VLSI DESIGN MONSOON PROJECT:-

THE DESIGN OF A 4 BIT ALU(ARITHMETIC LOGICAL UNIT)

(performing addition,subtraction,comparsion and logical and of two 4 bit numbers)

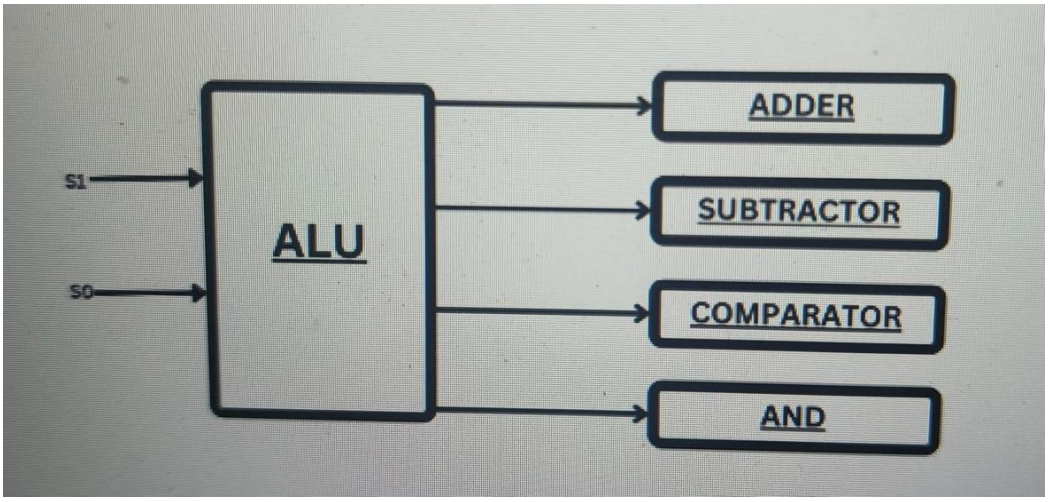
NAME: MANAS SACHIN DESHMUKH

ROLL NO: 2022102040

DATE: 27TH NOVEMBER 2023

1.1) INTRODUCTION:-

The project mainly deals with the design, layout and testing of a 4 bit ALU(Arithmetic logical unit) which performs four operations which are addition, subtraction, comparison and logical and of two 4 bit numbers. The layout of the above mentioned circuit is made in MAGIC. The testing and delay analysis is done with the help of NGSPICE. The functioning of the circuit is checked by VERILOG also. So in all MAGIC is for layout , NGSPICE for circuit design and VERILOG for checking the functionality of the circuit.

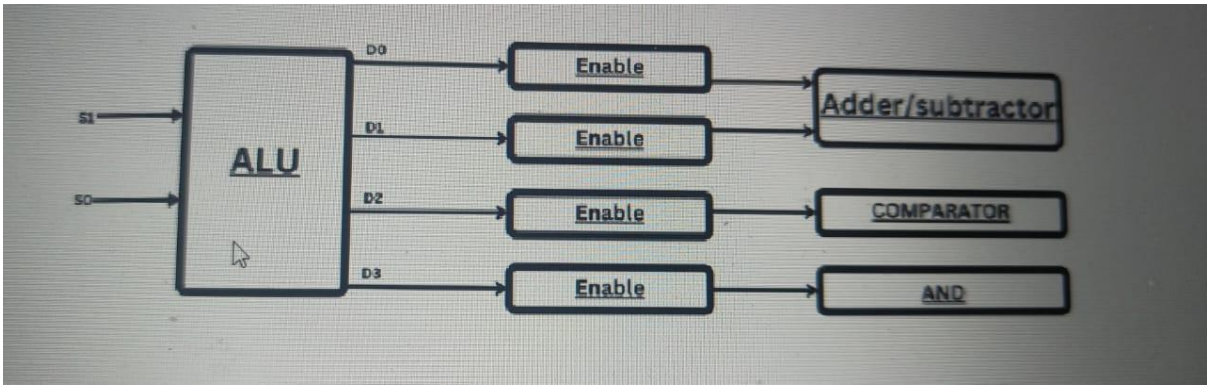


1.2) BASIC METHOD/APPROACH:-

For writing the netlist there is extensive use of subckt(SUBCIRCUITS) IN NGSPICE , this is so as to abstract all the individual blocks of the circuit, this extensively increases the readability and robustness of the final 4 bit ALU.

Firstly subcircuits are made for all logical gates like AND,OR,NAND,NOT,XOR,NOR AND XNOR. These subcircuits are made in magic(LAYOUT) and tested using NGSPICE. Separate netlists for gates are also made in NGSPICE.

THE MAIN IDEA OF DESIGN OF 4 BIT ALU:-

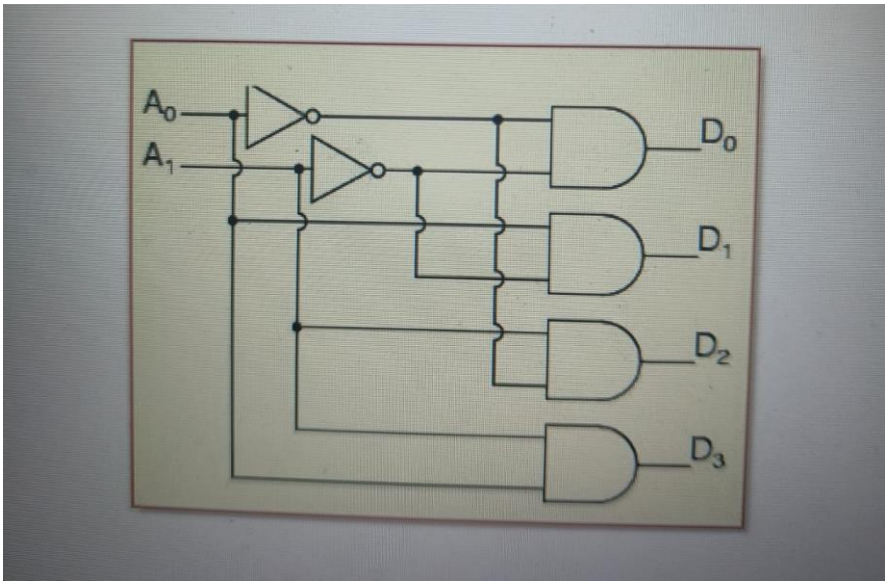


First the two select lines are passed to the ALU then for all 4 different Boolean combinations of the select lines(DECODER) one out of the four operations is selected then all 8 inputs(4 for bit1 and 4 for bit2) are passed to the enable block and the desired operation is performed and the rest three operation(s) output remains zero thus this can be similarly done for all four operations.

S1	S0	operation
0	0	Add
0	1	Subtract
1	0	Compare
1	1	And

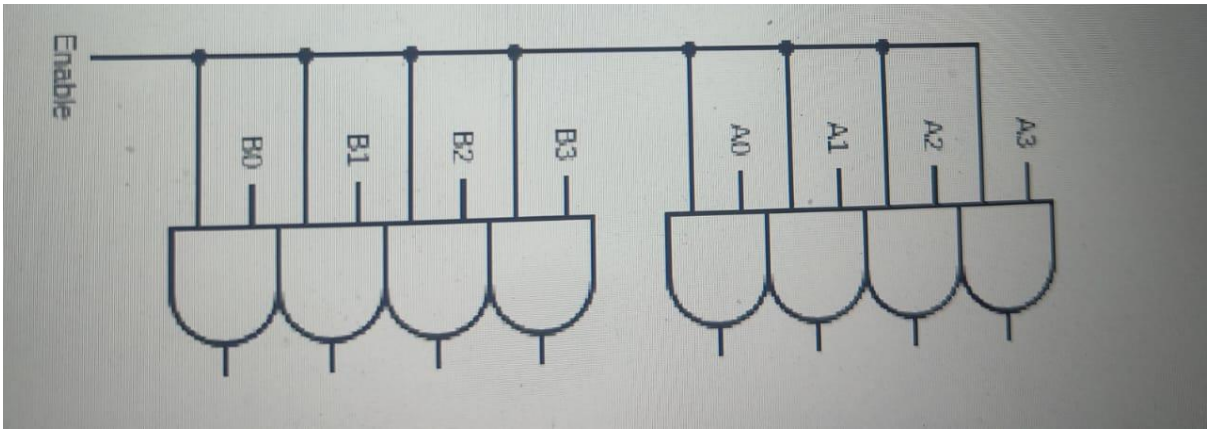
DECODER:-

DECODER CIRCUIT:-



If A0 and A1 are the select lines then we perform A0.A1, A0.NOT(A1), NOT(A0).A1, NOT(A0).NOT(A1) then we connect the outputs of the above decoder circuit (D0,D1,D2,D3) to the enable block to fulfil the functionality of the 4 bit ALU.

THE ENABLE BLOCK:-

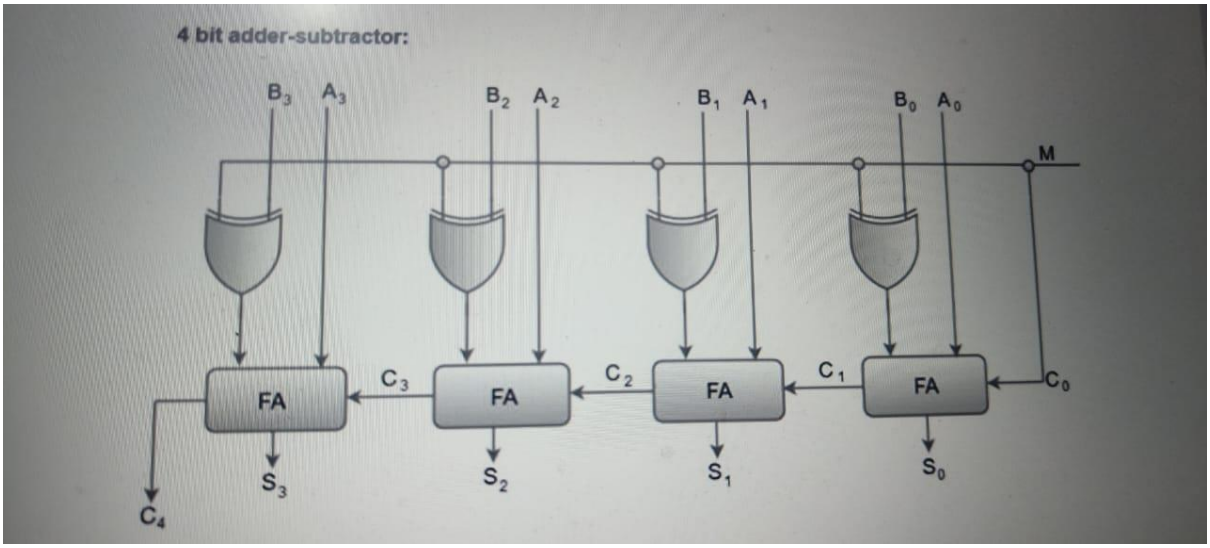


In the enable block the output of one of the Boolean operations of the two select lines is connected to the common line to all the 8 AND gates (B3B2B1B0 AND A3A2A1A0 are the two 4 bit numbers as input). We know that any logic value and with logic one is always the logic value itself and any logic value and with

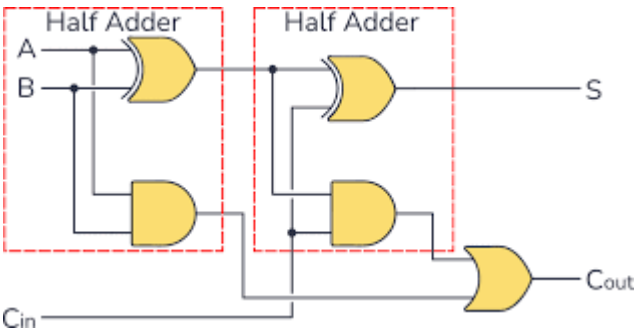
logic zero is always zero. Based upon this only the operation selected will be functional rest paths will not be functional.

THE FOUR BLOCKS:-

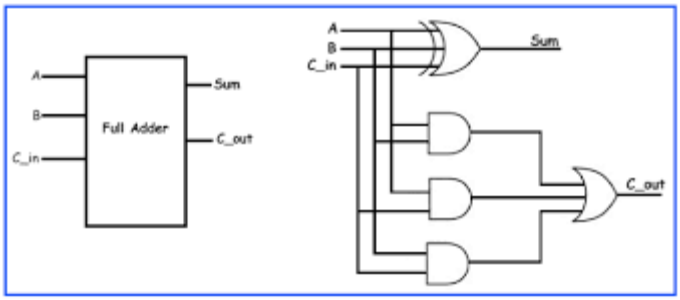
BLOCK1 2) ADDITION AND SUBTRACTION BLOCK(COMBINED CIRCUIT):-



This is the exact circuit implemented in magic and ngspice for the adder and the subtractor block. This circuit is basically a cascade of 4 full adders with output carry propagating through the circuit . The gates used above are XOR gates so when input carry(C0 in the diagram) is 0 this circuit functions as adder and when C0 is logic high(1) this functions as a subtractor.



FULL ADDER CIRUCIT:-



SUM =A XOR B XOR Cin

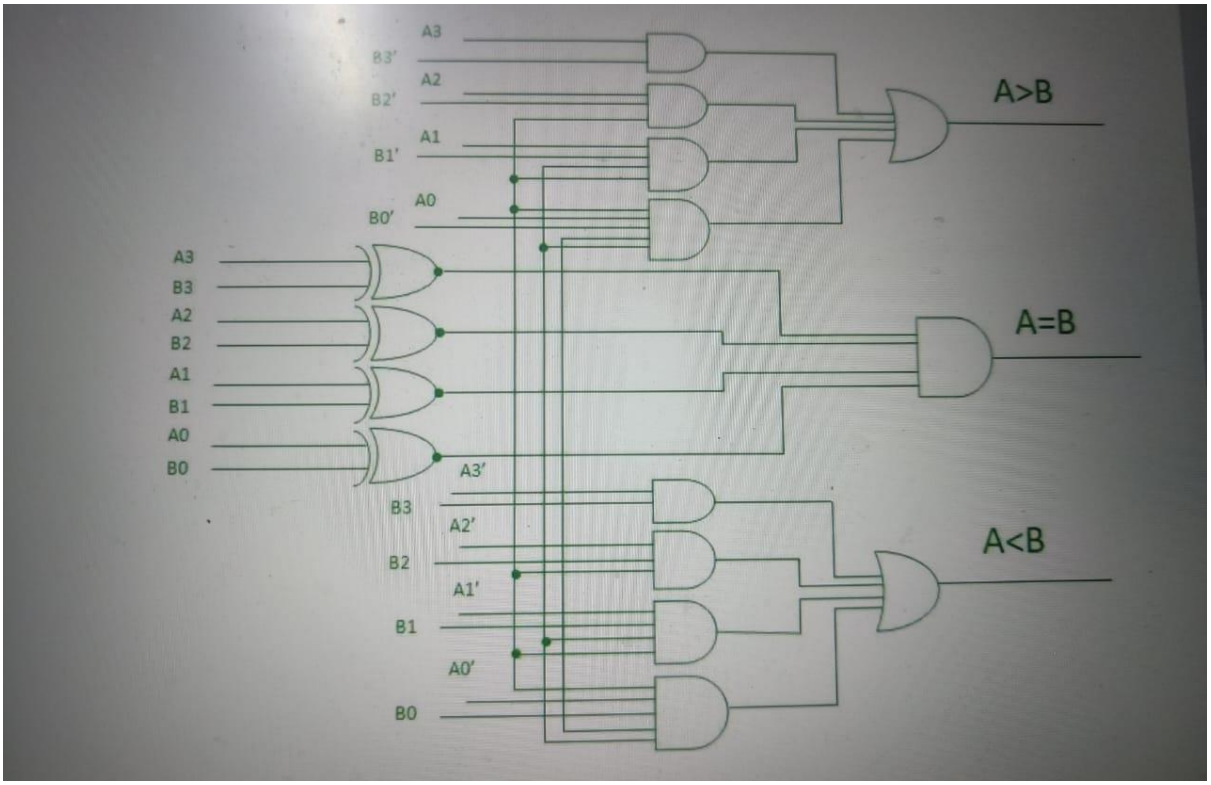
CARRY=A.B +B.Cin +Cin.A

<u>PROPERTY USED IN CIRCUIT DESIGN:-</u>

1) **Any logic value XOR with 0 is the logic value itself**
2) **Any logic value XOR with 1 is the complement or NOT of that logic value.**

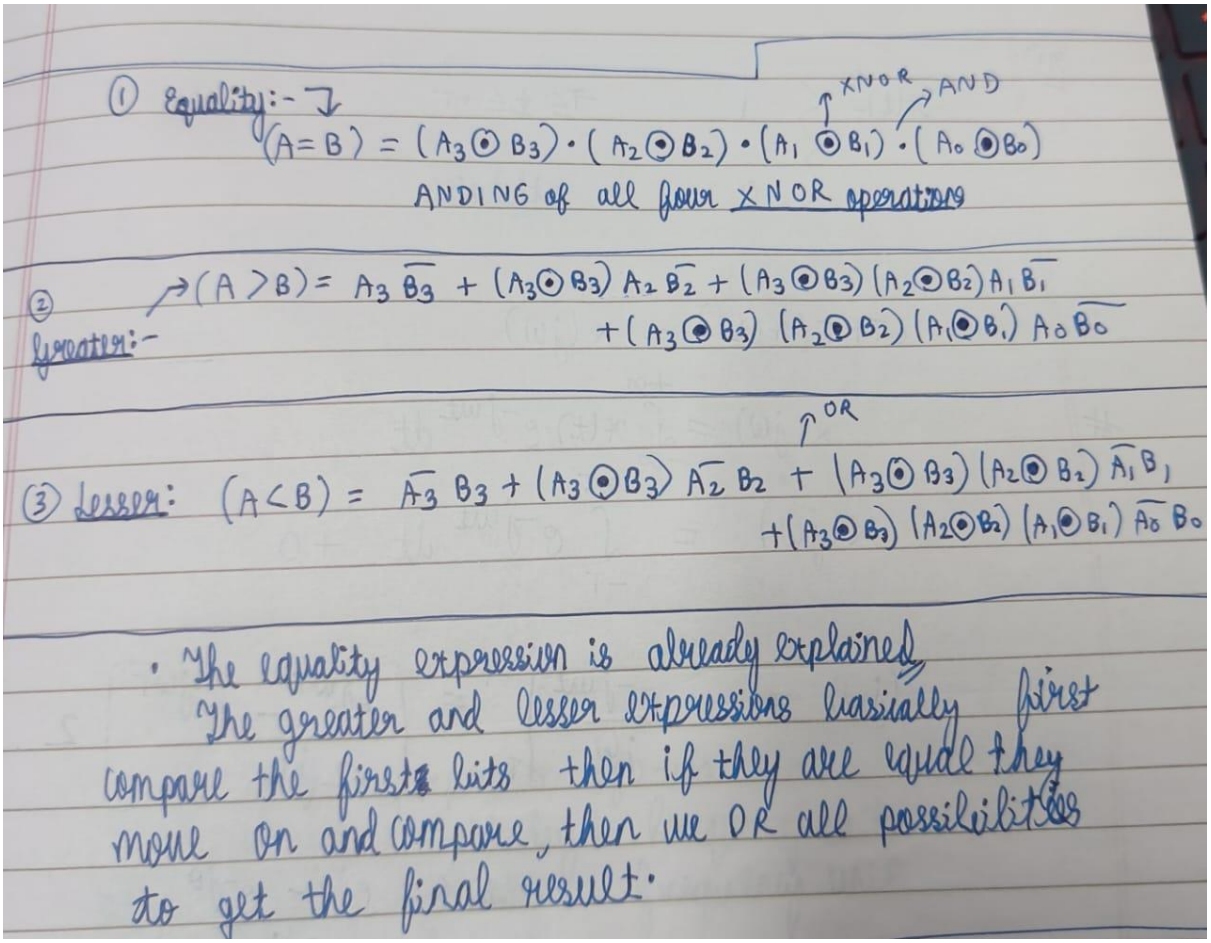
   So this ensures when C0=0 A0 AND B0 are passed to the first full adder(from right) (FOR ADDER) and when C0=1 then A0 NOT(B0) is passed to the full adder to fulfil the functionality of subtractor and give the correct final result.

   2)<u>COMPARATOR BLOCK:-</u>



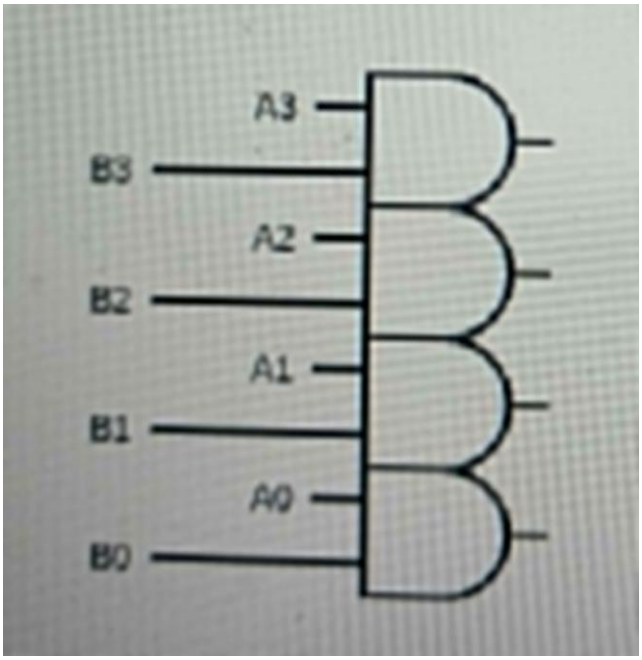
THE ABOVE IS THE CIRCUIT DIAGRAM FOR THE 4 BIT COMPARATOR:-

The equality functionality is directly obtained by xnor bit by bit (of the same position) of the two 4 bit numbers
The Boolean expression for the greater and lesser functionality are as follows:-



3) **THE AND BLOCK:-**

THE CIRCUIT DIAGRAM DISPLAYED BELOW:-



BASICALLY WE ARE JUST ANDING BIT BY BIT (A3 AND B3,A2 AND B2,A1 AND B1,A0 AND B0)

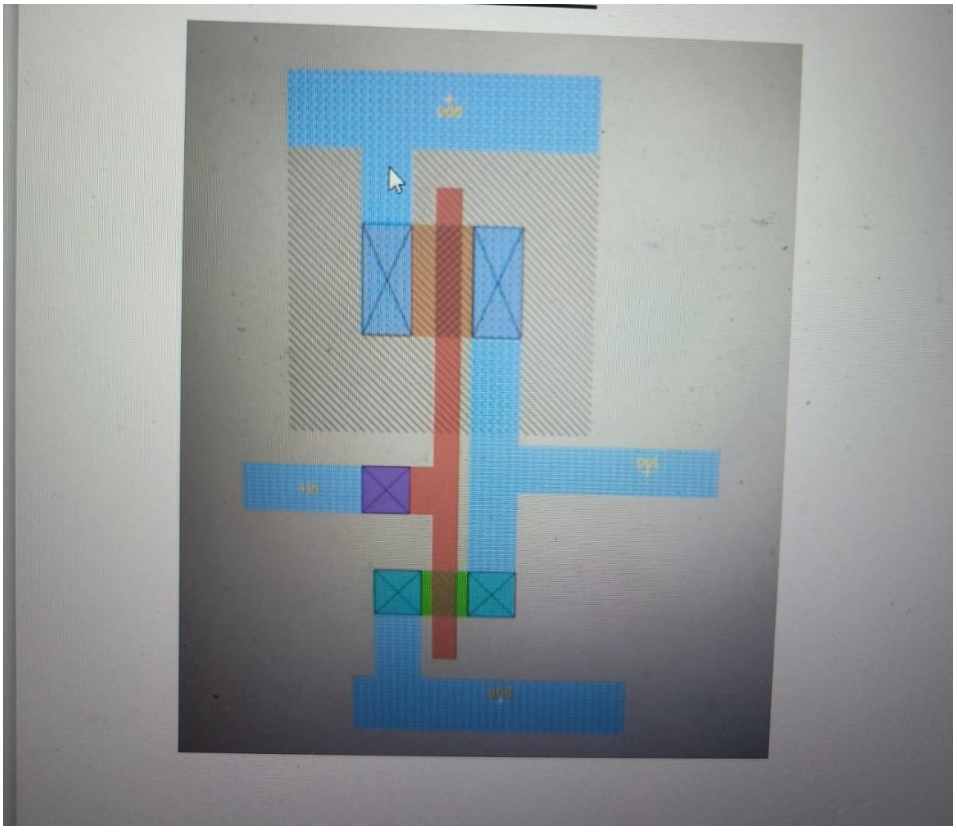
NOW WE CAN COMBINE THE DECODER CIRCUIT ENABLE BLOCK AND THE 4 BLOCKS(FOR ADDITION , SUBTRACTION , COMPARISON AND LOGICAL AND) TO OBTAIN OUR DESIRED ALU.

1.3) LAYOUT DETAILS IN MAGIC:-

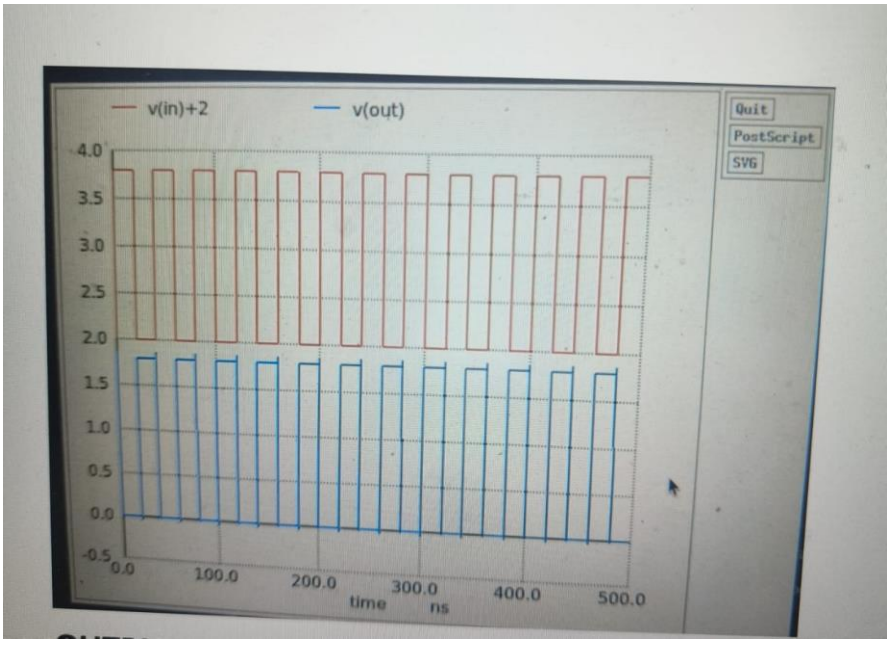
The blocks were made by various gates already implemented in magic like AND,OR,NOT,XOR,XNOR,NAND,OR.

Here are some of the layout used in the circuit design:-

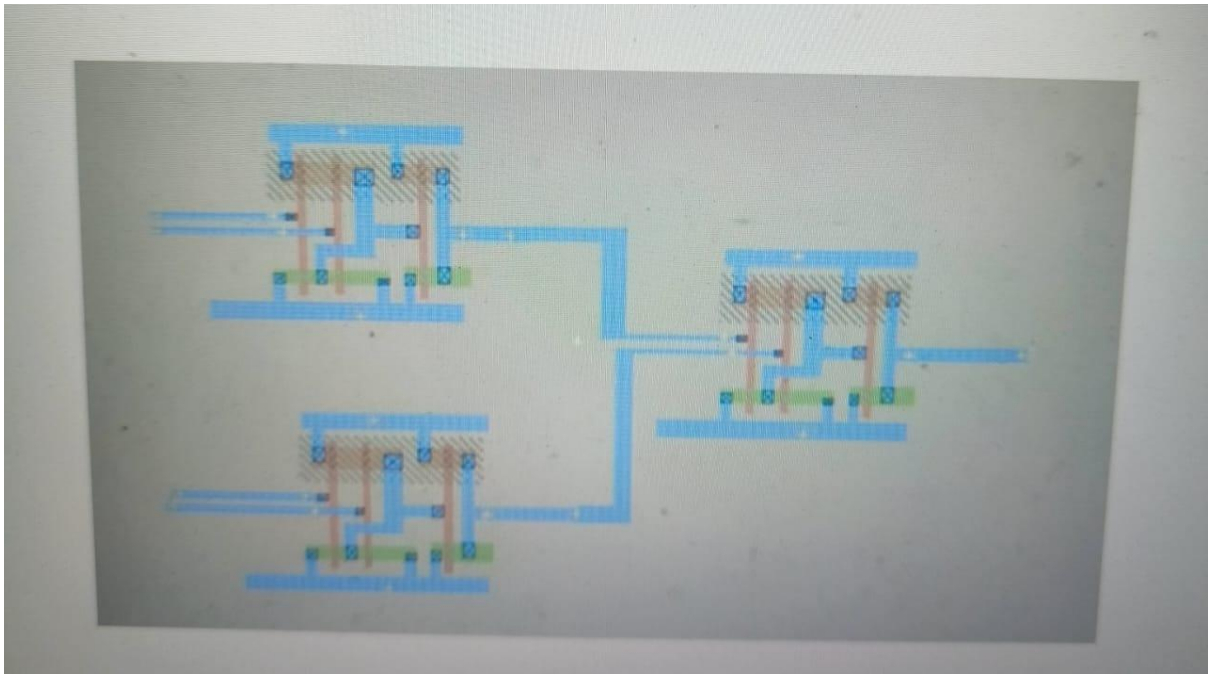
NOT GATE:-



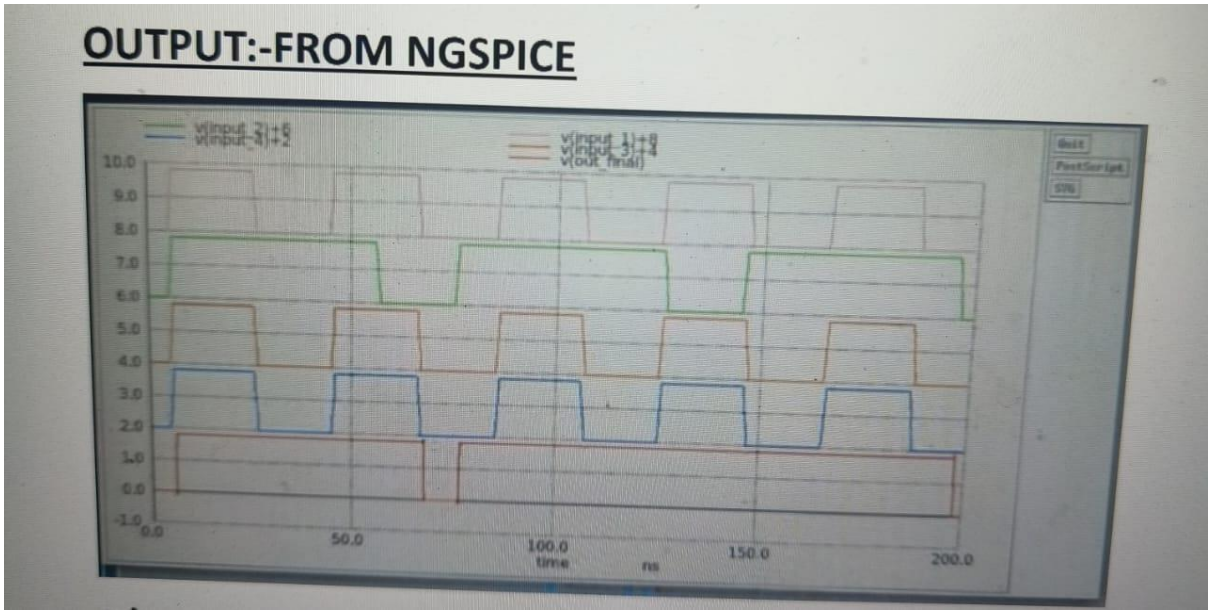
OUTPUT FROM NGSPICE:-



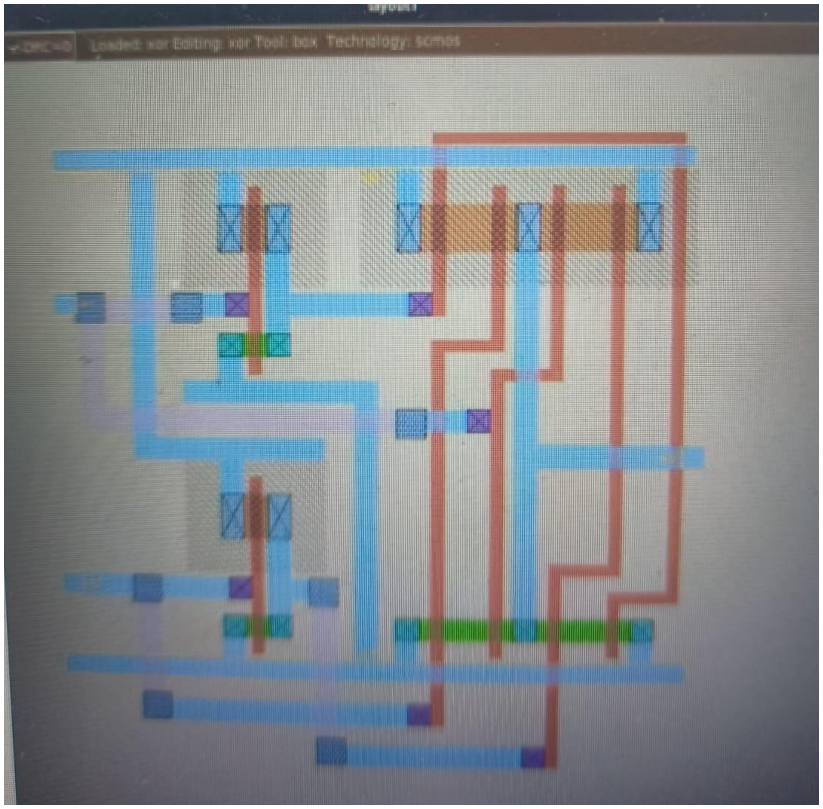
4 INPUT AND GATE:-

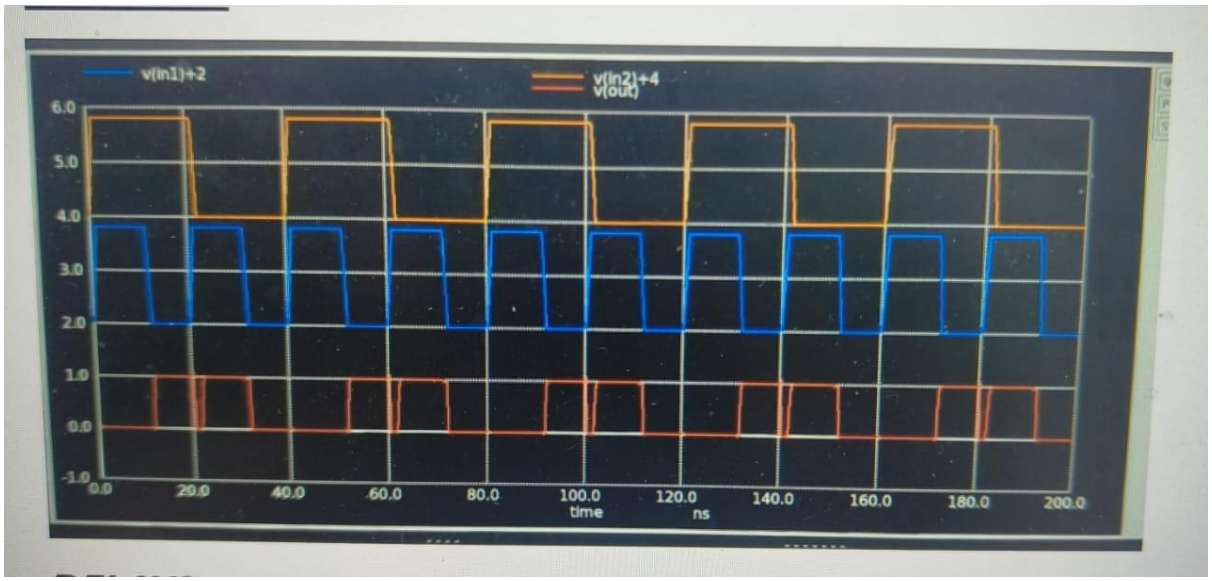


OUTPUT FROM NGSPICE:-



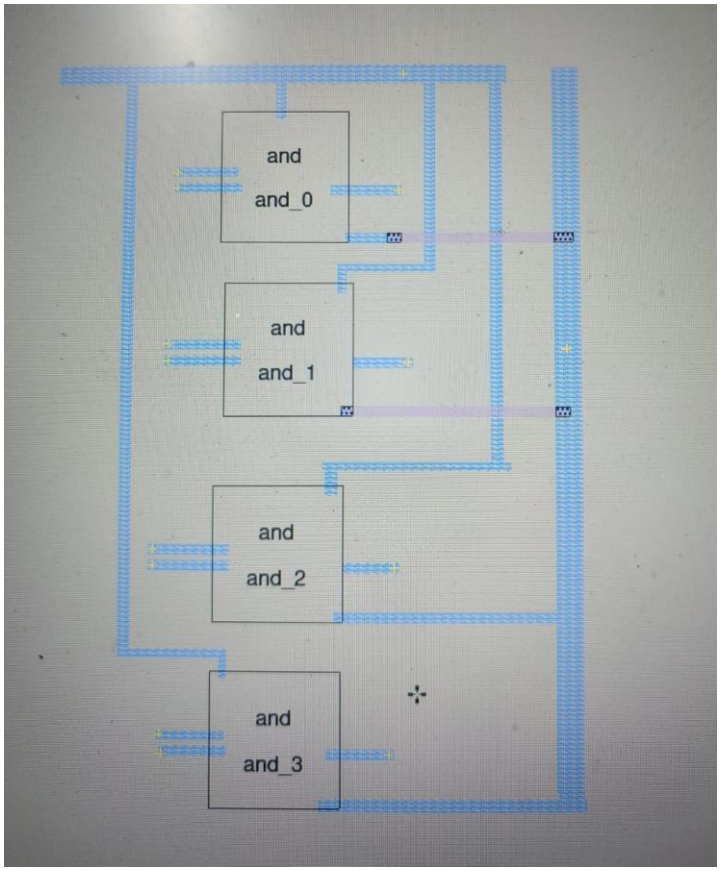
XOR GATE:-

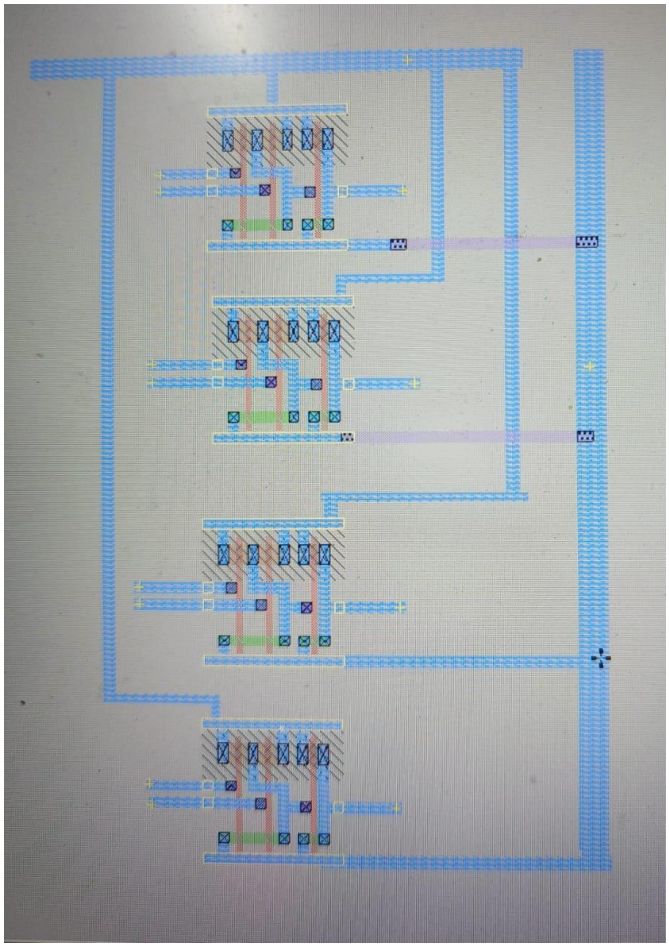




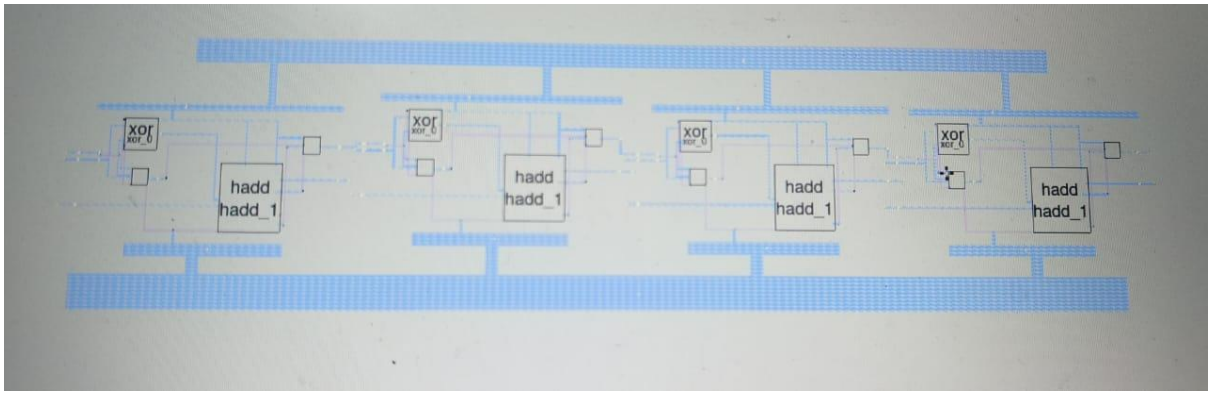
COMPONENTS OF THE ALU:-

1)ANDE BOCK:-

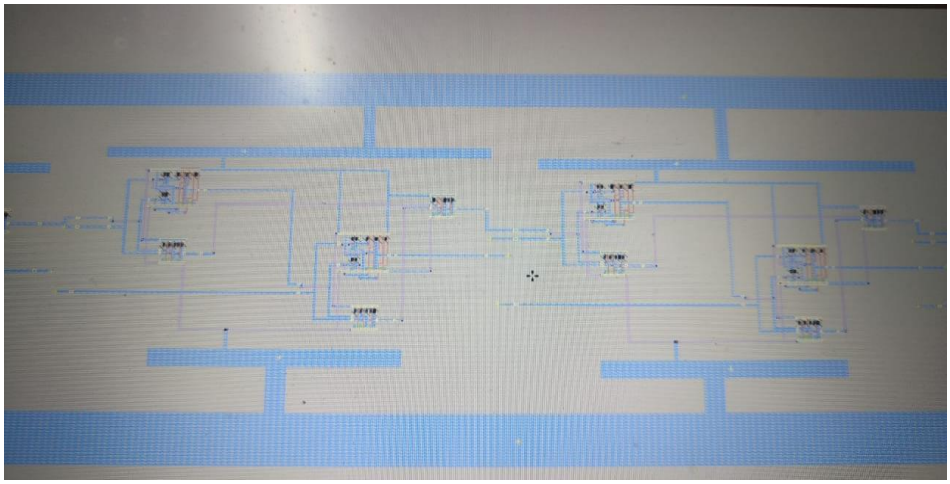

and
and_0
and
and_1
and
and_2
and
and_3



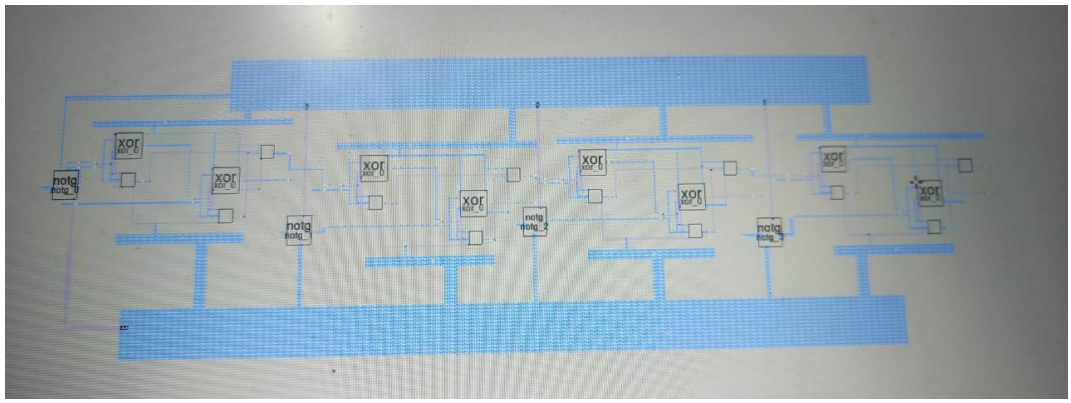
THE ADDER BLOCK:-



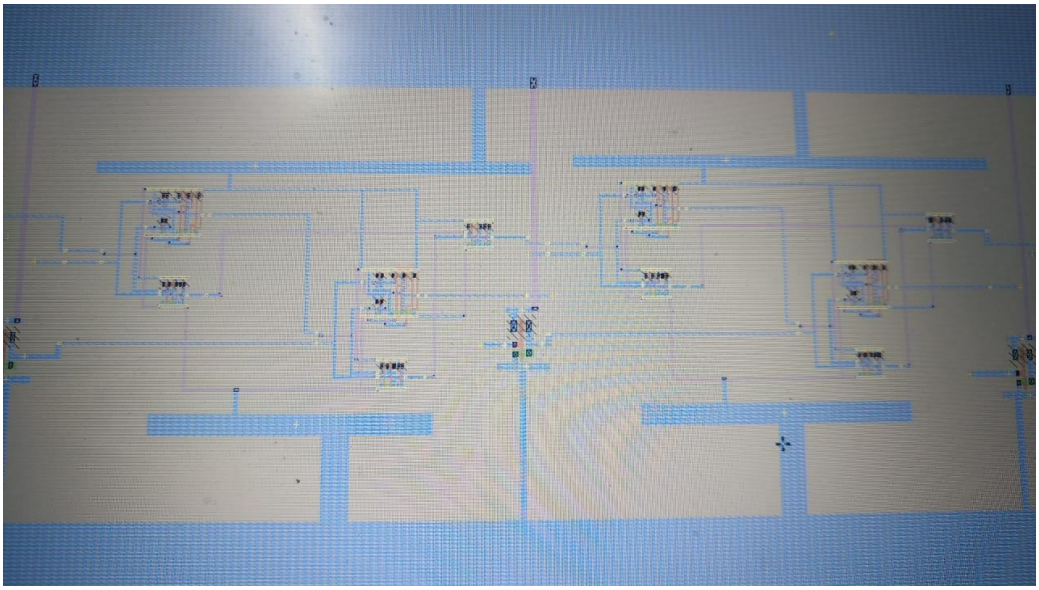
(ZOOMED IN VIEW)



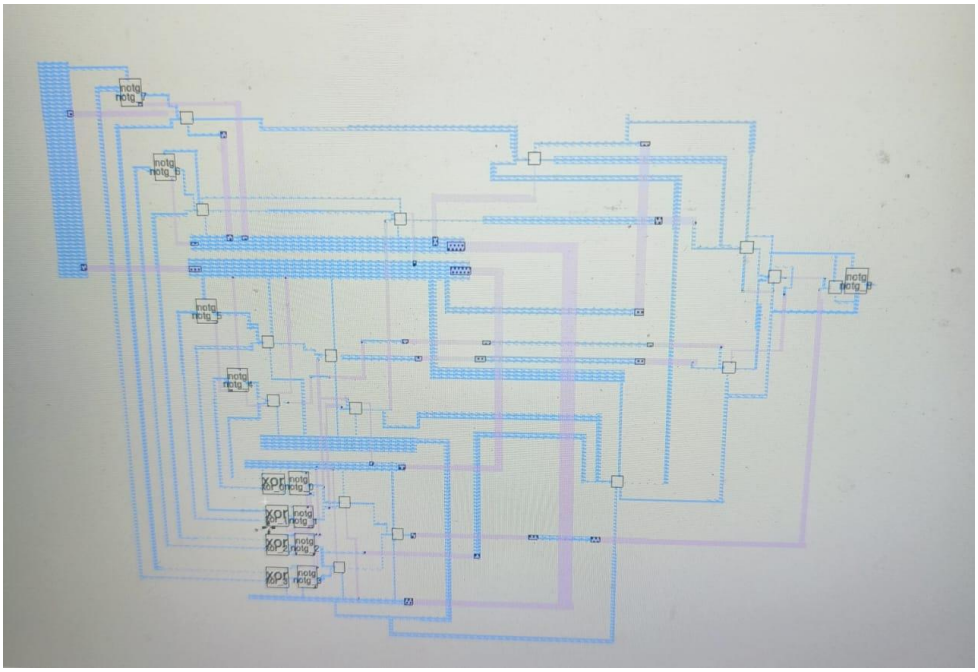
THE SUBTRACTOR BLOCK:-



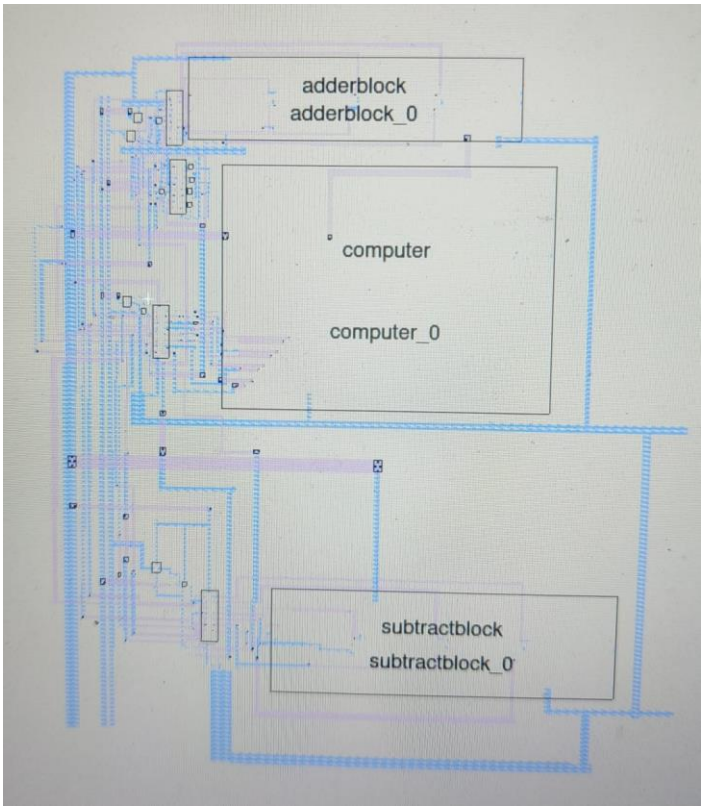
(ZOOMED IN VIEW)

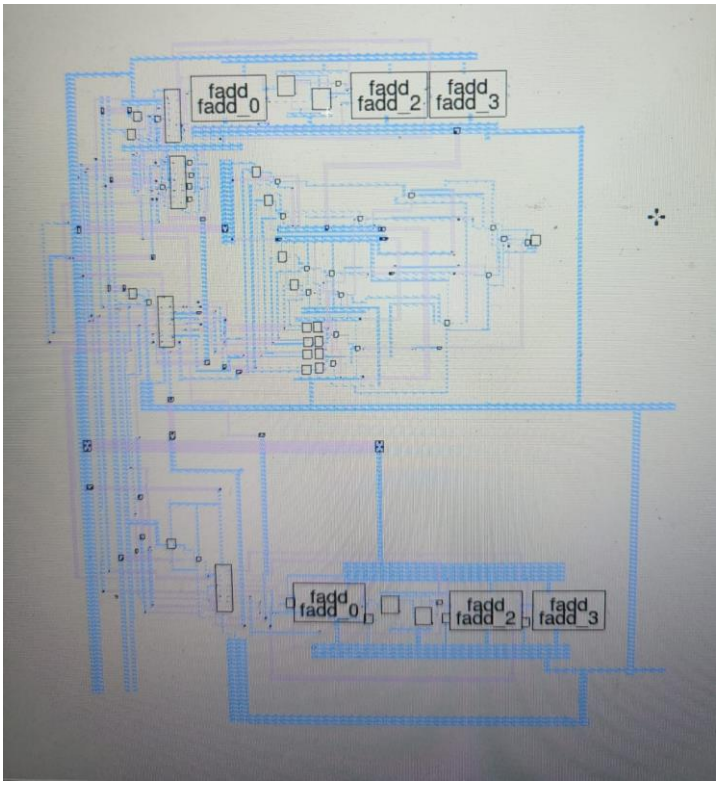


THE COMPARATOR BLOCK FOR COMPARISON(LESSER/GREATER/EQUAL) OF 2 4 BIT NUMBERS:-

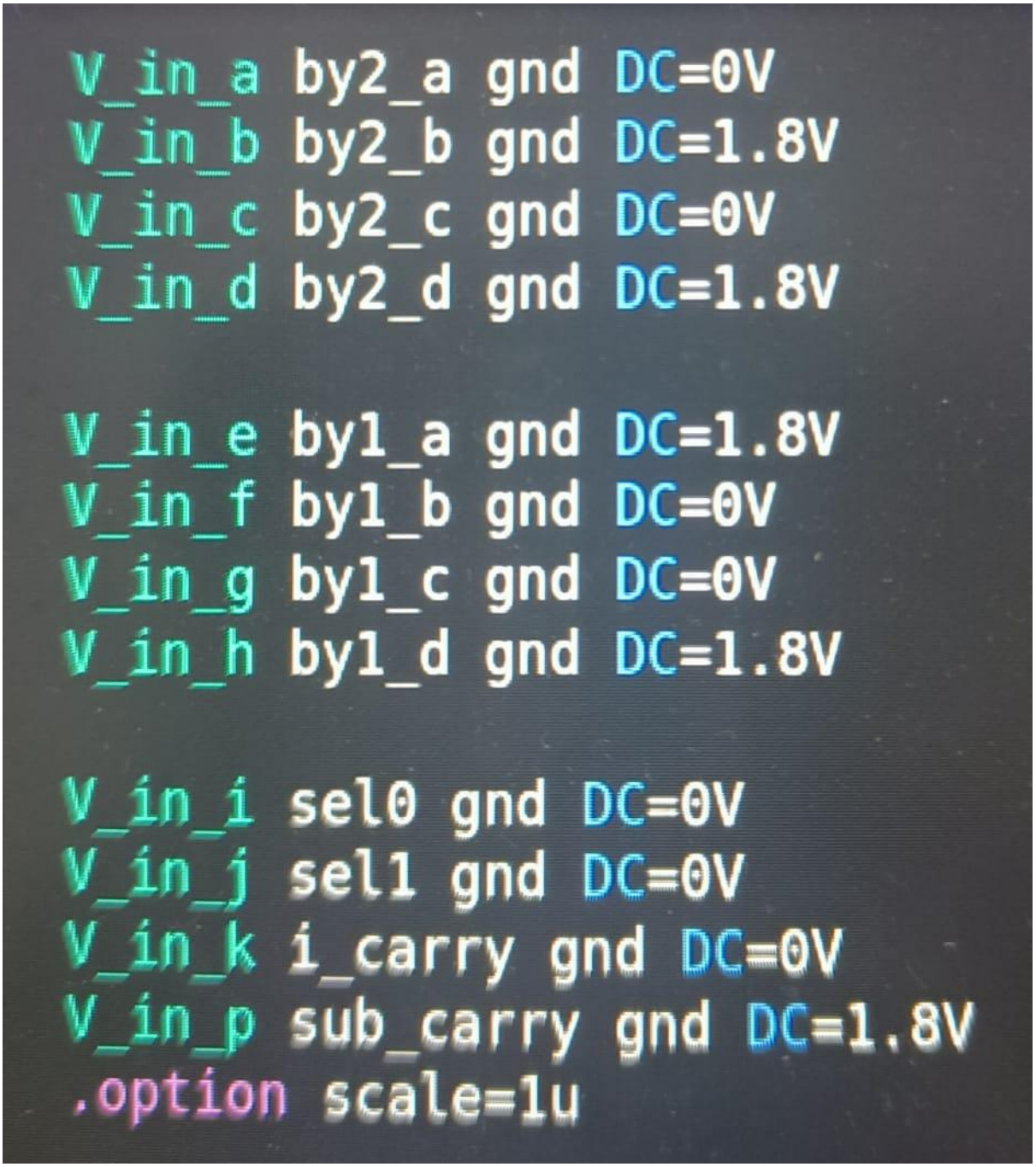


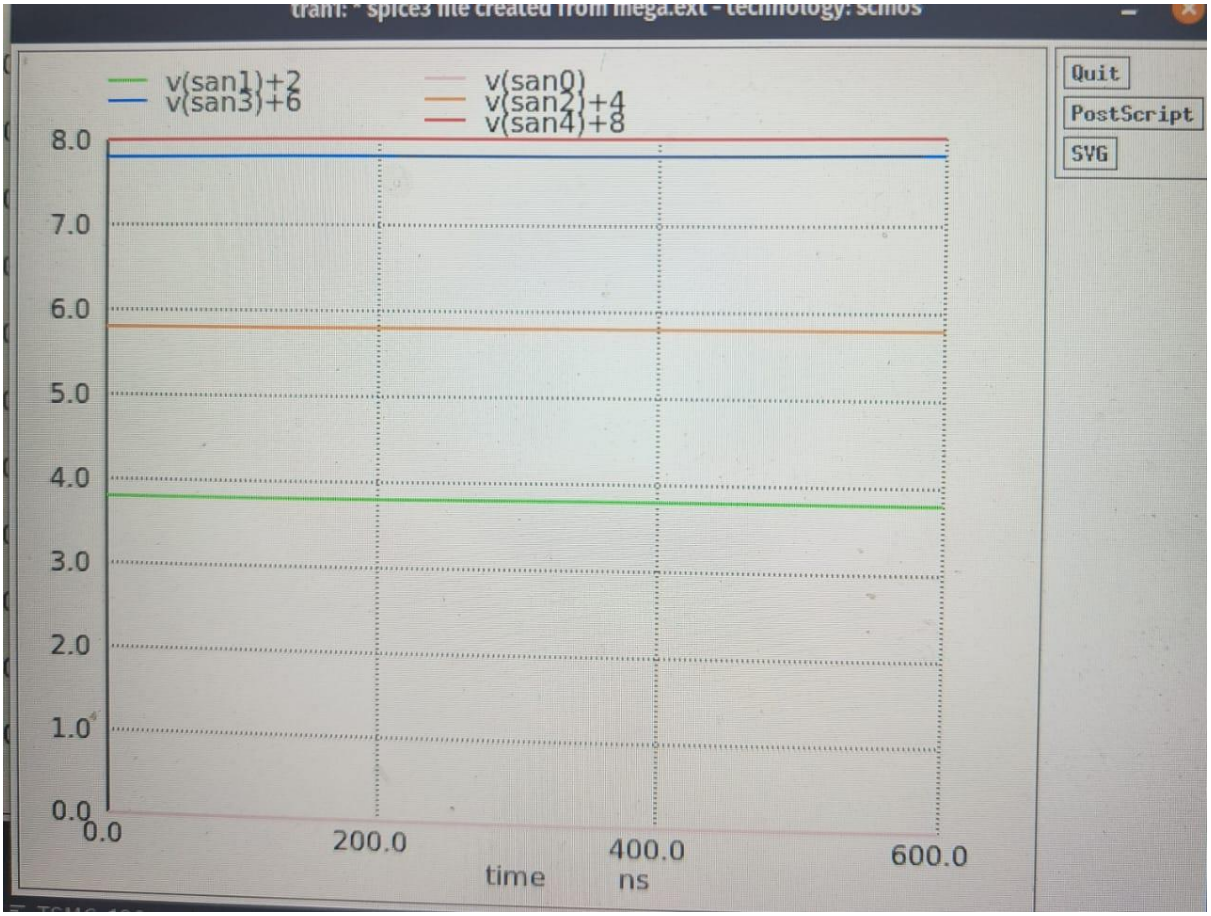
THE FINAL ALU DESIGN CIRCUIT:-



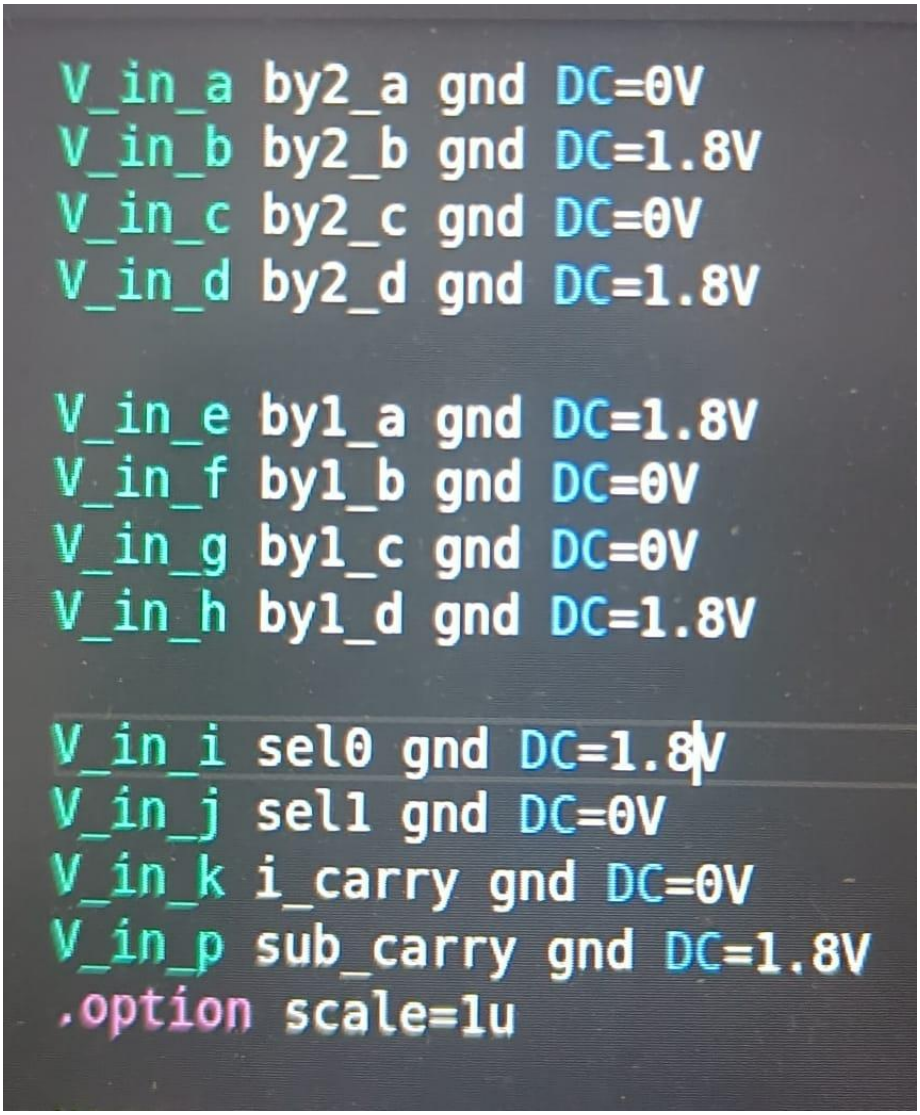


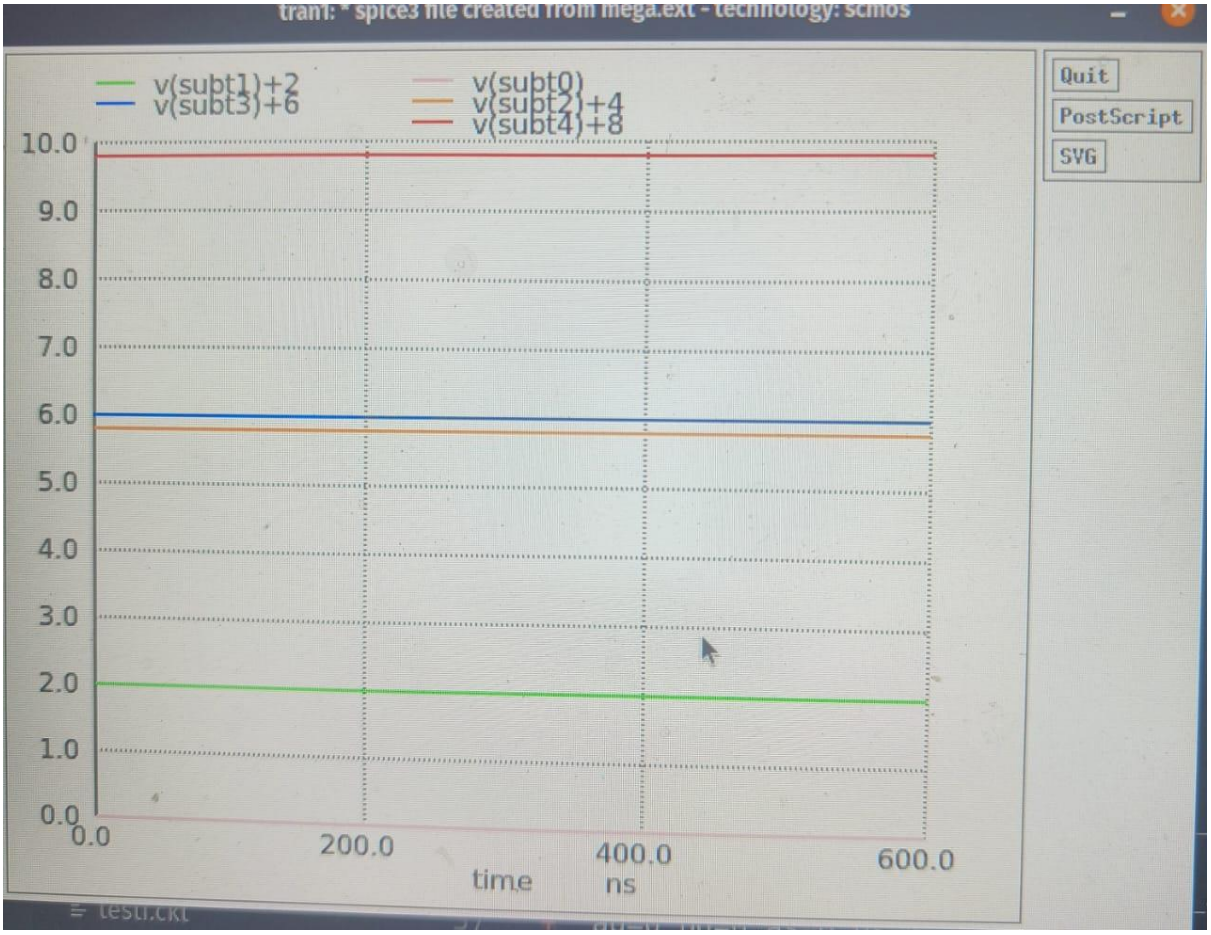
THE TESTING OF THE MAIN CIRCUIT USING NGSPICE:-



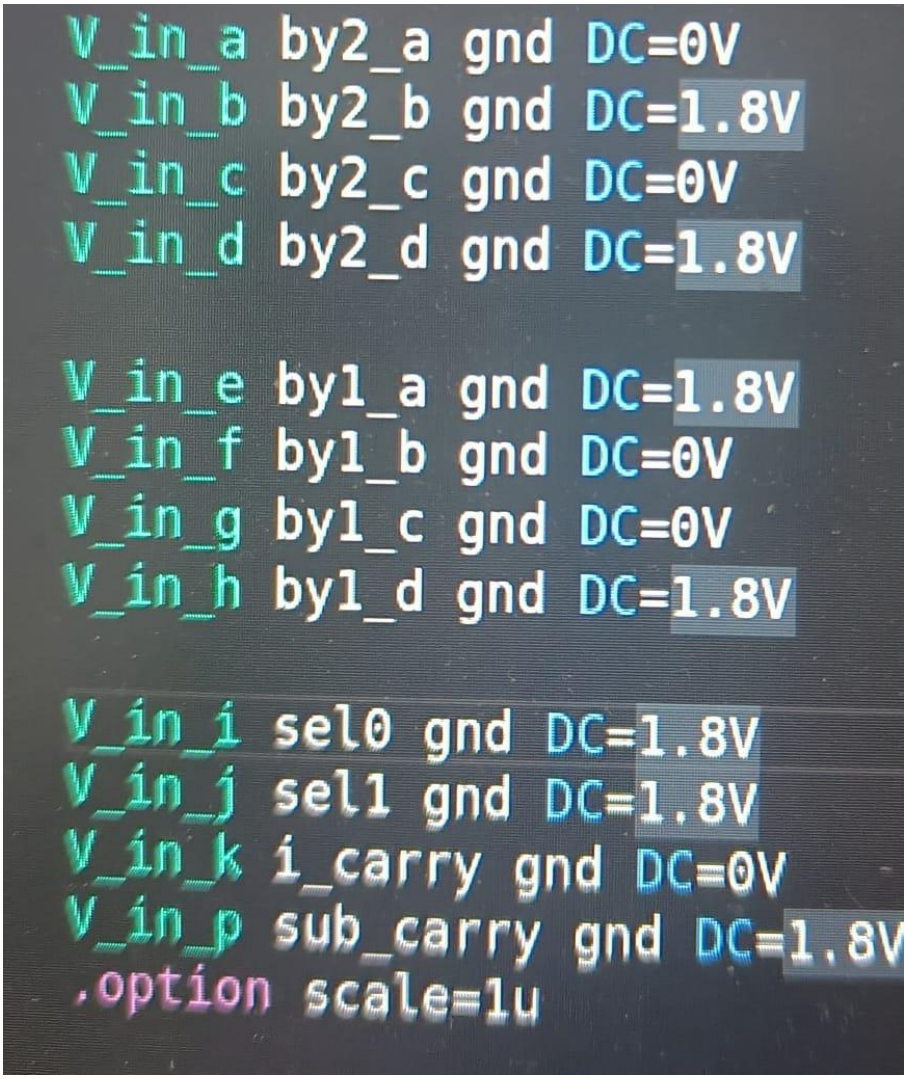


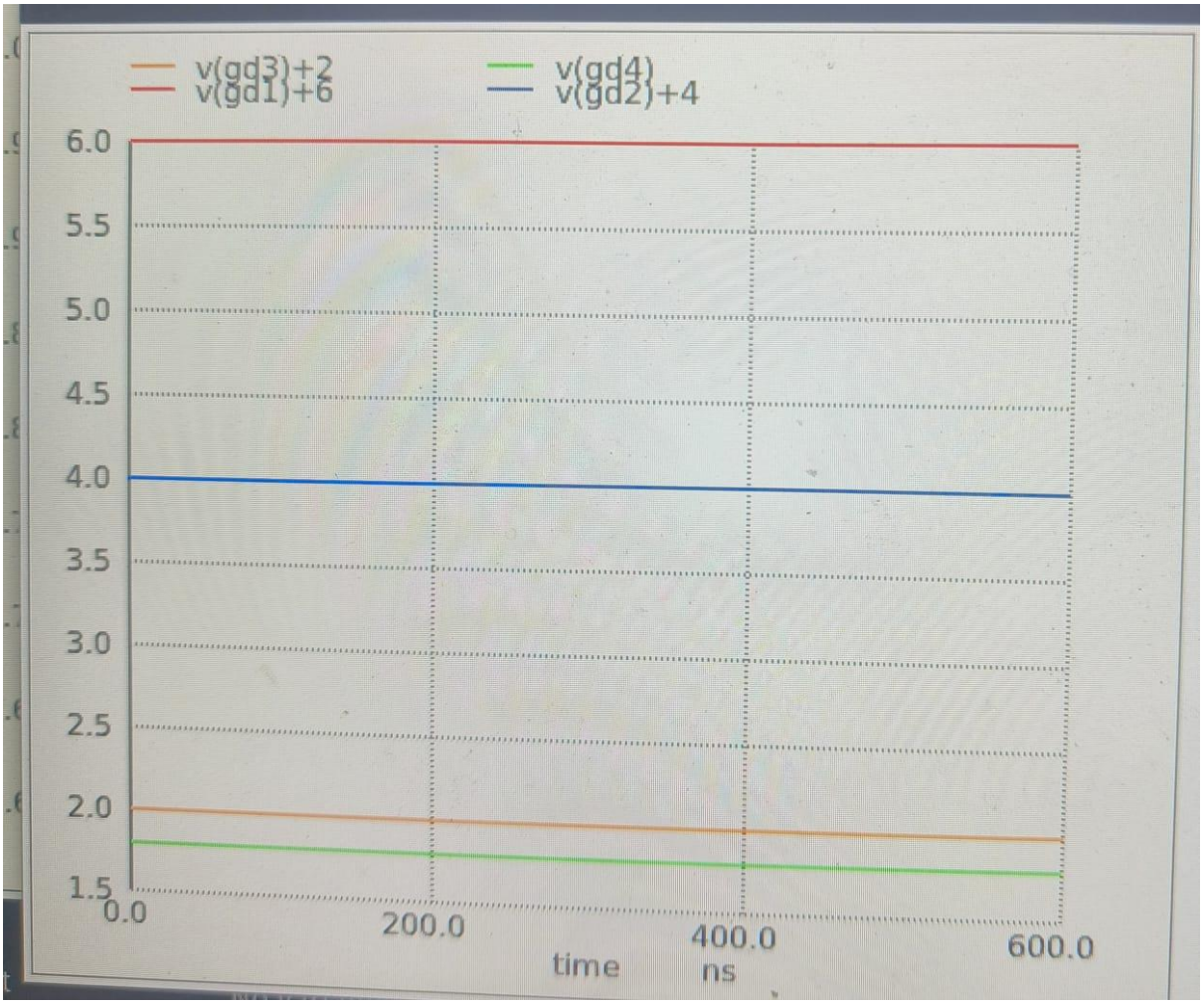
ADDITION RESULT


V_in_a by2_a gnd DC=0V
V_in_b by2_b gnd DC=1.8V
V_in_c by2_c gnd DC=0V
V_in_d by2_d gnd DC=1.8V
V_in_e by1_a gnd DC=1.8V
V_in_f by1_b gnd DC=0V
V_in_g by1_c gnd DC=0V
V_in_h by1_d gnd DC=1.8V
V_in_i sel0 gnd DC=1.8V
V_in_j sel1 gnd DC=0V
V_in_k i_carry gnd DC=0V
V_in_p sub_carry gnd DC=1.8V
.option scale=1u

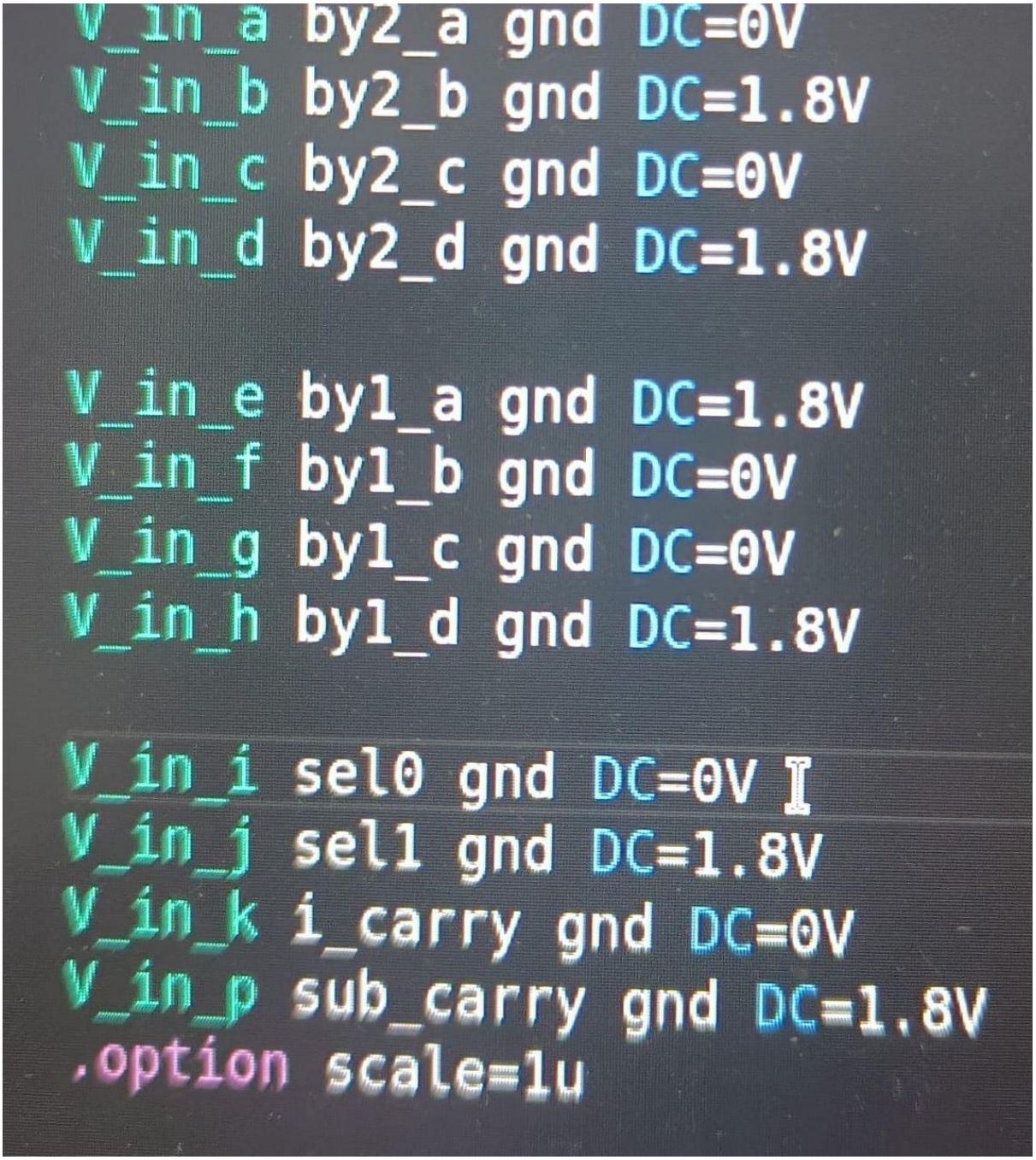


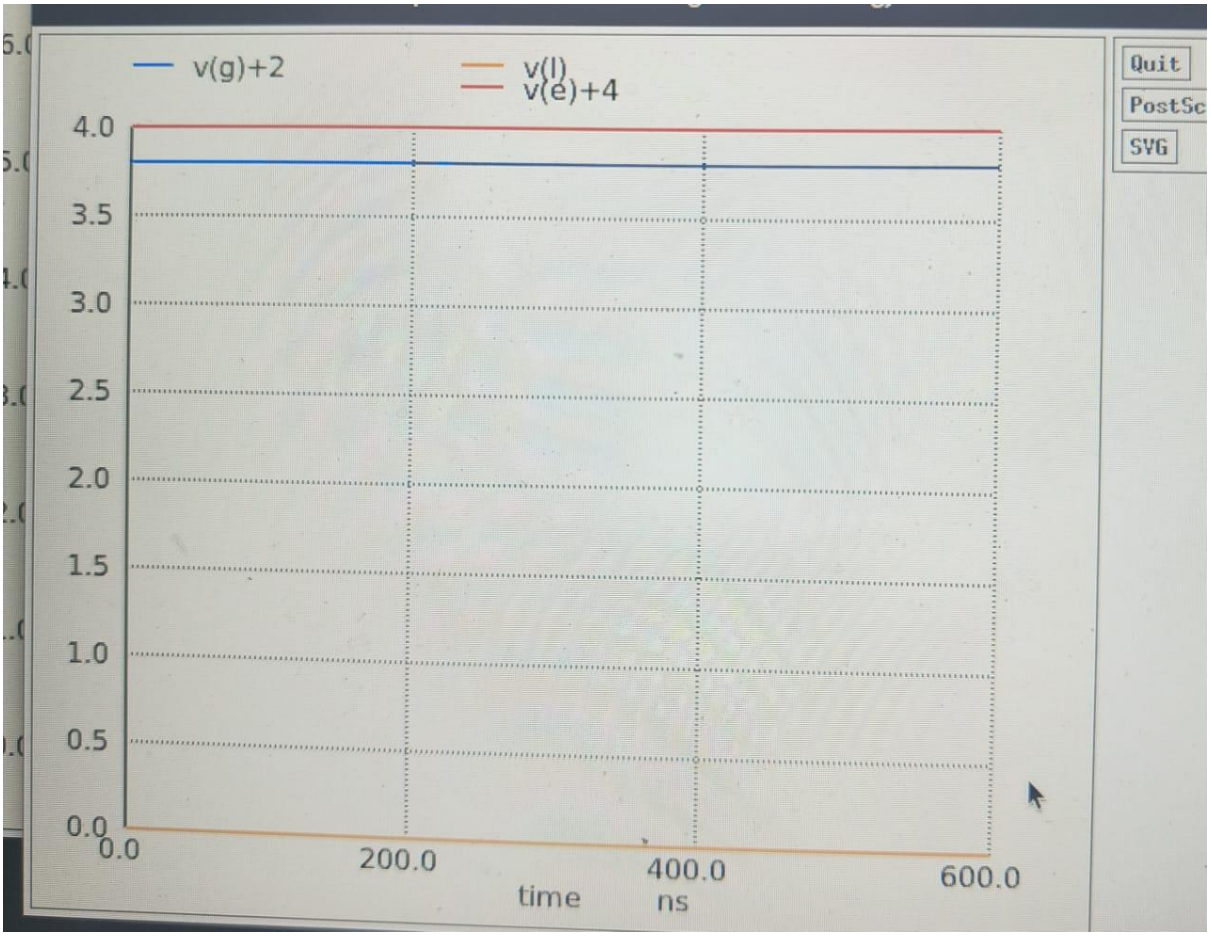
SUBTRACTION RESULT


V_in_a by2_a gnd DC=0V
V_in_b by2_b gnd DC=1.8V
V_in_c by2_c gnd DC=0V
V_in_d by2_d gnd DC=1.8V
V_in_e by1_a gnd DC=1.8V
V_in_f by1_b gnd DC=0V
V_in_g by1_c gnd DC=0V
V_in_h by1_d gnd DC=1.8V
V_in_i sel0 gnd DC=1.8V
V_in_j sel1 gnd DC=1.8V
V_in_k i_carry gnd DC=0V
V_in_p sub_carry gnd DC=1.8V
.option scale=1u



RESULT OF THE AND BLOCK


V_in_a by2_a gnd DC=0V
V_in_b by2_b gnd DC=1.8V
V_in_c by2_c gnd DC=0V
V_in_d by2_d gnd DC=1.8V
V_in_e by1_a gnd DC=1.8V
V_in_f by1_b gnd DC=0V
V_in_g by1_c gnd DC=0V
V_in_h by1_d gnd DC=1.8V
V_in_i sel0 gnd DC=0V
V_in_j sel1 gnd DC=1.8V
V_in_k i_carry gnd DC=0V
V_in_p sub_carry gnd DC=1.8V
.option scale=1u



RESULT OF THE COMPARATOR BLOCK

NGSPICE AND VERILOG:-

NGSPICE CODE FOR ALU:-

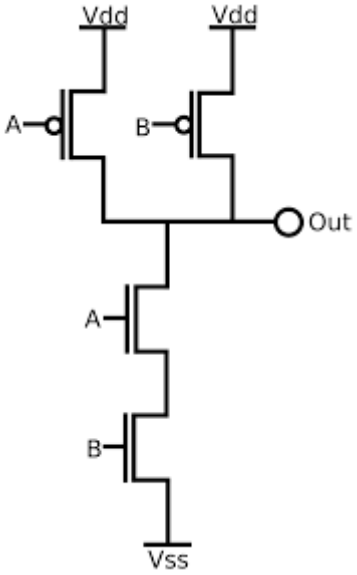
```
.include RING.sub
.include TSMC_180nm.txt
.include NAND.sub
.include enable.sub
.include adder.sub
.include subtractor.sub
.include comparator.sub
.include and.sub
.include make_XOR.sub
.include make_OR.sub
```

```
.include make_AND.sub
.include make_XNOR.sub
.param SUPPLY = 1.8
.param LAMBDA = 0.18u
.param wn1 = {10*LAMBDA}
.param wn2 = {10*LAMBDA}
.param ln1 = {2*LAMBDA}
.param ln2 = {2*LAMBDA}
.param wp1 = wn1
.param wp2 = wn1
.param lp1 = {LAMBDA}
.param lp2 = {LAMBDA}
.global gnd
Vdd vdd gnd 'SUPPLY'
V_in_a_dc bit1_a gnd DC=1.8V
V_in_b_dc bit1_b gnd DC=1.8V
V_in_c_dc bit1_c gnd DC=1.8V
V_in_d_dc bit1_d gnd DC=1.8V
V_in_e_dc bit2_a gnd DC=1.8v
V_in_f_dc bit2_b gnd DC=0V
V_in_g_dc bit2_c gnd DC=0V
V_in_h_dc bit2_d gnd DC=1.8V
V_in_i_dc select0 gnd DC 1.8V
V_in_j_dc select1 gnd DC 1.8V
X1 select0 sc0 vdd gnd RING
X2 select1 sc1 vdd gnd RING
X3 sc0 sc1 d0 vdd gnd make_AND
X4 sc1 select0 d1 vdd gnd make_AND
X5 select1 sc0 d2 vdd gnd make_AND
X6 select1 select0 d3 vdd gnd make_AND
X7 d0 bit1_a bit1_b bit1_c bit1_d bit2_a bit2_b bit2_c bit2_d
f1_a f1_b f1_c f1_d f2_a f2_b f2_c f2_d vdd gnd enable
```

```
X8 d1 bit1_a bit1_b bit1_c bit1_d bit2_a bit2_b bit2_c bit2_d g1_a g1_b g1_c g1_d g2_a g2_b g2_c g2_d vdd gnd enable
X9 d2 bit1_a bit1_b bit1_c bit1_d bit2_a bit2_b bit2_c bit2_d h1_a h1_b h1_c h1_d h2_a h2_b h2_c h2_d vdd gnd enable
X10 d3 bit1_a bit1_b bit1_c bit1_d bit2_a bit2_b bit2_c bit2_d i1_a i1_b i1_c i1_d i2_a i2_b i2_c i2_d vdd gnd enable
X11 f1_a f1_b f1_c f1_d f2_a f2_b f2_c f2_d mk1 mk2 mk3 mk4 mk5 vdd gnd adder
X12 g1_a g1_b g1_c g1_d g2_a g2_b g2_c g2_d mt1 mt2 mt3 mt4 mt5 vdd gnd subtractor
X13 h1_a h1_b h1_c h1_d h2_a h2_b h2_c h2_d equal greater lesser vdd gnd comparator
X14 i1_a i1_b i1_c i1_d i2_a i2_b i2_c i2_d rm1 rm2 rm3 rm4 vdd gnd and
.tran 1n 500n
.control
run
set color0 = rgb:f/f/e
set color1 = black
plot   v(rm1)+6 v(rm2)+4 v(rm3)+2 v(rm4)
hardcopy image.ps    v(rm1)+6 v(rm2)+4 v(rm3)+2 v(rm4)
plot   v(equal)+4 v(greater)+2 v(lesser)
hardcopy image1.ps   v(equal)+4 v(greater)+2 v(lesser)
plot   v(mt1)+8 v(mt2)+6 v(mt3)+4 v(mt4)+2 v(mt5)
hardcopy image2.ps  v(mt1)+8 v(mt2)+6 v(mt3)+4 v(mt4)+2 v(mt5)
plot   v(mk1)+8 v(mk2)+6 v(mk3)+4 v(mk4)+2 v(mk5)
hardcopy image3.ps  v(mk1)+8 v(mk2)+6 v(mk3)+4 v(mk4)+2 v(mk5)
plot   v(bit1_a)+6 v(bit1_b)+4 v(bit1_c)+2 v(bit1_d)
hardcopy  image4.ps     v(bit1_a)+6  v(bit1_b)+4  v(bit1_c)+2 v(bit1_d)
.end
```

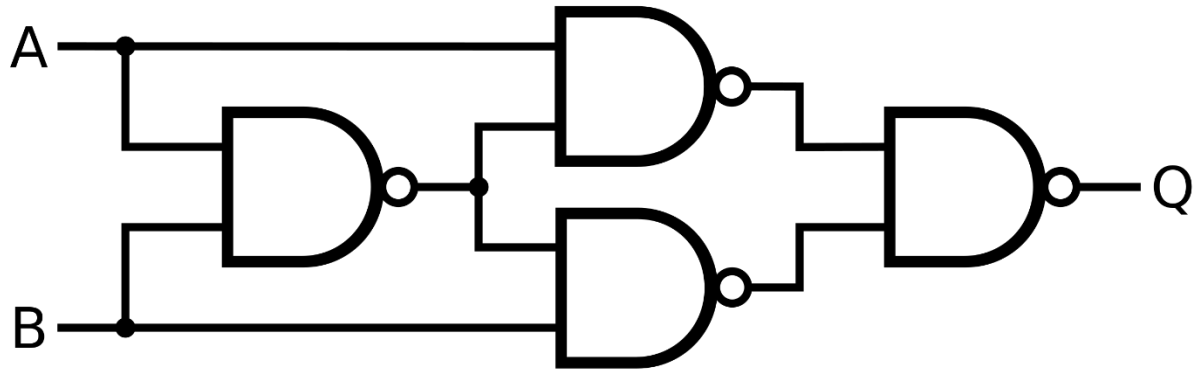
.endc

The code consists of importing lot of gates and the enable block separately made. The decoder block and the enable circuit ensures that the code remains purely structural and not behavioural ensuring correct functionality of the ALU.
I have created all gates with the help of NAND gates since NAND and NOR gates are universal in nature



This is how we create a NAND gate in CMOS technology
By 2 PMOS in parallel and 2 NMOS in serious and the output taken at the common node. ".subckt" feature of NGSPICE can be used to for calling other already implemented functions to ease out the code.

FOR EXAMPLE XOR GATE BY NAND GATE:-

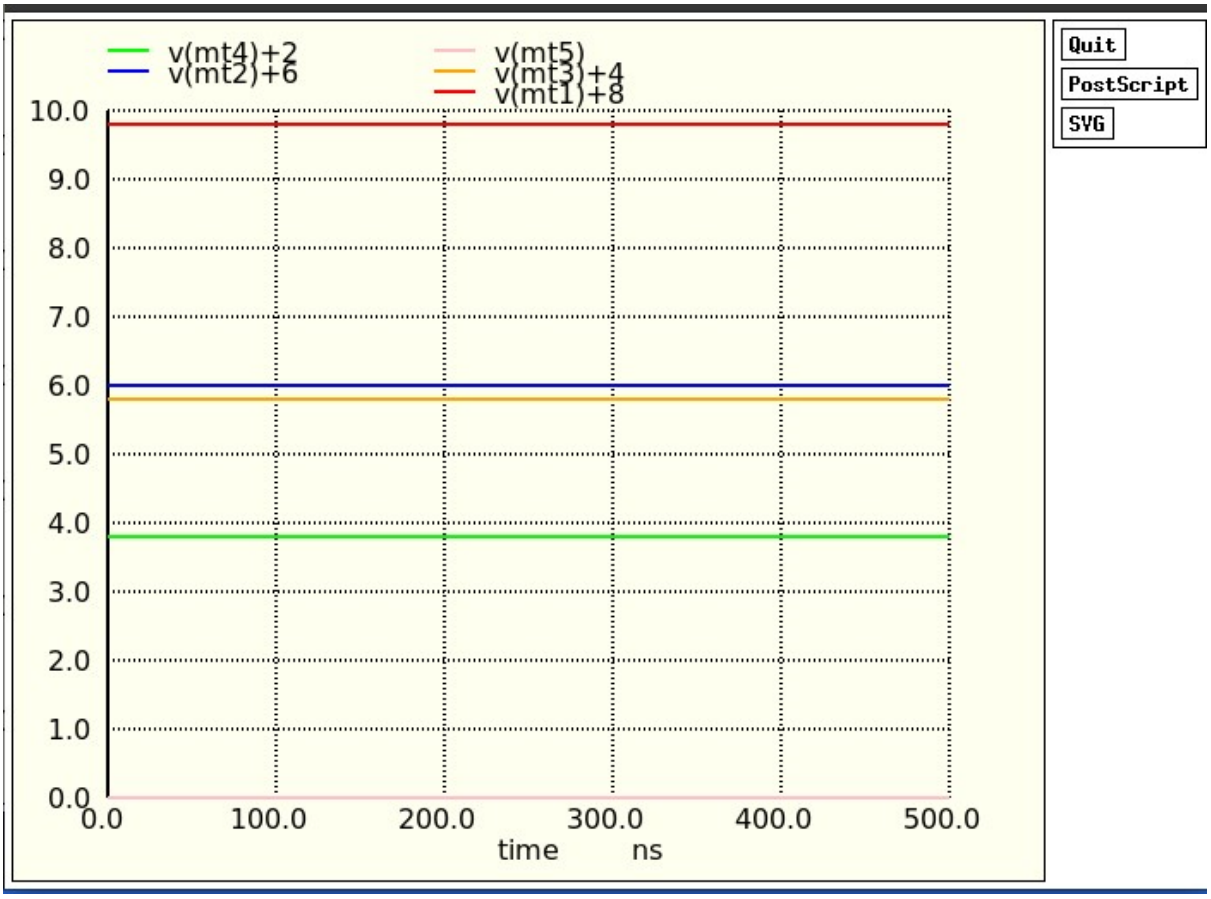


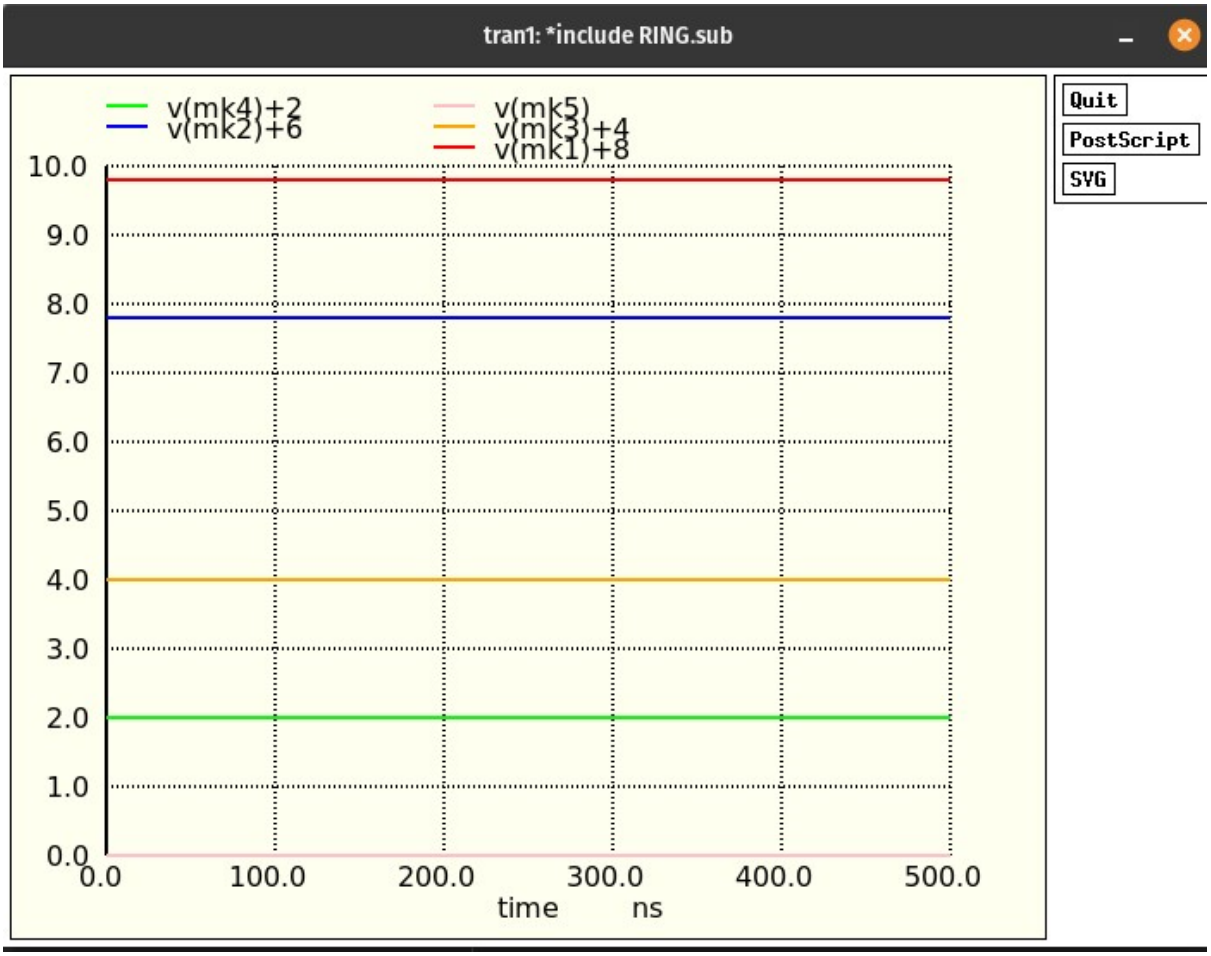
RESULTS OF NGSPICE:-

```
V_in_a_dc bit1_a gnd DC=1.8V
V_in_b_dc bit1_b gnd DC=1.8V
V_in_c_dc bit1_c gnd DC=1.8V
V_in_d_dc bit1_d gnd DC=1.8V
V_in_e_dc bit2_a gnd DC=1.8v
V_in_f_dc bit2_b gnd DC=0V
V_in_g_dc bit2_c gnd DC=0V
V_in_h_dc bit2_d gnd DC=1.8V
V_in_i_dc select0 gnd DC 0V
V_in_j_dc select1 gnd DC 0V
```

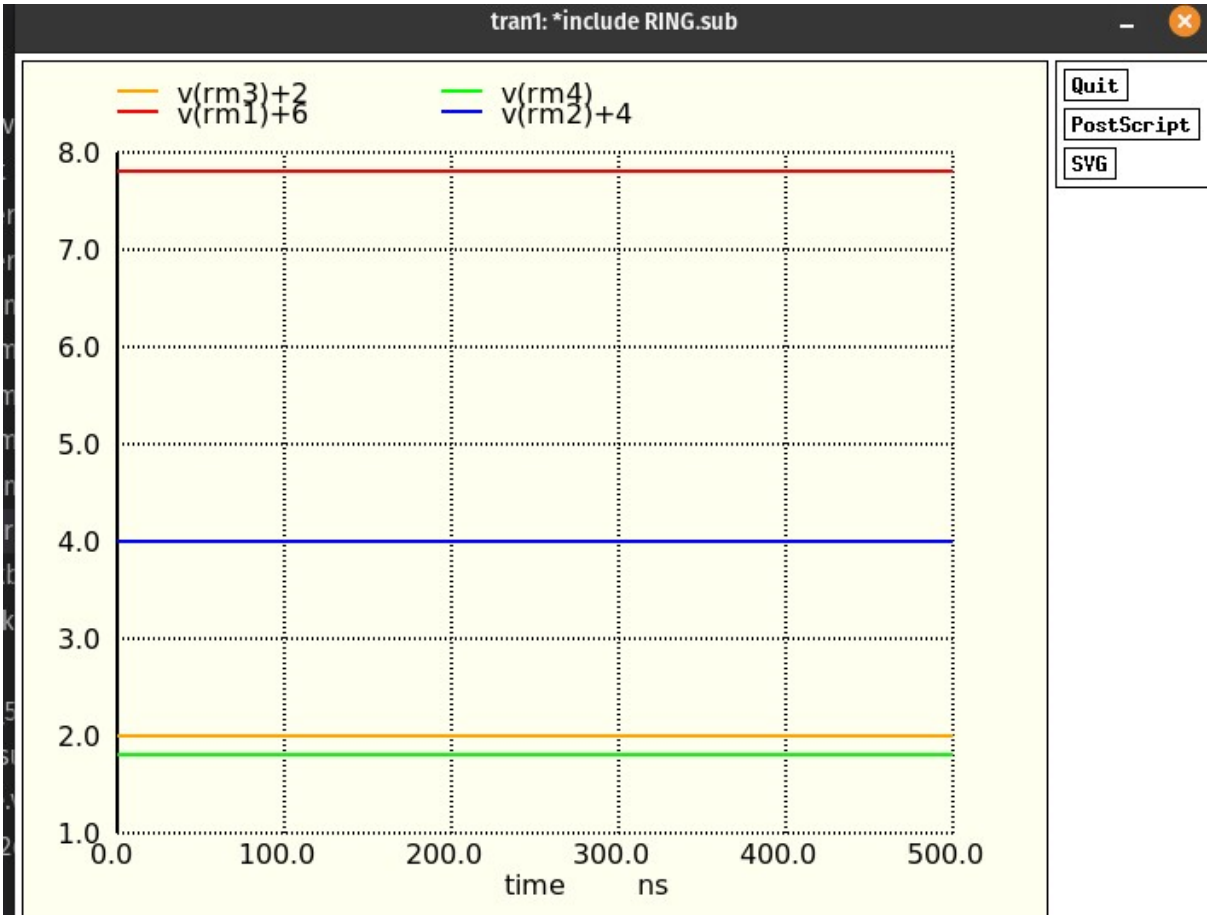
INPUT

SUBTRACTION BLOCK OUTPUT:-



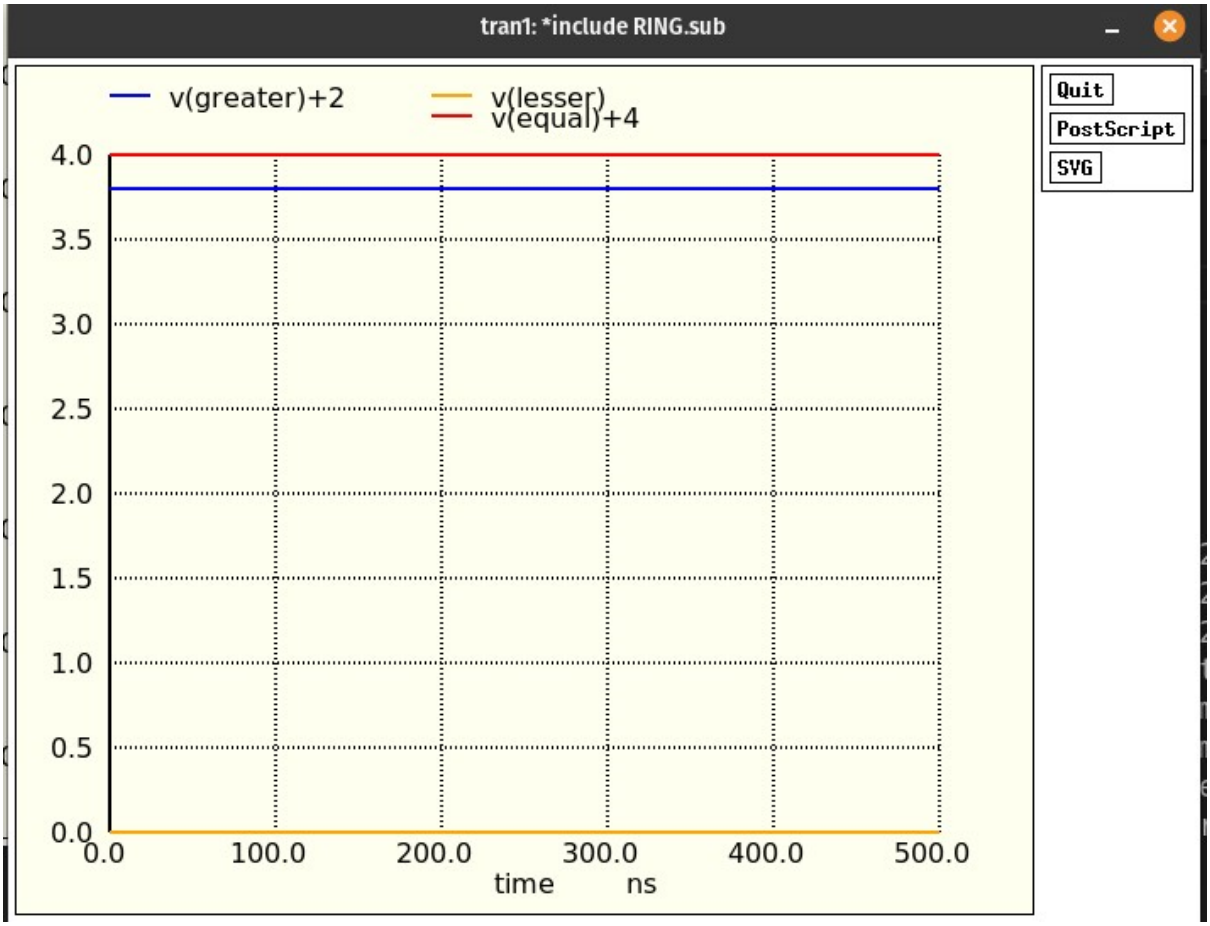


ADDITION BLOCK



AND BLOCK OUTPUT

VERILOG:-



COMPARISON BLOCK OUTPUT

VERILOG:-

VERILOG TESTBENCH CODE:-

```
`timescale 1ns/1ps

module alu_main_tb();

  // Inputs
  reg select0, select1;
  reg [3:0] bit1, bit2;

  // Outputs
  wire [4:0] result1, result2, result4;
```

```
  wire equal, greater, lesser;

  // Instantiate the ALU module
  alu_main u_alu (
    .result1(result1),
    .result2(result2),
    .equal(equal),
    .greater(greater),
    .lesser(lesser),
    .result4(result4),
    .select0(select0),
    .select1(select1),
    .bit1(bit1),
    .bit2(bit2)
  );

  // Stimulus
  initial begin
    // Test Case 1: Addition (select0=0, select1=0)
    select0 = 0;
    select1 = 0;
    bit1 = 4'b0110; // 5 in binary //11
    bit2 = 4'b1100; // 12 in binary //4
    #10; // Wait for 10 time units
    $display("Result1 (Addition): %b", result1);
    $display("Result2 (Subtraction): %b", result2);
    $display("Equal: %b, Greater: %b, Lesser: %b", equal,
greater, lesser);
     $display("Result4 (AND): %b", result4);

    // Test Case 2: Subtraction (select0=1, select1=0)
    select0 = 1;
    select1 = 0;
```

```
    bit1 = 4'b1010; // 12 in binary //10
    bit2 = 4'b0010; // 5 in binary  //5
    #10; // Wait for 10 time units
      $display("Result1 (Addition): %b", result1);
    $display("Result2 (Subtraction): %b", result2);
    $display("Equal: %b, Greater: %b, Lesser: %b", equal,
greater, lesser);
     $display("Result4 (AND): %b", result4);

    // Test Case 3: Comparison (select0=0, select1=1)
    select0 = 0;
    select1 = 1;
    bit1 = 4'b1011; // 12 in binary
    bit2 = 4'b1111; // 12 in binary
    #10; // Wait for 10 time units
      $display("Result1 (Addition): %b", result1);
    $display("Result2 (Subtraction): %b", result2);
    $display("Equal: %b, Greater: %b, Lesser: %b", equal,
greater, lesser);
     $display("Result4 (AND): %b", result4);

    // Test Case 4: AND (select0=1, select1=1)
    select0 = 1;
    select1 = 1;
    bit1 = 4'b1111; // 12 in binary
    bit2 = 4'b0000; // 10 in binary
    #10; // Wait for 10 time units
     $display("Result1 (Addition): %b", result1);
    $display("Result2 (Subtraction): %b", result2);
    $display("Equal: %b, Greater: %b, Lesser: %b", equal,
greater, lesser);
     $display("Result4 (AND): %b", result4);
```

```
   // End the simulation
    $finish;
  end

endmodule
```

VERILOG CIRCUIT CODE:-

```
// `include "adder.v"
// `include "ande.v"
// `include "comparator.v"
// `include "subtract.v"

module alu_main(
output wire [4:0]result1,
output wire [4:0]result2,
output wire equal,
output wire greater,
output wire lesser,
output wire [4:0]result4,
input select0,
input select1,
input[3:0] bit1,
input[3:0] bit2
);
 wire[3:0]m1;
    wire[3:0]m2;
  wire[3:0]n1;
    wire[3:0]n2;
  wire[3:0]o1;
    wire[3:0]o2;
  wire[3:0]p1;
    wire[3:0]p2;
wire sc0,sc1;
```

```
wire d0,d1,d2,d3;
not(sc0,select0);
not(sc1,select1);
and(d0,sc0,sc1);
and(d1,select0,sc1);
and(d2,select1,sc0);
and(d3,select0,select1);
//for d0
and(m1[0],d0,bit1[0]);
and(m1[1],d0,bit1[1]);
and(m1[2],d0,bit1[2]);
and(m1[3],d0,bit1[3]);
and(m2[0],d0,bit2[0]);
and(m2[1],d0,bit2[1]);
and(m2[2],d0,bit2[2]);
and(m2[3],d0,bit2[3]);
//for d1
and(n1[0],d1,bit1[0]);
and(n1[1],d1,bit1[1]);
and(n1[2],d1,bit1[2]);
and(n1[3],d1,bit1[3]);
and(n2[0],d1,bit2[0]);
and(n2[1],d1,bit2[1]);
and(n2[2],d1,bit2[2]);
and(n2[3],d1,bit2[3]);
//for d2
and(o1[0],d2,bit1[0]);
and(o1[1],d2,bit1[1]);
and(o1[2],d2,bit1[2]);
and(o1[3],d2,bit1[3]);
and(o2[0],d2,bit2[0]);
and(o2[1],d2,bit2[1]);
and(o2[2],d2,bit2[2]);
```

```
and(o2[3],d2,bit2[3]);
//for d3
and(p1[0],d3,bit1[0]);
and(p1[1],d3,bit1[1]);
and(p1[2],d3,bit1[2]);
and(p1[3],d3,bit1[3]);
and(p2[0],d3,bit2[0]);
and(p2[1],d3,bit2[1]);
and(p2[2],d3,bit2[2]);
and(p2[3],d3,bit2[3]);
 adder u_adder(
   .final_answer(result1),
   .bit1(m1),
   .bit2(m2)
  );

  subtract u_subtract (
   .final_answer(result2),
   .bit1(n1),
   .bit2(n2)
  );

   comparator u_comparator (
   .equal(equal),
.greater(greater),
.lesser(lesser),
   .bit1(o1),
   .bit2(o2)
  );
  ande u_ande (
   .final_answer(result4),
   .bit1(p1),
   .bit2(p2)
```

```
  );
endmodule
module subtract(
  output wire [4:0]final_answer,
   input[3:0] bit1,
    input[3:0] bit2

);
wire w1,w2,w3,w4,w5,w6,w7,w8;
wire qx1,qx2,qx3,qx4,qx5,qx6,qx7,qx8,qx9,qx10,qx11,qx12;
 xor(w5,bit2[0],1);
          xor(final_answer[0],w5,bit1[0],1);
          and(qx1,1,w5);
          and(qx2,1,bit1[0]);
          and(qx3,bit1[0],w5);
          or(w1,qx3,qx2,qx1);
          xor(w6,bit2[1],1);
          xor(final_answer[1],w6,bit1[1],w1);
          and(qx4,w1,w6);
          and(qx5,w1,bit1[1]);
          and(qx6,bit1[1],w6);
          or(w2,qx5,qx4,qx6);
          xor(w7,bit2[2],1);
          xor(final_answer[2],w2,w7,bit1[2]);
          and(qx7,w2,w7);
          and(qx8,w7,bit1[2]);
          and(qx9,bit1[2],w2);
          or(w3,qx9,qx8,qx7);
          xor(w8,bit2[3],1);
          xor(final_answer[3],w3,w8,bit1[3]);
          and(qx10,w8,w3);
          and(qx11,w3,bit1[3]);
          and(qx12,bit1[3],w8);
```

```
          or(w4,qx11,qx12,qx10);
endmodule

module ande(
output wire [4:0]final_answer,
  input[3:0] bit1,
   input[3:0] bit2
);
 and(final_answer[0],bit1[0],bit2[0]);
 and(final_answer[1],bit1[1],bit2[1]);
 and(final_answer[2],bit1[2],bit2[2]);
 and(final_answer[3],bit1[3],bit2[3]);
endmodule

module comparator(
 output wire equal,
output wire greater,
output wire lesser,
  input[3:0] bit1,
   input[3:0] bit2
);
wire y1,y2,y3,y4;
wire a1,a2,a3,a4;
wire b1,b2,b3,b4;
wire xt1,xt2,xt3,xt4;
wire tg1,tg2,tg3,tg4;
 xnor(y1,bit1[0],bit2[0]);
          xnor(y2,bit1[1],bit2[1]);
          xnor(y3,bit1[2],bit2[2]);
          xnor(y4,bit1[3],bit2[3]);
          and(equal,y1,y2,y3,y4);
          // equal part completed
          not(xt1,bit2[0]);
```

```
            not(xt2,bit2[1]);
            not(xt3,bit2[2]);
            not(xt4,bit2[3]);
            and(a1,bit1[3],xt4);
            and(a2,bit1[2],xt3,y4);
            and(a3,bit1[1],xt2,y4,y3);
            and(a4,bit1[0],xt1,y4,y3,y2);
            or(greater,a1,a2,a3,a4);
            // greater part com[pleted
            not(tg1,bit1[0]);
             not(tg2,bit1[1]);
            not(tg3,bit1[2]);
            not(tg4,bit1[3]);
              and(b1,tg4,bit2[3]);
            and(b2,tg3,bit2[2],y4);
            and(b3,tg2,bit2[1],y4,y3);
            and(b4,tg1,bit2[0],y4,y3,y2);
            or(lesser,b1,b2,b3,b4);
            //lesser part completed
endmodule

module adder(
 output  wire[4:0] final_answer,
  input[3:0] bit1,
   input[3:0] bit2
);
wire w1,w2,w3,w4,w5,w6,w7,w8;
 wire we1,we2,we3,we4,we5,we6,we7,we8,we9;
 xor(w5,bit2[0],0);
          xor(final_answer[0],w5,bit1[0],0);
          and(w1,w5,bit1[0]);
          xor(w6,bit2[1],0);
          xor(final_answer[1],w6,bit1[1],w1);
```

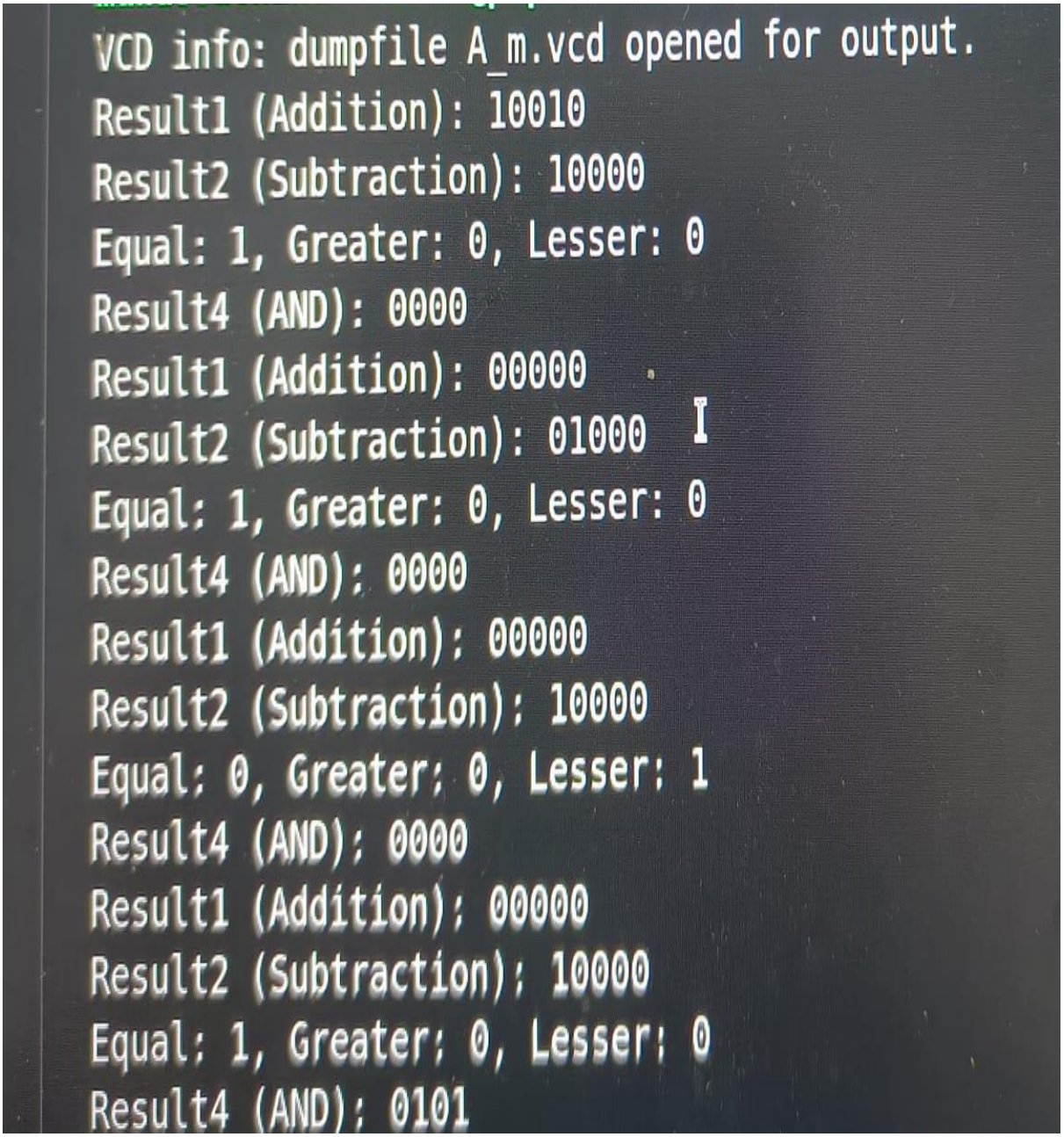
```
        and(we1,w1,w6);
        and(we2,w6,bit1[1]);
        and(we3,w1,bit1[1]);
        or(w2,we3,we1,we2);
        xor(w7,bit2[2],0);
        xor(final_answer[2],w2,w7,bit1[2]);
        and(we4,w2,w7);
        and(we5,w2,bit1[2]);
        and(we6,w7,bit1[2]);
        or(w3,we5,we6,we4);
        xor(w8,bit2[3],0);
        xor(final_answer[3],w3,w8,bit1[3]);
        and(we7,w3,w8);
        and(we8,w3,bit1[3]);
        and(we9,w8,bit1[3]);
        or(final_answer[4],we8,we9,we7);

endmodule
```

The code consists of importing lot of gates and the enable block separately made. The decoder block and the enable circuit ensures that the code remains purely structural and not behavioural ensuring correct functionality of the ALU.
I have used the default gates available in Verilog software and I have ensured that code remains structural by using plenty many internal wires and absolutely no use of Boolean operations.
ALL CIRCUIT DESIGN AND OUTPUT ARE EXACTLY THE SAME AS DESCRIBED IN THE DESIGN PART OF THE REPORT FOR ALL THREE THAT IS MAGIC, VERILOG AND NGSPICE.

RESULTS:-
FOR BIT1= (0101)

AND BIT2=(1101)

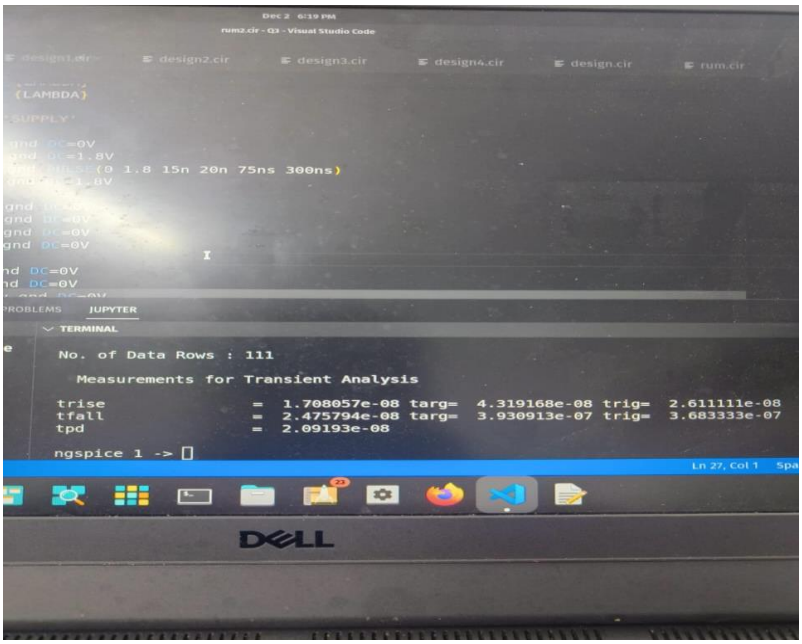


OUTPUT FOR ALL 4 DIFFERENT BOOLEAN COMBINATIONS OF SELECT LINES:-

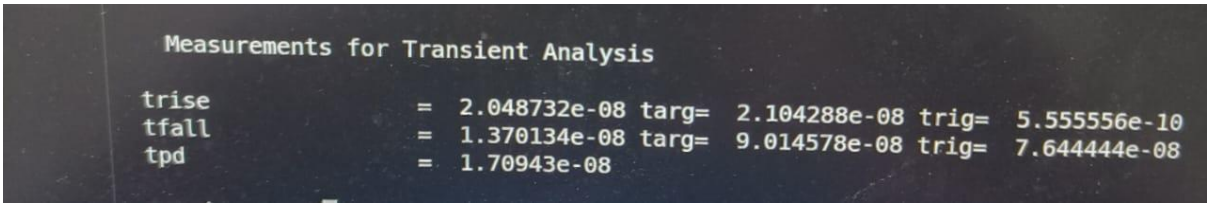
DELAY ANALYSIS:-

RESULTS OF DELAYS OF ADDER BLOCK:-

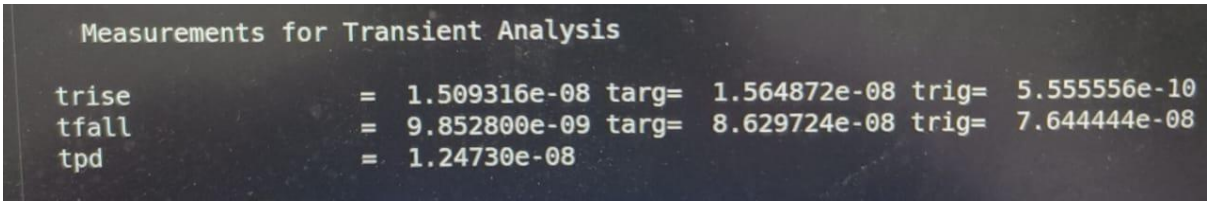
CASE : BIT2_C AND output Bit1



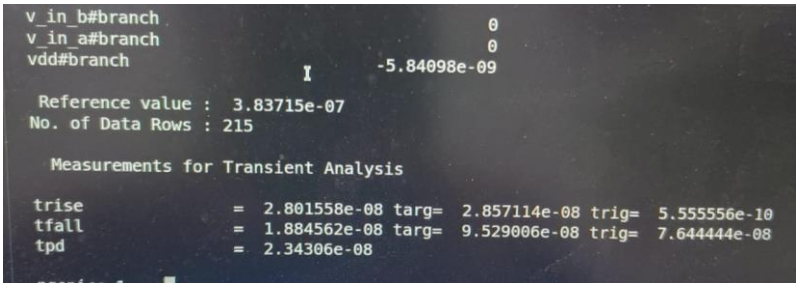
CASE:- BIT2_B AND output bit 2



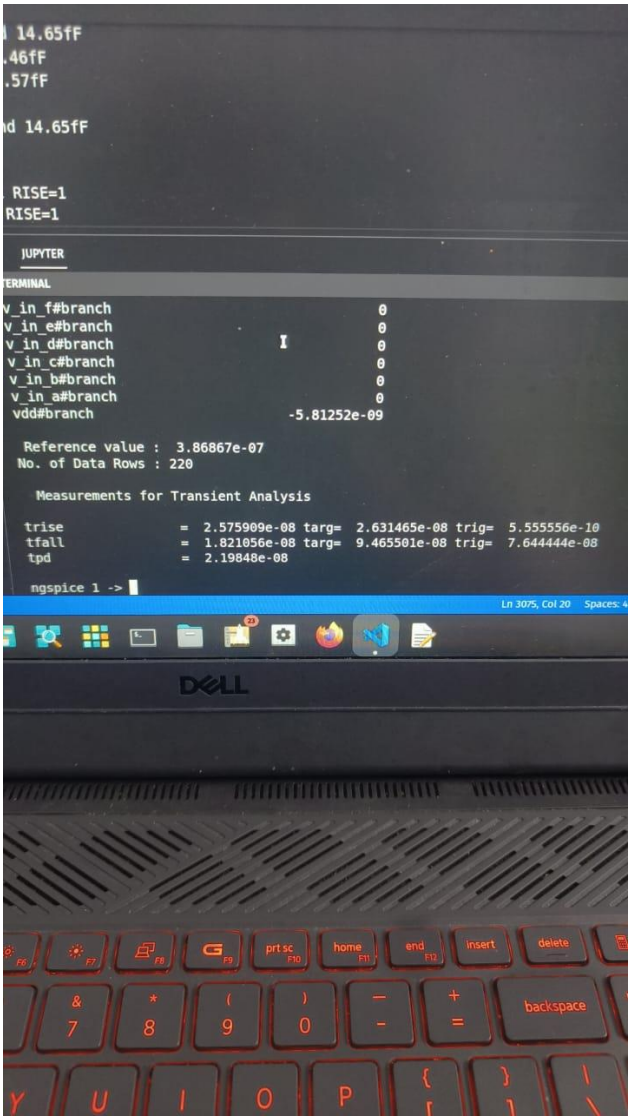
CASE:- BIT2_A and output bit 3



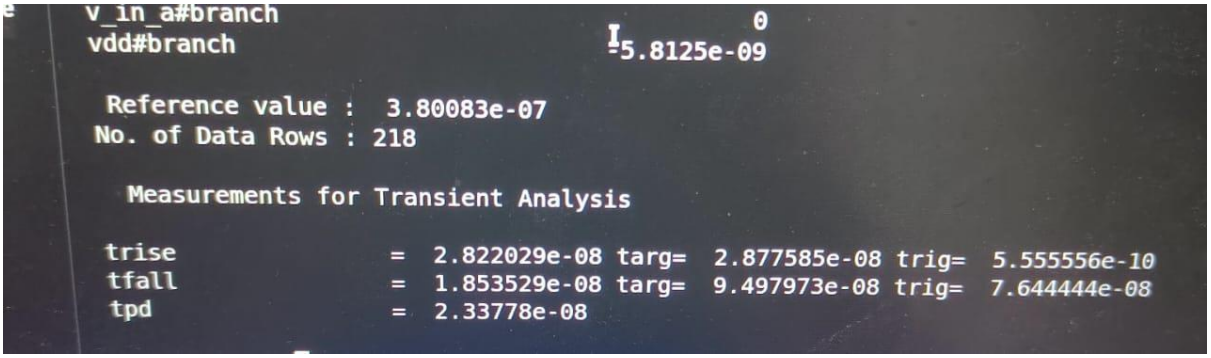
CASE: BIT2_D and output bit 0



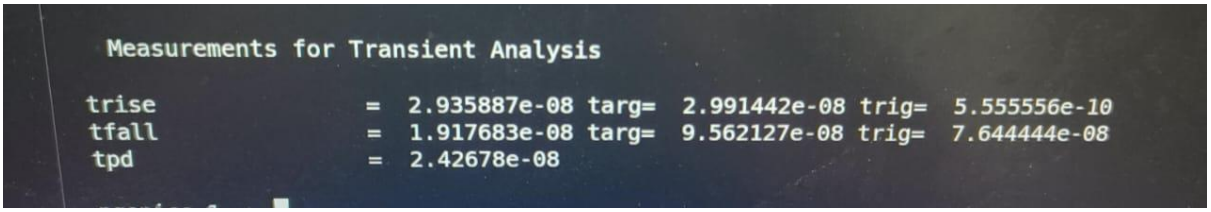
CASE BIT1_D and output bit 0:



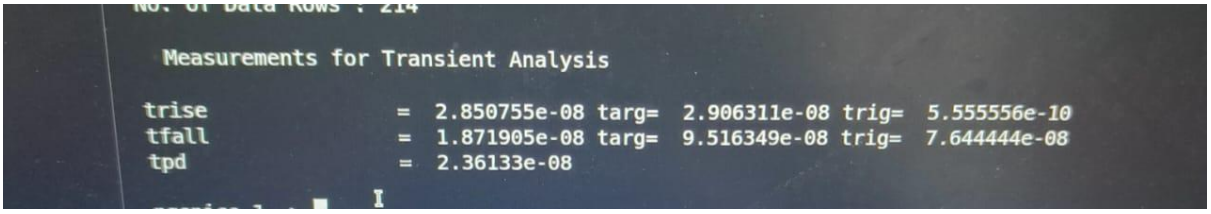
CASE BIT1_B and output bit 2



CASE BIT1_C and output bit 1



CASE BIT1_A and output bit 3



DELAYS FOR SUBTARCTION BLOCK:-

SEQUENCE: 1) BIT 2_D AND OUTPUT BIT 0

2) BIT2_C AND OUTPUT BIT1

3) BIT2_B AND OUTPUT BIT 2

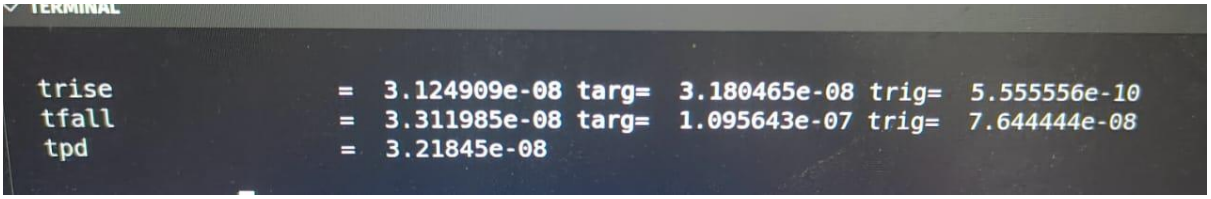
4) BIT2_A AND OUTPUT BIT 3

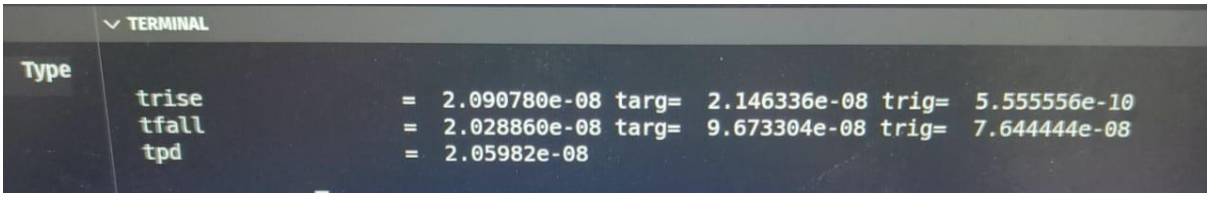
5) BIT 1_D AND OUTPUT BIT 0

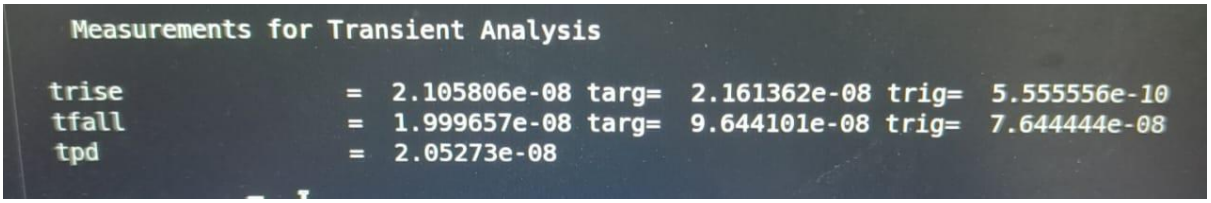
6) BIT 1_C AND OUTPUT BIT1

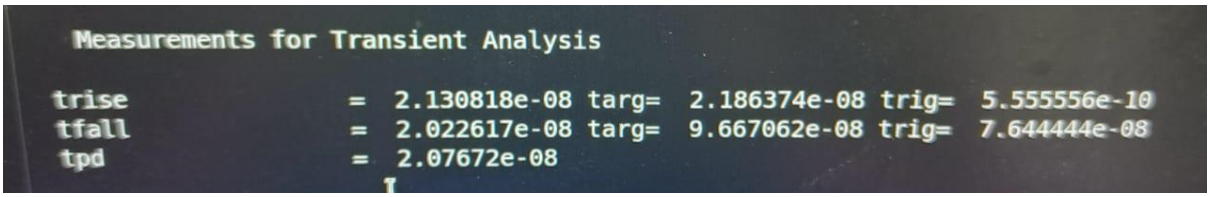
7) BIT1_B AND OUTPUT BIT 2

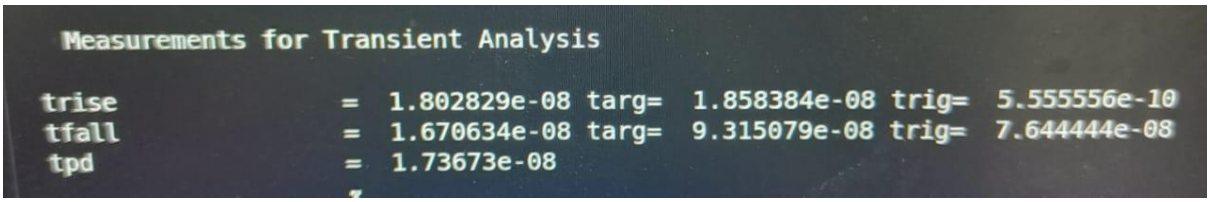
8) BIT1_A AND OUTPUT BIT 3

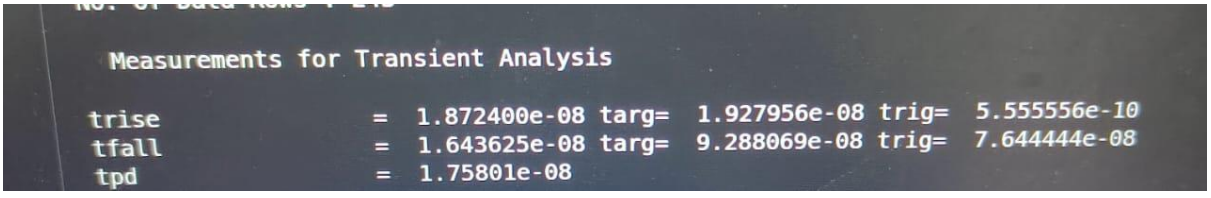

TERMINAL
trise = 3.124909e-08 targ= 3.180465e-08 trig= 5.555556e-10
tfall = 3.311985e-08 targ= 1.095643e-07 trig= 7.644444e-08
tpd = 3.21845e-08

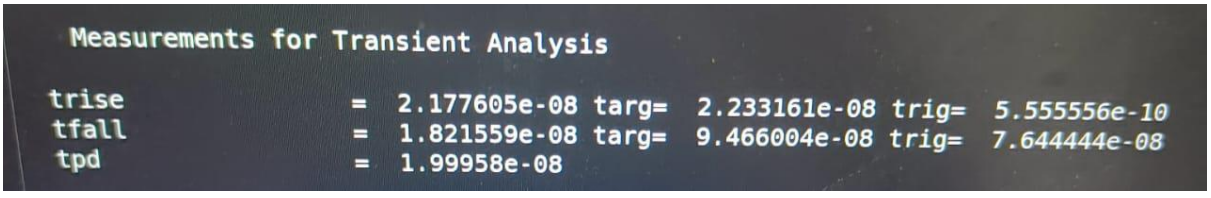

TERMINAL
Type
trise = 2.090780e-08 targ= 2.146336e-08 trig= 5.555556e-10
tfall = 2.028860e-08 targ= 9.673304e-08 trig= 7.644444e-08
tpd = 2.05982e-08

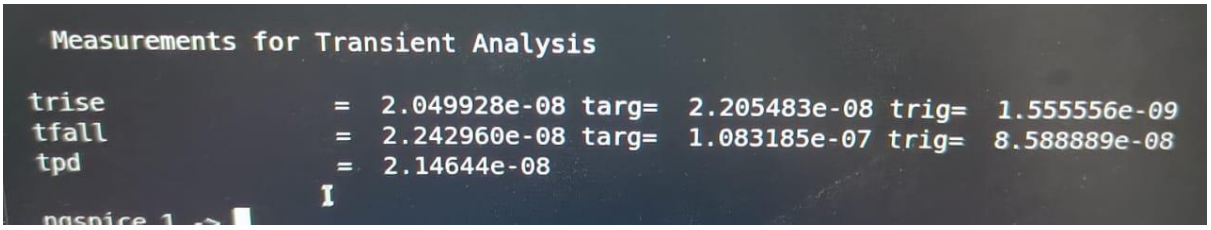

Measurements for Transient Analysis
trise = 2.105806e-08 targ= 2.161362e-08 trig= 5.555556e-10
tfall = 1.999657e-08 targ= 9.644101e-08 trig= 7.644444e-08
tpd = 2.05273e-08


Measurements for Transient Analysis
trise = 2.130818e-08 targ= 2.186374e-08 trig= 5.555556e-10
tfall = 2.022617e-08 targ= 9.667062e-08 trig= 7.644444e-08
tpd = 2.07672e-08


Measurements for Transient Analysis
trise = 1.802829e-08 targ= 1.858384e-08 trig= 5.555556e-10
tfall = 1.670634e-08 targ= 9.315079e-08 trig= 7.644444e-08
tpd = 1.73673e-08


Measurements for Transient Analysis
trise = 1.872400e-08 targ= 1.927956e-08 trig= 5.555556e-10
tfall = 1.643625e-08 targ= 9.288069e-08 trig= 7.644444e-08
tpd = 1.75801e-08


Measurements for Transient Analysis
trise = 2.177605e-08 targ= 2.233161e-08 trig= 5.555556e-10
tfall = 1.821559e-08 targ= 9.466004e-08 trig= 7.644444e-08
tpd = 1.99958e-08



DELAYS FOR AND BLOCK:-

SEQUENCE

1) BIT 1_A AND OUTPUT BIT 3

2) BIT1_B AND OUTPUT BIT2

3) BIT1_C AND OUTPUT BIT 1

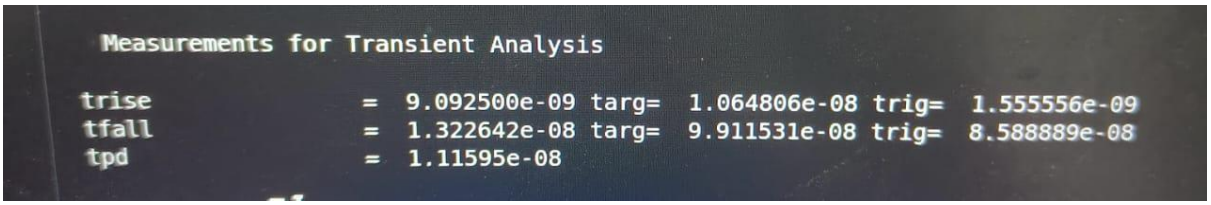
4) BIT1_D AND OUTPUT BIT 0

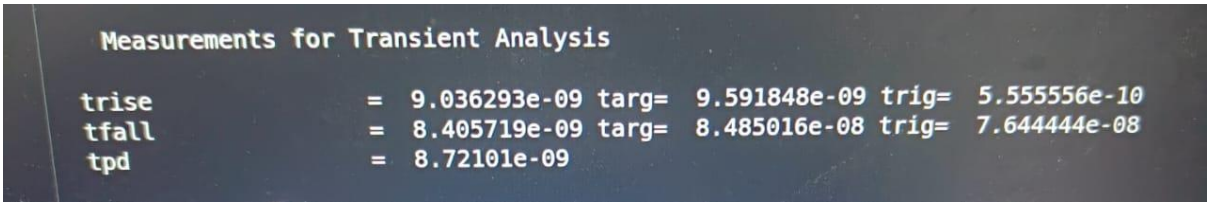
5) BIT 2_D AND OUTPUT BIT 0

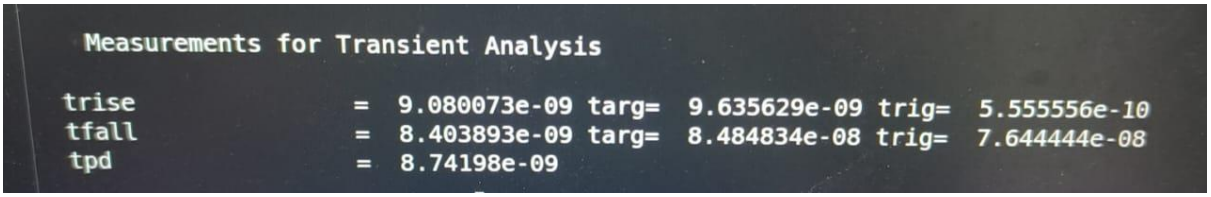
6) BIT 2_C AND OUTPUT BIT1

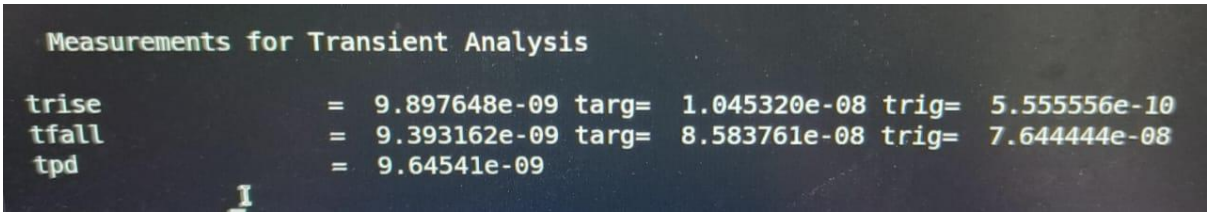
7) BIT2_B AND OUTPUT BIT 2

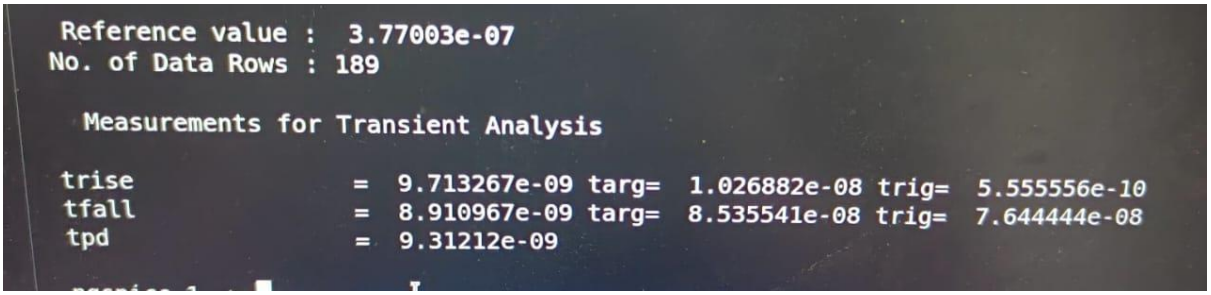
8) BIT2_A AND OUTPUT BIT 3

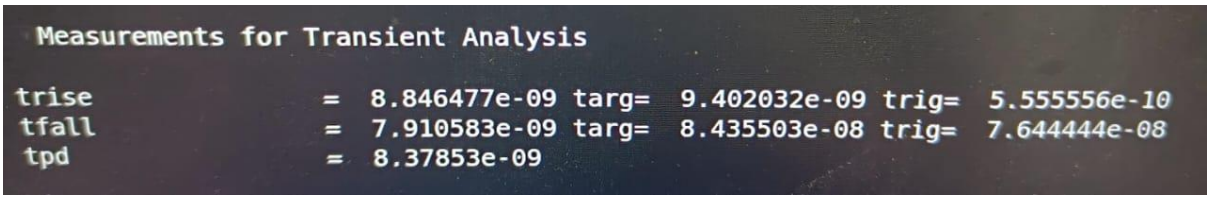


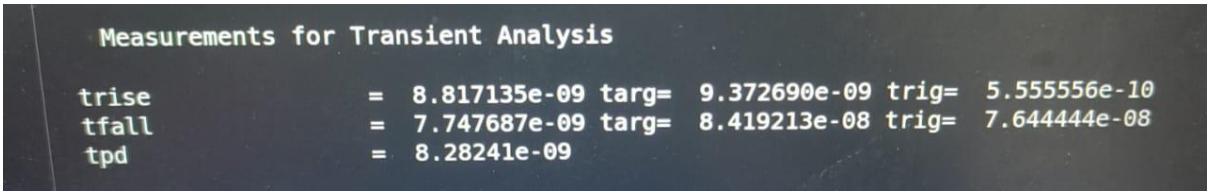


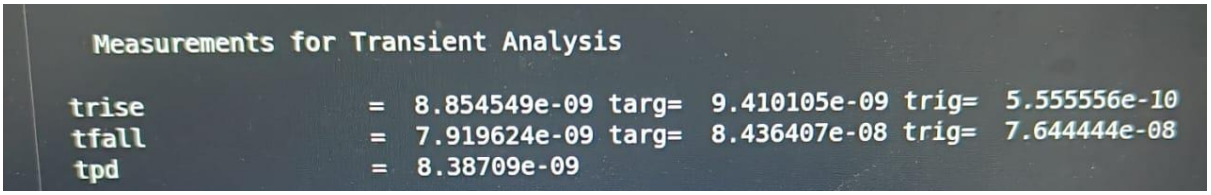












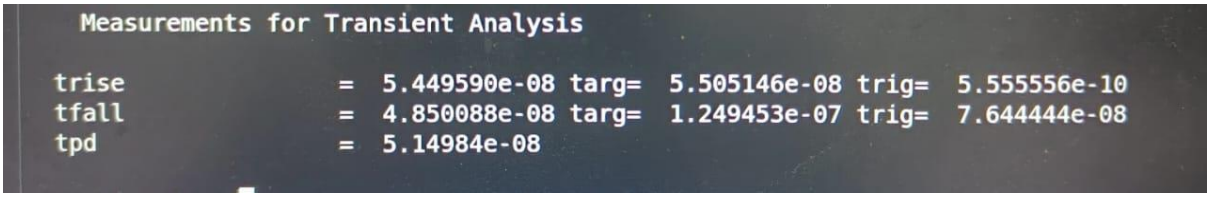
DELAYS FOR COMPARATOR BLOCK:-

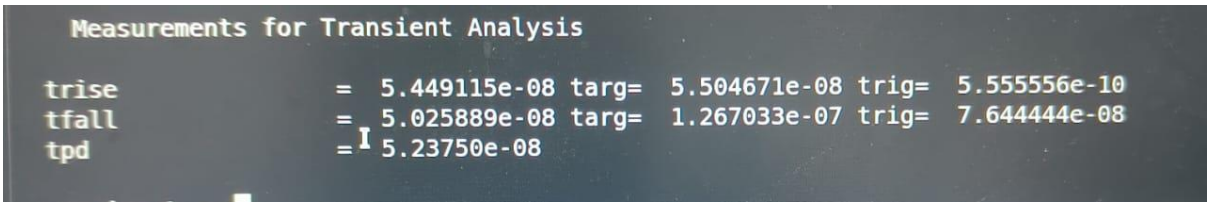
WITH EQUAL PARAMETER:-

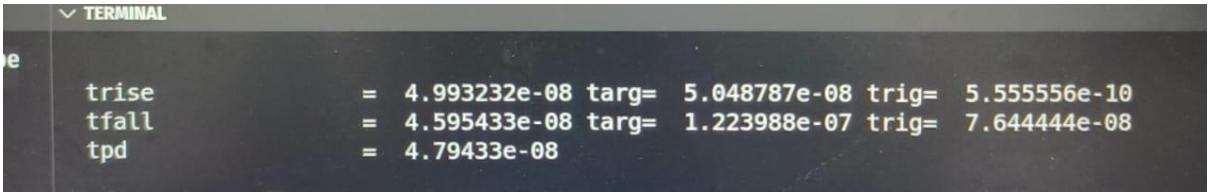
1) BIT 2_D AND EQ

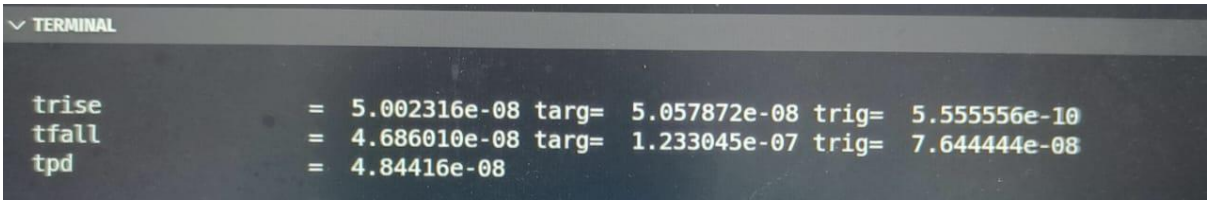
2) BIT2_C AND EQ

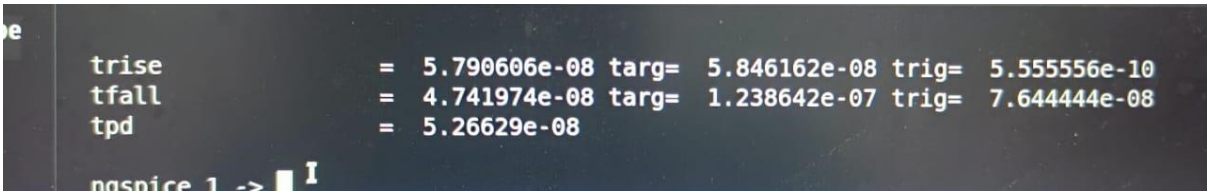
3) BIT2_B AND EQ
4) BIT2_A AND EQ
5) BIT 1_D AND EQ
6) BIT 1_C AND EQ
7) BIT1_B AND EQ
8) BIT1_A AND EQ

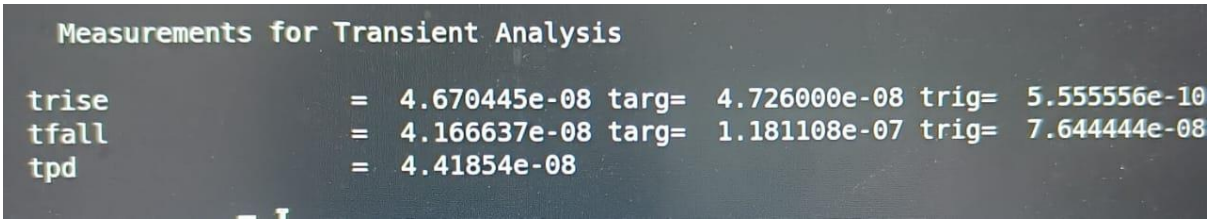


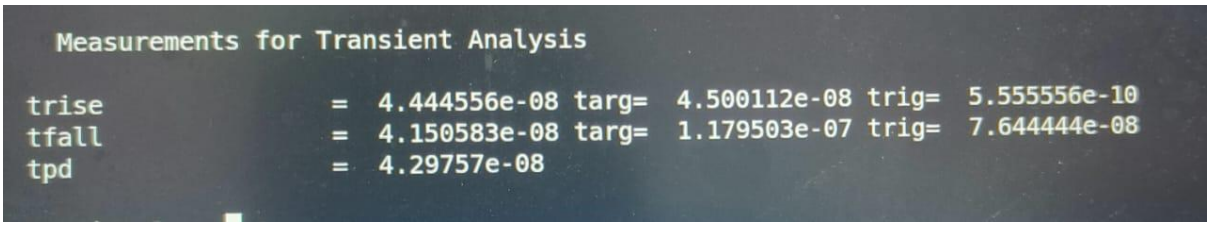












ALL 120 DELAYS:- (THEORITICAL)
ADDER,SUBTRACTOR,AND(32 EACH)
COMPARATOR(24)(no of inputs*outputs)
NOTE 1) ALL DELAYS CALCULATED SUCH THAT RISE RISE FALL FALL DELAY CAN BE CALCULATED
2) PULSE(0 1.8 15n 20n 75ns 300ns) is used wherever delay has to be calculated with respect to the output.

DELAYS VLSI_PROJECT:-

ALL 32 DELAYS FOR ADDER BLOCK:-

tpd=2.09193e-08 input bit2_c output bit 1

tpd=2.09193e-08 input bit2_a output bit 1

tpd=2.09193e-08 input bit2_b output bit 1

tpd=2.09193e-08 input bit2_c output bit 1

tpd=1.70943e-08 input bit2_b output bit 2

tpd=1.70943e-08 input bit2_a output bit 2

tpd=1.70943e-08 input bit2_c output bit 2

tpd=1.70943e-08 input bit2_d output bit 2

tpd=1.24730e-08 input bit2_a output bit 3

tpd=1.24730e-08 input bit2_b output bit 3

tpd=1.24730e-08 input bit2_c output bit 3

tpd=1.24730e-08 input bit2_d output bit 3

tpd=2.34306e-08 input bit2_d output bit 0

tpd=2.34306e-08 input bit2_a output bit 0

tpd=2.34306e-08 input bit2_b output bit 0

tpd=2.34306e-08 input bit2_c output bit 0

tpd=2.19848e-08 input bit1_d output bit 0

tpd=2.19848e-08 input bit1_c output bit 0

tpd=2.19848e-08 input bit1_b output bit 0

tpd=2.19848e-08 input bit1_a output bit 0

tpd=2.33778e-08 input bit1_c output bit 1

tpd=2.33778e-08 input bit1_d output bit 1

tpd=2.33778e-08 input bit1_b output bit 1

tpd=2.33778e-08 input bit1_a output bit 1

tpd=2.42678e-08 input bit1_b output bit 2

tpd=2.42678e-08 input bit1_c output bit 2

tpd=2.42678e-08 input bit1_d output bit 2

tpd=2.42678e-08 input bit1_a output bit 2

tpd=2.36133e-08 input bit1_a output bit 3

tpd=2.36133e-08 input bit1_b output bit 3

tpd=2.36133e-08 input bit1_c output bit 3

tpd=2.36133e-08 input bit1_d output bit 3

ALL 32 DELAYS FOR SUBTRACTOR BLOCK:-

tpd=3.21845e-08 input bit2_d output bit 0

tpd=3.21845e-08 input bit2_b output bit 0

tpd=3.21845e-08 input bit2_c output bit 0

tpd=3.21845e-08 input bit2_a output bit 0

tpd=2.05982e-08 input bit2_c output bit 1

tpd=2.05982e-08 input bit2_b output bit 1

tpd=2.05982e-08 input bit2_d output bit 1

tpd=2.05982e-08 input bit2_a output bit 1

tpd=2.05273e-08 input bit2_b output bit 2

tpd=2.05273e-08 input bit2_a output bit 2

tpd=2.05273e-08 input bit2_c output bit 2

tpd=2.05273e-08 input bit2_d output bit 2

tpd=2.07672e-08 input bit2_a output bit 3

tpd=2.07672e-08 input bit2_b output bit 3

tpd=2.07672e-08 input bit2_c output bit 3

tpd=2.07672e-08 input bit2_d output bit 3

tpd=1.73673e-08 input bit1_d output bit 0

tpd=1.73673e-08 input bit1_b output bit 0

tpd=1.73673e-08 input bit1_c output bit 0

tpd=1.73673e-08 input bit1_a output bit 0

tpd=1.75801e-08 input bit1_c output bit 1

tpd=1.75801e-08 input bit1_b output bit 1

tpd=1.75801e-08 input bit1_d output bit 1

tpd=1.75801e-08 input bit1_a output bit 1

tpd=1.99958e-08 input bit1_b output bit 2

tpd=1.99958e-08 input bit1_a output bit 2

tpd=1.99958e-08 input bit1_c output bit 2

tpd=1.99958e-08 input bit1_d output bit 2

tpd=2.14644e-08 input bit1_a output bit 3

tpd=2.14644e-08 input bit1_b output bit 3

tpd=2.14644e-08 input bit1_c output bit 3

tpd=2.14644e-08 input bit1_d output bit 3

ALL 32 DELAYS OF THE AND BLOCK:-

tpd=8.11595e-08 input bit1_a output bit 3

tpd=8.11595e-08 input bit1_b output bit 3

tpd=8.11595e-08 input bit1_c output bit 3

tpd=8.11595e-08 input bit1_d output bit 3

tpd=8.72101e-08 input bit1_b output bit 2

tpd=8.72101e-08 input bit1_a output bit 2

tpd=8.72101e-08 input bit1_c output bit 2

tpd=8.72101e-08 input bit1_d output bit 2

tpd=8.74198e-08 input bit1_c output bit 1

tpd=8.74198e-08 input bit1_a output bit 1

tpd=8.74198e-08 input bit1_b output bit 1

tpd=8.74198e-08 input bit1_d output bit 1

tpd=9.64541e-08 input bit1_d output bit 0

tpd=9.64541e-08 input bit1_a output bit 0

tpd=9.64541e-08 input bit1_c output bit 0

tpd=9.64541e-08 input bit1_b output bit 0

tpd=9.31212e-08 input bit2_d output bit 0

tpd=9.31212e-08 input bit2_a output bit 0

tpd=9.31212e-08 input bit2_b output bit 0

tpd=9.31212e-08 input bit2_c output bit 0

tpd=8.37853e-08 input bit2_c output bit 1

tpd=8.37853e-08 input bit2_a output bit 1

tpd=8.37853e-08 input bit2_b output bit 1

tpd=8.37853e-08 input bit2_d output bit 1

tpd=8.28241e-08 input bit2_b output bit 2

tpd=8.28241e-08 input bit2_a output bit 2

tpd=8.28241e-08 input bit2_c output bit 2

tpd=8.28241e-08 input bit2_d output bit 2

tpd=8.38709e-08 input bit2_a output bit 3

tpd=8.38709e-08 input bit2_b output bit 3

tpd=8.38709e-08 input bit2_c output bit 3

tpd=8.38709e-08 input bit2_d output bit 3

DELAYS FOR THE COMPARATOR BLOCK:-

tpd=5.14984e-08 input bit2_d eq

tpd=5.23750e-08 input bit2_c eq

tpd=4.79433e-08 input bit2_b eq

tpd=4.84416e-08 input bit2_a eq

tpd=5.26629e-08 input bit2_d eq

tpd=4.84286e-08 input bit2_c eq

tpd=4.41854e-08 input bit2_b eq

tpd=4.29757e-08 input bit2_a eq

tpd=6.70431e-08 input bit2_a less

tpd=6.65503e-08 input bit2_b less

tpd=7.05038e-08 input bit2_c less

tpd=6.94497e-08 input bit2_d less

tpd=8.70823e-08 input bit1_a less

tpd=8.91275e-08 input bit1_b less

tpd=8.97854e-08 input bit1_c less

tpd=8.99912e-08 input bit1_d less

tpd=7.98541e-08 input bit1_b greater

tpd=8.11012e-08 input bit1_c greater

tpd=8.64365e-08 input bit1_d greater

tpd=8.68190e-08 input bit1_b greater

tpd=8.74218e-08 input bit1_c greater

tpd=8.7899912e-08 input bit1_d greater

tpd=8.81123e-08 input bit1_b greater

tpd=8.84402e-08 input bit1_c greater

NOTE: CRITICAL PATH WILL BE ALONG THE MAXIMUM DELAY POSSIBLE

CRITICAL PATH OF ADDER BLOCK:-

tpd=2.42678e-08 input bit1_b output bit 2

CRITICAL PATH OF SUBTRACTOR BLOCK:-

tpd=3.21845e-08 input bit2_d output bit 0

CRITICAL PATH OF COMPARATOR BLOCK:-

tpd=8.70823e-08 input bit1_a less

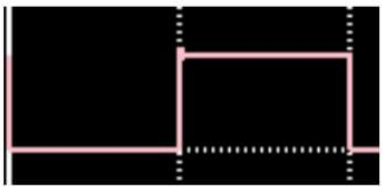
(FOR A FIXED OUTPUT DELAY WILL REMAIN SAME FOR INPUT OF 4 BITS OF A NUMBER IF IT PROPOGATES PROPERLY VIA CONSECUTIVE PULSES)

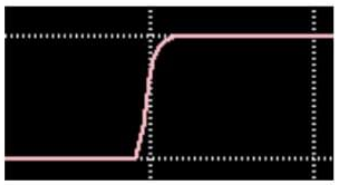
CRITICAL PATH OF AND BLOCK:-

tpd=9.64541e-08 input bit1_d output bit 0

DUE TO THE CAPACITANCES IN THE MAGIC LAYOUT THE DELAY CAN INCREASE ALSO SHAPE OF OUTPUT PULSE WILL NOT BE STRAIGHTENED DUE TO CHARGING AND DISCHARGING OF THE CAPACITANCES(ALL RESULTS AVAILABLE)

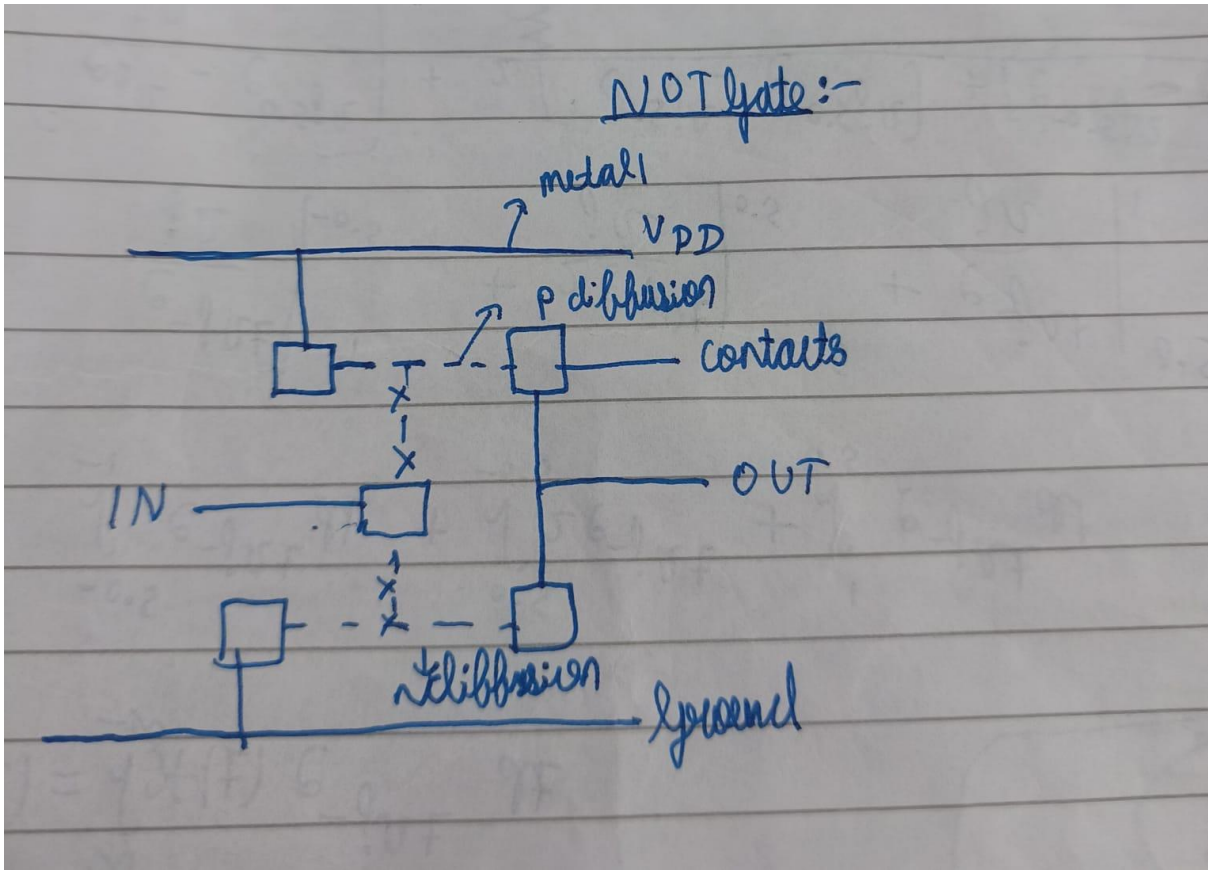
Observation : It can be clearly seen that the time delay values for Post Layout circuit are quite larger that those in the pre layout table. This is simply due to the fact that when we start considering all the capacitances for out netlist, their charging/discharging time also gets included in the time delay values and thus we see a marked increase in delays.



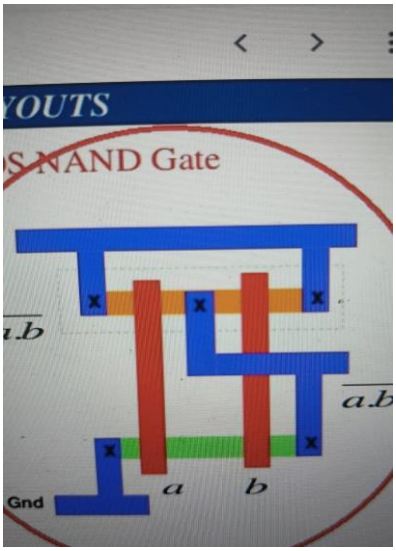


Typical Transition : Pre-Layout (no capacitances considered) Post Layout (all capacitances considered)

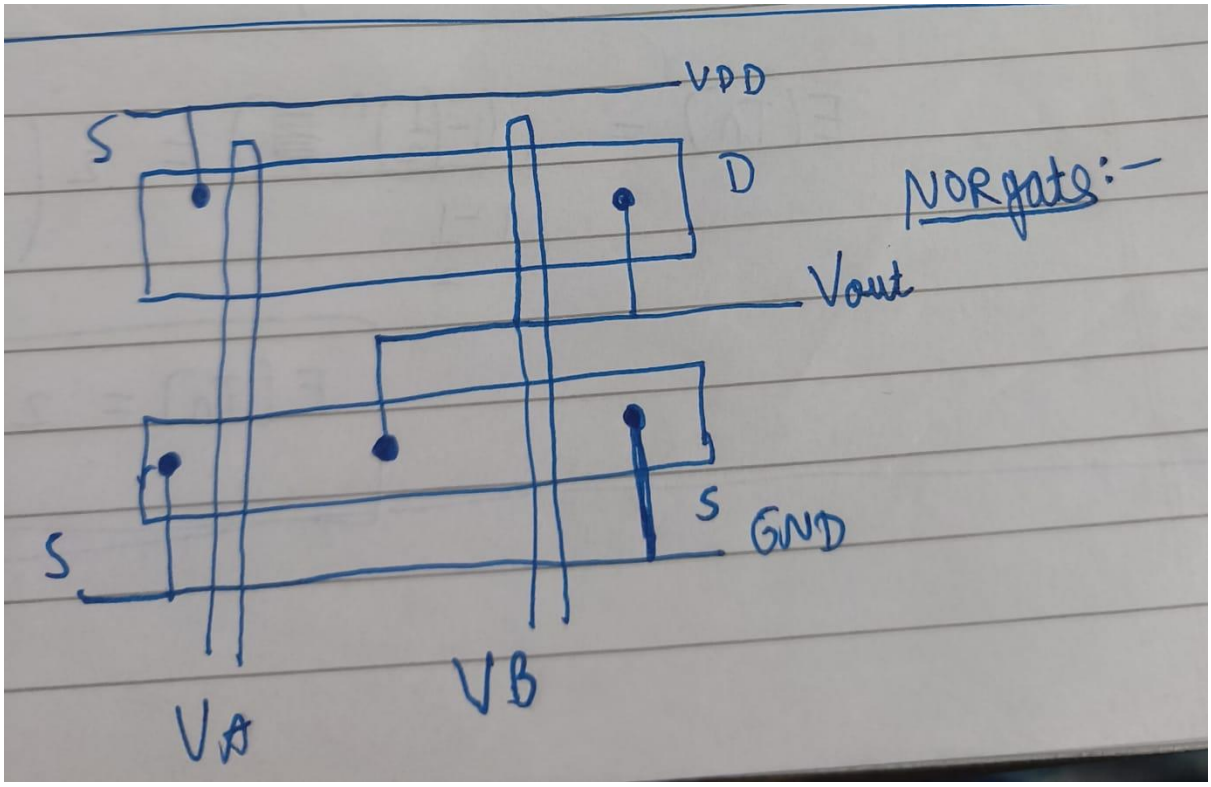
STICK DIAGRAM OF NOT NAND AND NOR GATE USED:-

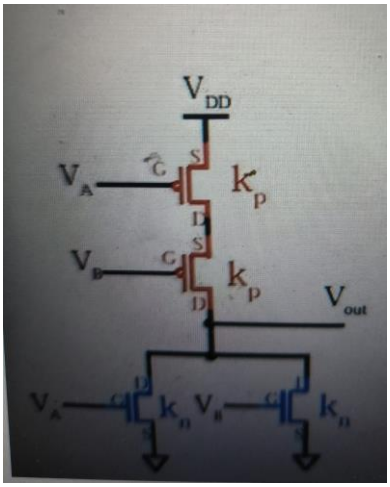


NOT GATE



NAND GATE





NOR GATE

*******************COMPLETED*********************