 . So Carry's of 4- Bit additions are as below :

C1 =G0

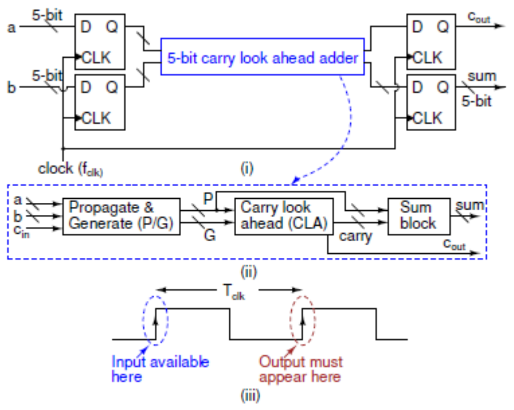
C2 =G1+P1G0

C3 =G2+P2G1+P2P1G0

C4 =G3+P3G2+P3P2G1+P3P2P1G0

C5=G5+P5G4+P5P4G3+P5P4P3G2+P5P4P3P2G1

VIII. FINAL CIRCUIT



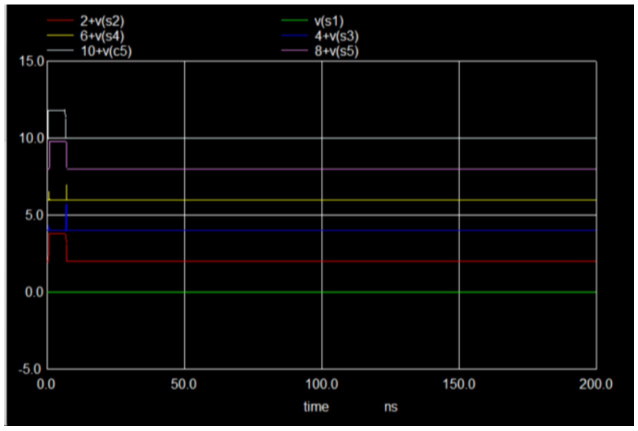
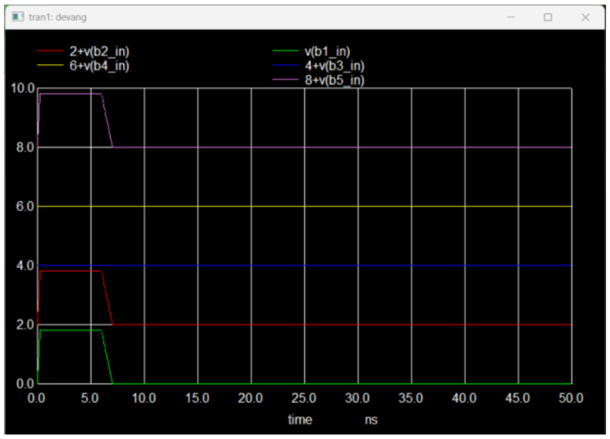


Fig. OUTPUTS SUM AND C5 AFTER OUTPUT FLIPFLOP AND CORRESPONDING DELAY



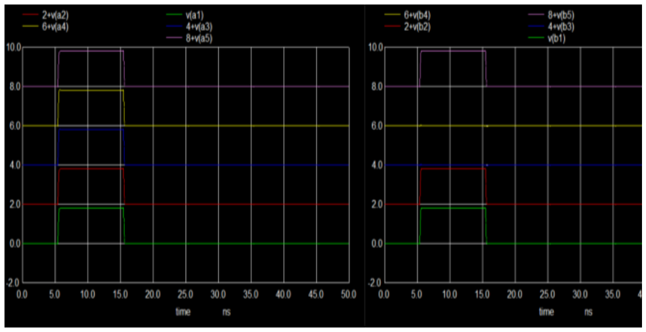
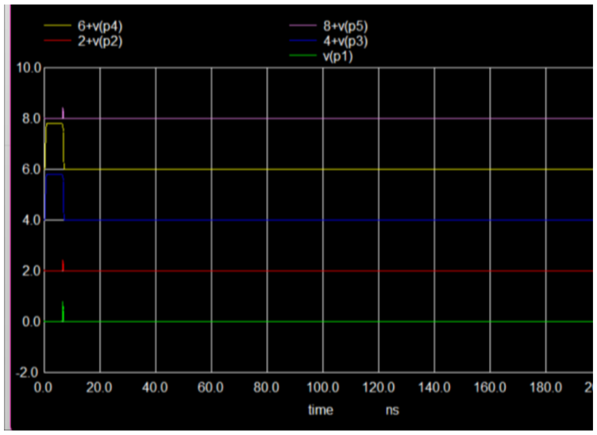


Fig. “A” INPUTS BEFORE FLIP FLOP

Fig. PROPAGATES



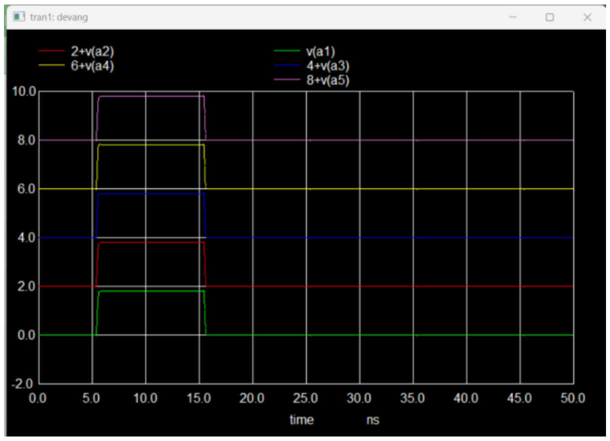


Fig. GENERGATES

Fig. “A” INPUTS AFTERE FLIP FLOP

Inputs given: a5a4a3a2a1 = 11111 & b5b4b3b2b1 = 10011 Expected Output: expected output = > c5 = 1 s5s4s3s2s1 = 10010

PRE AND POST LAYOUT COMPARISION

Using the below formula,
Worst case and Minimum clock period is calculated for both cases

$$T_{min} = T_{clk-q} + T_{adder\_delay} + T_{setup}$$

Case	PreLayout	Post layout
Worst delay	364 ps	412.6ps
Minimum time period	464 ps	581ps
frequency	2.155GHz	1.9157GHz

MAGIC LAYOUTS

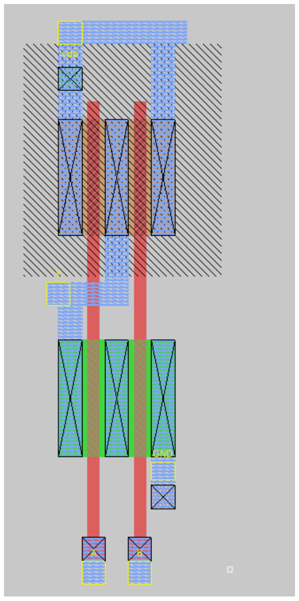


Fig:Inverter

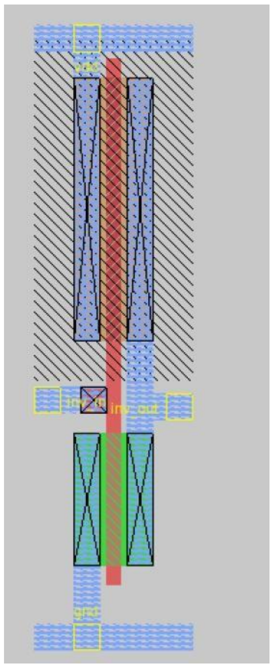
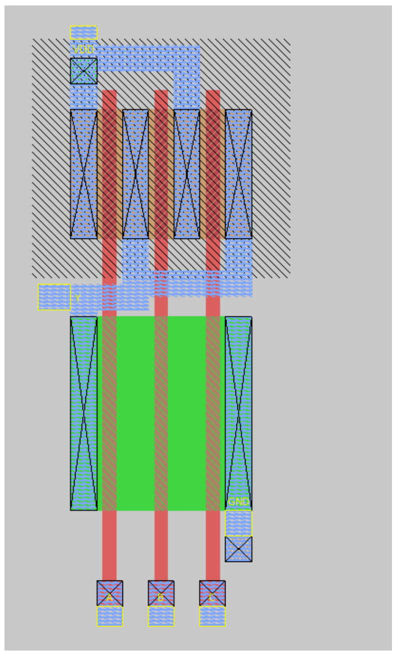


Fig: 2 input NAND Gate



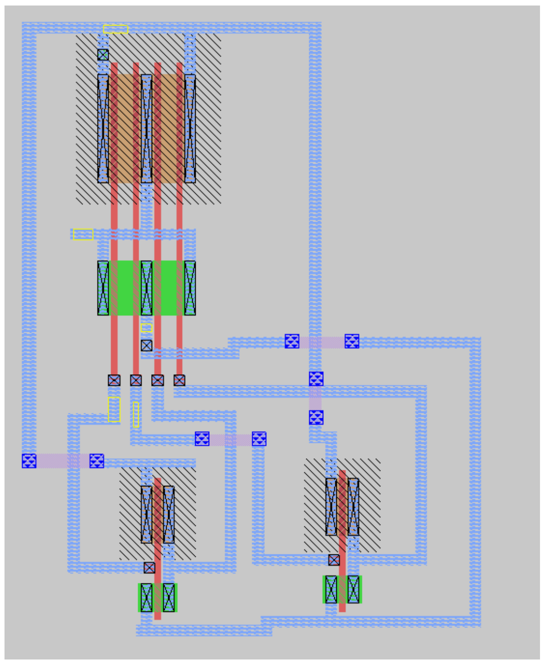
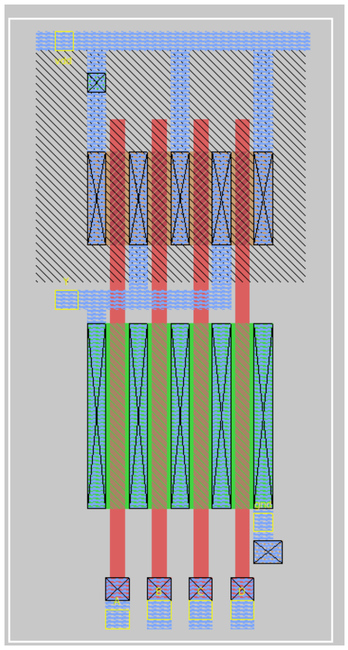


Fig: 2 input XOR gate

Fig: 3 input NAND Gate



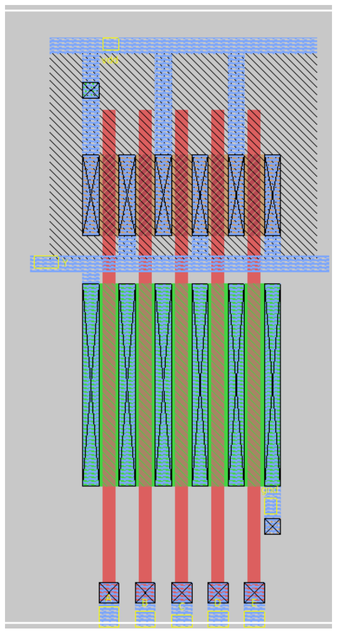


Fig: 4 input NAND Gate

Fig: 5 input NAND Gat

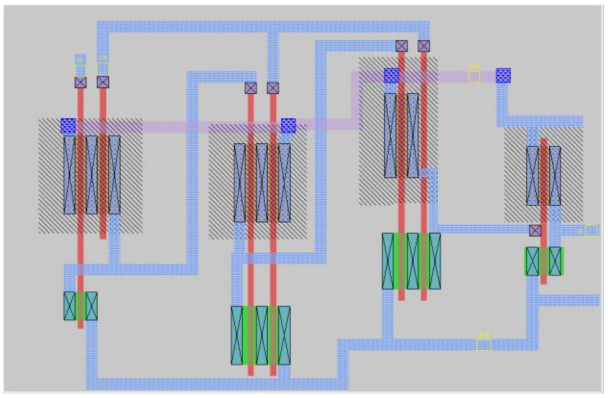


Fig: D Flip Flop (TSPC)

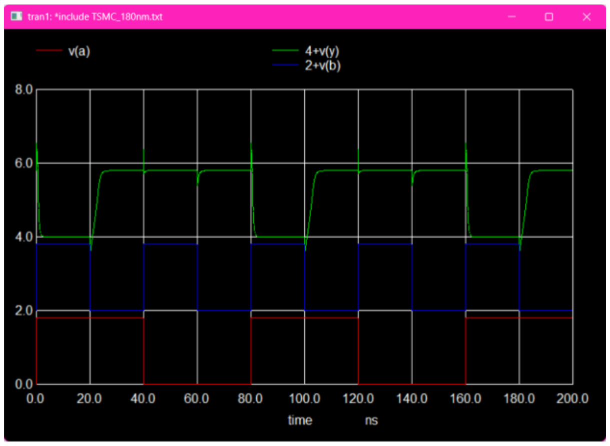


Fig: 2 input XOR gate

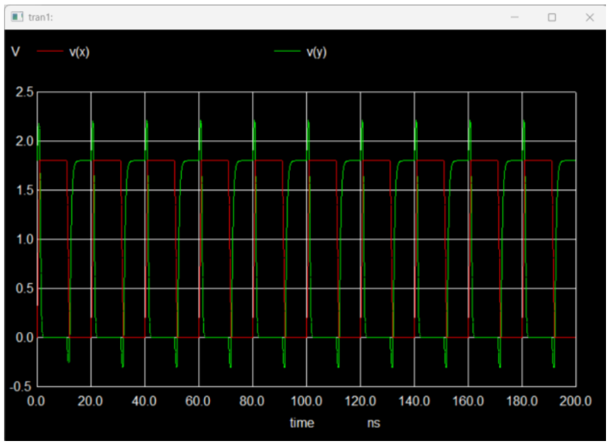


Fig: Inverter Waveform

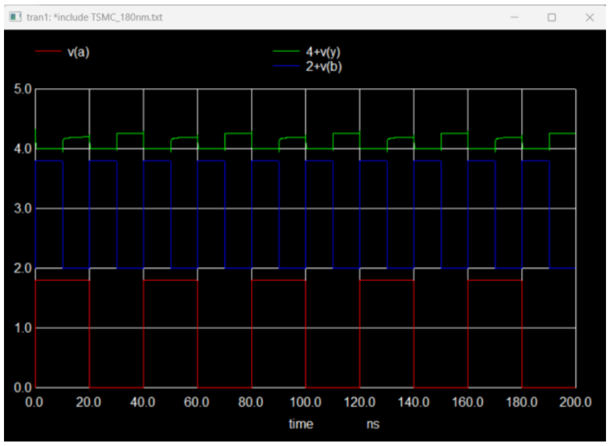
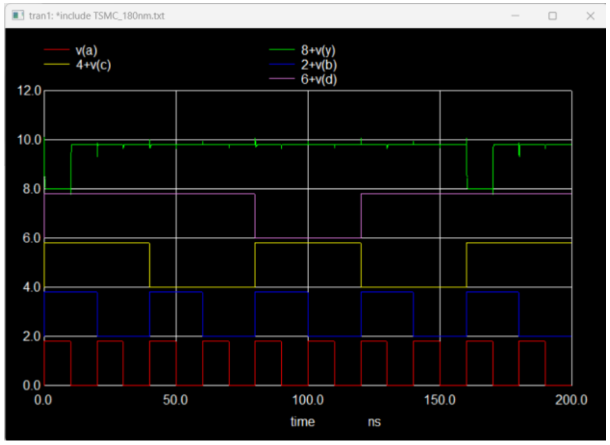


Fig: 2 input Nand Gate



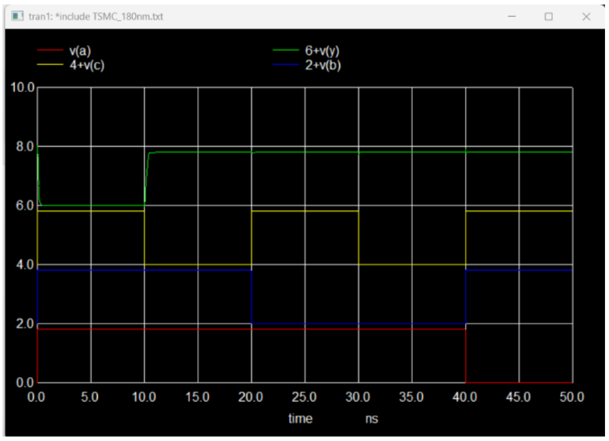
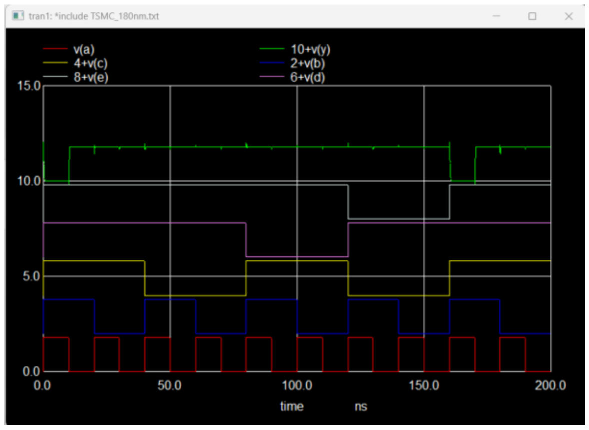


Fig : 3 input Nand gate

Fig: 4 input Nand gate

jkbFig: Inverter Waveforasdadad

asd



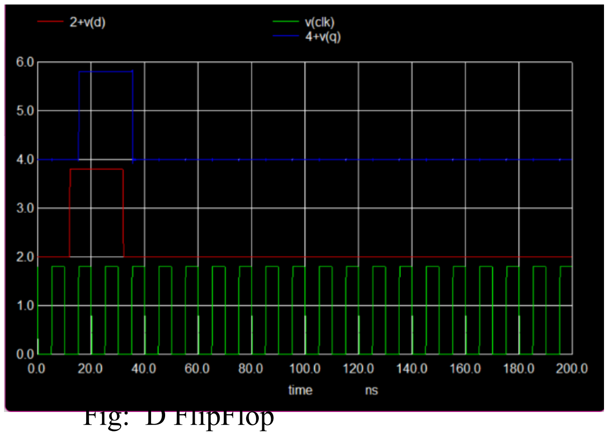
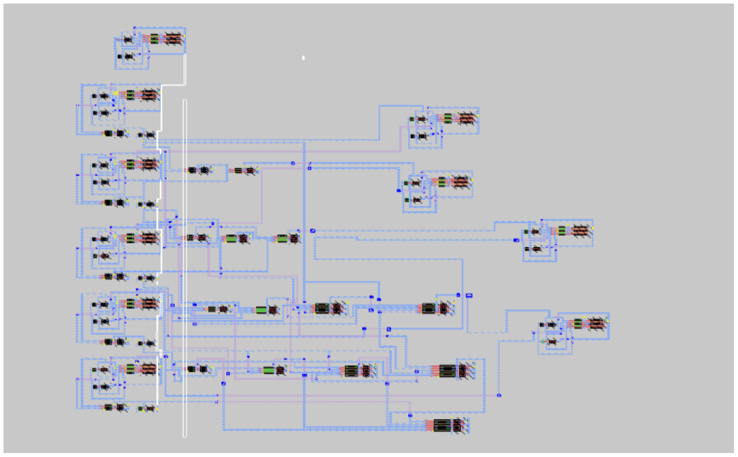


Fig: D FlipFlop

Fig: 5 input Nand gate

FLOOR PLAN



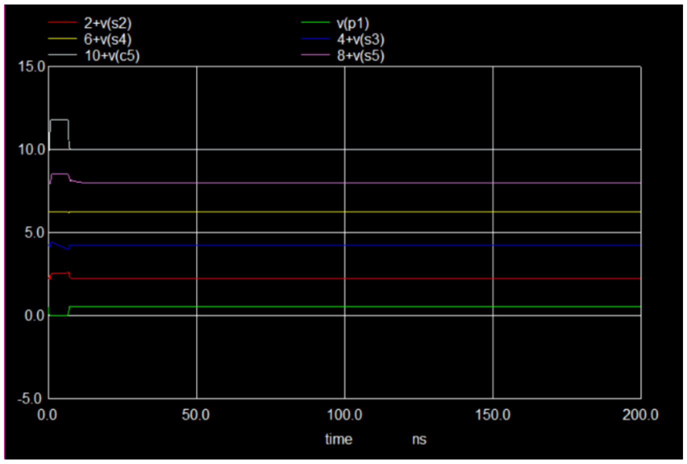
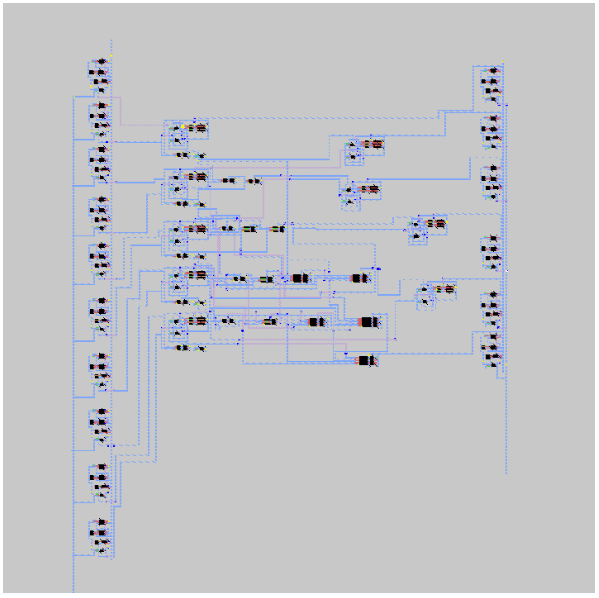


Fig: Output Waveform

horizontal pitches=170.55 microns

vertical pitches=0.99 microns



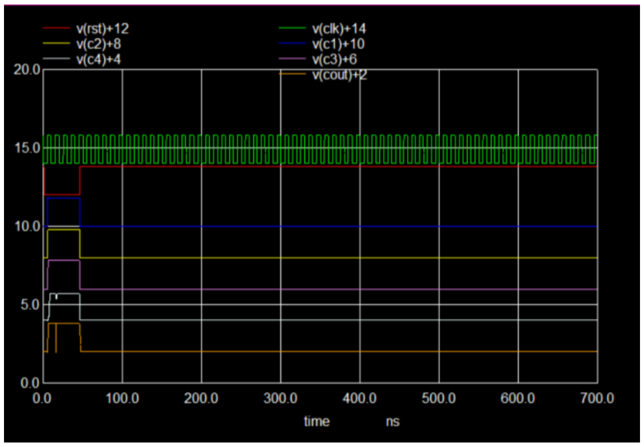
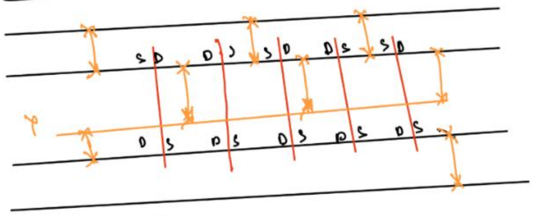


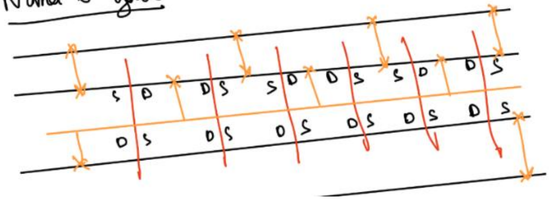
Fig: - floor plan of the layout for the complete circuit with flip flop

Stick diagrams of all unique gates

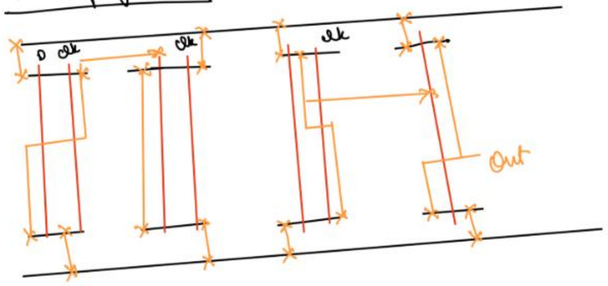
Nand 5 gate



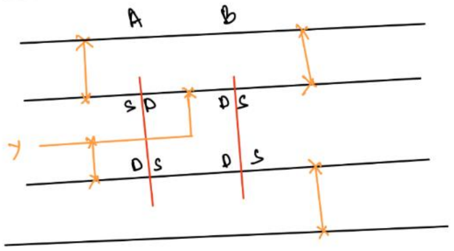
Nand 6 gate



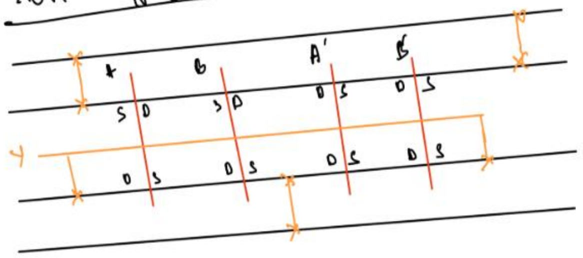
D Flip-flop



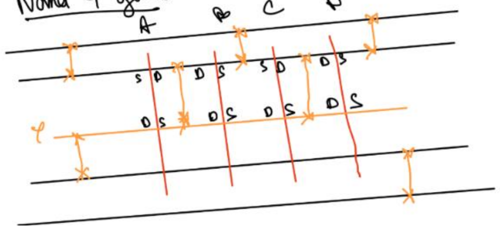
Nand 2 gate



XOR 2 i/p gate



Nand 4 gate



VERILOG HDL

This code describes a **5bit Carry-Lookahead Adder (CLA)** implemented in Verilog. The design uses flip-flops for input and output registers, making it a **pipelined** version of the adder. The CLA architecture improves addition speed by precomputing carry signals, unlike simpler ripple-carry adders, which must wait for each bit's carry to propagate sequentially.

Module-wise Breakdown

1. Input and Output Declarations:

```
input wire clk,

input wire [4:0] a, b,

output wire [4:0] sum,

output wire cout
```

- **Inputs:** Two 5-bit numbers (a and b), and a clock signal (clk).
- **Outputs:** 5-bit sum (sum) and a carry-out bit (cout).

2. Register Declarations:

```
wire [4:0] a_reg, b_reg;
```

- Temporary registers hold input values after being latched by flip-flops, ensuring stable signals during addition.

3. Flip-Flop Instantiation:

```
dff a_reg0 (.clk(clk), .d(a[0]),
.q(a_reg[0]));
```

- Each bit of inputs a and b is latched using D flip-flops (dff). This captures the inputs on each rising clock edge, ensuring synchronization.

4. Generate (g) and Propagate (p) Signals:

```
xor p0_xor(p[0], a_reg[0],
b_reg[0]);

and g0_and(g[0], a_reg[0],
b_reg[0]);
```

- **Generate (g)**: Indicates whether a carry is generated at a bit position.
- **Propagate (p)**: Indicates whether a carry will propagate from the previous position.

5. Carry Calculation (c):

assign c[0] = 1'b0;

- The carry logic uses CLA principles to compute carries in parallel, reducing delay:
    - c[1] depends on g[0] and p[0].
    - c[2] depends on g[1], p[1], and previous carries, and so on up to c[5].

6. Sum Calculation:

```
xor sum0_xor(sum_wire[0], p[0],
c[0]);
```

- The sum for each bit is calculated by XOR-ing the corresponding p and c values.

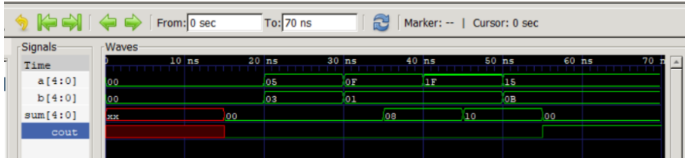
7. Output Flip-Flops:

```
dff sum_reg0 (.clk(clk),
.d(sum_wire[0]), .q(sum[0]));
```

- The calculated sum and carry-out are stored in output flip-flops, ensuring the outputs are registered and synchronized with the clock signal.

Summary:

This pipelined CLA adder improves speed and efficiency for 4-bit addition by leveraging carry-lookahead logic. Input and output registers provide stability and synchronization, crucial for high-speed digital systems.



VERILOG IMPLEMENTATION AND VERIFICATION WITH OSCILLOSCOPE

The code was verified by using different inputs on the FPGA board and getting the required input