VLSI Project
4-Bit Carry Lookahead Adder

Sudhanva Joshi - 2023102022
ECE, IIIT Hyderabad, Monsoon 2024
Instructor: Dr. Abhishek Shrivastava

***Abstract*—This report details the design of a 4-bit Carry Lookahead Adder, emphasizing its high-speed performance through parallel carry computation by eliminating sequential carry propagation. The pipeline to test this model has been expounded upon below.**
Note: All corresponding files are included in their respective folders.

I. Adder Structure

The following CLA Adder, along with D Flip-Flop conditions given, is implemented. Each sum output drives an inverter sized to meet project specifications.

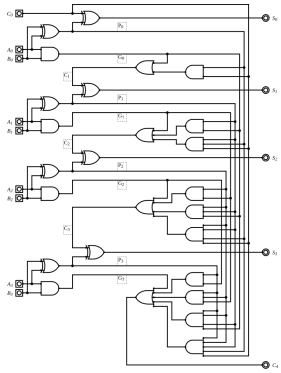


Fig. 1: Circuit Diagram, 4-Bit CLA Adder

The CLA adder accelerates binary addition by precomputing carry signals. It uses the concepts of **generate** (G_i) and **propagate** (P_i) signals defined for each bit as:

$$G_i = A_i \cdot B_i, \quad P_i = A_i + B_i$$

where A_i and B_i are the input bits. The carry signals are calculated recursively as:

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

The sum bits are computed as:

$$S_i = P_i \oplus C_i$$

This approach eliminates the sequential carry propagation delay, enabling faster computation for large bit-width adders. The CLA uses lookahead logic to compute all carries simultaneously, leveraging the recursive nature of C_i.

II. Design Details

Given below are design choices of each implementation used along with general justification for the choice of topology used.
Transistor sizing follows $W_p/W_n = 20\lambda/10\lambda$, where $\lambda = 0.09\mu m$, ensuring balanced performance and area.
Static CMOS Logic is used in N-Input OR, AND Gate implementation. i.e.
Count for N-Input OR & AND Gates = 2*N+2 Transistors.
Sizing Transistors to keep Rise & Fall Times even while being able to drive given load, we have the following relation to account for load.

$$\frac{W_p}{W_n} = 2$$

$$W_p = 2 \cdot W_{inv}$$

$$W_n = 1 \cdot W_{inv}$$

A. XOR Logic

Transmission Gate is used here for the reasons below.

- **Reduced Transistor Count:** Compared to the Static CMOS Logic using 8 Transistors, 6 Transistors are used.
- **Output Voltage Levels:** The PTL Implementation for XOR using MUX Logic yields poor Output Voltage Levels, which isn't a problem for the topology used here.
- **Delay, Power Benefits:** Due to better sizing and performance, power and delay offered by this design is more favourable.

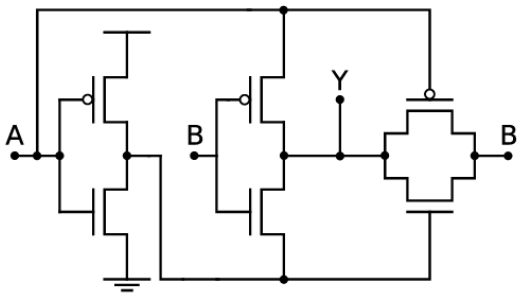


Fig. 2: XOR Circuit Diagram, Transmission Gate

B. D Flip-Flop

True Single Phase Clock (TSPC) is used here for the following reasons:

- **Reduced Transistor Count:** Compared to the Static CMOS Logic using 18 Transistors, 11 Transistors are used here.
- **Single Clock Advantage:** Single Clock Implementation alleviates concerns of multi-clock synchronization, improving clock skew discrepancies.

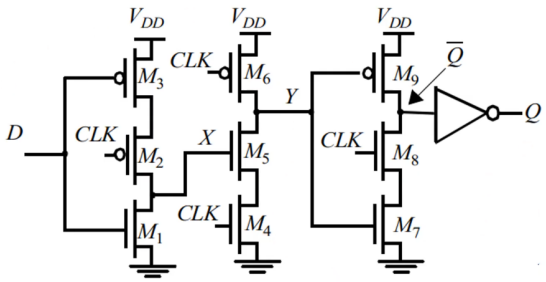


Fig. 3: Waveform, NGSpice Netlist

III. Simulation Results

A. AND Logic

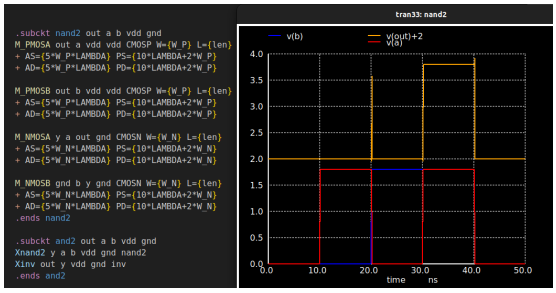


Fig. 4: AND2, Static CMOS Logic

1) AND2:

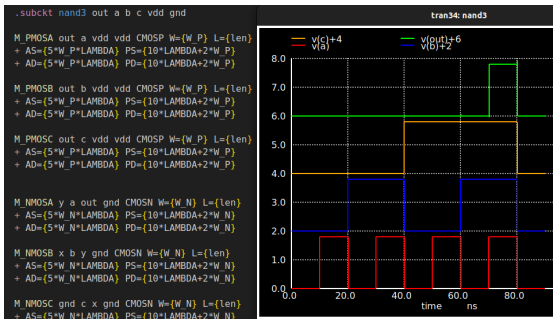


Fig. 5: AND3, Static CMOS Logic

2) AND3:

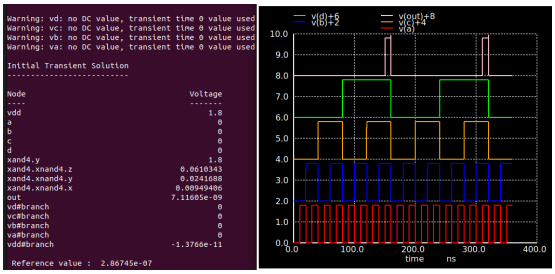


Fig. 6: AND4, Static CMOS Logic

3) AND4:

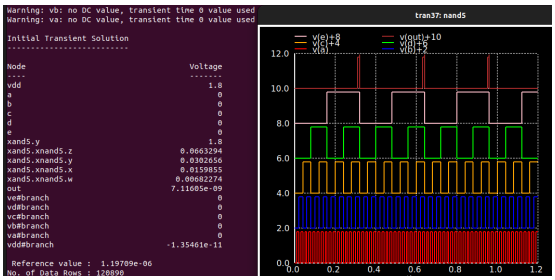


Fig. 7: AND5, Static CMOS Logic

4) AND5:

B. OR Logic

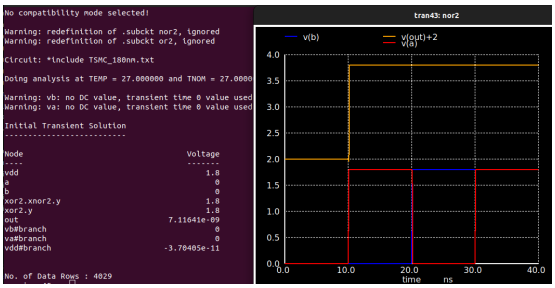


Fig. 8: OR2, Static CMOS Logic

1) OR2:

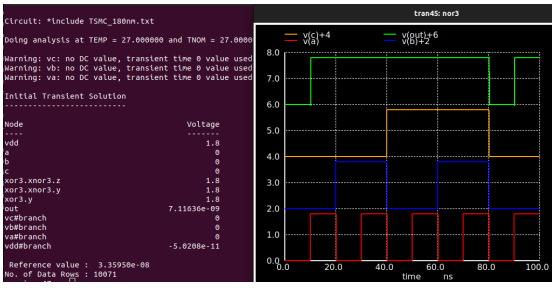


Fig. 9: OR3, Static CMOS Logic

2) OR3:

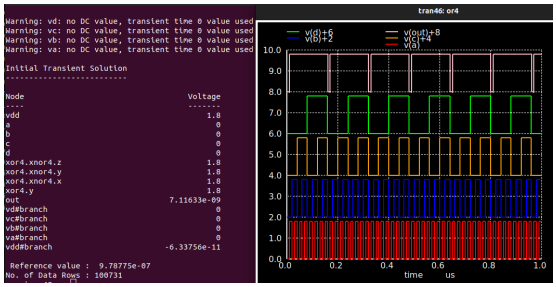


Fig. 10: OR4, Static CMOS Logic

3) OR4:

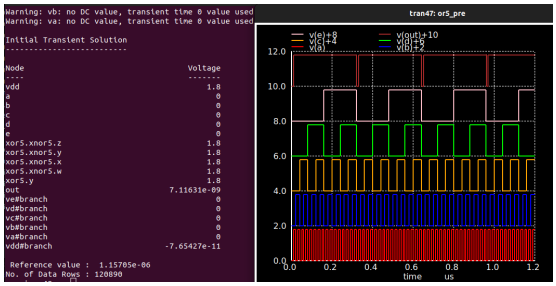


Fig. 11: OR5, Static CMOS Logic

4) OR5:

C. XOR Logic

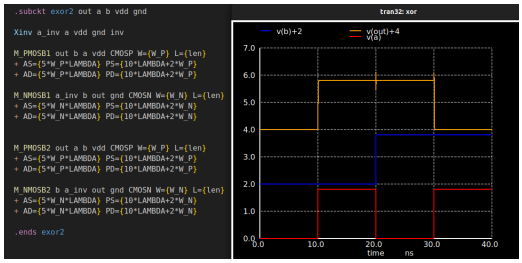


Fig. 12: Transmission Gate XOR, NGSpice Netlist

D. D Flip-Flop

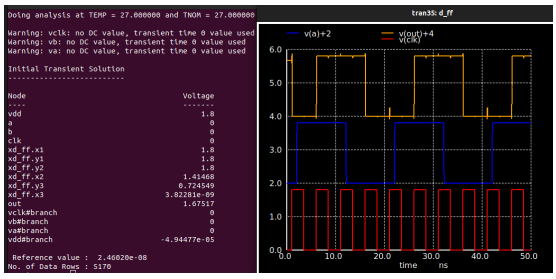


Fig. 13: TSPC d_ff, NGSpice Netlist

IV. Setup, Hold, CtQ Time of D_FF

For setup, hold and CtQ time delays, the given parameters are found graphically.

Defining **Clock-to-Q** delay as the time taken for the input change to propagate from the rising edge of the clock to the output Q, we obtain the result through plot observation:

$$t_{CtQ} \approx 0.2731 \cdot 10^{-9} s$$

$$t_{prop} \approx 0.20438 \cdot 10^{-9} s$$

Given conditions as:

$$T_{CtQ} + T_{prop} > T_{hold}$$

$$\therefore T_{hold} \approx .47748 \cdot 10^{-9} s$$

$$T_{CtQ} + T_{prop} + T_{setup} < T_{TP}$$

$$\therefore T_{setup} \approx .41964 \cdot 10^{-9} s$$

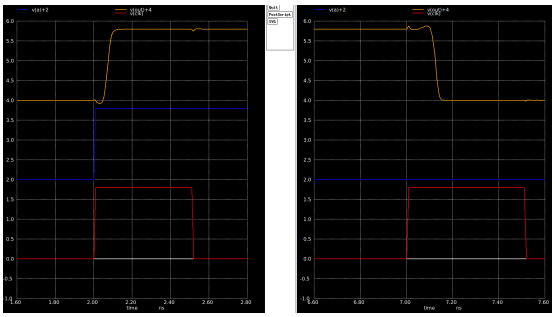


Fig. 14: *Rise & Fall*

V. Stick Diagram

Simplified, abstract representations of CMOS layouts, showing connections without exact scaling. They use colored lines to represent polysilicon, metal, and diffusion layers, with PMOS transistors placed in the n-well and NMOS transistors in the substrate.

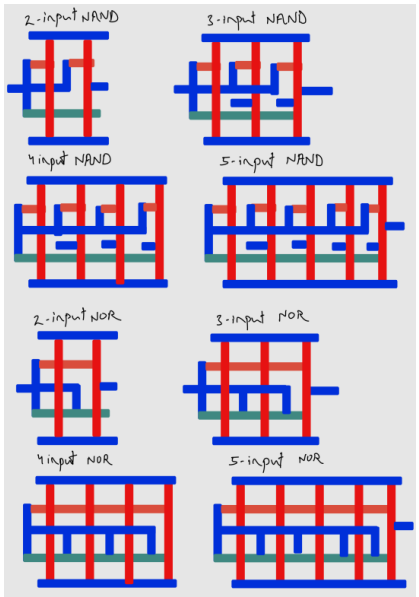


Fig. 15: Stick Diagrams

VI. MAGIC & Post-Layout Simulations

A. AND Logic

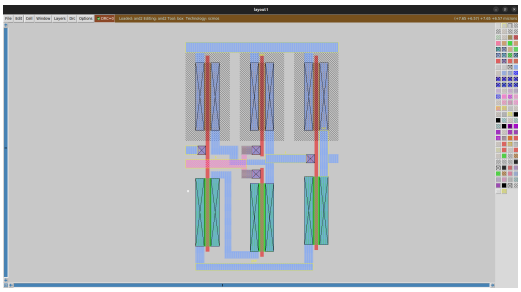


Fig. 16: MAGIC Layout, AND2

1) AND2:

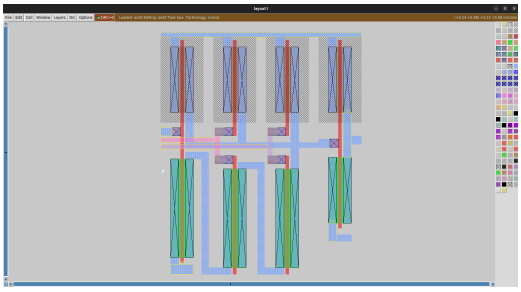


Fig. 17: MAGIC Layout, AND3

2) AND3:

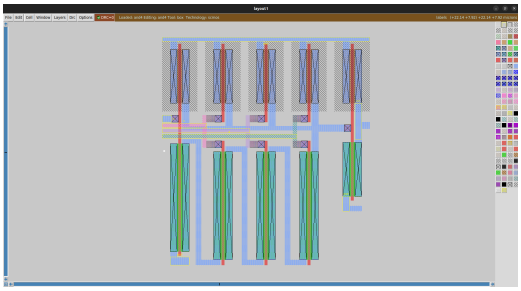


Fig. 18: MAGIC Layout, AND4

3) AND4:

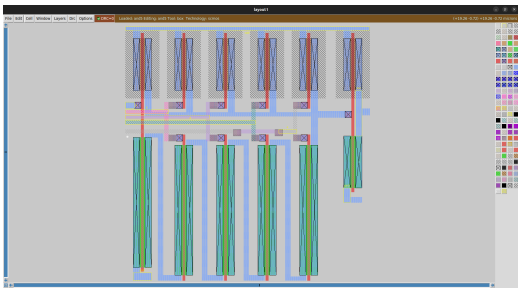


Fig. 19: MAGIC Layout, AND5

4) AND5:

B. OR Logic

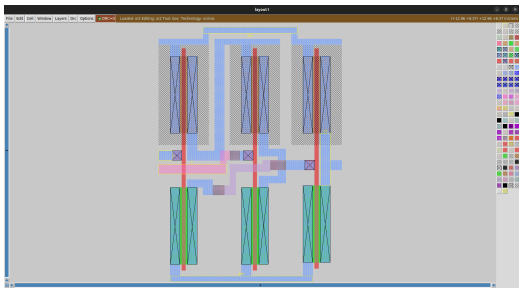


Fig. 20: MAGIC Layout, OR2

1) OR2:

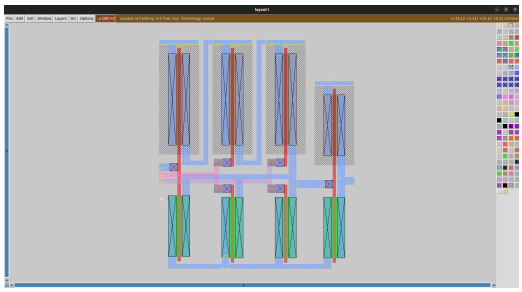


Fig. 21: MAGIC Layout, OR3

2) OR3:

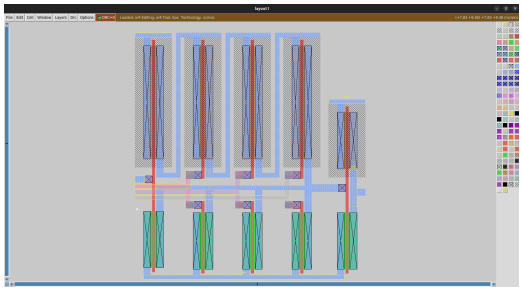


Fig. 22: MAGIC Layout, OR4

3) OR4:

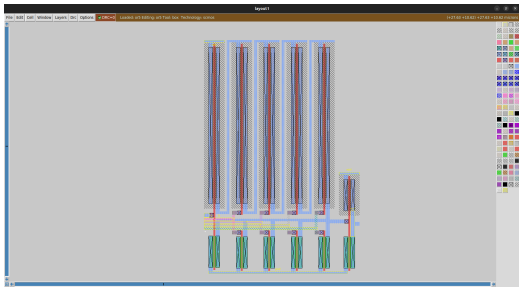


Fig. 23: MAGIC Layout, OR5

4) OR5:

C. XOR Logic

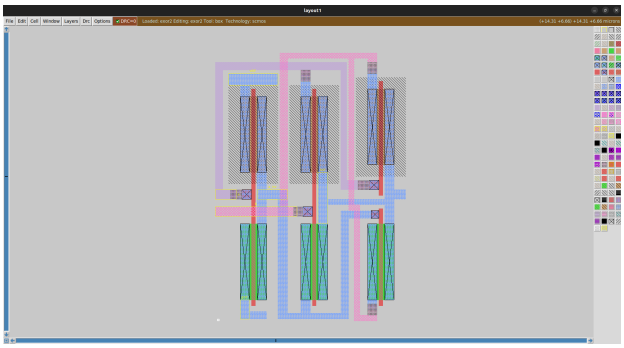


Fig. 24: MAGIC Layout, XOR

D. D Flip-Flop

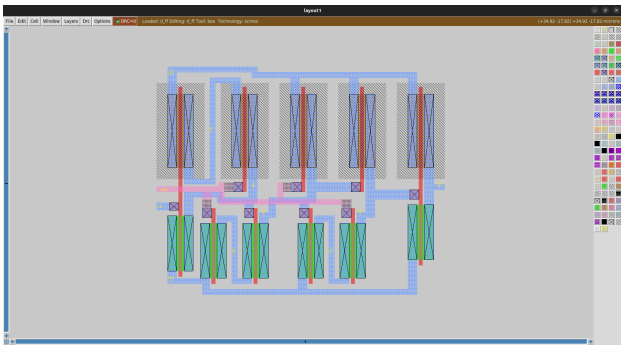


Fig. 25: MAGIC Layout, D_FF

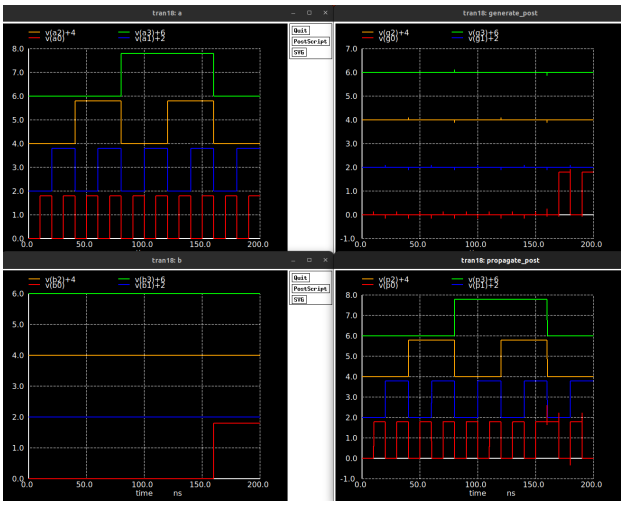


Fig. 26: PropGen & Sum Block Analysis

VII. Block Integration

Note: The modular implementation is in the **netlist-CLA** folder. All Logic Gates are referenced in a subcircuit file for readibility. Netlists, Plots, etc are all available in respective folders.

```
Xblock_A a3 a2 a1 a0 a3_in a2_in a1_in a0_in clk vdd gnd d_block_4bit
Xblock_B b3 b2 b1 b0 b3_in b2_in b1_in b0_in clk vdd gnd d_block_4bit

.subckt propgen a0 a1 a2 a3 b0 b1 b2 b3 p0 g0 p1 g1 p2 g2 p3 g3 vdd gnd
* Propagate using XOR
Xexor0 p0 a0 b0 vdd gnd exor2
Xexor1 p1 a1 b1 vdd gnd exor2
Xexor2 p2 a2 b2 vdd gnd exor2
Xexor3 p3 a3 b3 vdd gnd exor2

* Generate using AND
Xand0 g0 a0 b0 vdd gnd and2
Xand1 g1 a1 b1 vdd gnd and2
Xand2 g2 a2 b2 vdd gnd and2
Xand3 g3 a3 b3 vdd gnd and2
.ends propgen


.subckt carry_logic C Gi Pi Ci vdd gnd
* Carry Logic: Ci+1 = Gi+(Pi.Ci)
Xprod prod Pi Ci vdd gnd and2
Xsum C Gi prod vdd gnd or2
.ends carry_logic

.subckt carry_block c1 c2 c3 c4 c0 p0 p1 p2 p3 g0 g1 g2 g3 vdd gnd
* Carry Logic: Ci+1 = Gi+(Pi.Ci)
Xcarry1 c1 g0 p0 c0 vdd gnd carry_logic
Xcarry2 c2 g1 p1 c1 vdd gnd carry_logic
Xcarry3 c3 g2 p2 c2 vdd gnd carry_logic
Xcarry4 c4 g3 p3 c3 vdd gnd carry_logic
.ends carry_block

.subckt sum_block p0 p1 p2 p3 c0 c1 c2 c3 s0 s1 s2 s3 vdd gnd
* Sum using XOR
Xexor0 s0 p0 c0 vdd gnd exor2
Xexor1 s1 p1 c1 vdd gnd exor2
Xexor2 s2 p2 c2 vdd gnd exor2
Xexor3 s3 p3 c3 vdd gnd exor2
.ends sum_block

* * <A> <B> <G> <P> <S> <C>
.subckt CLA a0 a1 a2 a3 b0 b1 b2 b3 p0 g0 p1 g1 p2 g2 p3 g3 s0 s1 s2 s3 c1 c2 c3 c4 c0 vdd gnd
* Sum using XOR
X_prop_gen a0 a1 a2 a3 b0 b1 b2 b3 p0 g0 p1 g1 p2 g2 p3 g3 vdd gnd propgen
X_carry_block c1 c2 c3 c4 c0 p0 p1 p2 p3 g0 g1 g2 g3 vdd gnd carry_block
X_sum p0 p1 p2 p3 c0 c1 c2 c3 s0 s1 s2 s3 vdd gnd sum_block
.ends CLA

X_CLA a0 a1 a2 a3 b0 b1 b2 b3 p0 g0 p1 g1 p2 g2 p3 g3 s0 s1 s2 s3 c1 c2 c3 c4 c0 vdd gnd CLA
```

Fig. 27: Modular Netlist

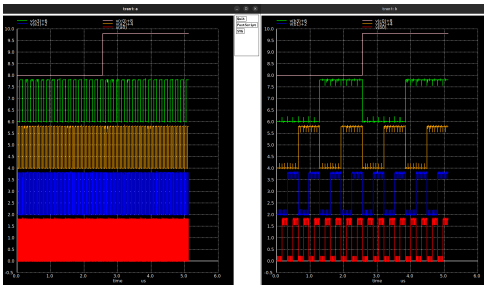


Fig. 28: A, B

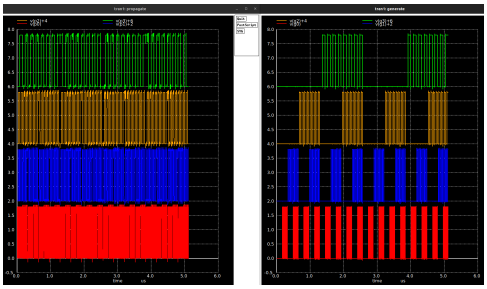


Fig. 29: Propagate, Generate

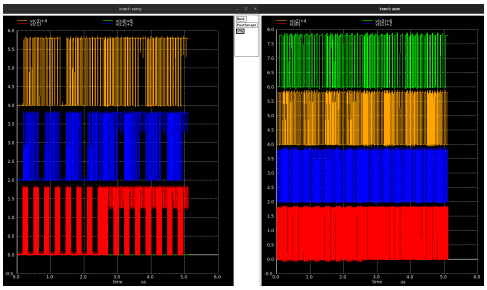


Fig. 30: Carry, Sum

Worst case delay will naturally occur for C_{out}, due to largest gates and heaviest logic function leading into this output.

$$d_{max} \approx 0.26282 \cdot 10^{-9} s$$

$$f_{max} \approx \frac{1}{0.26282} \cdot 10^{9}$$

$$\therefore f_{max} \approx 3.8 \cdot 10^{9} Hz$$

VIII. Floor Plan

Assuming that the floor plan refers to the comprehensive final layout & that 'pitches' refer to dimensions of our adder, the following design is implemented with corresponding labels for extraction to **spice netlist**.

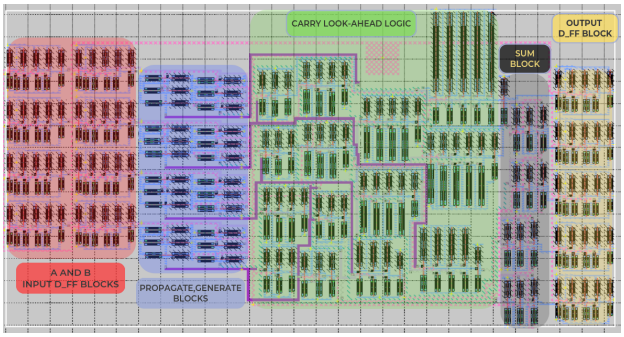


Fig. 31: Highlighted Layout

Dimensions occupied by the entire CLA block are:

$$Area \approx 140 \cdot 72 \; microns$$

IX. Complete Circuit Post-Layout Simulation

Final Layout is as follows:

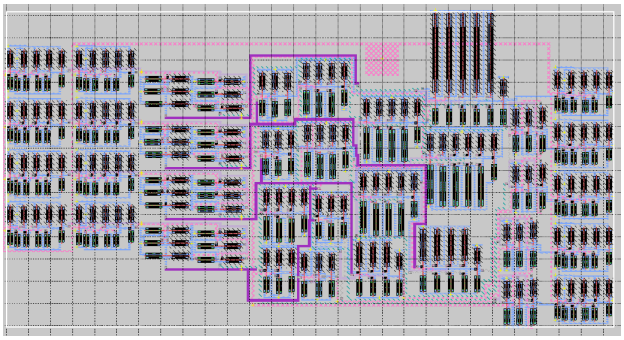


Fig. 32: Final CLA Layout

Comparing our schematic netlists with the post-layout extracted netlists:

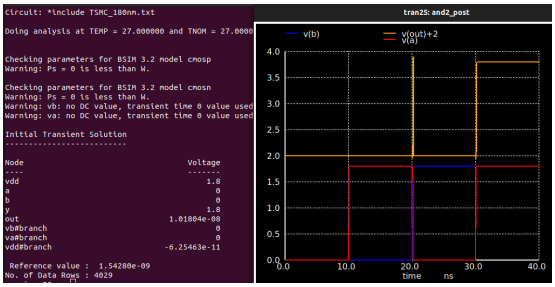


Fig. 33: AND2, Post Layout

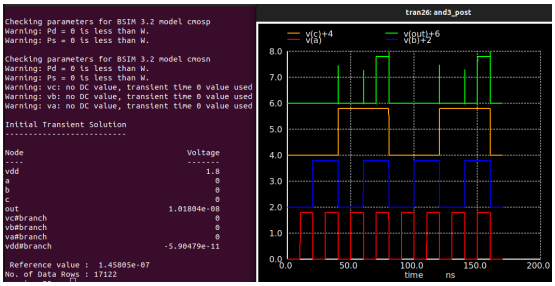


Fig. 34: AND3, Post Layout

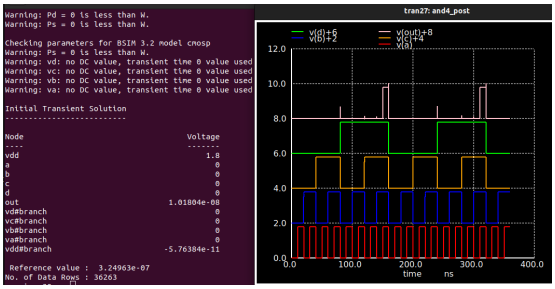


Fig. 35: AND4, Post Layout

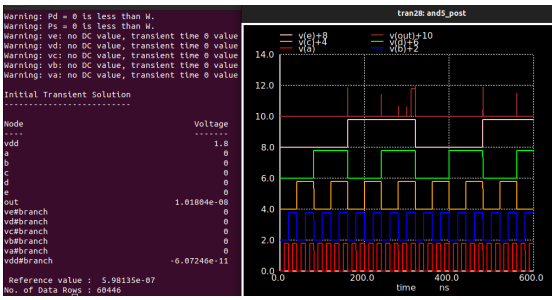


Fig. 36: AND5, Post Layout

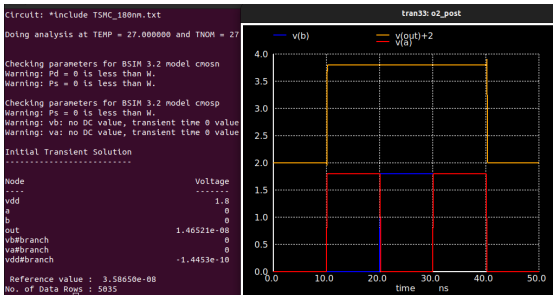


Fig. 37: OR2, Post Layout

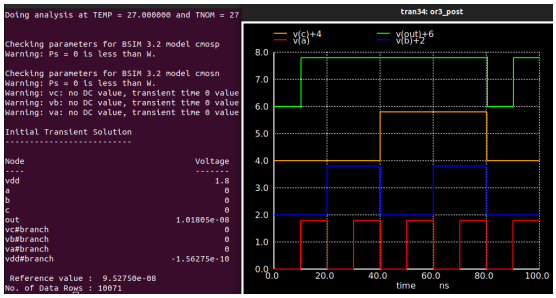


Fig. 38: OR3, Post Layout

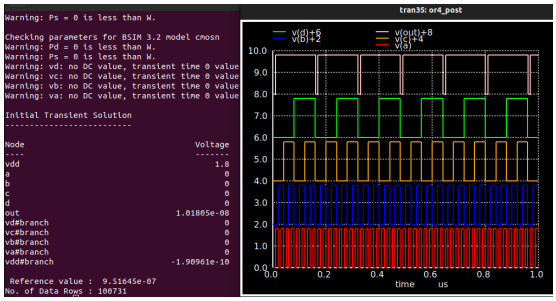


Fig. 39: OR4, Post Layout

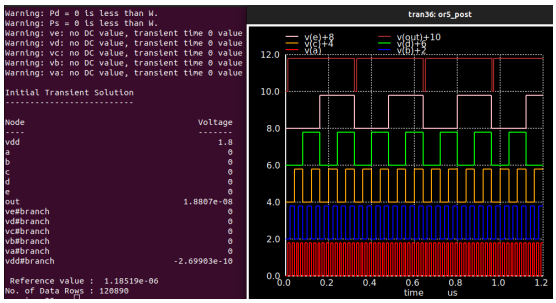


Fig. 40: OR5, Post Layout

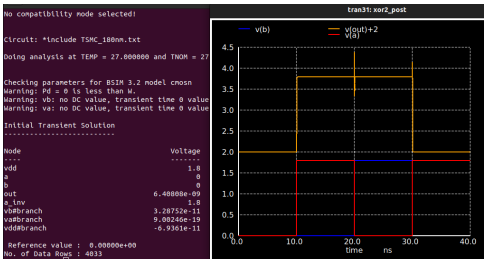


Fig. 41: XOR2, Post Layout

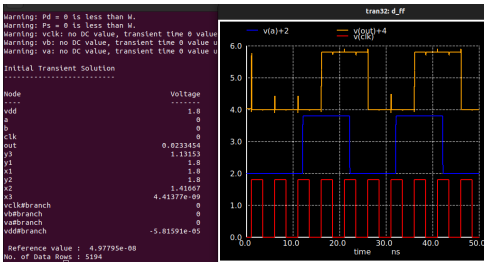


Fig. 42: D_FF, Post Layout

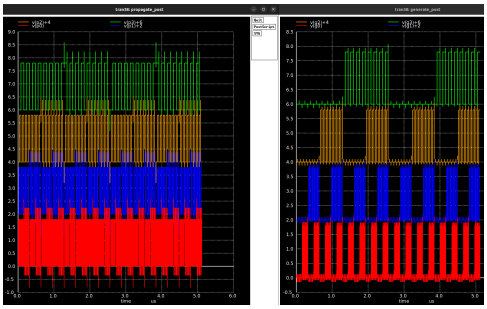


Fig. 43: Propagate + Generate, Post Layout

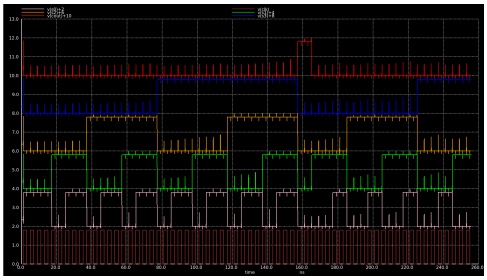


Fig. 44: Sum, Post Layout, Low Sweep

Parameter	Schematic Simulation	Post-Layout Simulation
Rise Time	$0.1342 \cdot 10^{-9} ns$	$0.1567 \cdot 10^{-9} ns$
Fall Time	$0.1041 \cdot 10^{-9} ns$	$0.1338 \cdot 10^{-9} ns$
Propagation Delay	$0.2903 \cdot 10^{-9} ns$	$0.3092 \cdot 10^{-9} ns$
Propagate Signal Delay	$0.2706 \cdot 10^{-9} ns$	$0.3105 \cdot 10^{-9} ns$
Generate Signal Delay	$0.2691 \cdot 10^{-9} ns$	$0.2874 \cdot 10^{-9} ns$
Sum Output Delay	$0.2952 \cdot 10^{-9} ns$	$0.3109 \cdot 10^{-9} ns$
Carry Output Delay	$0.2709 \cdot 10^{-9} ns$	$0.3013 \cdot 10^{-9} ns$

TABLE I: Schematic & Post-Layout Comparison

X. Delay, Clock Frequency Contraints

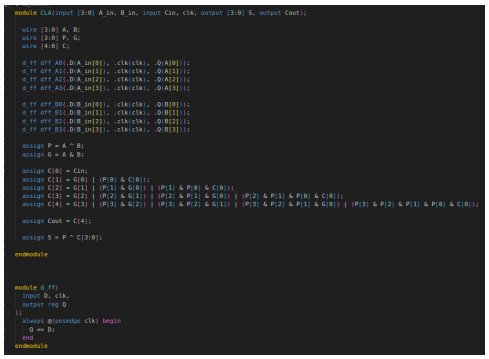
Worst case delay will naturally occur for C_{out} again.

$$d_{max} \approx 0.3047 \cdot 10^{-9} s$$

$$f_{max} \approx \frac{1}{0.3047} \cdot 10^{9}$$

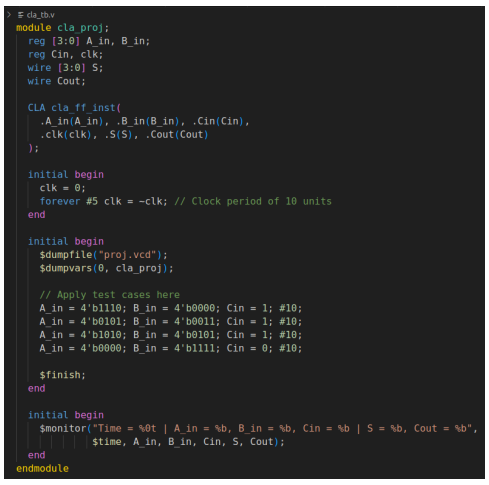
$$\therefore f_{max} \approx 3.2819 \cdot 10^{9} Hz$$

XI. Verilog HDL



```
module CLA(input [3:0] A_in, B_in, input Cin, clk, output [3:0] S, output Cout);

  wire [3:0] A, B;
  wire [3:0] P, G;
  wire [4:0] C;

  d_ff dff_A0(.D(A_in[0]), .clk(clk), .Q(A[0]));
  d_ff dff_A1(.D(A_in[1]), .clk(clk), .Q(A[1]));
  d_ff dff_A2(.D(A_in[2]), .clk(clk), .Q(A[2]));
  d_ff dff_A3(.D(A_in[3]), .clk(clk), .Q(A[3]));

  d_ff dff_B0(.D(B_in[0]), .clk(clk), .Q(B[0]));
  d_ff dff_B1(.D(B_in[1]), .clk(clk), .Q(B[1]));
  d_ff dff_B2(.D(B_in[2]), .clk(clk), .Q(B[2]));
  d_ff dff_B3(.D(B_in[3]), .clk(clk), .Q(B[3]));

  assign P = A ^ B;
  assign G = A & B;

  assign C[0] = Cin;
  assign C[1] = G[0] | (P[0] & C[0]);
  assign C[2] = G[1] | (P[1] & G[0]) | (P[1] & P[0] & C[0]);
  assign C[3] = G[2] | (P[2] & G[1]) | (P[2] & P[1] & G[0]) | (P[2] & P[1] & P[0] & C[0]);
  assign C[4] = G[3] | (P[3] & G[2]) | (P[3] & P[2] & G[1]) | (P[3] & P[2] & P[1] & G[0]) | (P[3] & P[2] & P[1] & P[0] & C[0]);

  assign Cout = C[4];

  assign S = P ^ C[3:0];

endmodule


module d_ff(
  input D, clk,
  output reg Q
);
  always @(posedge clk) begin
    Q <= D;
  end
endmodule
```

Fig. 45: Verilog Modules, CLA & D Flip-Flop



```
module cla_proj;
  reg [3:0] A_in, B_in;
  reg Cin, clk;
  wire [3:0] S;
  wire Cout;

  CLA cla_ff_inst(
    .A_in(A_in), .B_in(B_in), .Cin(Cin),
    .clk(clk), .S(S), .Cout(Cout)
  );

  initial begin
    clk = 0;
    forever #5 clk = ~clk; // Clock period of 10 units
  end

  initial begin
    $dumpfile("proj.vcd");
    $dumpvars(0, cla_proj);

    // Apply test cases here
    A_in = 4'b1110; B_in = 4'b0000; Cin = 1; #10;
    A_in = 4'b0101; B_in = 4'b0011; Cin = 1; #10;
    A_in = 4'b1010; B_in = 4'b0101; Cin = 1; #10;
    A_in = 4'b0000; B_in = 4'b1111; Cin = 0; #10;

    $finish;
  end

  initial begin
    $monitor("Time = %0t | A_in = %b, B_in = %b, Cin = %b | S = %b, Cout = %b",
             $time, A_in, B_in, Cin, S, Cout);
  end
endmodule
```

Fig. 46: Testbench, Verilog HDL

Fig. 47: Waveforms, GTKWave

```
Time = 0 | A_in = 0000, B_in = 0000, Cin = 0 | S = xxxx, Cout = x
Time = 5 | A_in = 0000, B_in = 0000, Cin = 0 | S = 0000, Cout = 0
Time = 10 | A_in = 0001, B_in = 0001, Cin = 0 | S = 0000, Cout = 0
Time = 15 | A_in = 0001, B_in = 0001, Cin = 0 | S = 0010, Cout = 0
Time = 20 | A_in = 0010, B_in = 0010, Cin = 0 | S = 0010, Cout = 0
Time = 25 | A_in = 0010, B_in = 0010, Cin = 0 | S = 0100, Cout = 0
Time = 30 | A_in = 0011, B_in = 0011, Cin = 0 | S = 0100, Cout = 0
Time = 35 | A_in = 0011, B_in = 0011, Cin = 0 | S = 0110, Cout = 0
Time = 40 | A_in = 0100, B_in = 0100, Cin = 0 | S = 0110, Cout = 0
Time = 45 | A_in = 0100, B_in = 0100, Cin = 0 | S = 1000, Cout = 0
Time = 50 | A_in = 0101, B_in = 0101, Cin = 0 | S = 1000, Cout = 0
Time = 55 | A_in = 0101, B_in = 0101, Cin = 0 | S = 1010, Cout = 0
Time = 60 | A_in = 0110, B_in = 0110, Cin = 0 | S = 1010, Cout = 0
Time = 65 | A_in = 0110, B_in = 0110, Cin = 0 | S = 1100, Cout = 0
Time = 70 | A_in = 0111, B_in = 0111, Cin = 0 | S = 1100, Cout = 0
Time = 75 | A_in = 0111, B_in = 0111, Cin = 0 | S = 1110, Cout = 0
Time = 80 | A_in = 1000, B_in = 1000, Cin = 0 | S = 1110, Cout = 0
Time = 85 | A_in = 1000, B_in = 1000, Cin = 0 | S = 0000, Cout = 1
Time = 90 | A_in = 1001, B_in = 1001, Cin = 0 | S = 0000, Cout = 1
Time = 95 | A_in = 1001, B_in = 1001, Cin = 0 | S = 0010, Cout = 1
Time = 100 | A_in = 1010, B_in = 1010, Cin = 0 | S = 0010, Cout = 1
Time = 105 | A_in = 1010, B_in = 1010, Cin = 0 | S = 0100, Cout = 1
Time = 110 | A_in = 1011, B_in = 1011, Cin = 0 | S = 0100, Cout = 1
Time = 115 | A_in = 1011, B_in = 1011, Cin = 0 | S = 0110, Cout = 1
Time = 120 | A_in = 1100, B_in = 1100, Cin = 0 | S = 0110, Cout = 1
Time = 125 | A_in = 1100, B_in = 1100, Cin = 0 | S = 1000, Cout = 1
Time = 130 | A_in = 1101, B_in = 1101, Cin = 0 | S = 1000, Cout = 1
Time = 135 | A_in = 1101, B_in = 1101, Cin = 0 | S = 1010, Cout = 1
Time = 140 | A_in = 1110, B_in = 1110, Cin = 0 | S = 1010, Cout = 1
Time = 145 | A_in = 1110, B_in = 1110, Cin = 0 | S = 1100, Cout = 1
Time = 150 | A_in = 1111, B_in = 1111, Cin = 0 | S = 1100, Cout = 1
Time = 155 | A_in = 1111, B_in = 1111, Cin = 0 | S = 1110, Cout = 1
Time = 160 | A_in = 0000, B_in = 0000, Cin = 1 | S = 1111, Cout = 1
Time = 165 | A_in = 0000, B_in = 0000, Cin = 1 | S = 0001, Cout = 0
Time = 170 | A_in = 0001, B_in = 0001, Cin = 1 | S = 0001, Cout = 0
Time = 175 | A_in = 0001, B_in = 0001, Cin = 1 | S = 0011, Cout = 0
Time = 180 | A_in = 0010, B_in = 0010, Cin = 1 | S = 0011, Cout = 0
Time = 185 | A_in = 0010, B_in = 0010, Cin = 1 | S = 0101, Cout = 0
Time = 190 | A_in = 0011, B_in = 0011, Cin = 1 | S = 0101, Cout = 0
Time = 195 | A_in = 0011, B_in = 0011, Cin = 1 | S = 0111, Cout = 0
Time = 200 | A_in = 0100, B_in = 0100, Cin = 1 | S = 0111, Cout = 0
Time = 205 | A_in = 0100, B_in = 0100, Cin = 1 | S = 1001, Cout = 0
Time = 210 | A_in = 0101, B_in = 0101, Cin = 1 | S = 1001, Cout = 0
Time = 215 | A_in = 0101, B_in = 0101, Cin = 1 | S = 1011, Cout = 0
Time = 220 | A_in = 0110, B_in = 0110, Cin = 1 | S = 1011, Cout = 0
Time = 225 | A_in = 0110, B_in = 0110, Cin = 1 | S = 1101, Cout = 0
Time = 230 | A_in = 0111, B_in = 0111, Cin = 1 | S = 1101, Cout = 0
Time = 235 | A_in = 0111, B_in = 0111, Cin = 1 | S = 1111, Cout = 0
Time = 240 | A_in = 1000, B_in = 1000, Cin = 1 | S = 1111, Cout = 0
Time = 245 | A_in = 1000, B_in = 1000, Cin = 1 | S = 0001, Cout = 1
Time = 250 | A_in = 1001, B_in = 1001, Cin = 1 | S = 0001, Cout = 1
Time = 255 | A_in = 1001, B_in = 1001, Cin = 1 | S = 0011, Cout = 1
Time = 260 | A_in = 1010, B_in = 1010, Cin = 1 | S = 0011, Cout = 1
Time = 265 | A_in = 1010, B_in = 1010, Cin = 1 | S = 0101, Cout = 1
Time = 270 | A_in = 1011, B_in = 1011, Cin = 1 | S = 0101, Cout = 1
Time = 275 | A_in = 1011, B_in = 1011, Cin = 1 | S = 0111, Cout = 1
Time = 280 | A_in = 1100, B_in = 1100, Cin = 1 | S = 0111, Cout = 1
Time = 285 | A_in = 1100, B_in = 1100, Cin = 1 | S = 1001, Cout = 1
Time = 290 | A_in = 1101, B_in = 1101, Cin = 1 | S = 1001, Cout = 1
Time = 295 | A_in = 1101, B_in = 1101, Cin = 1 | S = 1011, Cout = 1
Time = 300 | A_in = 1110, B_in = 1110, Cin = 1 | S = 1011, Cout = 1
Time = 305 | A_in = 1110, B_in = 1110, Cin = 1 | S = 1101, Cout = 1
Time = 310 | A_in = 1111, B_in = 1111, Cin = 1 | S = 1101, Cout = 1
Time = 315 | A_in = 1111, B_in = 1111, Cin = 1 | S = 1111, Cout = 1
```

Fig. 48: Truth Table, Verilog

XII. FPGA, Oscilloscope Implementation

The following links contain brief video demonstrations of FPGA Implementation and Generations of Oscilloscope Waveforms using an Arduino UNO.

FPGA & Oscilloscope Demo, Google Drive

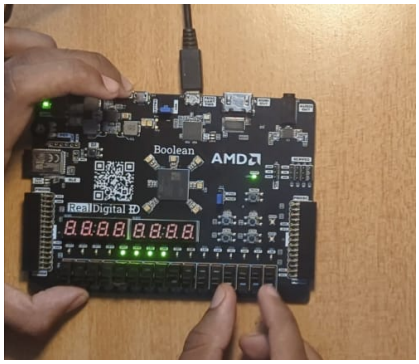


Fig. 49: Vivado Test, AMD Boolean Board

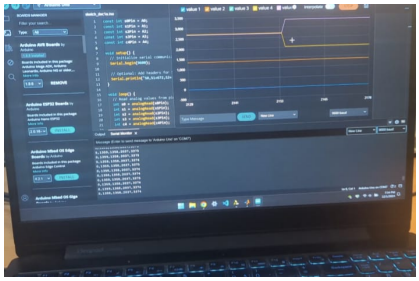


Fig. 50: Waveforms, GTKWave

ACKNOWLEDGMENT

I sincerely thank Professor Abhishek Shrivastava and the TAs for their invaluable guidance and support throughout this project, as their assistance and constructive feedback greatly contributed to the successful completion of this work.

REFERENCES

1. International Journal of Recent Technology and Engineering (IJRTE), ISSN: 2277-3878 (Online), Volume-8, Issue-1, May 2019. Available online

2. Logic Design, Dinesh Sharma, Microelectronics Group, EE Department, IIT Bombay.

3. Carry-lookahead Adder, Wikipedia. Available online