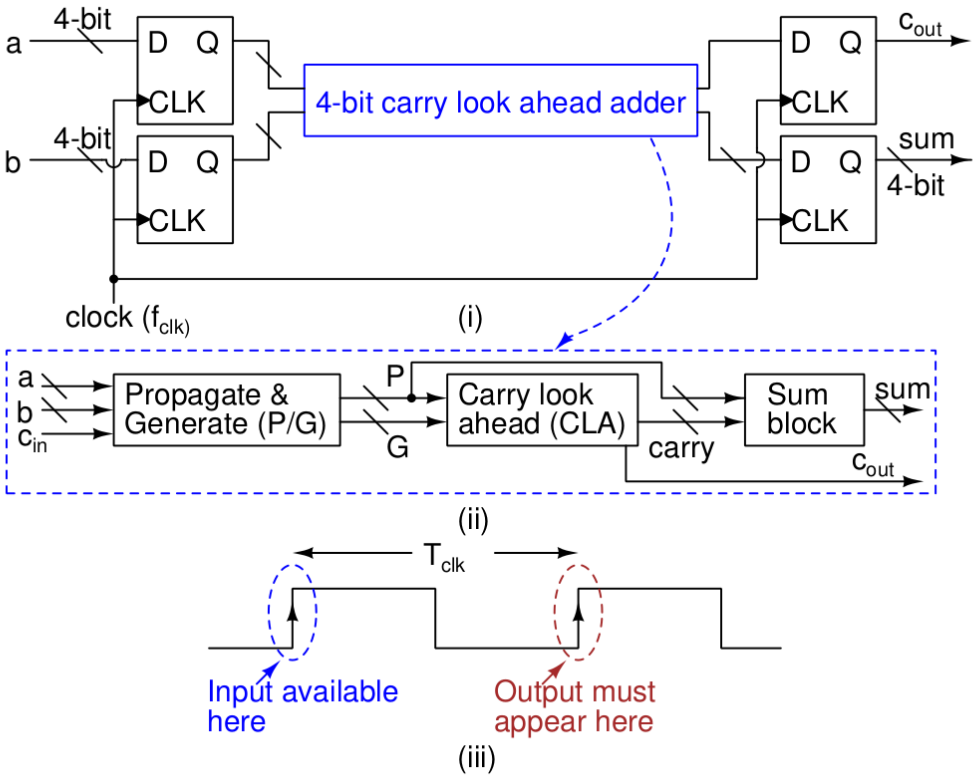
# *VLSI Project*

Charchit Gupta
2019102034

## ***I & II***

Circuit overview




D-flipflops

I have used tristate-inverter based latch, then used two latches in master-slave combination to produce a 20-transistor positive-edge triggered clocked CMOS D flipflop. At the rising edge of the clock, the input bits are passed to the Combinatorial logic which consists of the propagate & generate block, CLA block and the sum block. This logic achieves the desired value well before the next positive edge (assuming low clock frequency) but is passed onto the final outputs through the output-4-bit flipflop and output-1-bit flipflop on the next positive edge only.

Propagate and Generate block (P/G)

This block is used to generate the propagate and generate bits for the inputs, these bits are later used by the CLA block and the sum block to get the carry and the sum bits. If the numbers to be added are $a_3a_2a_1a_0$ and $b_3b_2b_1b_0$ , then the propagate ($p_i$) and generate ($g_i$) signals for each bit position can be defined as (for i = 0,1, 2, 3)

$$p_i = a_i \oplus b_i$$
$$g_i = a_i \,.\, b_i$$

These bits are produced by the 2-input XOR and 2-input AND gates.

Carry look ahead (CLA) block

This block is used to generate the carry bits. The carry out $c_{(i+1)}$ of the ith bit position can be written as (assuming $c_0 = 0$) follows:

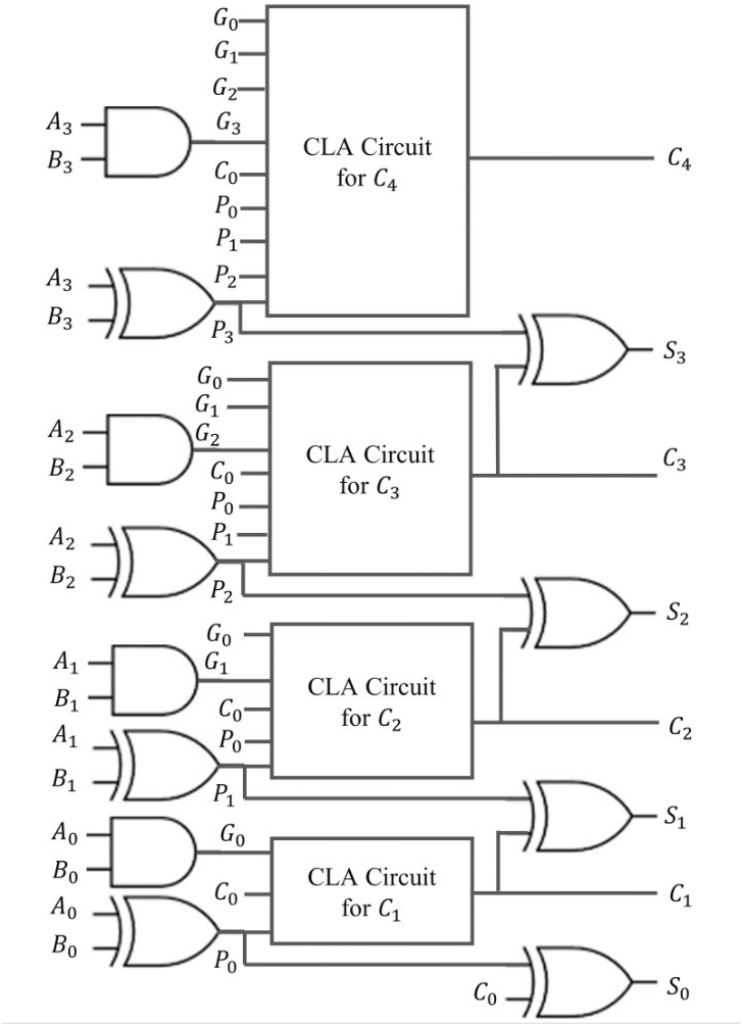
$$c_{(i+1)} = (p_i.c_i) + g_i \text{ , } i = 0,1, 2, 3$$

This gives us the following logic:

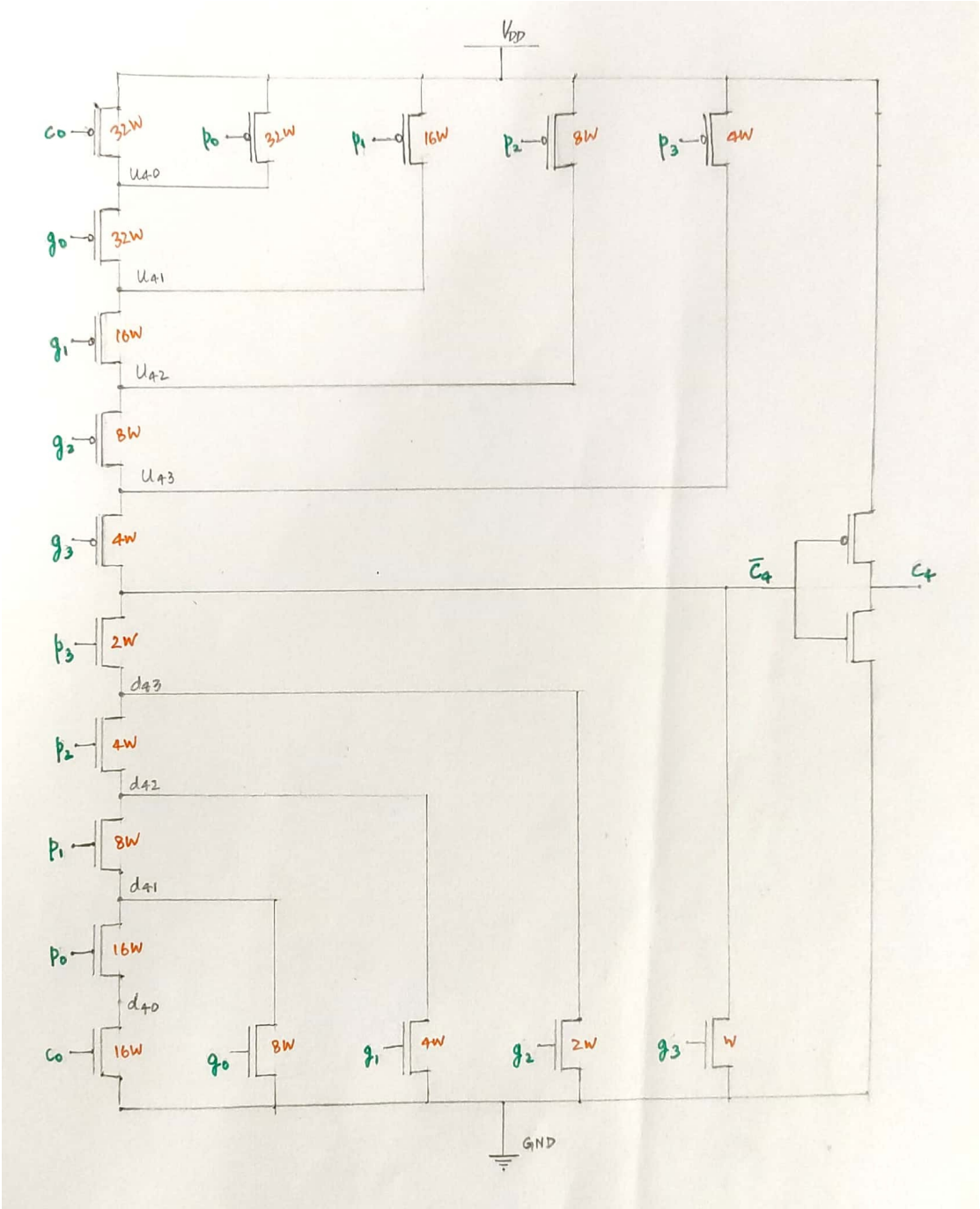
$$c_1 = g_0 + p_0.c_0$$
$$c_2 = g_1 + p_1.c_1 = g_1 + p_1.g_0 + p_1.p_0.c_0$$
$$c_3 = g_2 + p_2.c_2 = g_2 + p_2.g_1 + p_2.p_1.g_0 + p_2.p_1.p_0.c_0$$
$$c_4 = g_3 + p_3.c_3 = g_3 + p_3.g_2 + p_3.p_2.g_1 + p_3.p_2.p_1.g_0 + p_3.p_2.p_1.p_0.c_0$$

This can be implemented using AND & OR gates, but we would need a lot of them. Fortunately, there's a more efficient way to implement this. We implement it in the transistor level itself with specific Pull-Up and Pull-Down networks.

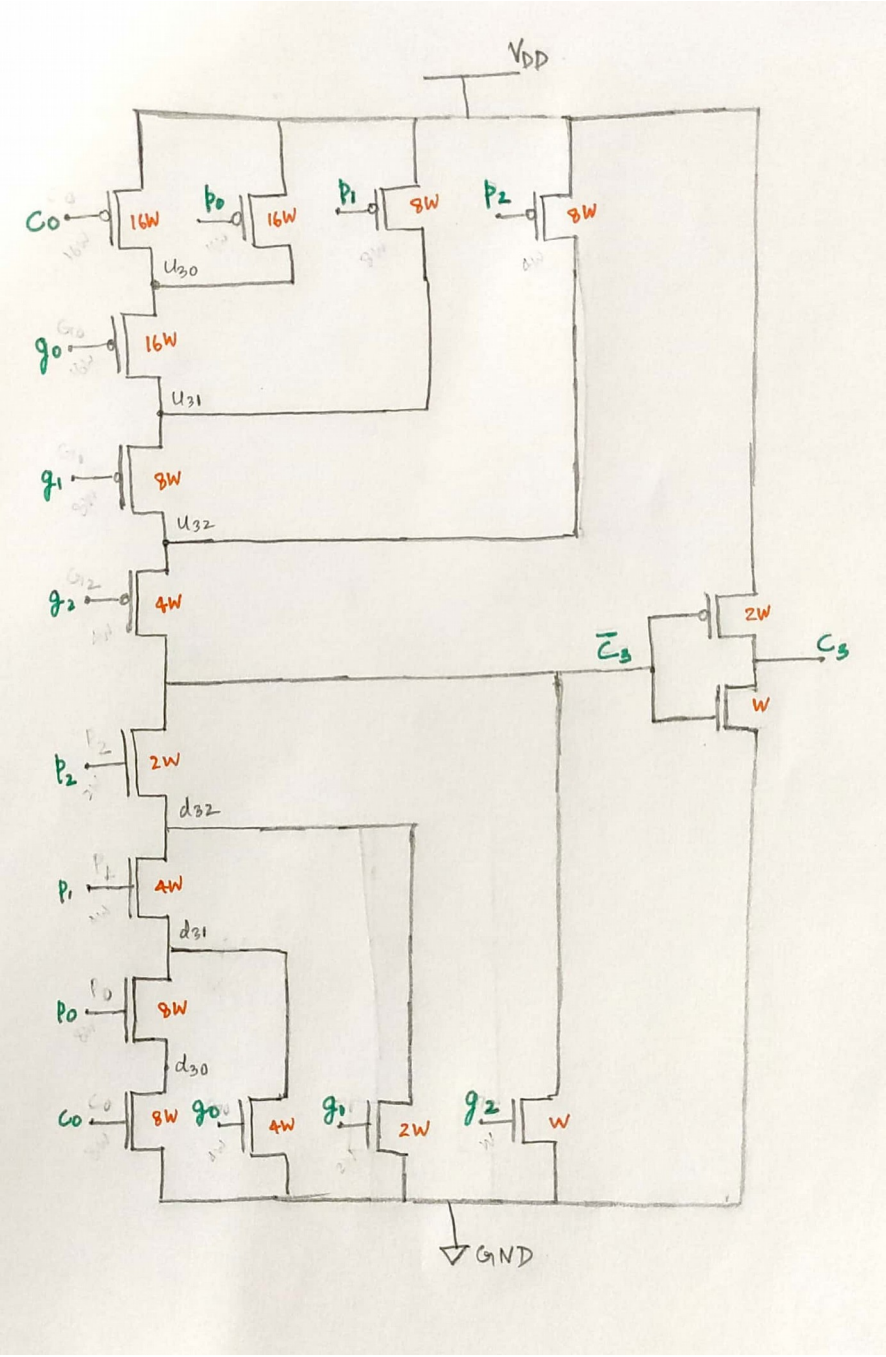
Circuit diagram:



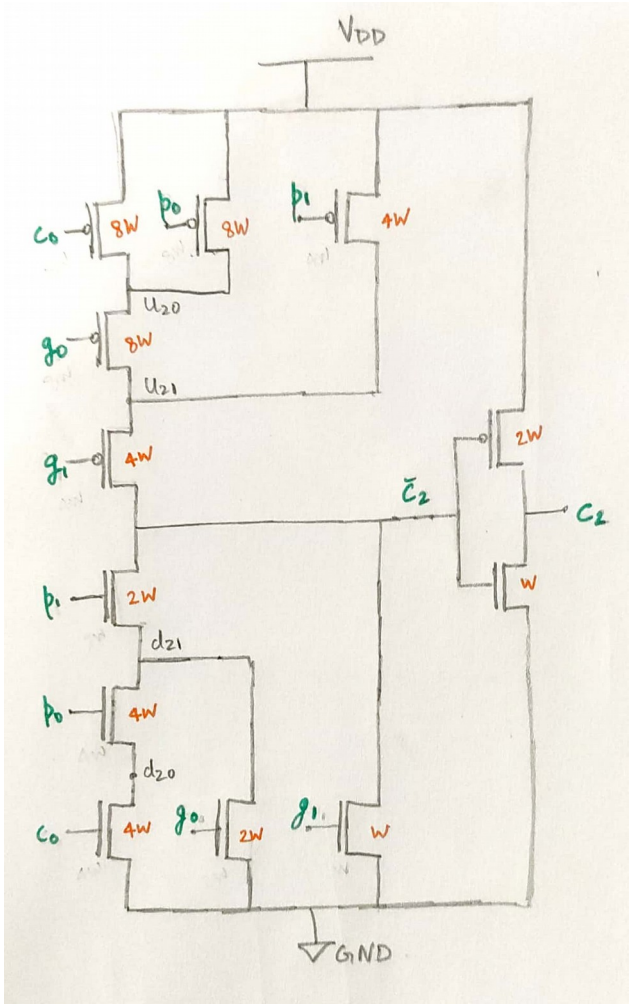
CLA for $C_4$:


VDD
c0
32W
p0
32W
p1
16W
p2
8W
p3
4W
u40
g0
32W
u41
g1
16W
u42
g2
8W
u43
g3
4W
C4
C4
p3
2W
d43
p2
4W
d42
p1
8W
d41
p0
16W
d40
c0
16W
g0
8W
g1
4W
g2
2W
g3
W
GND

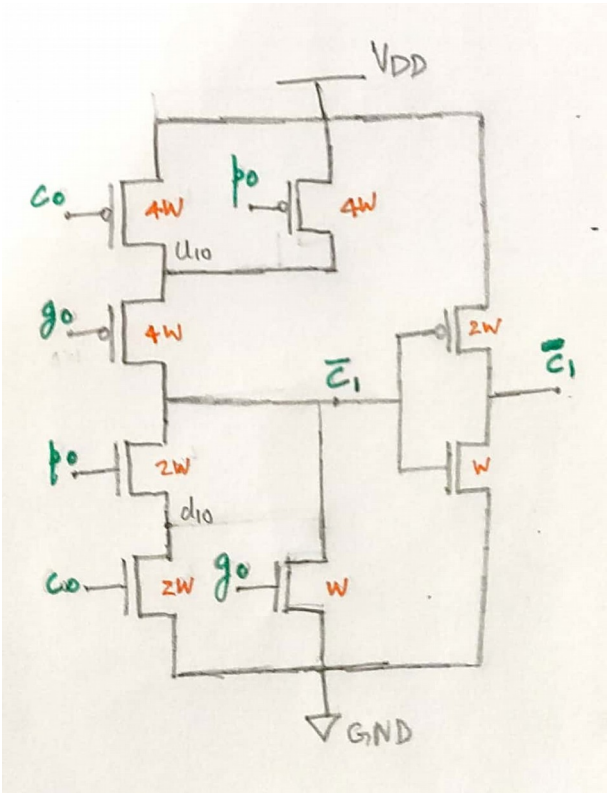
CLA for $C_3$:


VDD
C0
16W
p0
16W
p1
8W
p2
8W
U30
g0
16W
U31
g1
8W
U32
g2
4W
2W
C3
C3
W
p2
2W
d32
p1
4W
d31
p0
8W
d30
C0
8W
g0
4W
g1
2W
g2
W
GND

CLA for $C_2$:




CLA for $C_1$:



Sum block

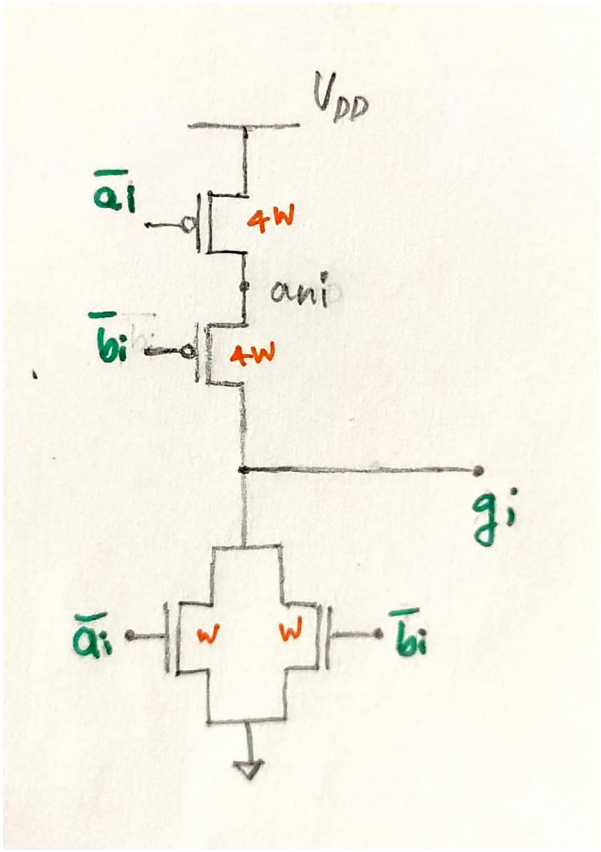
This block is used to generate the output bit stream, i.e., the sum bits.

The sum bits can be represented as follows:

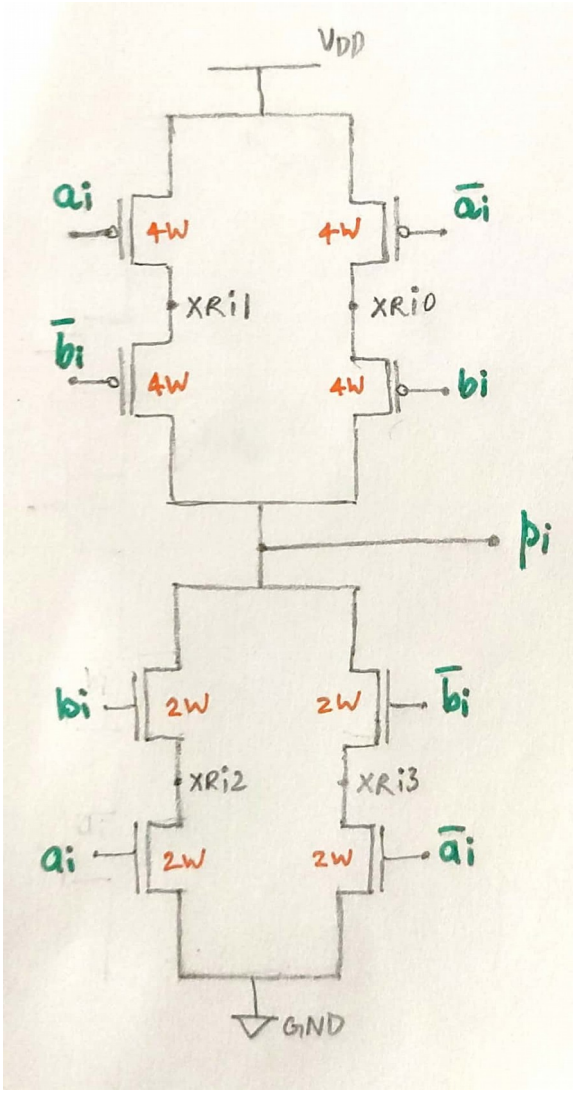
$$s_i = p_i \oplus c_i \text{ , } i=0,1,2,3$$

As done for the propagate and generate block, these sum bits are produced by the 2-input XOR gates.

AND gate



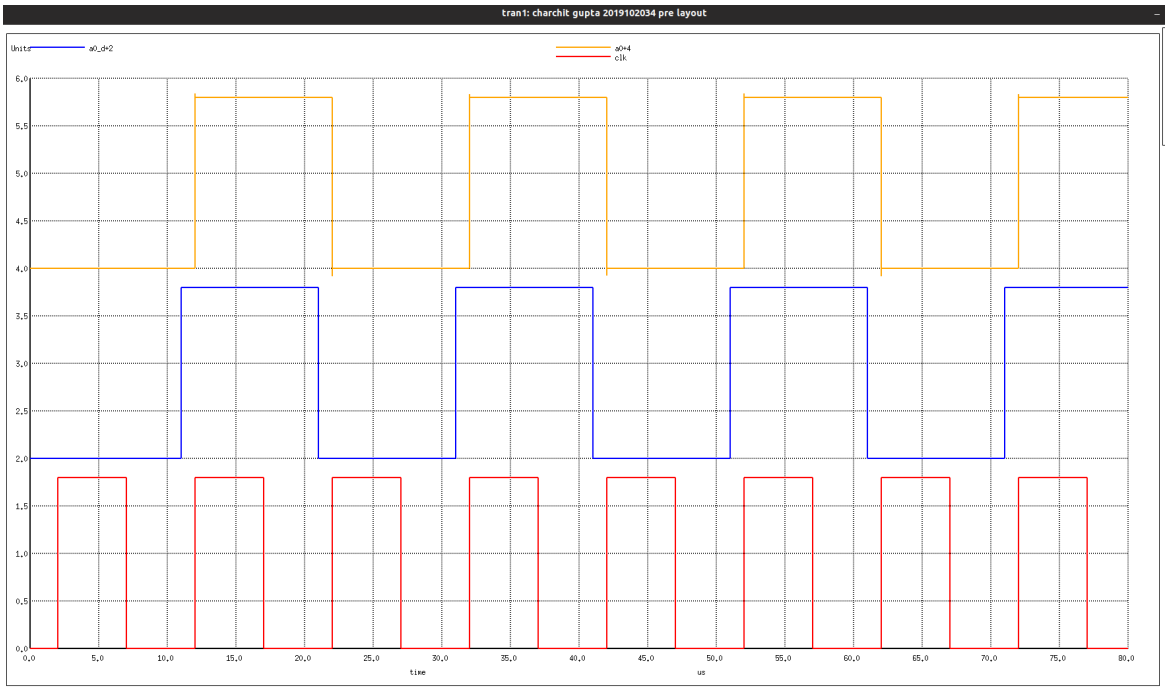

XOR gate



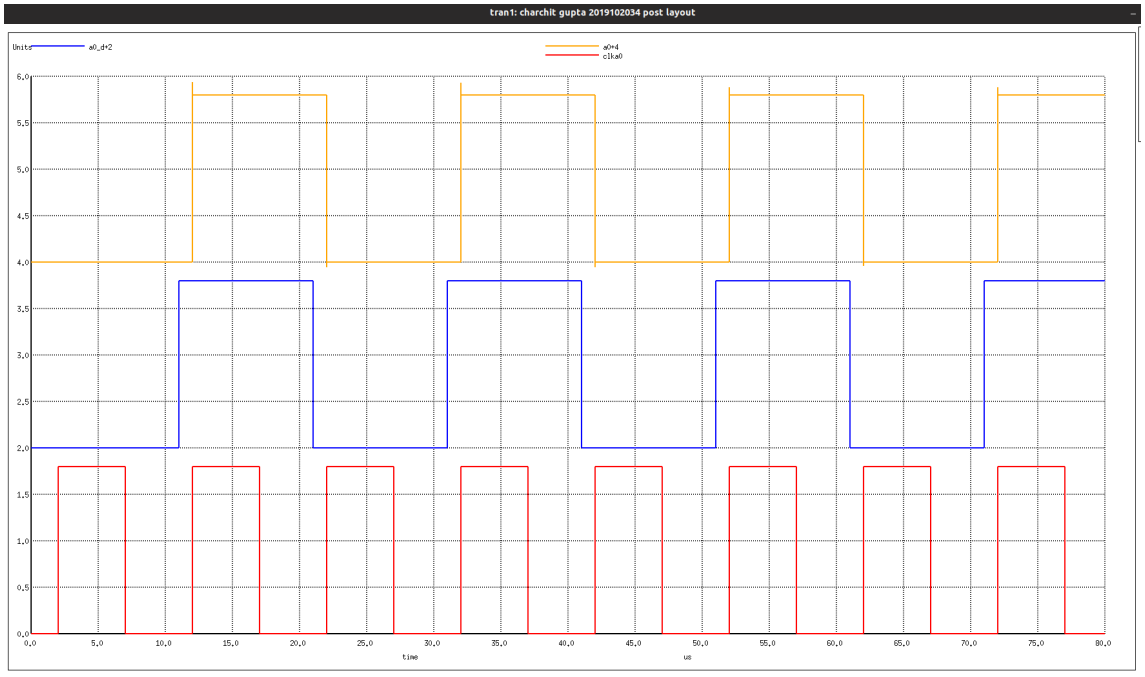
## ***III & VI Simulations***

D-flipflop

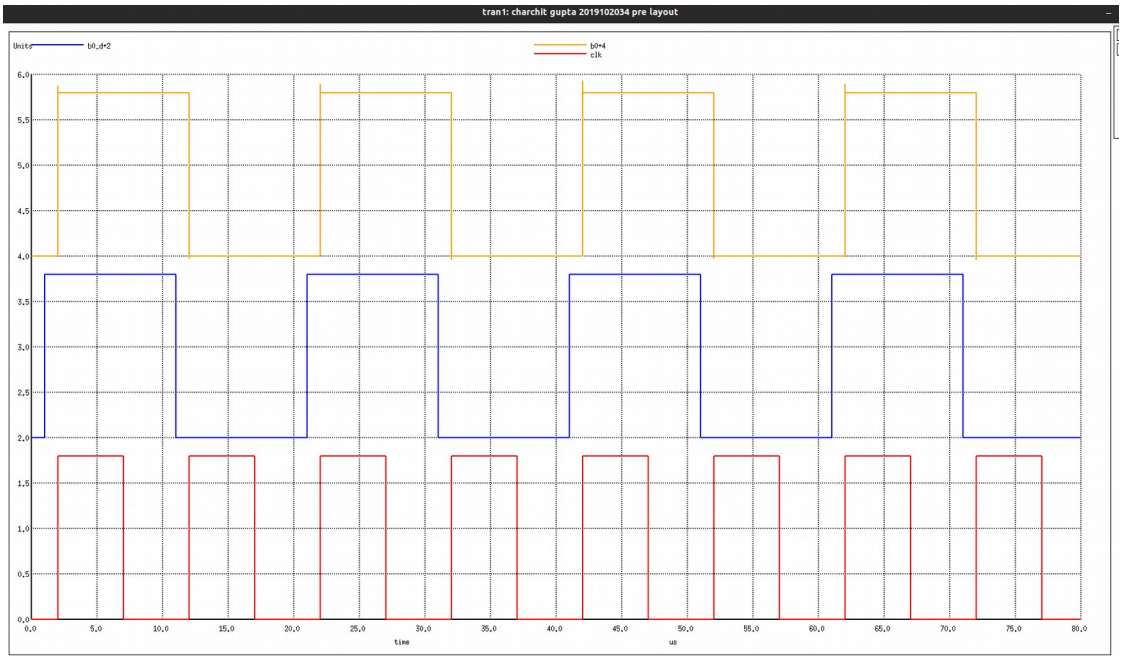
Case1: Pre-layout:-



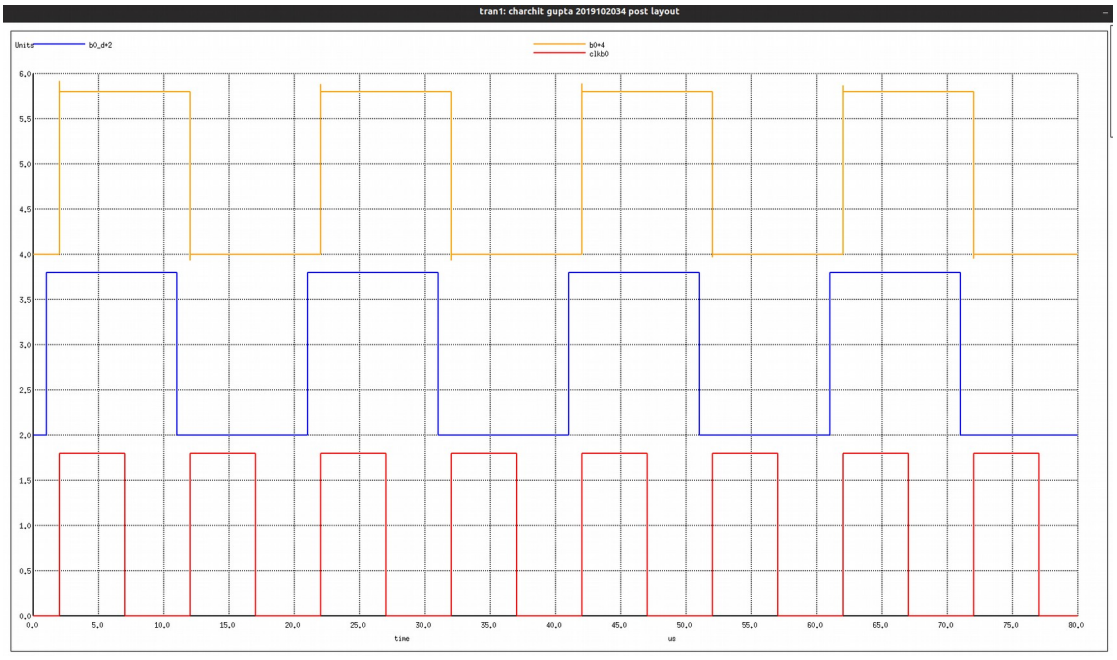

Post-layout:-



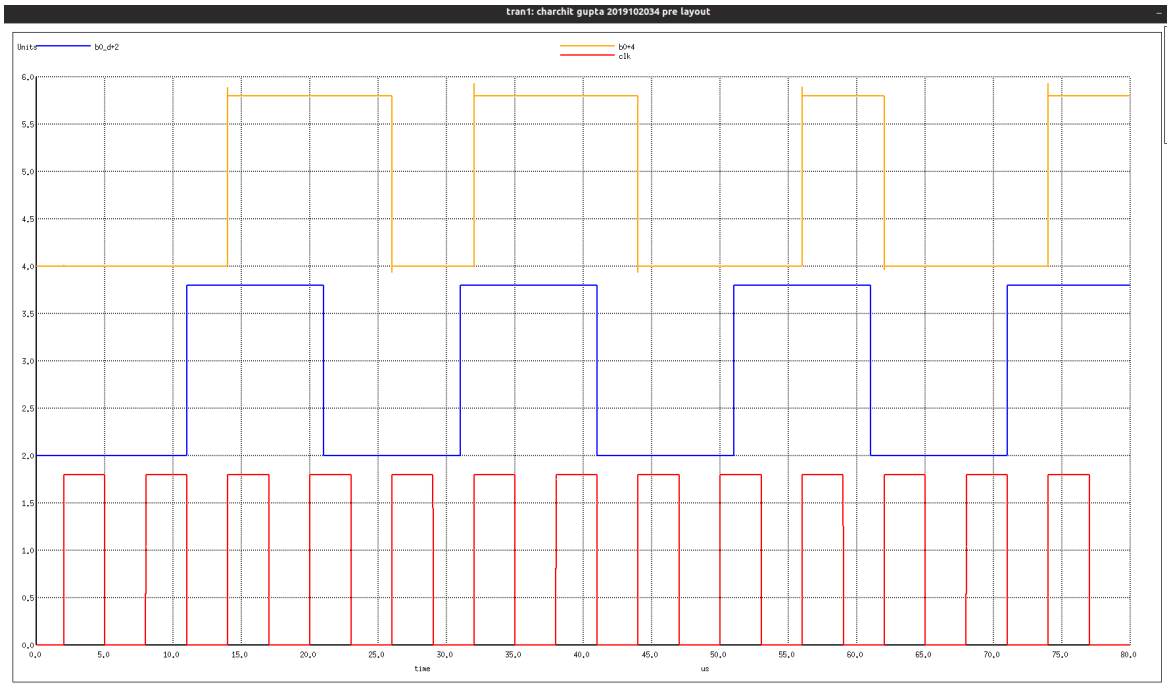
Case2: Pre-layout:-



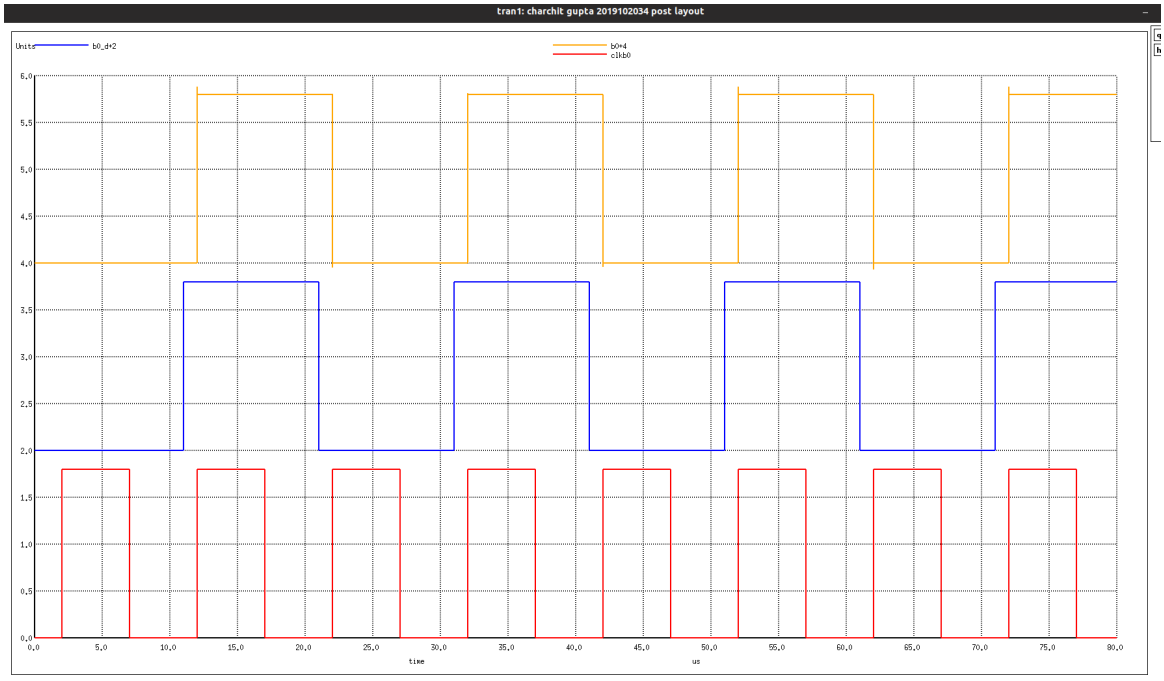

Post-layout:-



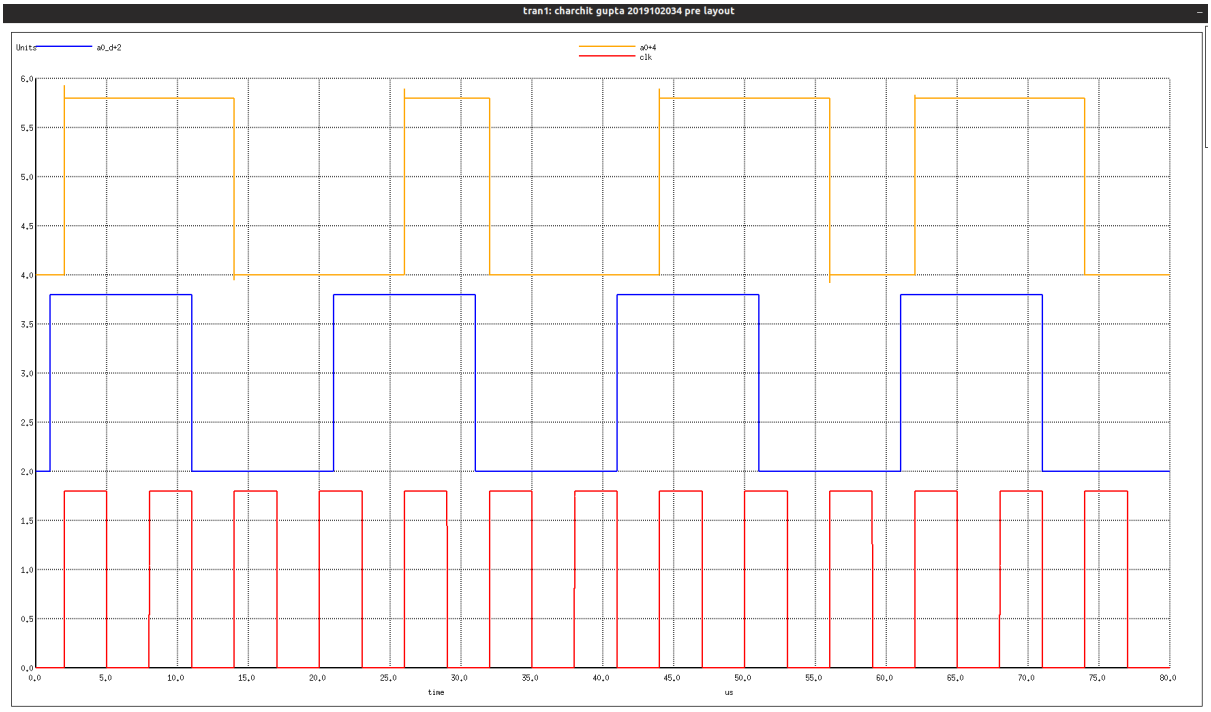
Case3: Pre-layout:-



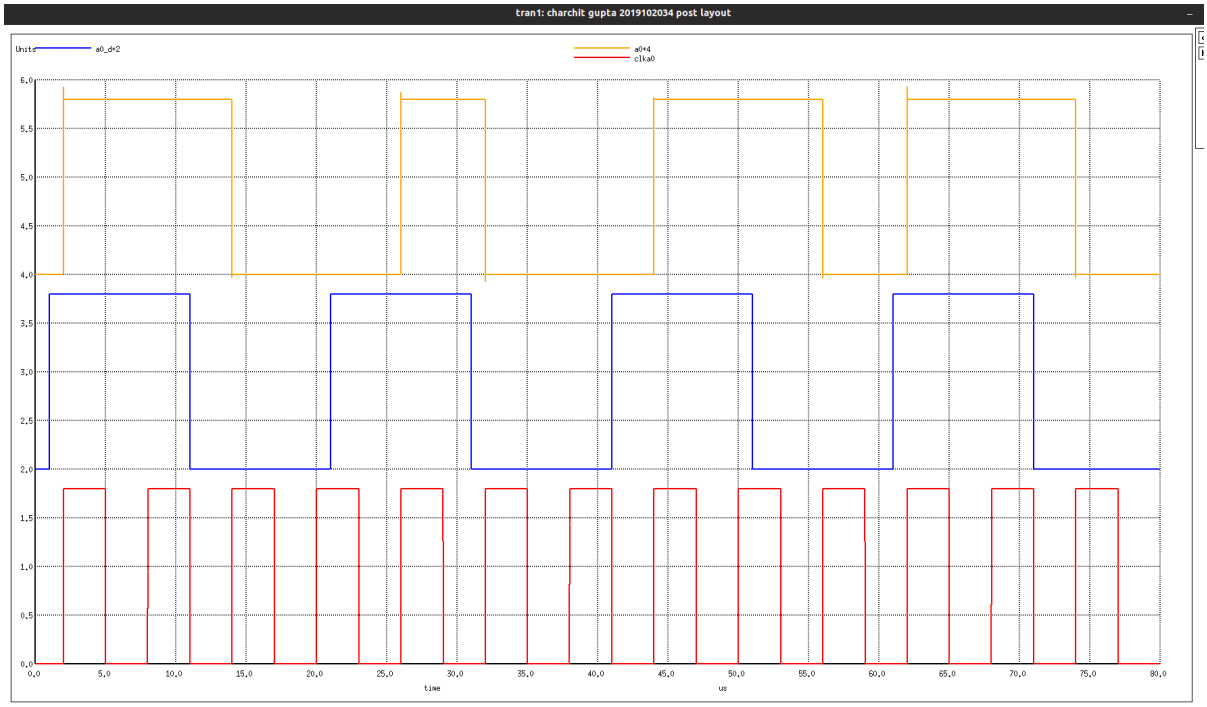

Post-layout:-



Case4: Pre-layout:-




Post-layout:-



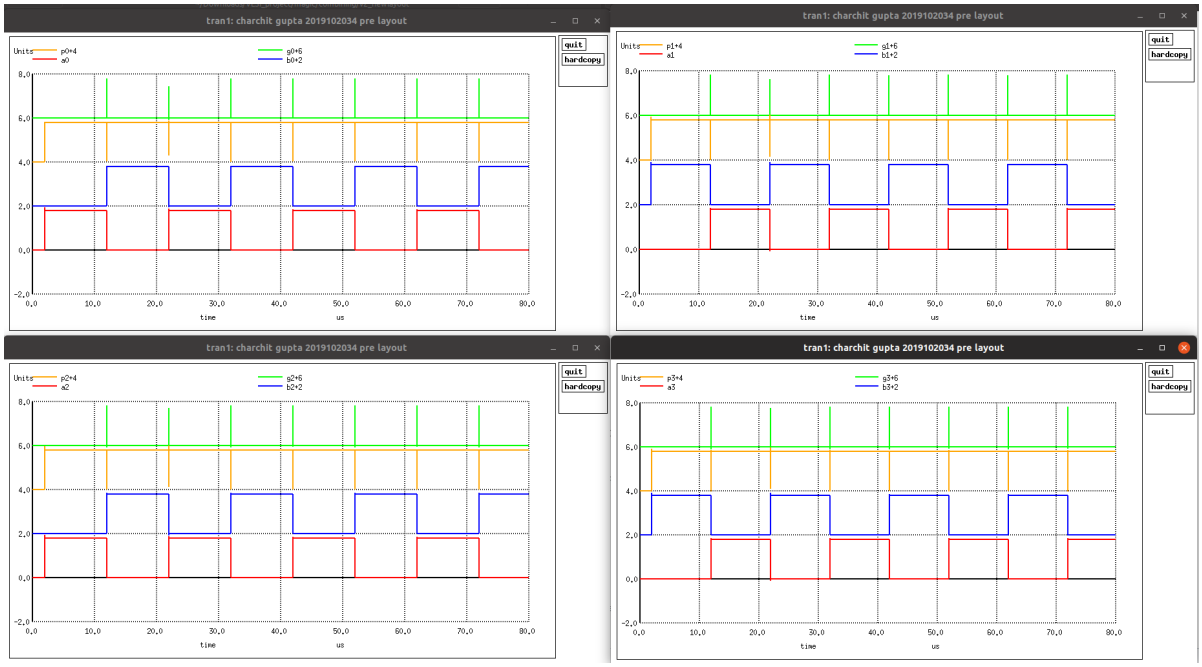
## P/G block

Let A = ($a_3,a_2,a_1,a_0$) and B = ($b_3,b_2,b_1,b_0$)

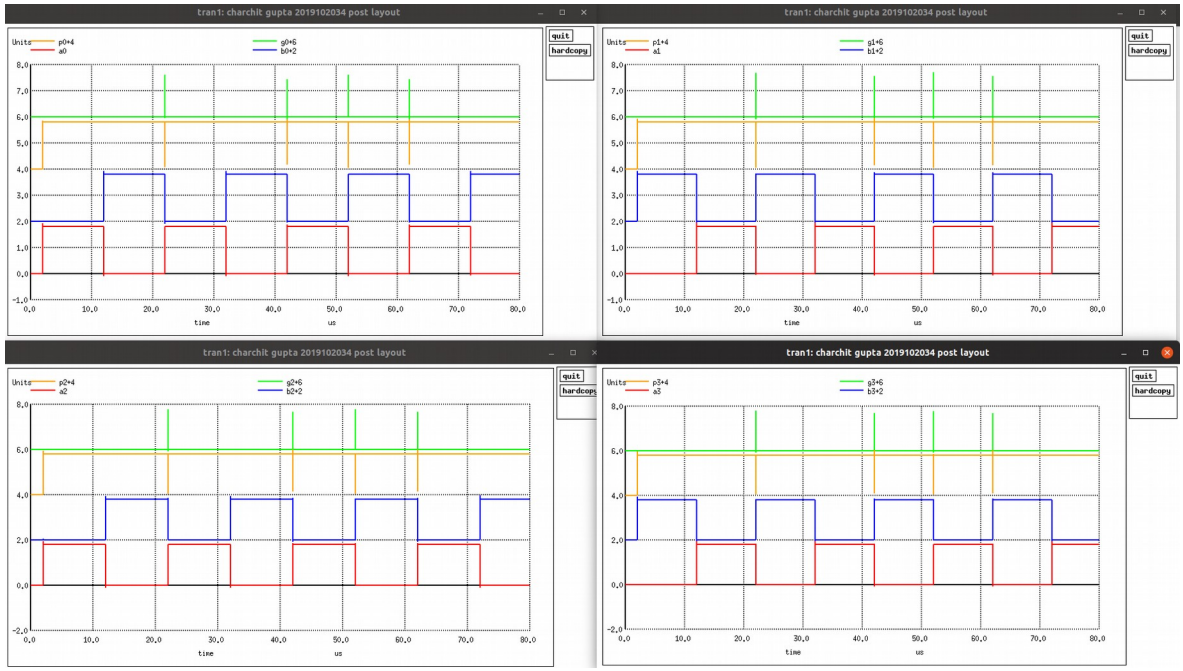
Case1: A = (0,1,0,1) and B = (1,0,1,0) {evaluating at first rising edge}

Output should be P = (1,1,1,1) and G = (0,0,0,0)

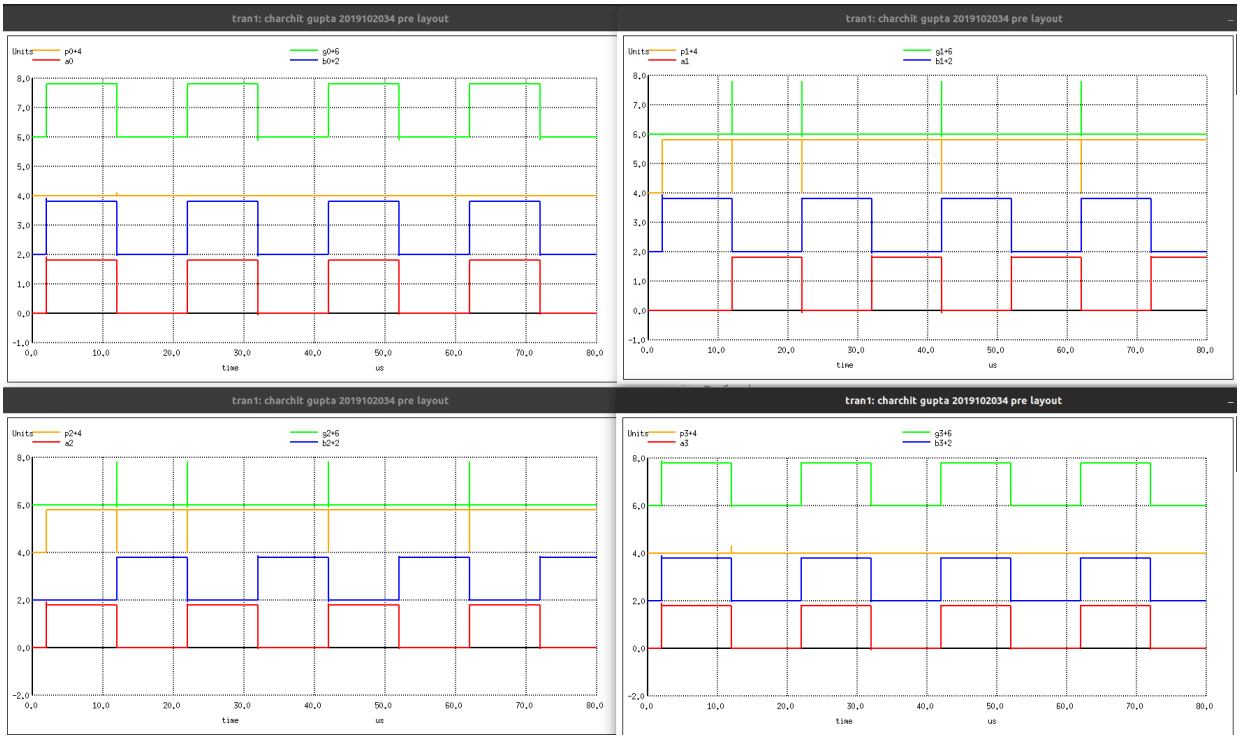
Pre-layout:-



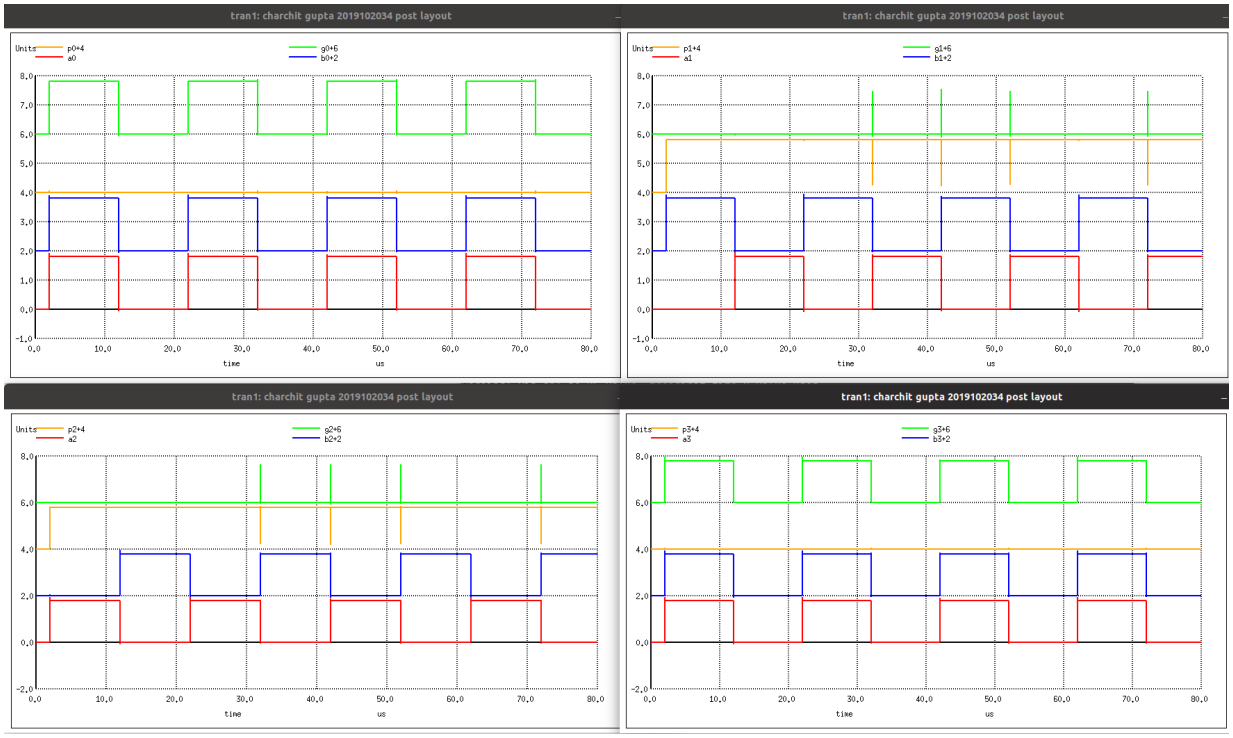

Post-layout:-



Case2:  A = (1,1,0,1) and B = (1,0,1,1)  {evaluating at first rising edge}
Output should be P = (0,1,1,0) and G = (1,0,0,1)
Pre-layout:-




Post-layout:-



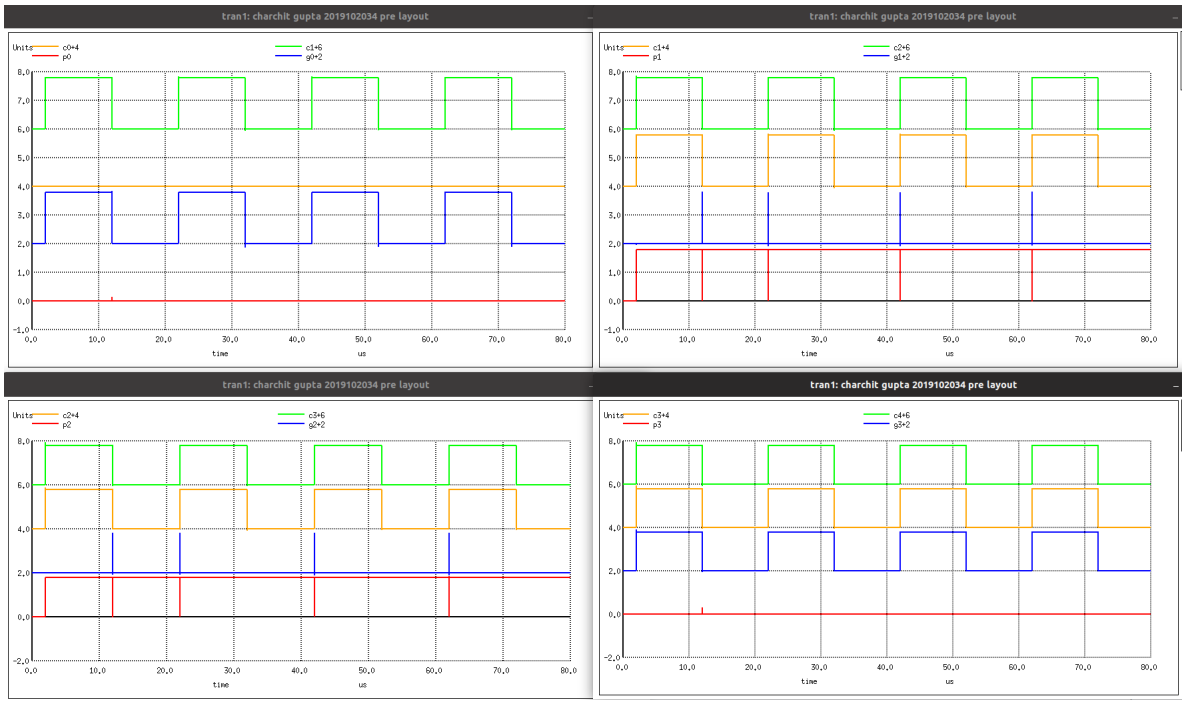
## CLA block

Let P = ($p_3$,$p_2$,$p_1$,$p_0$), G = ($g_3$,$g_2$,$g_1$,$g_0$) and C = ($c_4$,$c_3$,$c_2$,$c_1$)

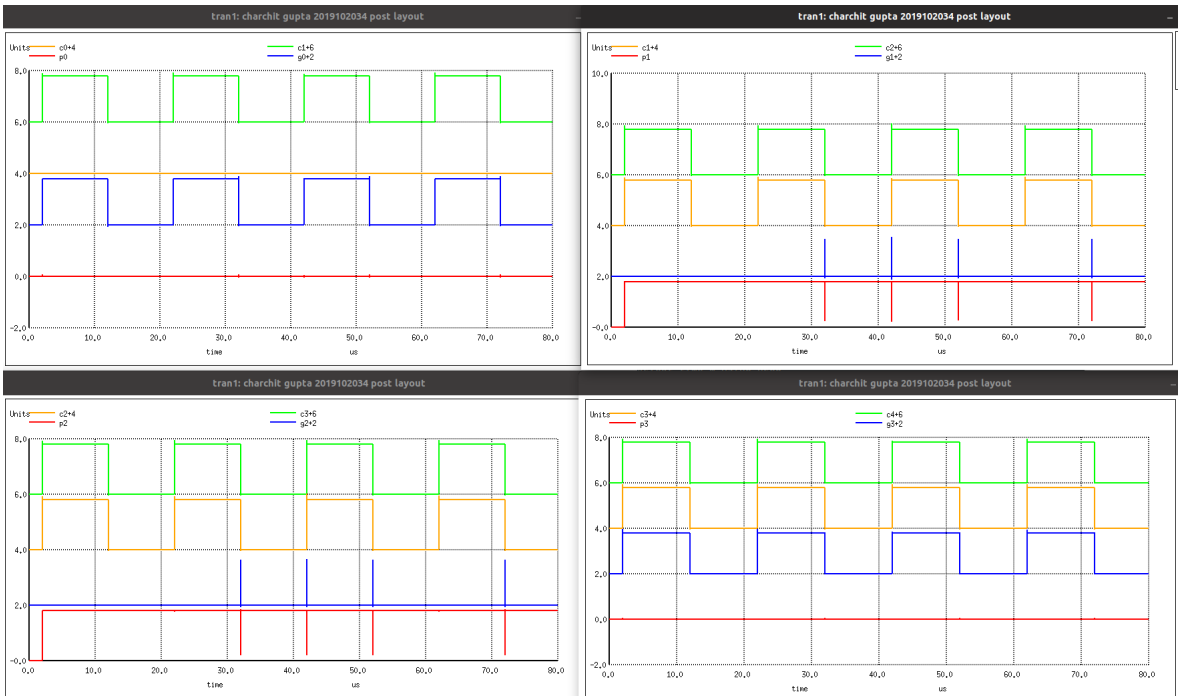
Case1: P = (0,1,1,0) and G = (1,0,0,1) {evaluating at first rising edge}

Output should be C = (1,1,1,1)

Pre-layout:-




Post-layout:-



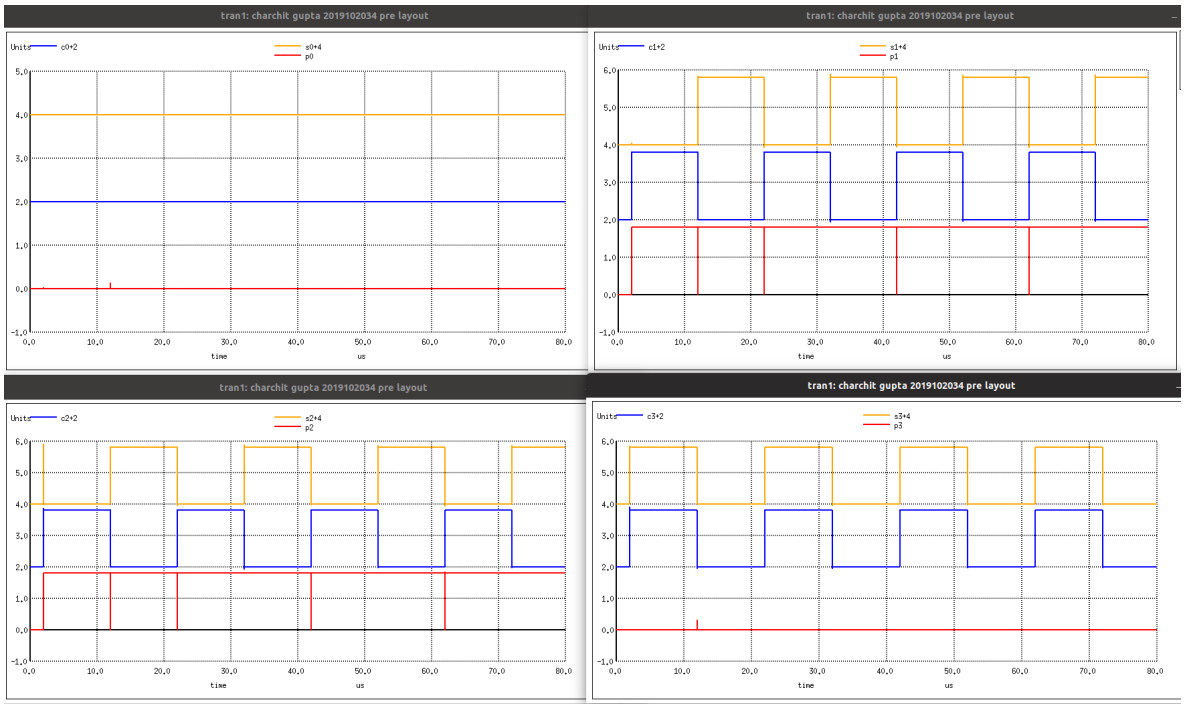
## Sum block

Let P = ($p_3$,$p_2$,$p_1$,$p_0$) and C = ($c_4$,$c_3$,$c_2$,$c_1$)

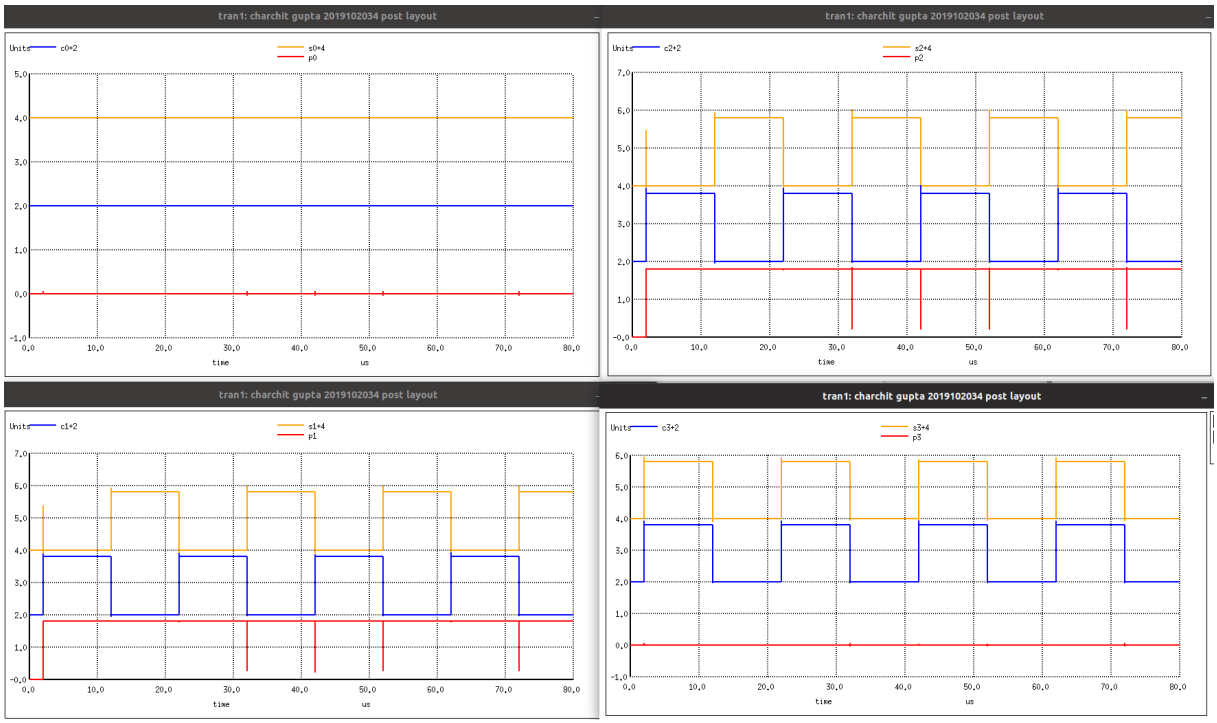
Case1: P = (0,1,1,0) and C = (1,1,1,1) {evaluating at first rising edge}

Output should be S = (1,0,0,0)

Pre-layout:-

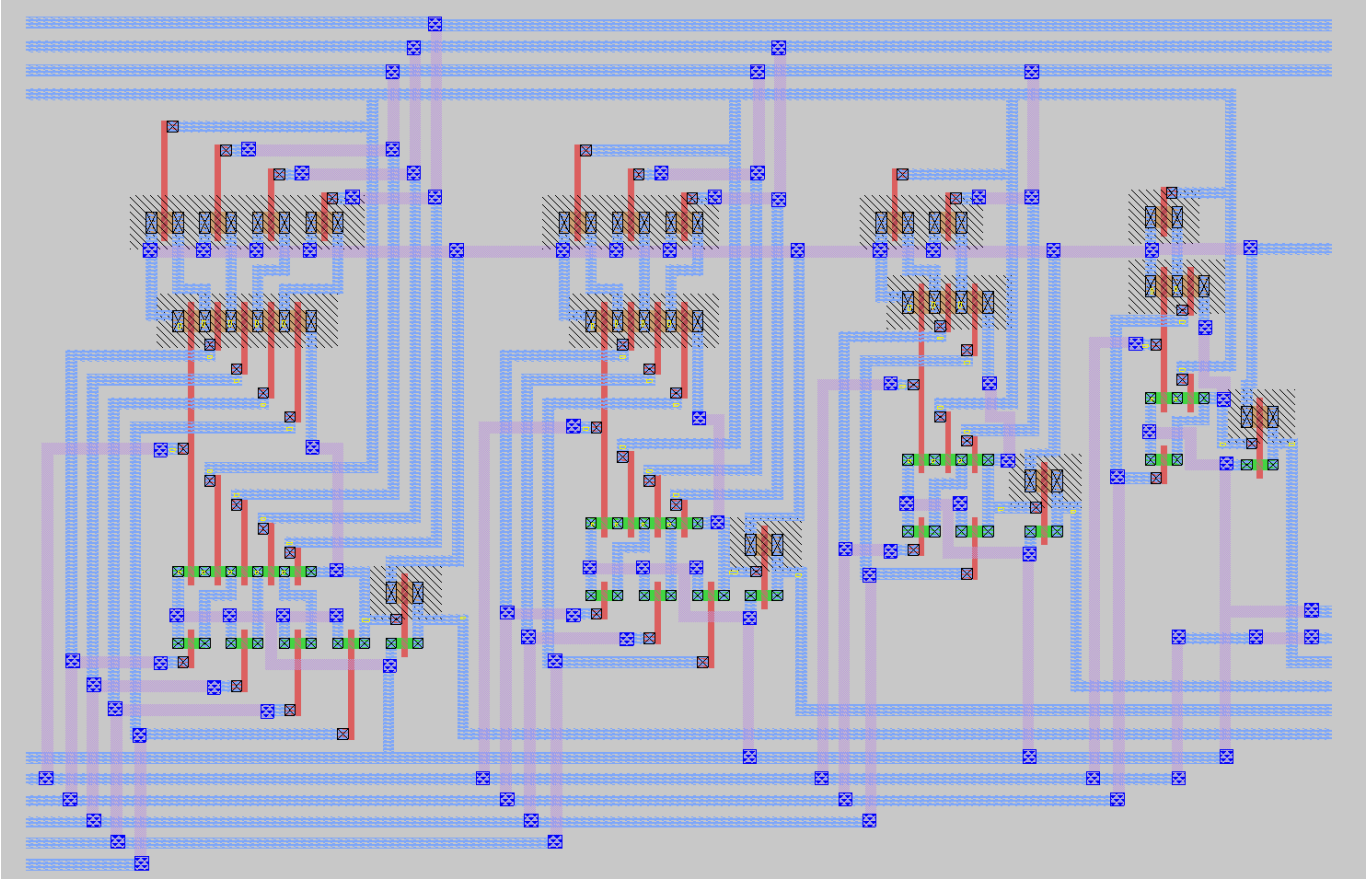



Post-layout:-

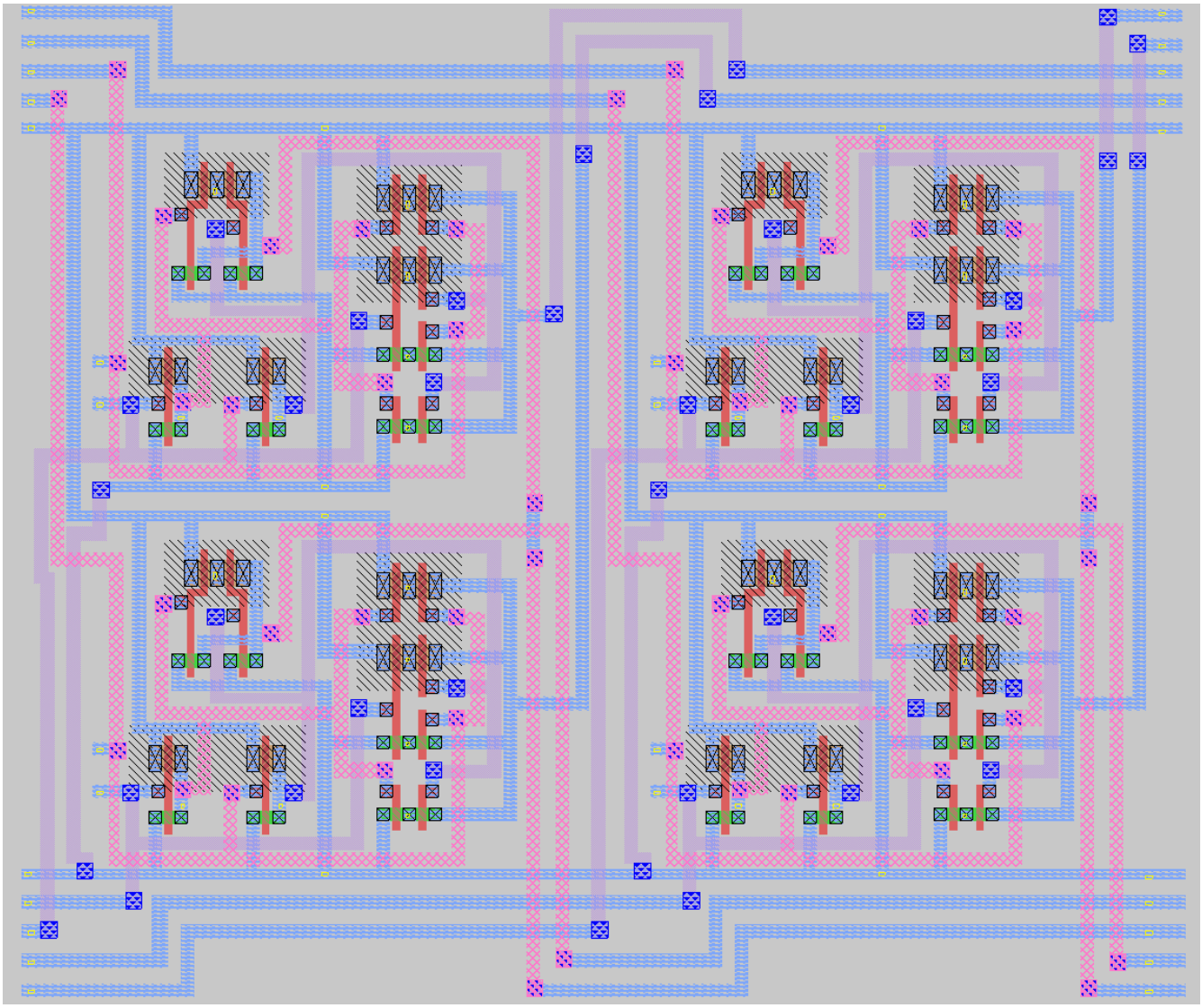


## ***Layouts***

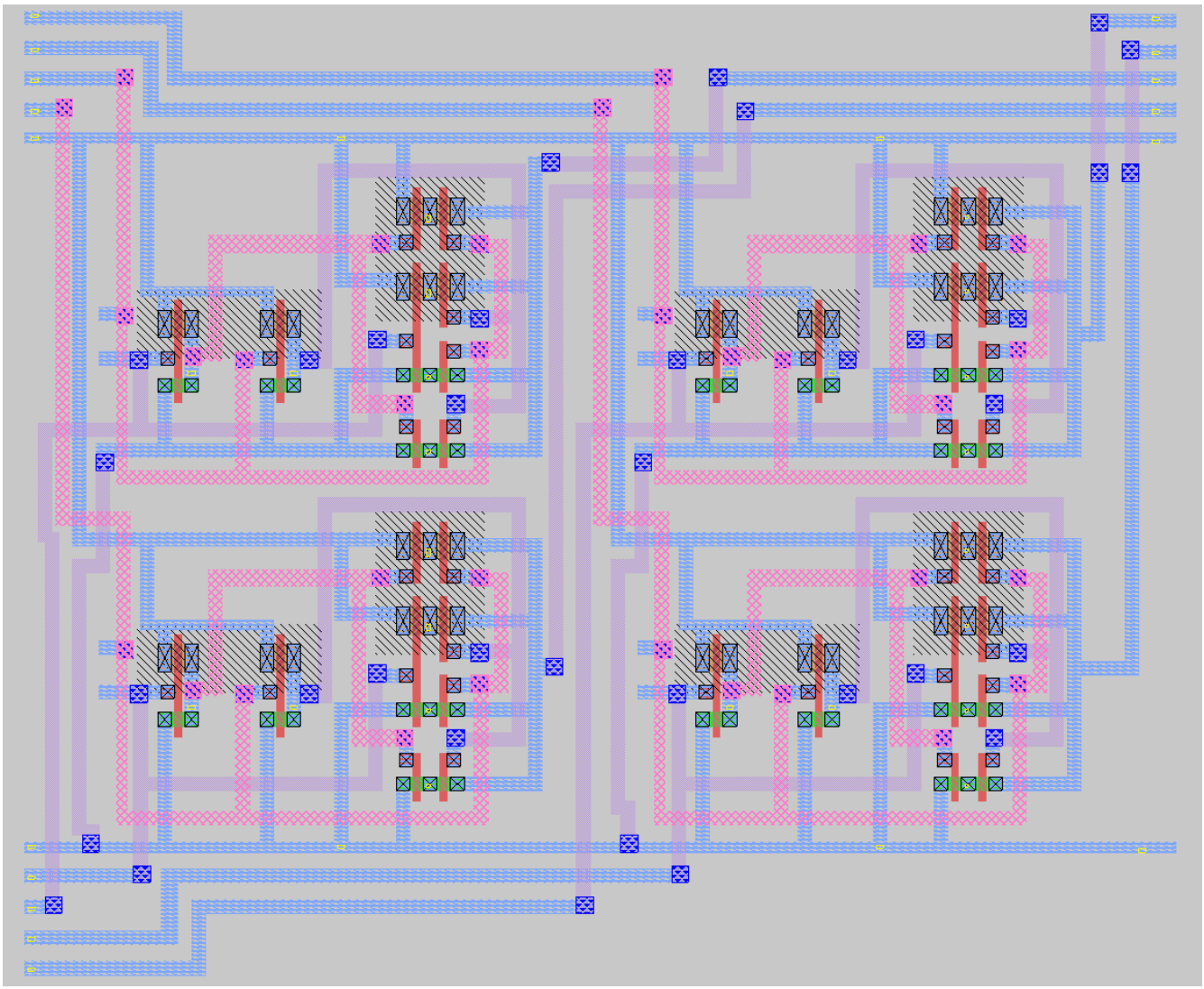
CLA block



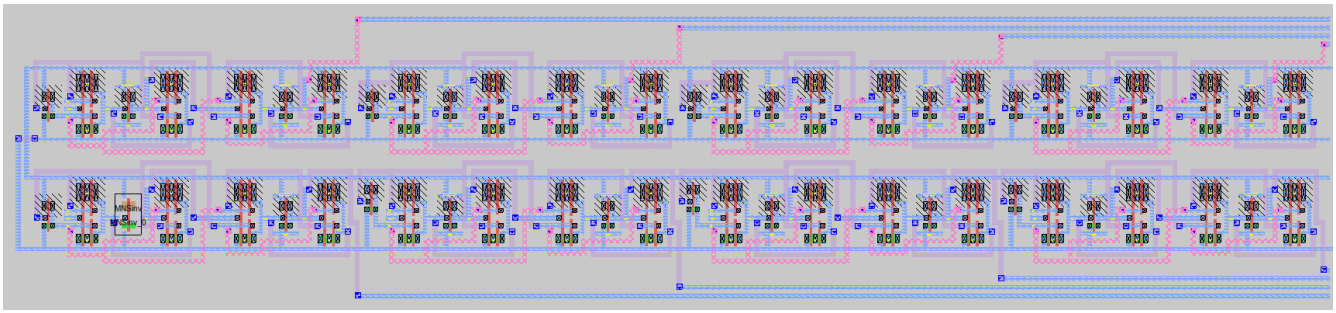
PG Block



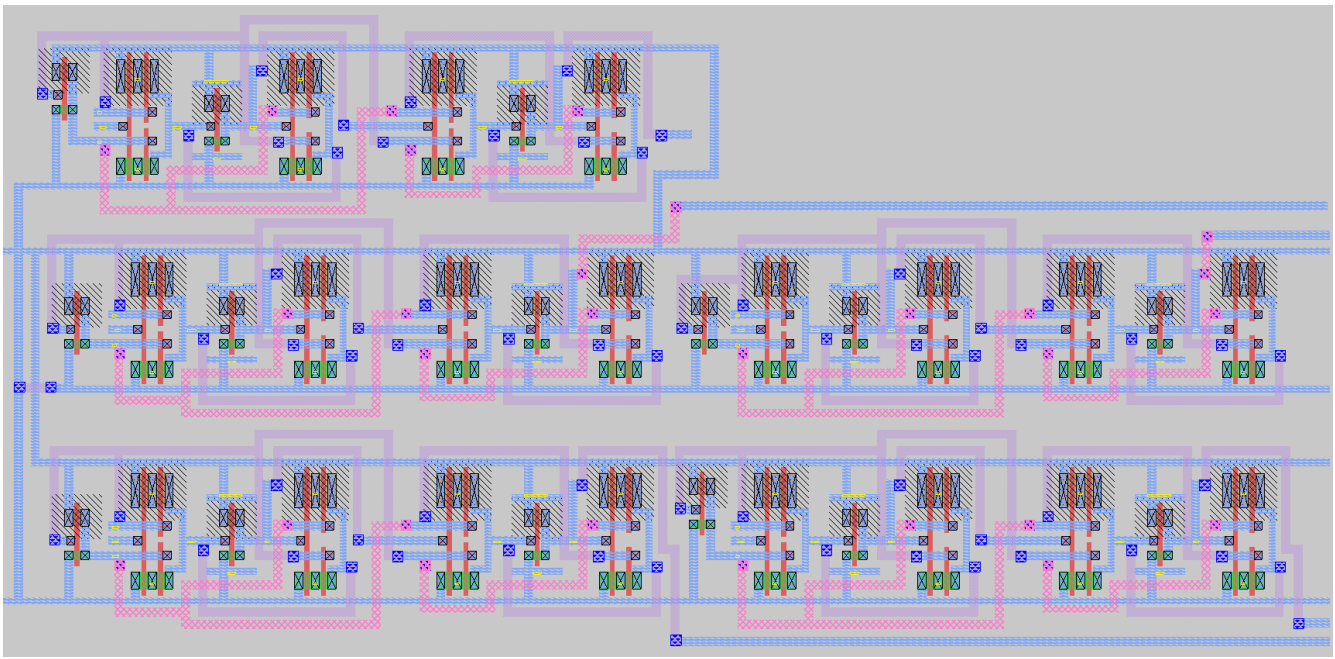
Sum block



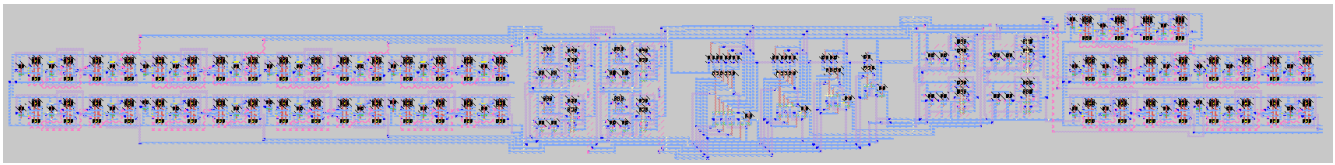
D-flipflop-input



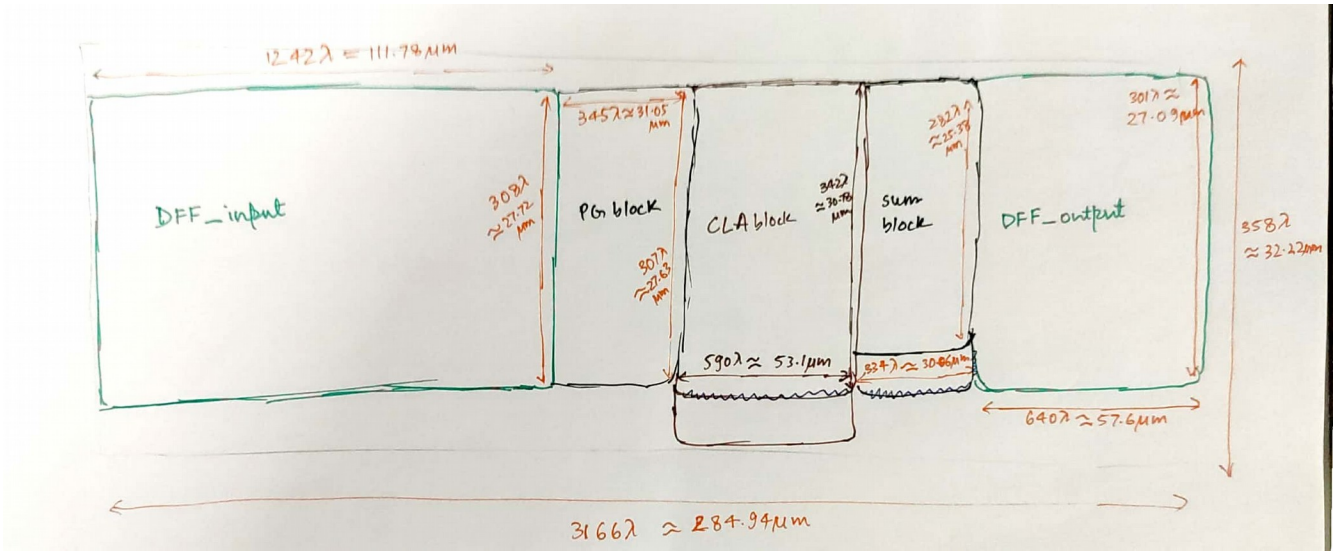
D-flipflop-output



Full-Layout

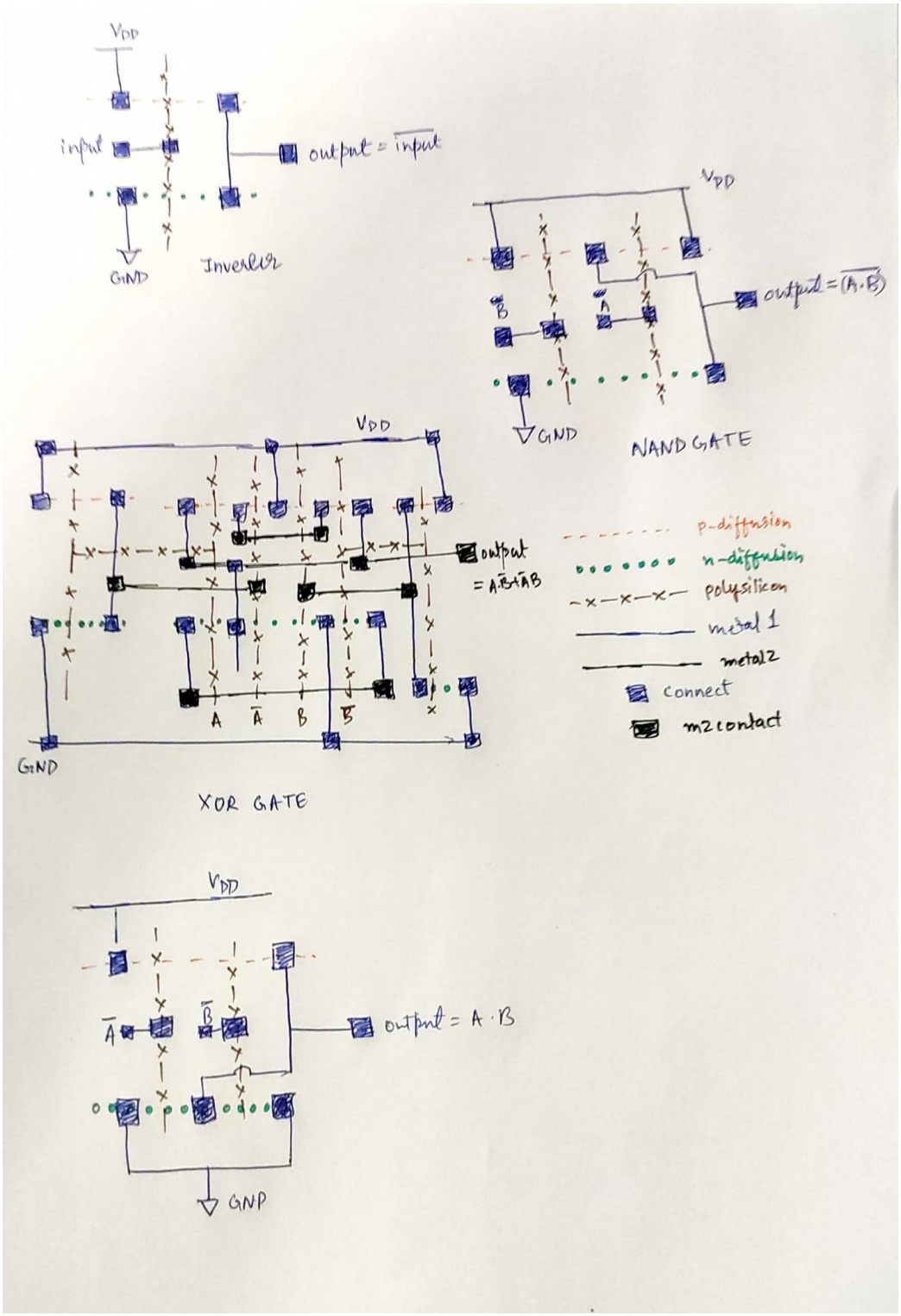


Floor-Plan




Horizontal pitch:3166*lambda
Vertical pitch:358*lambda
Area:1133428*lambda$^2$

## ***V. Stick Diagrams***



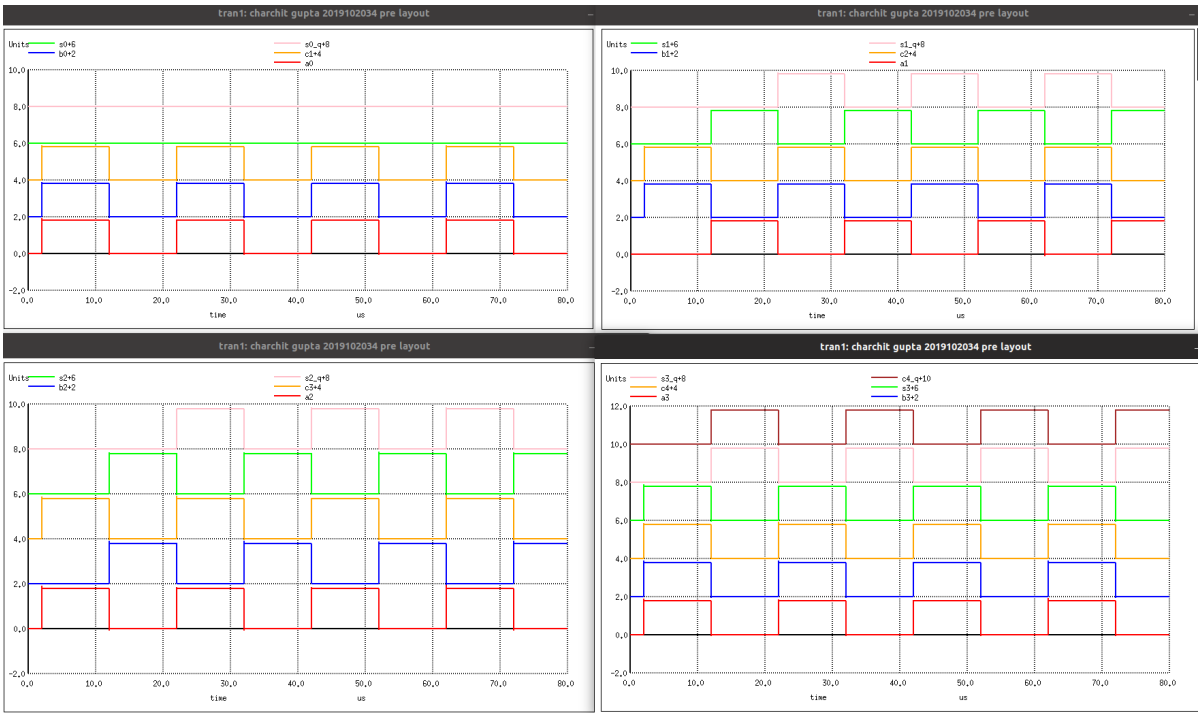
## ***VII &IX. Final Simulations***

Let A = ($a_3$,$a_2$,$a_1$,$a_0$) and B = ($b_3$,$b_2$,$b_1$,$b_0$)

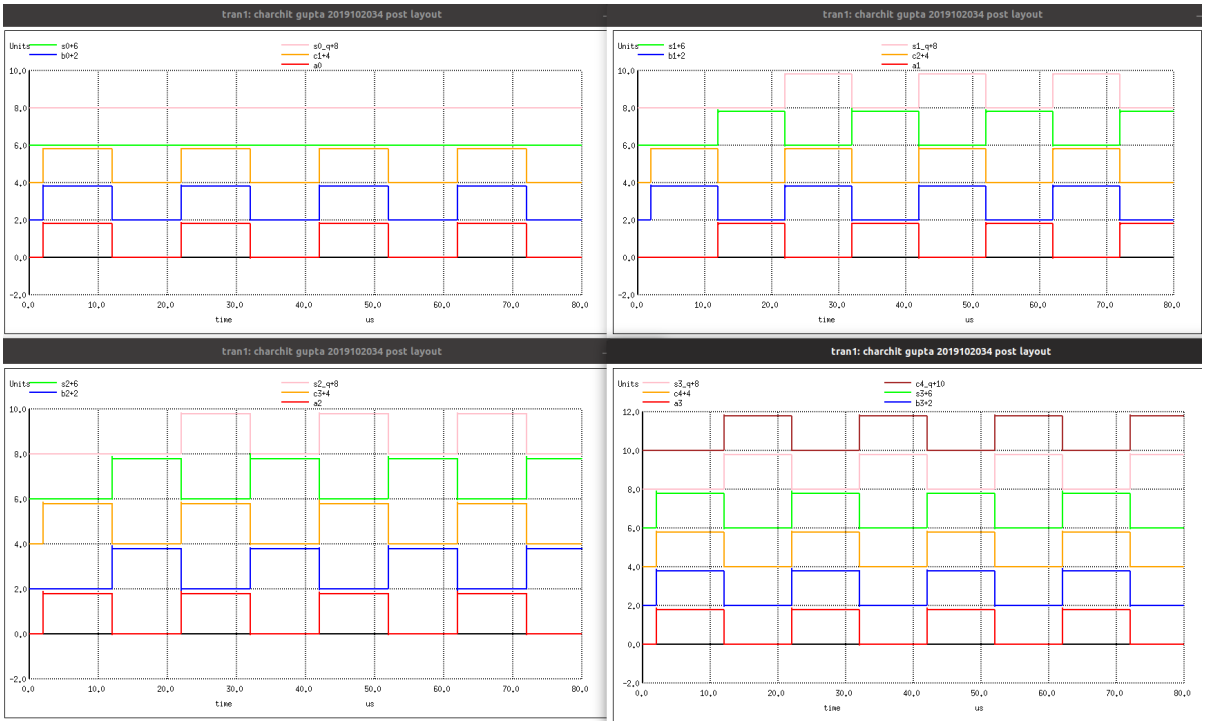
Case1: A = (1,1,0,1) and B = (1,0,1,1) {evaluating at first rising edge}

Output should be S = (1,0,0,0), C = (1,1,1,1) and c4_q, S_q should be found at the next clock cycle.

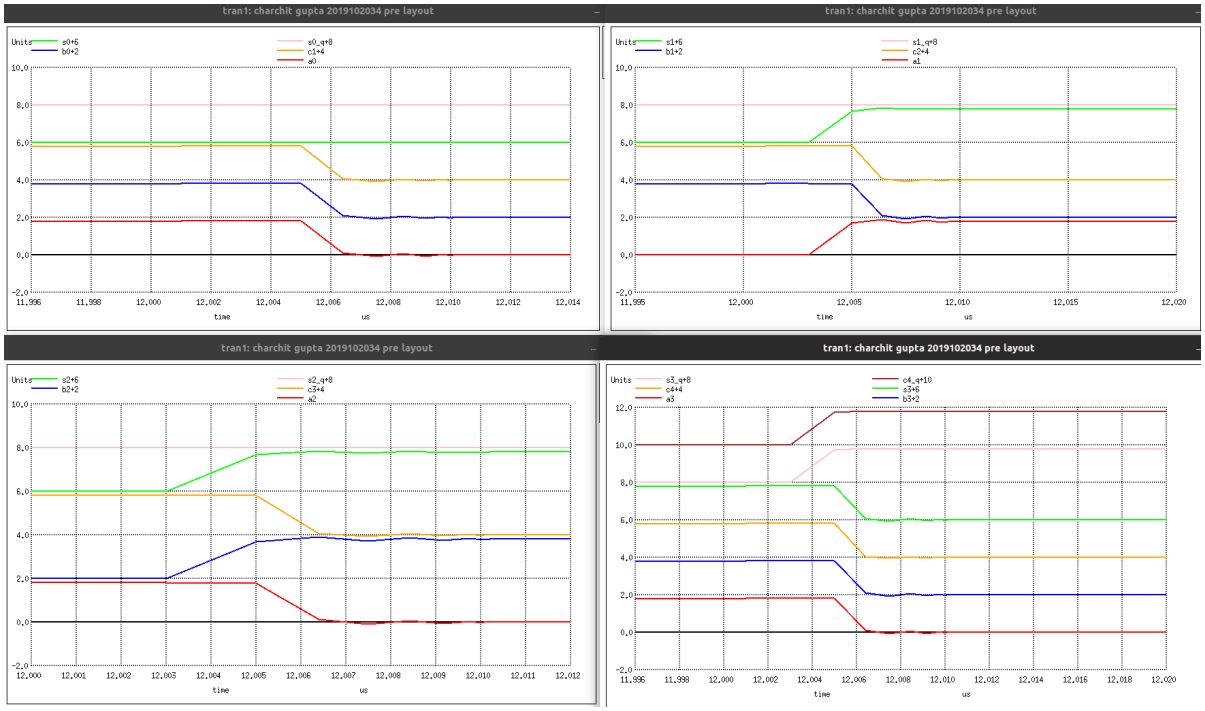
Pre-layout:-



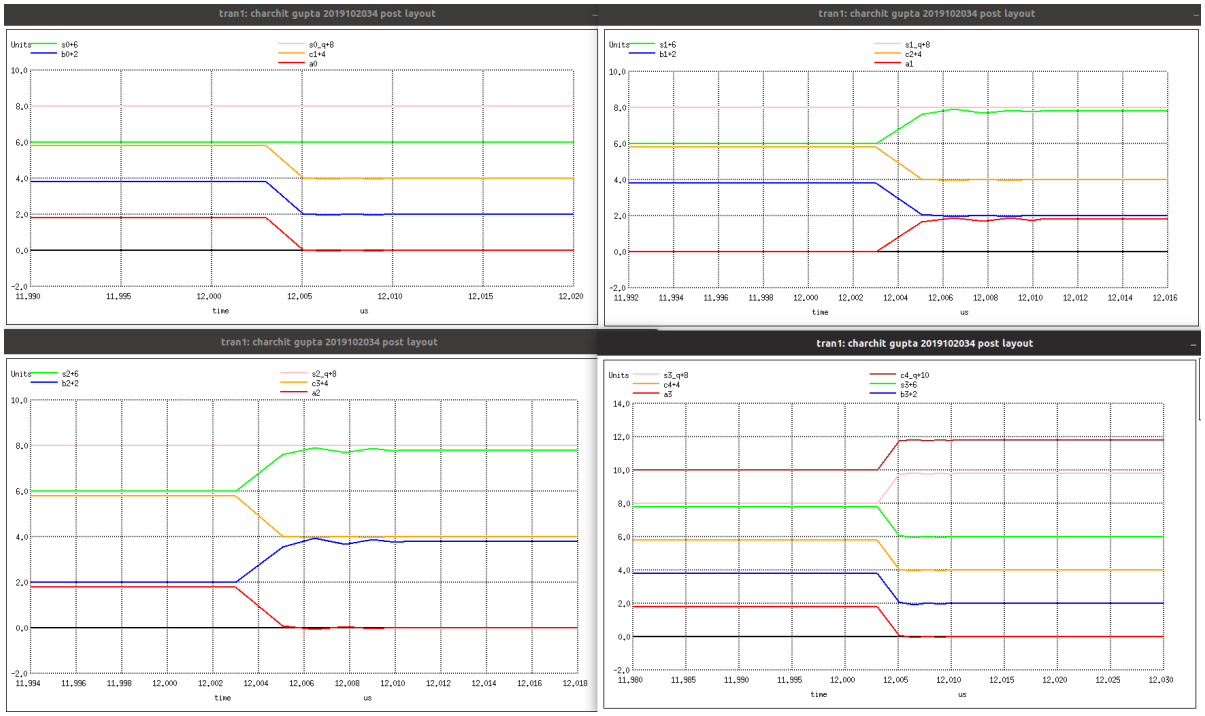

Post-layout:-



Pre-layout-zoom:

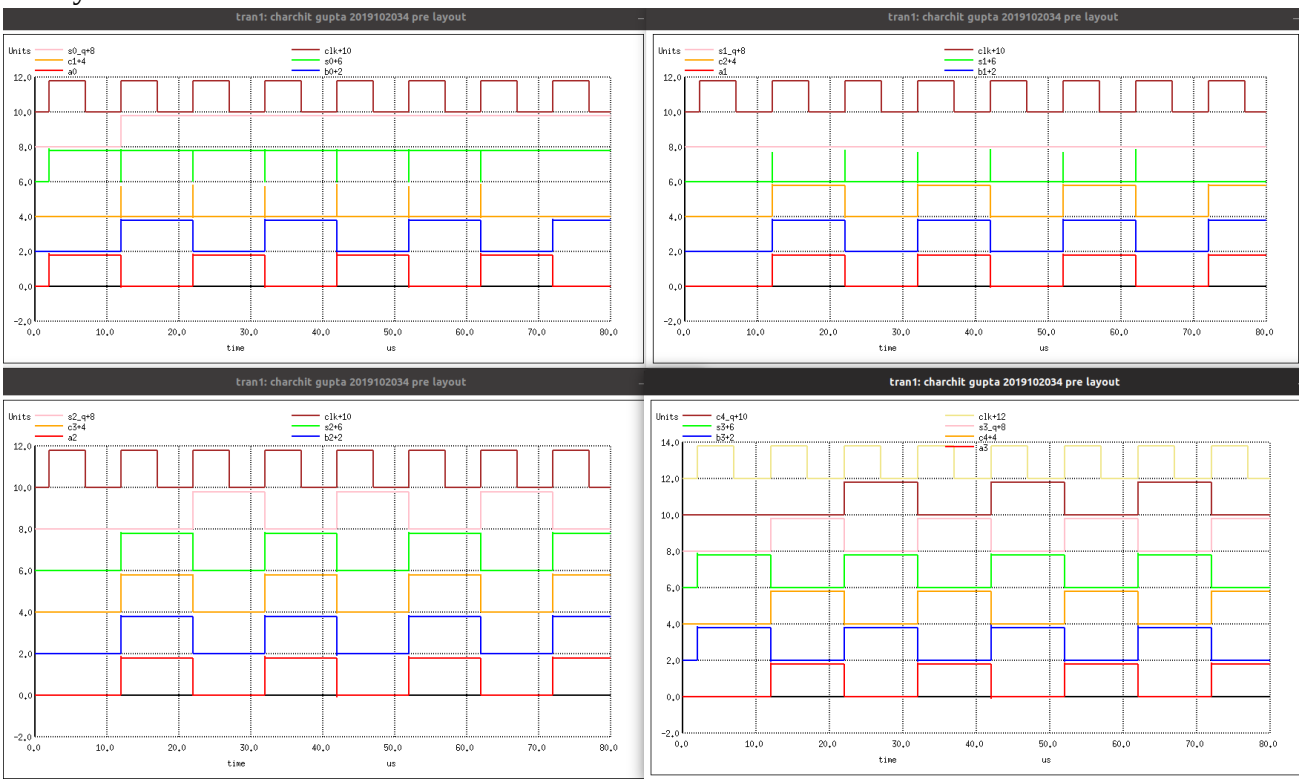



Post-layout-zoom:

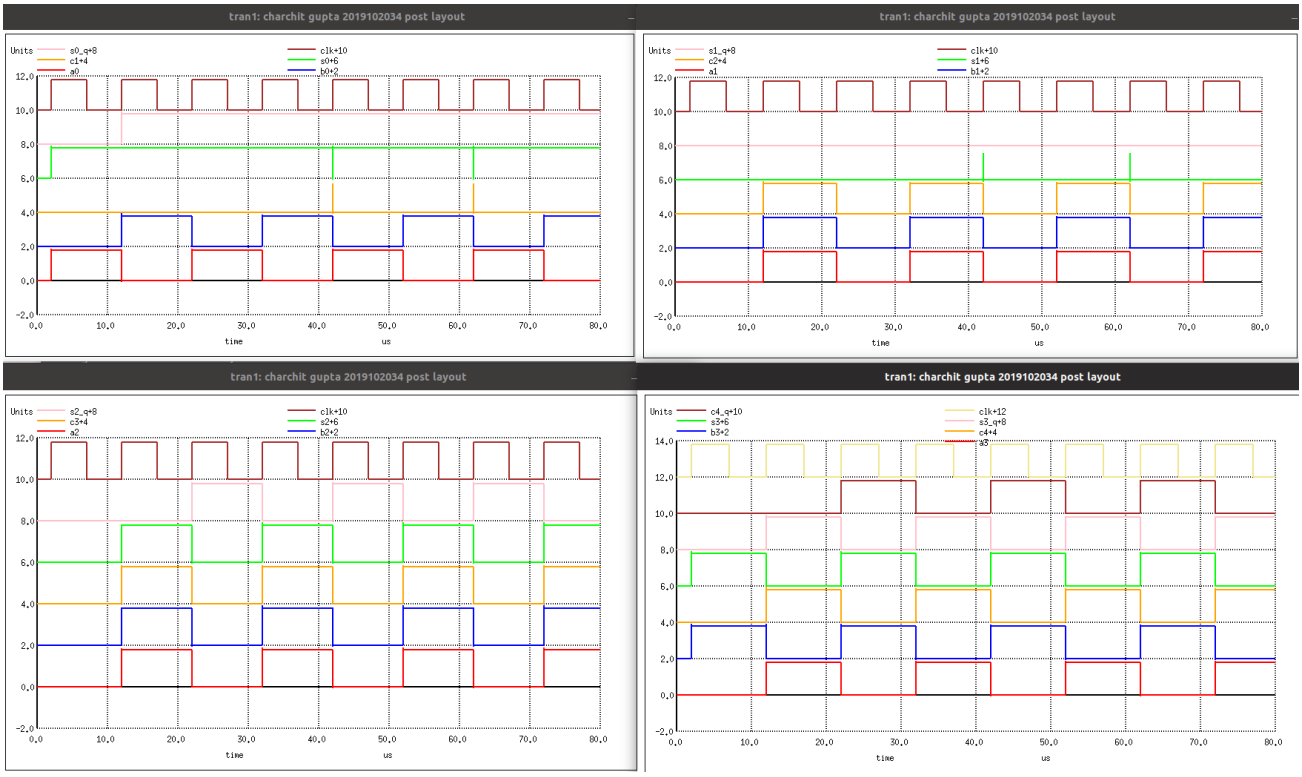



**Clock starts at 2 micro secs, and has a time period of 10 micro seconds. Will be shown in the next cases.*

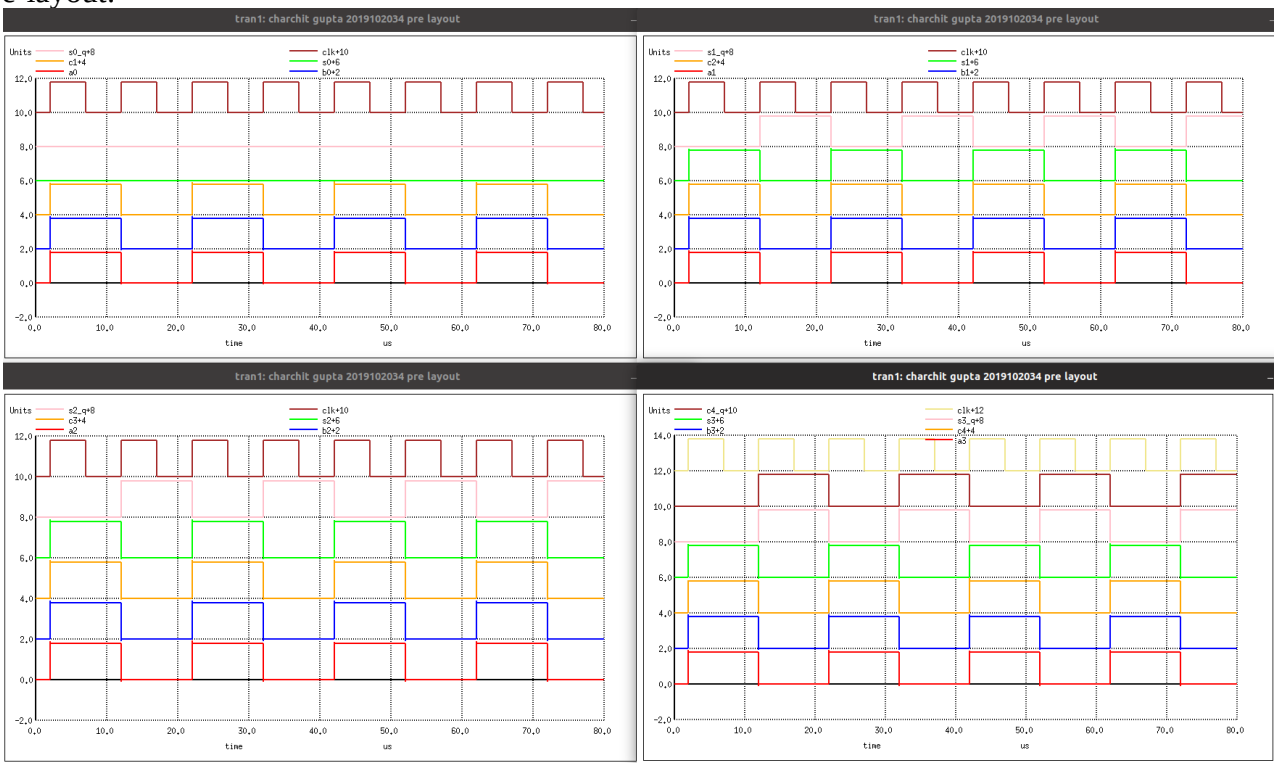
Case2:  A = (0,0,0,1) and B = (1,0,0,0)  {evaluating at first rising edge}
Output should be S = (1,0,0,1), C = (0,0,0,0) and c4_q, S_q should be found at the next clock cycle.
Pre-layout:-



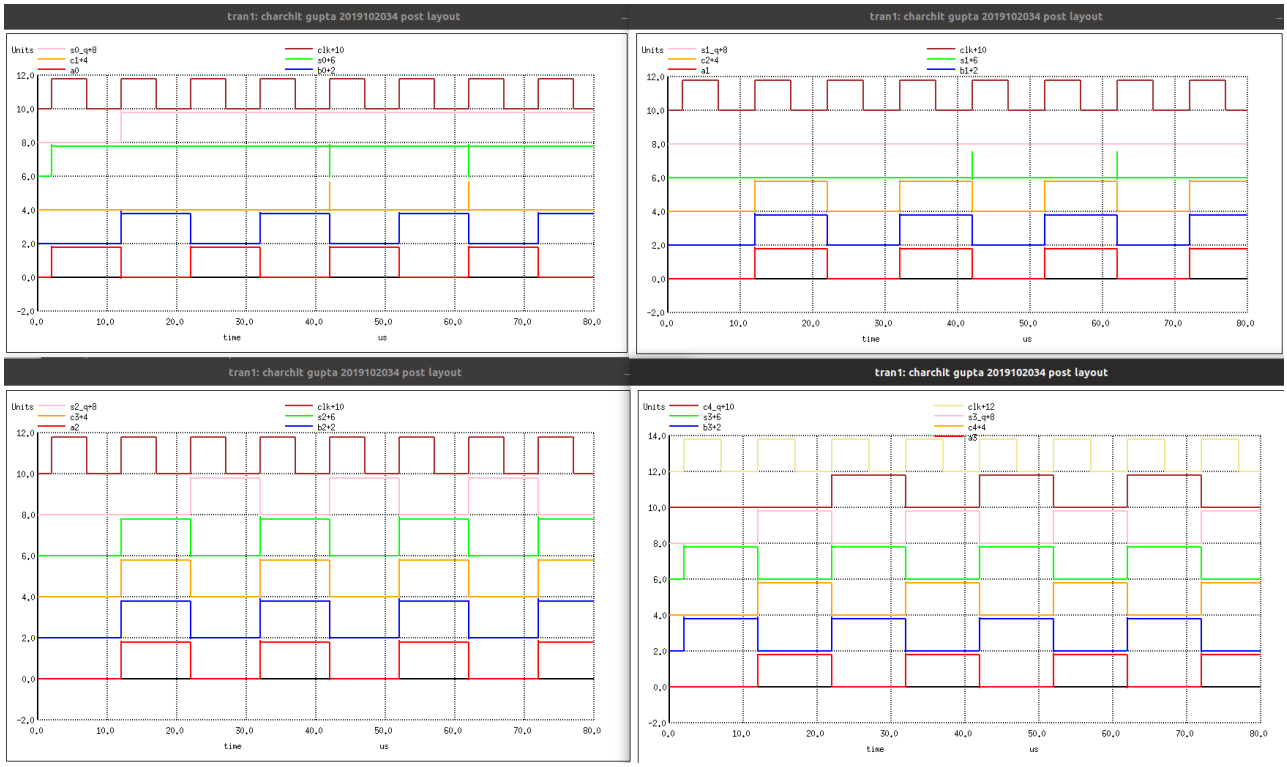

Post-layout:-



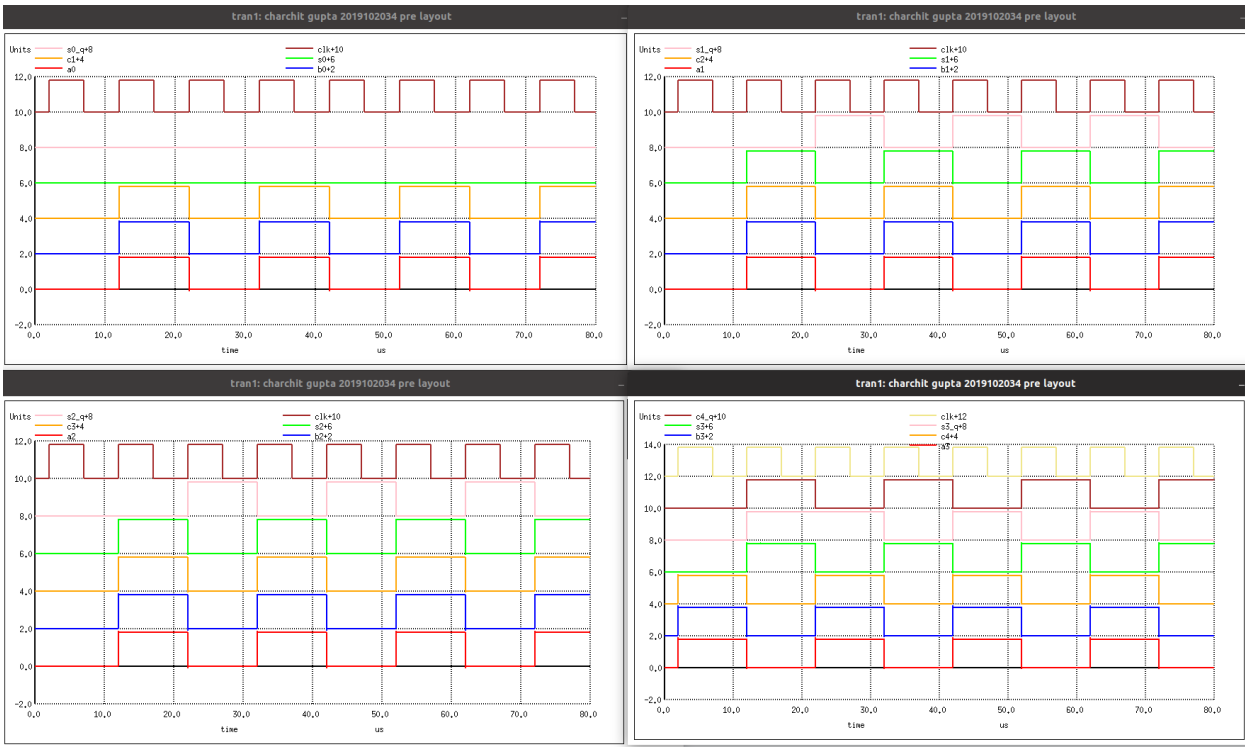
Case3:  A = (1,1,1,1) and B = (1,1,1,1)  {evaluating at first rising edge}
Output should be S = (1,1,1,0), C = (1,1,1,1) and c4_q, S_q should be found at the next clock cycle.
Pre-layout:-



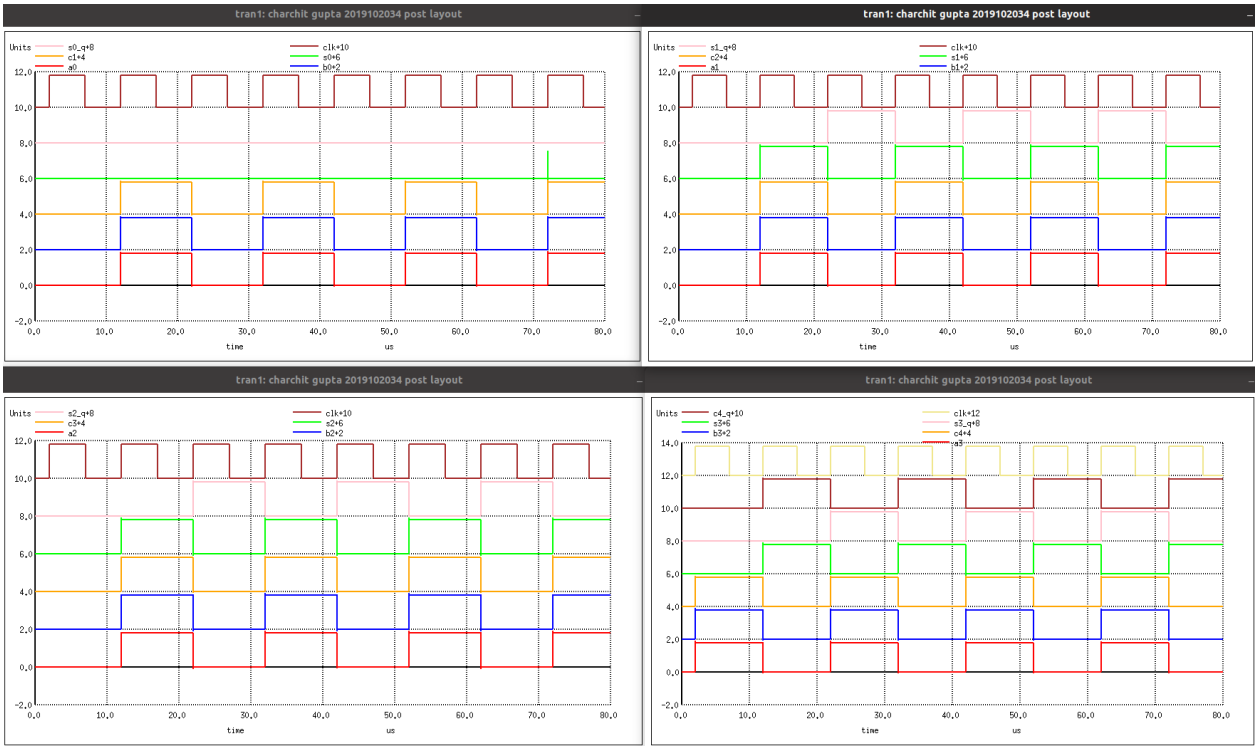

Post-layout:-



Case4:  A = (1,0,0,0) and B = (1,0,0,0)  {evaluating at first rising edge}
Output should be S = (0,0,0,0), C = (1,0,0,0) and c4_q, S_q should be found at the next clock cycle.
Pre-layout:-

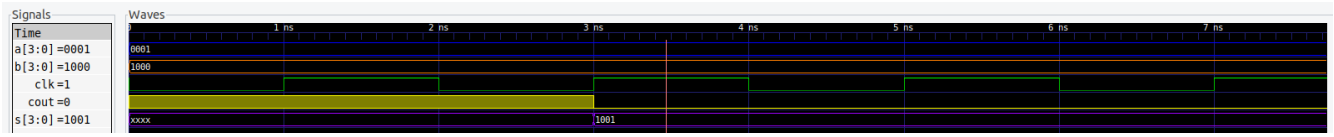



Post-layout:-

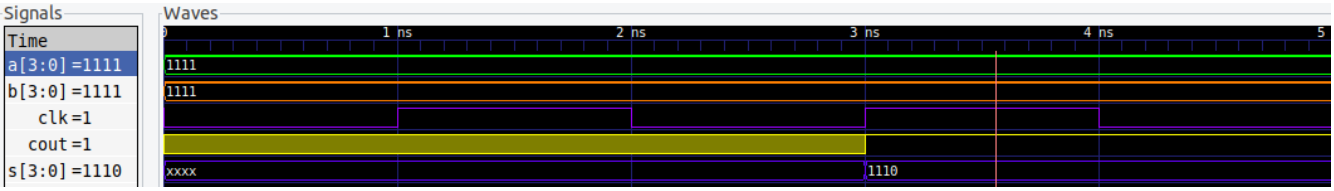


## ***XI. Verilog***

Correctness of the above outputs can be verified by the following gtkwaves:






Structural description:

```
`timescale 1ns / 1ps
module test_bench;

reg clk;
reg [3:0] a, b;
wire [3:0] s;
wire cout;

adder runadd(
    .a(a),
    .b(b),
    .clk(clk),
    .cout(cout),
    .s(s)
);

initial clk = 0;

initial begin
    $dumpfile("adder.vcd");
    $dumpvars(0,test_bench);
    a = 8;
    b = 8;
    #1 clk = 1;
    #1 clk = 0;
    #1 clk = 1;
    #1 clk = 0;
#1 clk = 1;
#1 clk = 0;
#1 clk = 1;
#1 clk = 0;

end

endmodule
```

```
module dflipflop(
    input d,
    input enable,
    output res
);


reg resreg;

always @(posedge enable) begin
    if(enable)
        resreg <= d;
end

assign res = resreg;

endmodule
```

```
module adder(
    input [3:0] a,
    input [3:0] b,
    input clk,
    output cout,
    output [3:0] s
);

wire a0, a1, a2, a3;
wire b0, b1, b2, b3;
wire p0, p1, p2, p3;
wire g0, g1, g2, g3;
wire c1, c2, c3, c4;
wire s0, s1, s2, s3;

dflipflop da0(a[0],clk,a0);
dflipflop da1(a[1],clk,a1);
dflipflop da2(a[2],clk,a2);
dflipflop da3(a[3],clk,a3);
dflipflop db0(b[0],clk,b0);
dflipflop db1(b[1],clk,b1);
dflipflop db2(b[2],clk,b2);
dflipflop db3(b[3],clk,b3);

assign p0 = a0 ^ b0;
assign p1 = a1 ^ b1;
assign p2 = a2 ^ b2;
assign p3 = a3 ^ b3;
assign g0 = a0 & b0;
assign g1 = a1 & b1;
assign g2 = a2 & b2;
assign g3 = a3 & b3;

assign c1 = g0;
assign c2 = (p1 & g0) | g1;
assign c3 = (p2 & ((p1 & g0) | g1)) | g2;
assign c4 = (p3 & ((p2 & ((p1 & g0) | g1)) | g2)) | g3;

assign s0 = p0;
assign s1 = p1 ^ c1;
assign s2 = p2 ^ c2;
assign s3 = p3 ^ c3;

dflipflop ds0(s0,clk,s[0]);
dflipflop ds1(s1,clk,s[1]);
dflipflop ds2(s2,clk,s[2]);
dflipflop ds3(s3,clk,s[3]);
dflipflop dcout(c4,clk,cout);

endmodule
```

***Table***

$$T_{clk} >= t_{cq} + t_{pdmax} + t_{setup}$$

| | D-FLIPFLOP | | | ADDER | |
|---|---|---|---|---|---|
| | Setup time | Hold time | Clock to Q delay | Worst case delay | $(F_{clk})_{max}$ |
| **PRE-LAYOUT** | 0.217 | 0 | 0.94 | 1.01 | 0.2 |
| **POST-LAYOUT** | 0.34 | 0 | 0.99 | 1.37 | 0.12 |

**time period is in ns and clock frequency is in Ghz*

According to the formula we get maximum clock frequencies as 0.46 Ghz ang 0.37 Ghz for pre and post layout respectively.