VLSI FINAL PROJECT REPORT

1 . VERILOG

2. NGSPICE

3. MAGIC LAYOUT

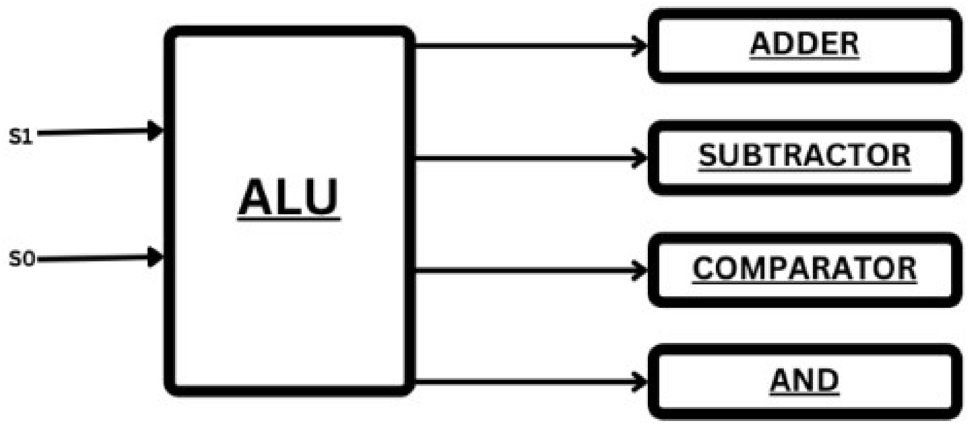
ANIRUTH S

2022102055

PROJECT OBJECTIVE

Design an ALU that can perform a 4-Bit addition, subtraction, comparison, ANDing. Estimate the critical path, maximum delay possible in the circuit. Design the layout of your ALU, clearly indicatethe location of each standard cell in the design. Compare your pre- and post-layout results. Also, Verify the functionality of your ALU using Verilog.

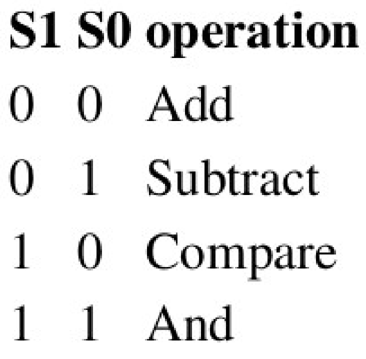
The basic block diagram describing the entire project :



So , we need to design 5 basic sub – blocks :

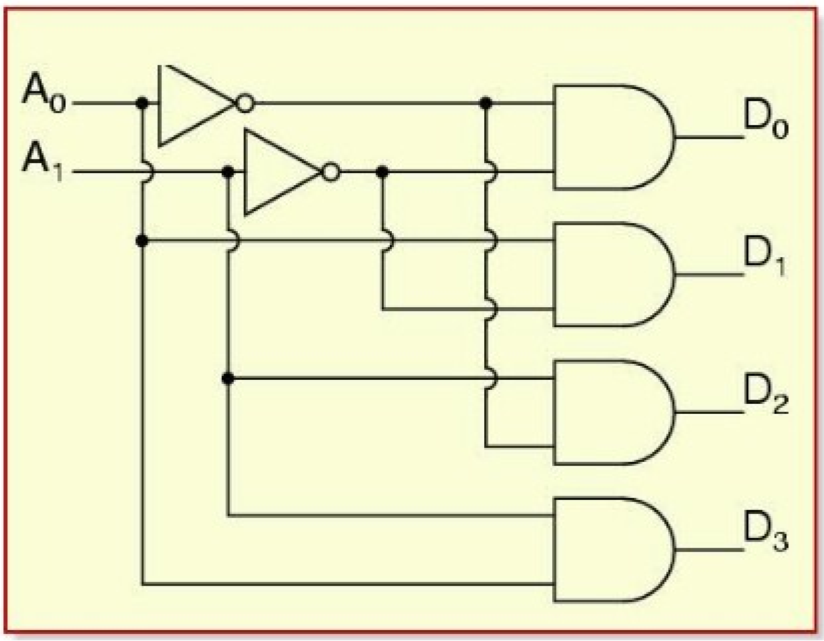
1. **ALU :**

Here ALU block acts as router to out computational circuit. The operations that needed to be done by ALU is as follows :



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

To establish this, we can use a 2-4 decoder :



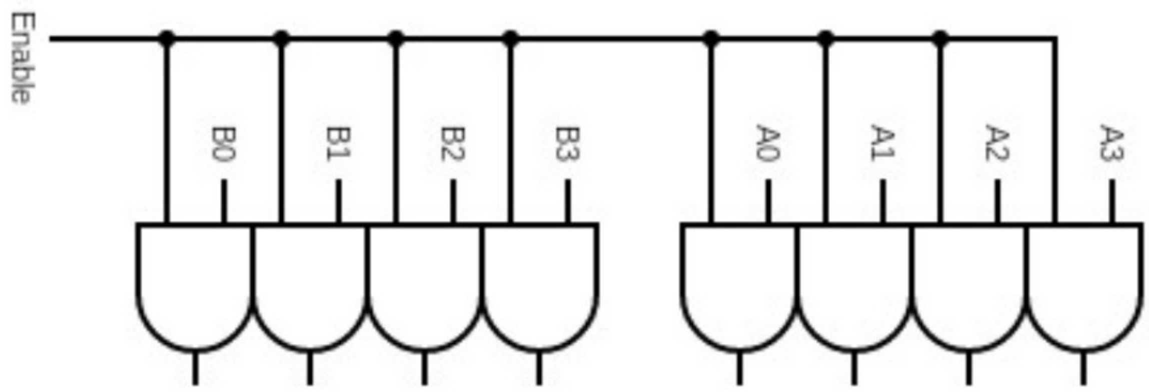
The enables for the following circuits are as follows :

D0 – Adder; D1 – Subtractor; D2 – Comparator; D3 – And

The enable block described above is given by :

Enable Block:

This is made-up of 8 AND gates whose main purpose is send our values A3A2A1A1, B3B2B1B0 to their respective block if enable is 1 else 0.



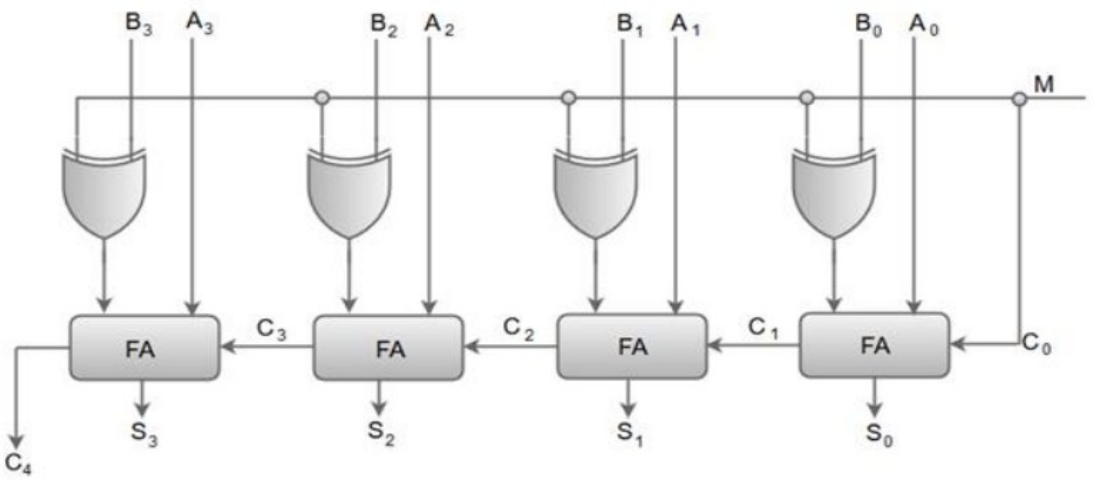
2. ADDER – SUBTRACTOR :

Here instead of making a separate Adder and Subtractor we can use a single block which can both act as adder and subtractor .

TO achieve this , we pass S1 to M i.e, in case of subtractor M = 1 , and in case of adder M = 0 .

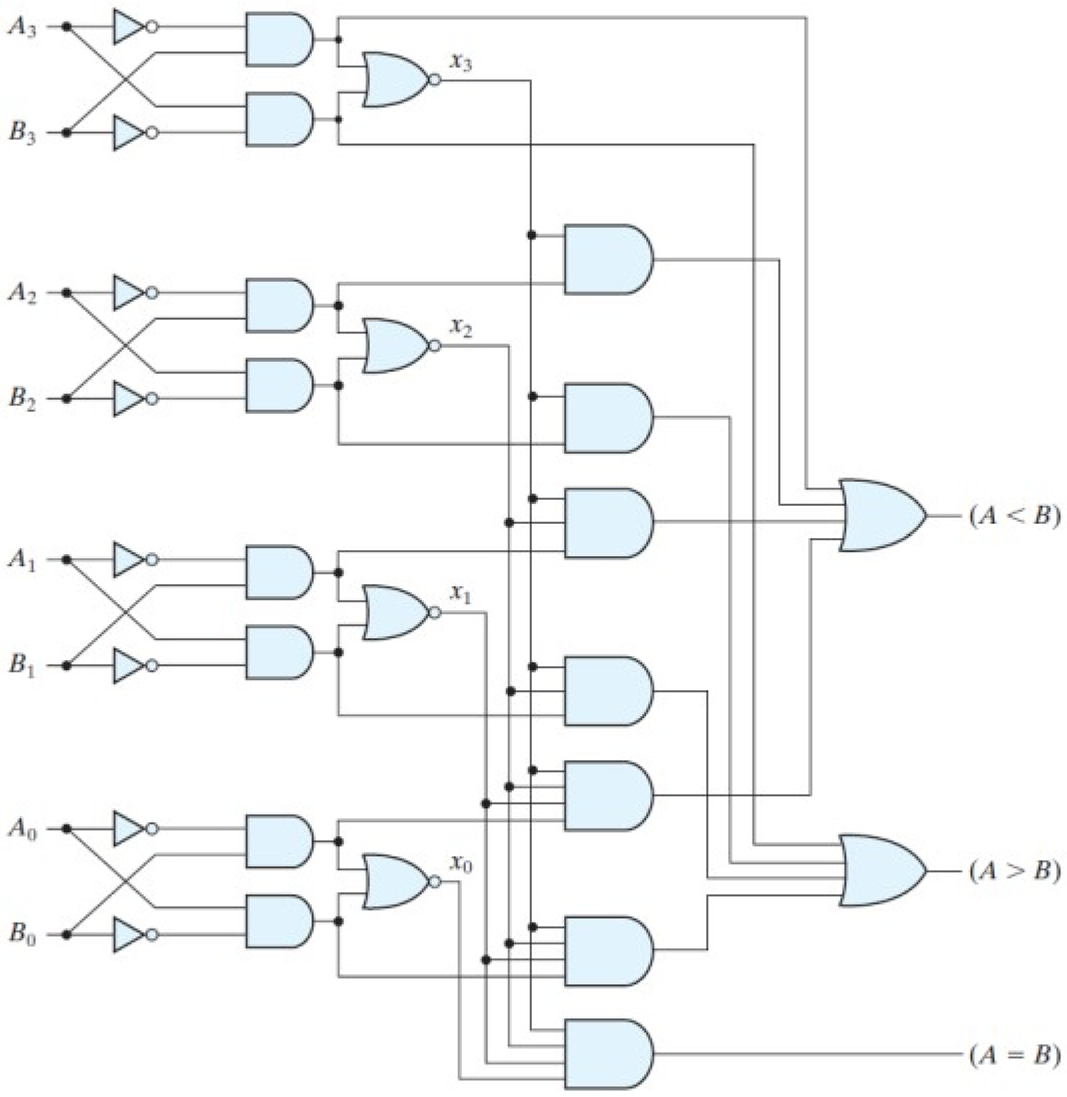
Adder operation is **A3A2A1A1+ B3B2B1B0**
Subtractor operation is **A3A2A1A1- B3B2B1B0**

4 bit adder-subtractor:



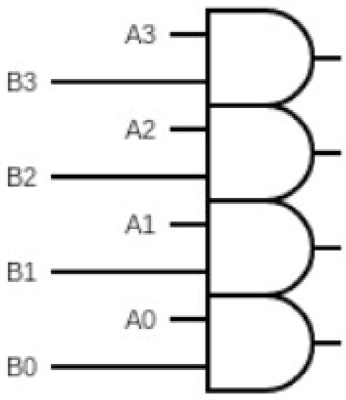
3. COMPARATOR :

This block would compare our 4-Bit number and give result whether A3A2A1A1 is greater than or less than or equal to B3B2B1B0.



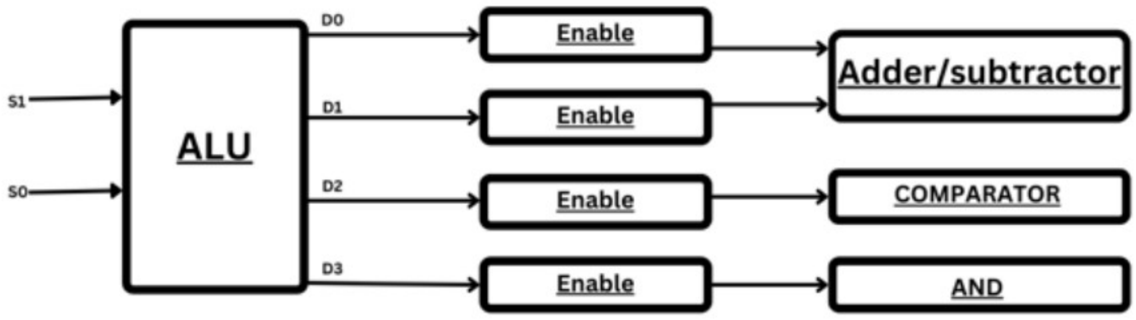
4 . AND BLOCK

This block Performs AND operation on A3&B3; A0&B0; A1&B1; A0&B0.



Combining all these blocks would give us our ALU .

The final design included would be as follows :



We make two major optimisations in our implementation :

1. Instead of using 4 enable blocks , we use only 3 i.e. We use only one enable block for both adder and subtractor . We do this by **OR** - ing the outputs D0 and D1 and passing that as an input to the enable block (OR will always be 1 irrespective of adder / subtractor)

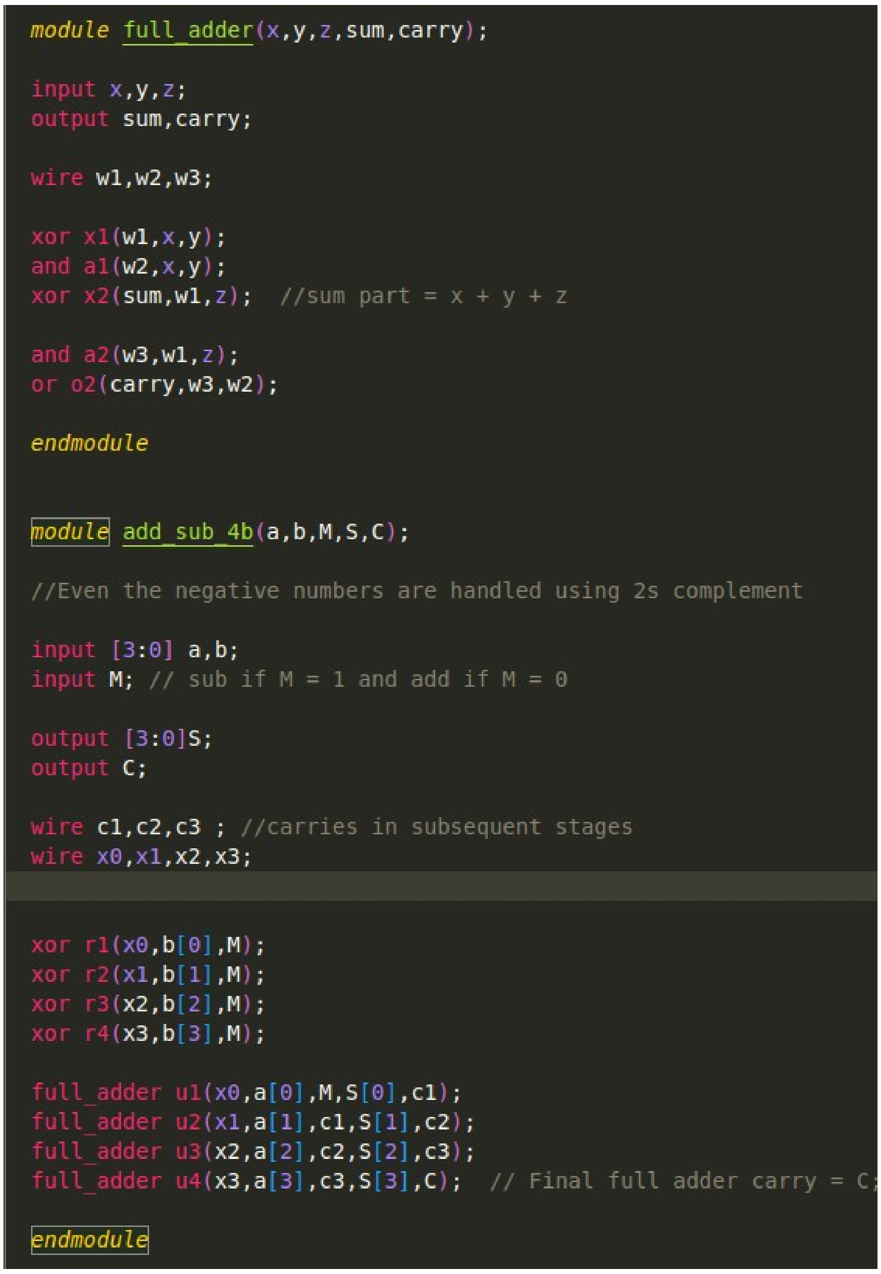
2 . Instead of implementing adder and subtractor block separately , we combine them by sending S1 as M (M =1 for subtractor and M = 0 for adder) .

VERILOG

We test our validity of the circuit by implementing it first in verilog

Here we define modules to indicate the blocks

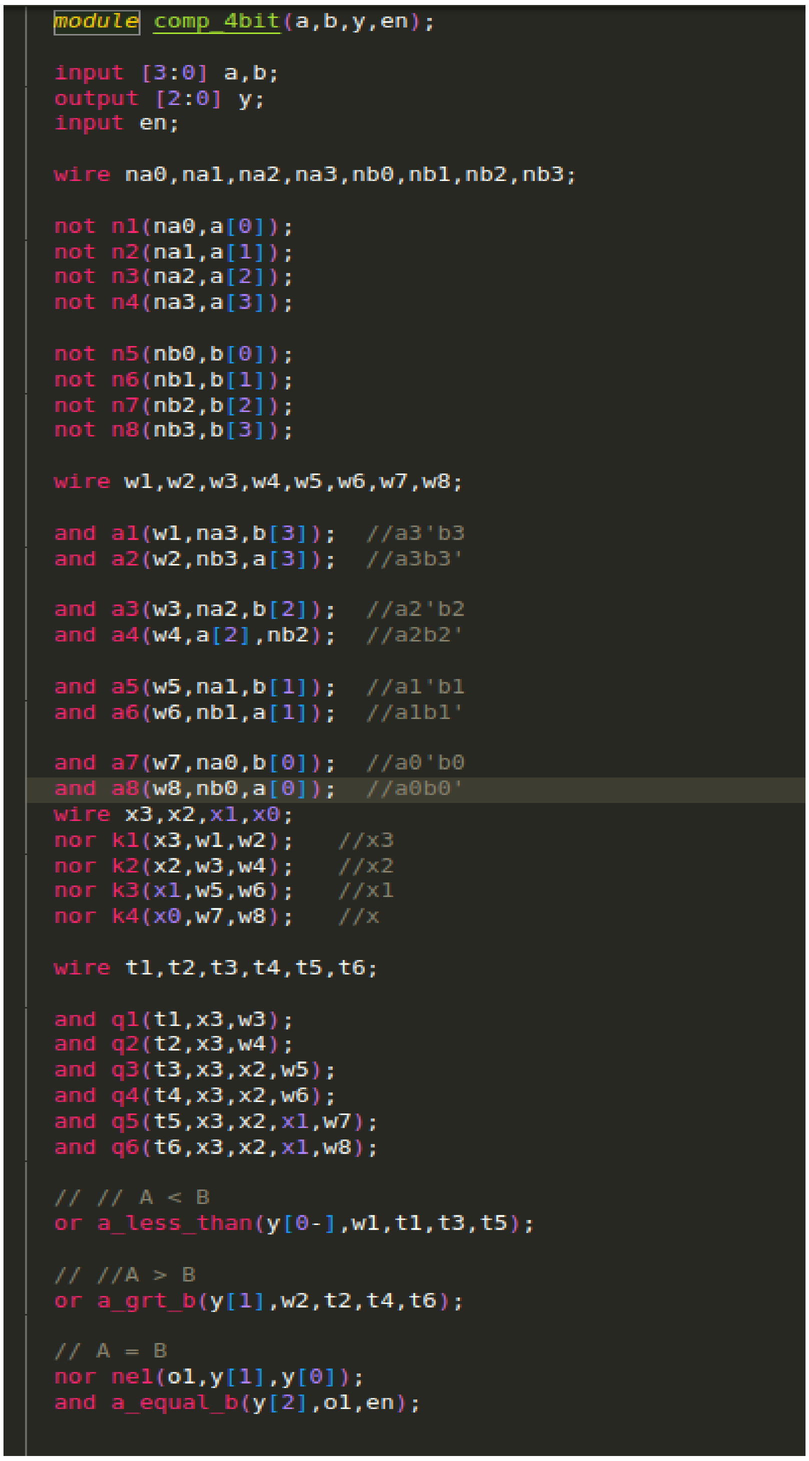
1) Adder – Subtractor



```
module full_adder(x,y,z,sum,carry);

input x,y,z;
output sum,carry;

wire w1,w2,w3;

xor x1(w1,x,y);
and a1(w2,x,y);
xor x2(sum,w1,z);  //sum part = x + y + z

and a2(w3,w1,z);
or o2(carry,w3,w2);

endmodule


module add_sub_4b(a,b,M,S,C);

//Even the negative numbers are handled using 2s complement

input [3:0] a,b;
input M; // sub if M = 1 and add if M = 0

output [3:0]S;
output C;

wire c1,c2,c3 ; //carries in subsequent stages
wire x0,x1,x2,x3;


xor r1(x0,b[0],M);
xor r2(x1,b[1],M);
xor r3(x2,b[2],M);
xor r4(x3,b[3],M);

full_adder u1(x0,a[0],M,S[0],c1);
full_adder u2(x1,a[1],c1,S[1],c2);
full_adder u3(x2,a[2],c2,S[2],c3);
full_adder u4(x3,a[3],c3,S[3],C);  // Final full adder carry = C;

endmodule
```

Initially we
implement the full adder block and call the same in the adder – subtractor block module

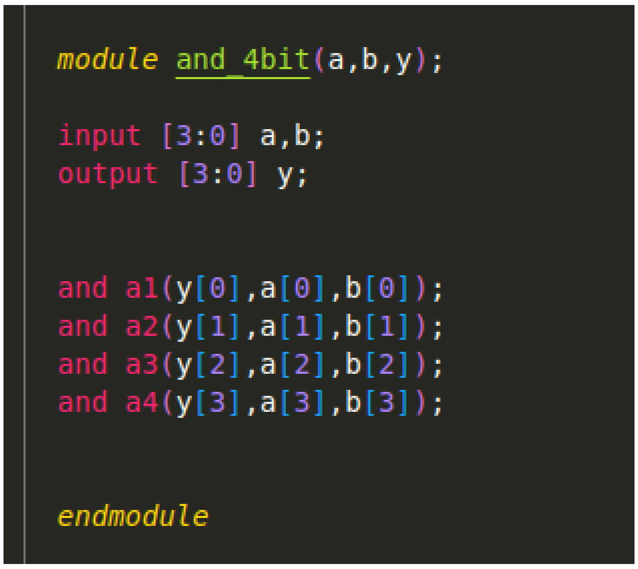
2. Comparator block



```
module comp_4bit(a,b,y,en);

input [3:0] a,b;
output [2:0] y;
input en;

wire na0,na1,na2,na3,nb0,nb1,nb2,nb3;

not n1(na0,a[0]);
not n2(na1,a[1]);
not n3(na2,a[2]);
not n4(na3,a[3]);

not n5(nb0,b[0]);
not n6(nb1,b[1]);
not n7(nb2,b[2]);
not n8(nb3,b[3]);

wire w1,w2,w3,w4,w5,w6,w7,w8;

and a1(w1,na3,b[3]);  //a3'b3
and a2(w2,nb3,a[3]);  //a3b3'

and a3(w3,na2,b[2]);  //a2'b2
and a4(w4,a[2],nb2);  //a2b2'

and a5(w5,na1,b[1]);  //a1'b1
and a6(w6,nb1,a[1]);  //a1b1'

and a7(w7,na0,b[0]);  //a0'b0
and a8(w8,nb0,a[0]);  //a0b0'
wire x3,x2,x1,x0;
nor k1(x3,w1,w2);   //x3
nor k2(x2,w3,w4);   //x2
nor k3(x1,w5,w6);   //x1
nor k4(x0,w7,w8);   //x

wire t1,t2,t3,t4,t5,t6;

and q1(t1,x3,w3);
and q2(t2,x3,w4);
and q3(t3,x3,x2,w5);
and q4(t4,x3,x2,w6);
and q5(t5,x3,x2,x1,w7);
and q6(t6,x3,x2,x1,w8);

// // A < B
or a_less_than(y[0-],w1,t1,t3,t5);

// //A > B
or a_grt_b(y[1],w2,t2,t4,t6);

// A = B
nor ne1(o1,y[1],y[0]);
and a_equal_b(y[2],o1,en);
```

Comparator block is made just by using the basic gates .

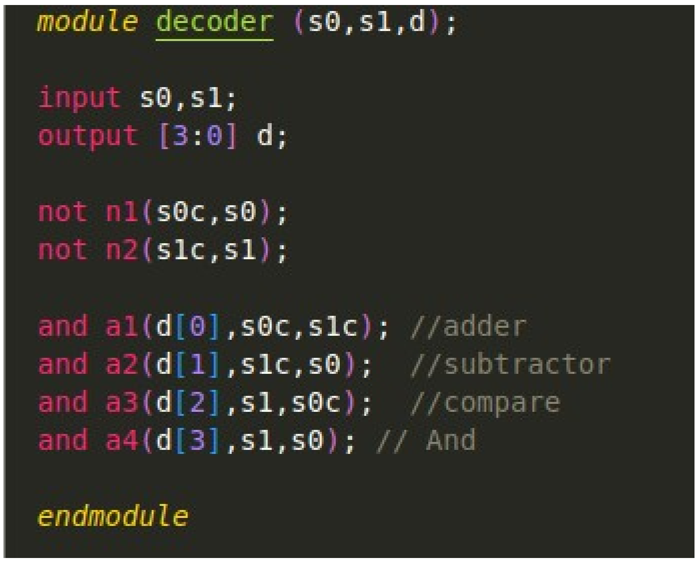
3 . And Block



```
module and_4bit(a,b,y);

input [3:0] a,b;
output [3:0] y;


and a1(y[0],a[0],b[0]);
and a2(y[1],a[1],b[1]);
and a3(y[2],a[2],b[2]);
and a4(y[3],a[3],b[3]);


endmodule
```

And block is just anding the inputs bit – wise .The enable block is also implemented the same way as the and block using 4 and gates .

4. Decoder

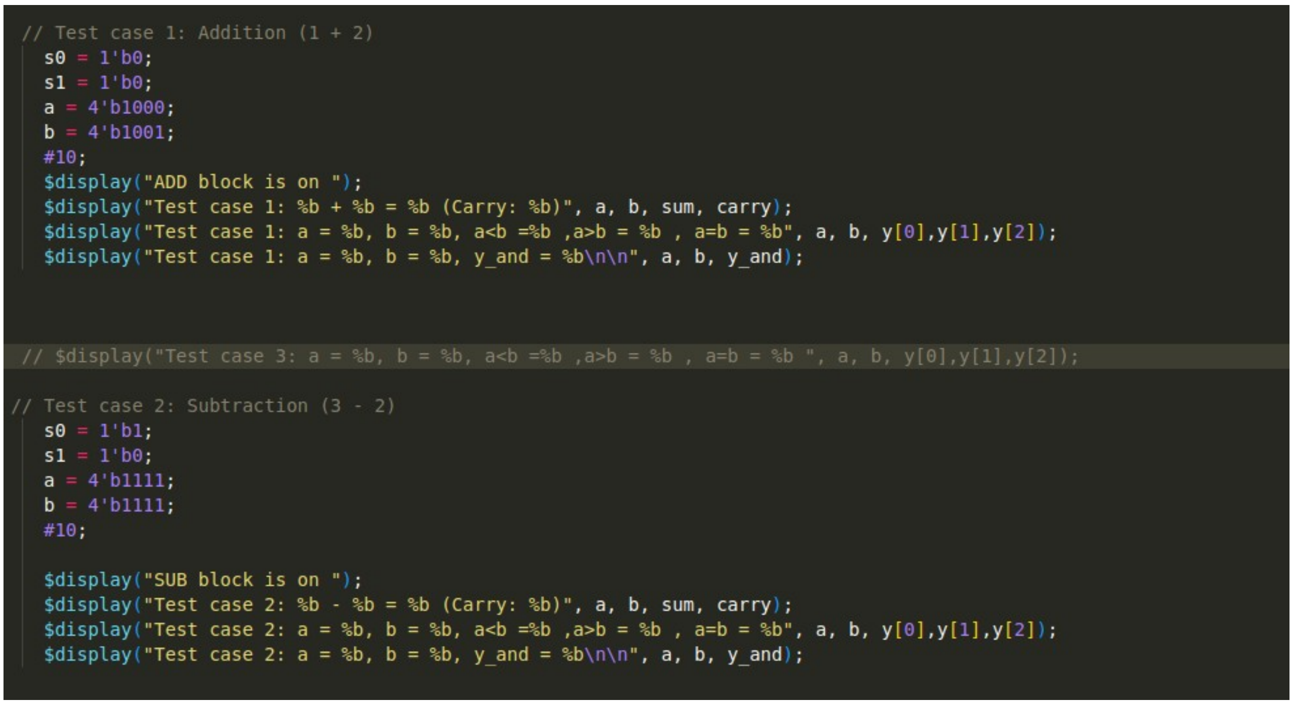


```
module decoder (s0,s1,d);

input s0,s1;
output [3:0] d;

not n1(s0c,s0);
not n2(s1c,s1);

and a1(d[0],s0c,s1c); //adder
and a2(d[1],s1c,s0);  //subtractor
and a3(d[2],s1,s0c);  //compare
and a4(d[3],s1,s0); // And

endmodule
```

Decoder takes in S0,S1 as inputs and returns D0 , D1 , D2 , D3 using the same circuit diagram mentioned earlier .

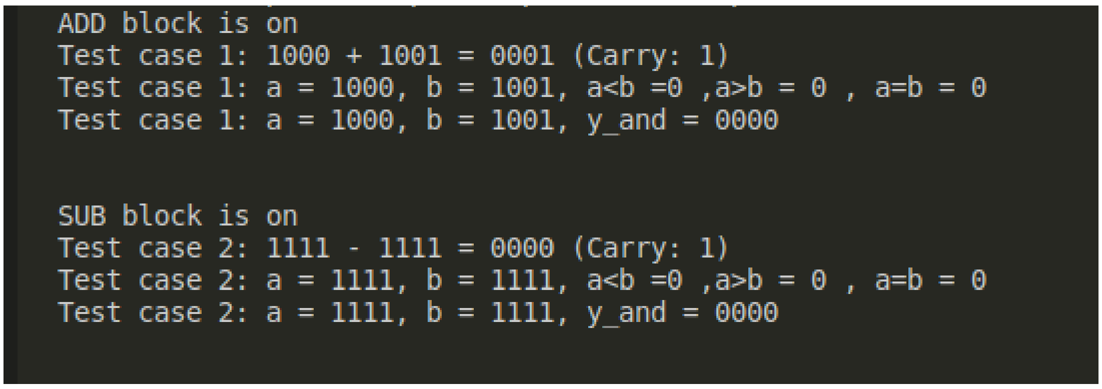
Testing the circuit in VERILOG

1 . INPUT : ADDER – SUBTRACTOR BLOCK



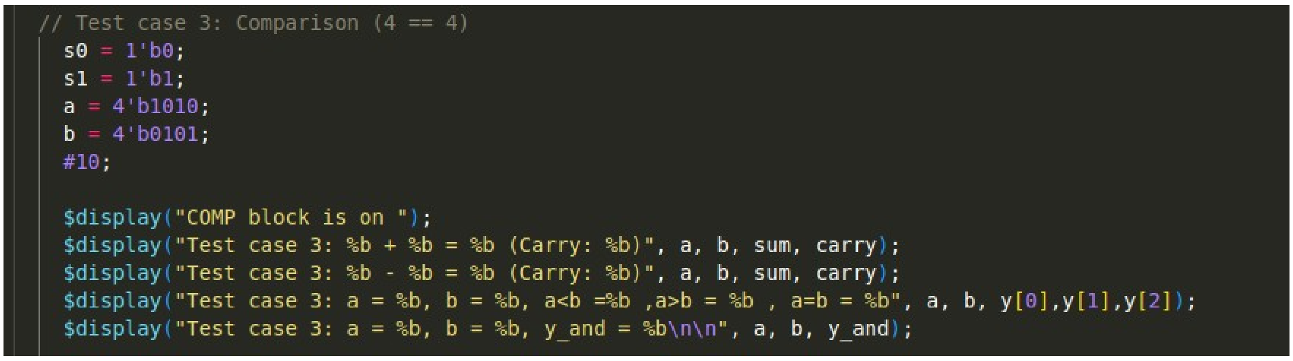
```
// Test case 1: Addition (1 + 2)
   s0 = 1'b0;
   s1 = 1'b0;
   a = 4'b1000;
   b = 4'b1001;
   #10;
   $display("ADD block is on ");
   $display("Test case 1: %b + %b = %b (Carry: %b)", a, b, sum, carry);
   $display("Test case 1: a = %b, b = %b, a<b =%b ,a>b = %b , a=b = %b", a, b, y[0],y[1],y[2]);
   $display("Test case 1: a = %b, b = %b, y_and = %b\n\n", a, b, y_and);



// $display("Test case 3: a = %b, b = %b, a<b =%b ,a>b = %b , a=b = %b ", a, b, y[0],y[1],y[2]);

// Test case 2: Subtraction (3 - 2)
   s0 = 1'b1;
   s1 = 1'b0;
   a = 4'b1111;
   b = 4'b1111;
   #10;

   $display("SUB block is on ");
   $display("Test case 2: %b - %b = %b (Carry: %b)", a, b, sum, carry);
   $display("Test case 2: a = %b, b = %b, a<b =%b ,a>b = %b , a=b = %b", a, b, y[0],y[1],y[2]);
   $display("Test case 2: a = %b, b = %b, y_and = %b\n\n", a, b, y_and);
```

OUTPUT : ADDER – SUBTRACTOR BLOCK



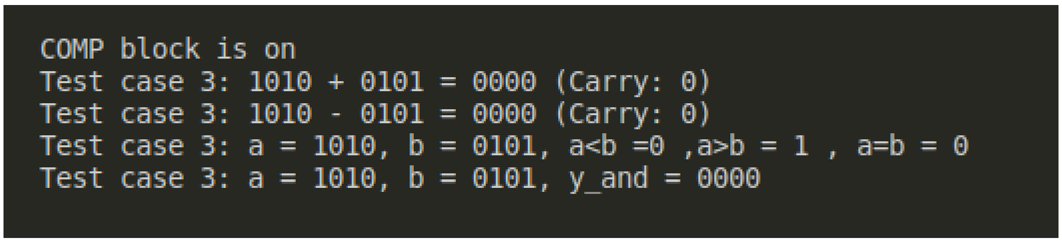
```
ADD block is on
Test case 1: 1000 + 1001 = 0001 (Carry: 1)
Test case 1: a = 1000, b = 1001, a<b =0 ,a>b = 0 , a=b = 0
Test case 1: a = 1000, b = 1001, y_and = 0000


SUB block is on
Test case 2: 1111 - 1111 = 0000 (Carry: 1)
Test case 2: a = 1111, b = 1111, a<b =0 ,a>b = 0 , a=b = 0
Test case 2: a = 1111, b = 1111, y_and = 0000
```

2. INPUT – COMPARATOR



```
// Test case 3: Comparison (4 == 4)
 s0 = 1'b0;
 s1 = 1'b1;
 a = 4'b1010;
 b = 4'b0101;
 #10;

 $display("COMP block is on ");
 $display("Test case 3: %b + %b = %b (Carry: %b)", a, b, sum, carry);
 $display("Test case 3: %b - %b = %b (Carry: %b)", a, b, sum, carry);
 $display("Test case 3: a = %b, b = %b, a<b =%b ,a>b = %b , a=b = %b", a, b, y[0],y[1],y[2]);
 $display("Test case 3: a = %b, b = %b, y_and = %b\n\n", a, b, y_and);
```

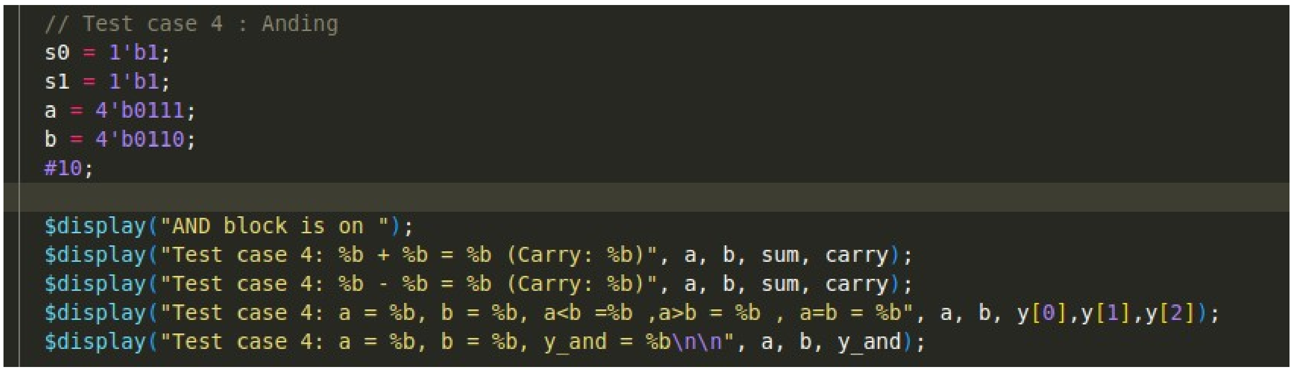
OUTPUT – COMPARATOR



```
COMP block is on
Test case 3: 1010 + 0101 = 0000 (Carry: 0)
Test case 3: 1010 - 0101 = 0000 (Carry: 0)
Test case 3: a = 1010, b = 0101, a<b =0 ,a>b = 1 , a=b = 0
Test case 3: a = 1010, b = 0101, y_and = 0000
```

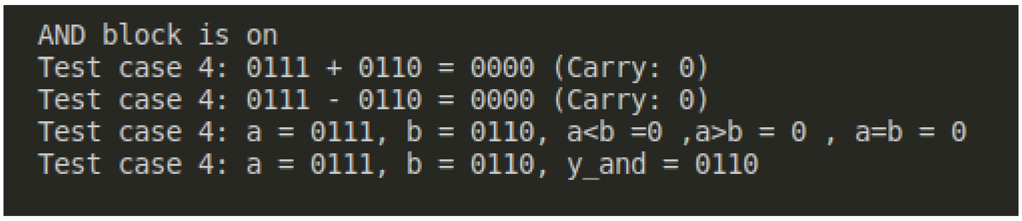
From the test case clearly A = 12 and B =5 , so A > B must satisfy and hence its ON whereas the rest of the ouputs are OFF

3. INPUT – ANDING



```
// Test case 4 : Anding
s0 = 1'b1;
s1 = 1'b1;
a = 4'b0111;
b = 4'b0110;
#10;

$display("AND block is on ");
$display("Test case 4: %b + %b = %b (Carry: %b)", a, b, sum, carry);
$display("Test case 4: %b - %b = %b (Carry: %b)", a, b, sum, carry);
$display("Test case 4: a = %b, b = %b, a<b =%b ,a>b = %b , a=b = %b", a, b, y[0],y[1],y[2]);
$display("Test case 4: a = %b, b = %b, y_and = %b\n\n", a, b, y_and);
```

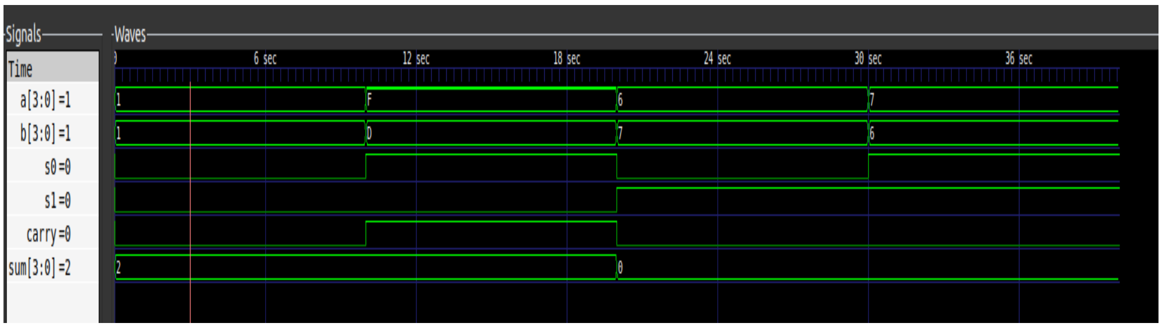
OUTPUT – ANDING



```
AND block is on
Test case 4: 0111 + 0110 = 0000 (Carry: 0)
Test case 4: 0111 - 0110 = 0000 (Carry: 0)
Test case 4: a = 0111, b = 0110, a<b =0 ,a>b = 0 , a=b = 0
Test case 4: a = 0111, b = 0110, y_and = 0110
```

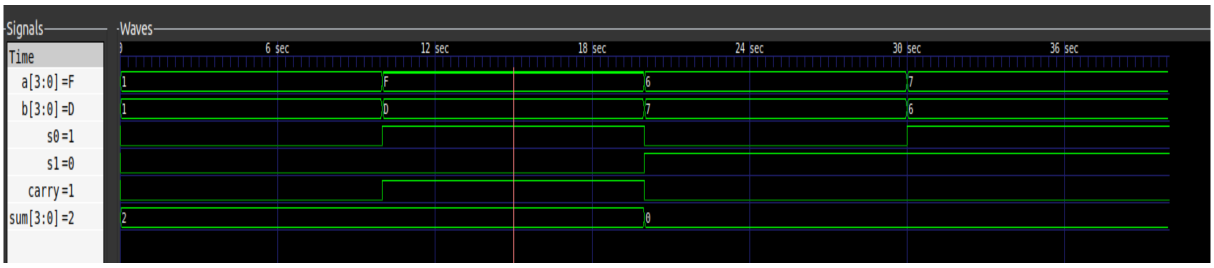
To visualise better , we plot the outputs in GTKWAVE and observe :

1. Adder Block



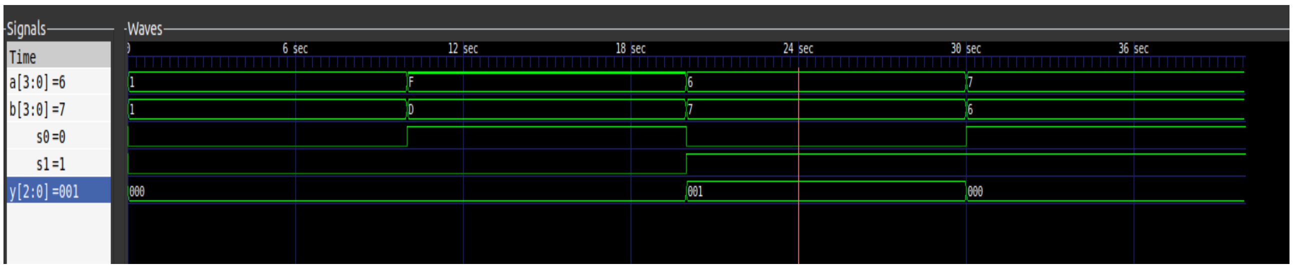
We see that till 10 secs , A = 1 and B = 1 and the corresponding output is **SUM = 2 and carry = 0** (indicated by the red vertical line) .

2 . Subtractor Block



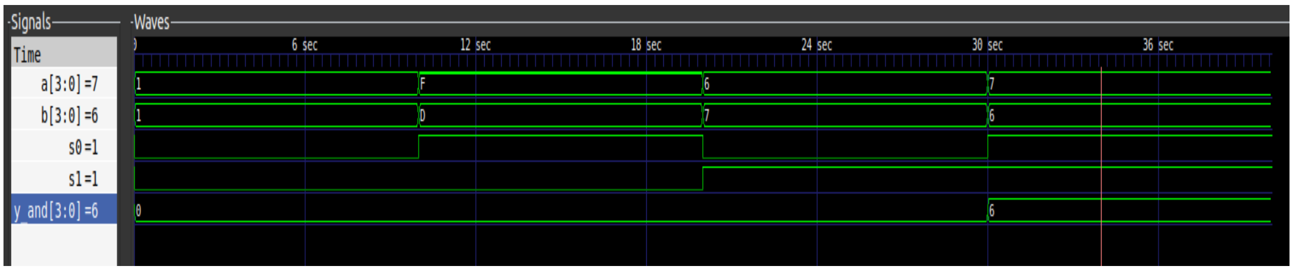
We observe that from 10 - 20 secs , A = 15 (F) and B = 13(D) and the corresponding output is **SUM = 2 and carry = 1** (indicated by the red vertical line) .

3 . Comparator Block



From 20 to 30 seconds , comparator block is ON . A = 6 and B = 7 , so A < B and hence **y0 = 1** and rest are all 0 .

4 . Anding Block



From 30 to 40 seconds , and - block is ON . A = 7 (0111) and B = 6 (0110) , so **y = 0110 (6)** and we can observe the same from the above plot .

AREAS OF CONCERN :

1. PROBLEM ENCOUNTERED :

When the comparator block is OFF , the input received to it is A = 0000 and B = 0000 . In this case , we want y2 = 0 (the equal to case) because the comparator block is OFF . But our logic we will get 1 as they satisfy the equal to condition .

SOLUTION :

To print y2 = 0 , we **AND** the output y2 with the enable D2 . So y2 will 1 only if the comparator block is enabled and the equal to condition is satisfied . This fixes the problem we encountered .

2 . Make sure the arguments passed for each modules are correct especially in cases where we pass the decoder output as enable .

NGSPICE – CIRCUIT DESIGN

Now , we implement the circuit in transistor level before moving on to implementing it in magic layout .

STEPS TO FOLLOW :

1. We first define the NAND gate in transistor level i.e, using PMOS and NMOS and their parameters (source area , source perimeter , drain area , drain perimeter) .
2. Then , we make the other gates by calling NAND gate using subcircuit .
3. Once all the gates are defined in terms of NAND , then all the blocks can be implemented just as in case of verilog by calling them using subcircuits .

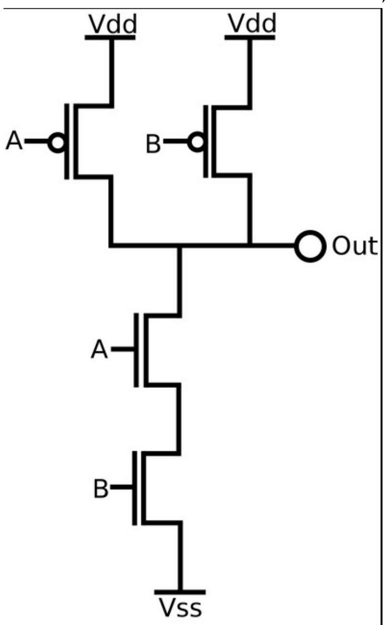
BUILDING THE BASIC GATES :

1. NAND GATE

NAND gate is made in transistor level with well – defined parameters as follows :

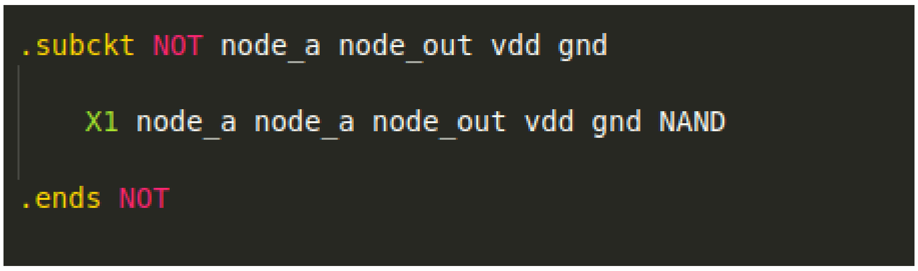
```
.subckt NAND node_a node_b node_out vdd gnd

    Mn1 node_out node_a node_y gnd CMOSN W = {wn1} L = {ln1}
    + AS = {5*wn1*LAMBDA} PS = {10*LAMBDA + 2*wn1} AD = {5*wn1*LAMBDA} PD = {10*LAMBDA + 2*wn1}

    Mn2 node_y node_b gnd gnd CMOSN W = {wn2} L = {ln2}
    + AS = {5*wn2*LAMBDA} PS = {10*LAMBDA + 2*wn2} AD = {5*wn2*LAMBDA} PD = {10*LAMBDA + 2*wn2}

    Mp1 node_out node_a vdd vdd CMOSP W = {wp1} L = {lp1}
    + AS = {5*wp1*LAMBDA} PS = {10*LAMBDA + 2*wp1} AD = {5*wp1*LAMBDA} PD = {10*LAMBDA + 2*wp1}

    Mp2 node_out node_b vdd vdd CMOSP W = {wp2} L = {lp2}
    + AS = {5*wp2*LAMBDA} PS = {10*LAMBDA + 2*wp2} AD = {5*wp2*LAMBDA} PD = {10*LAMBDA + 2*wp2}

.ends NAND
```

This is implemented using the below circuit :



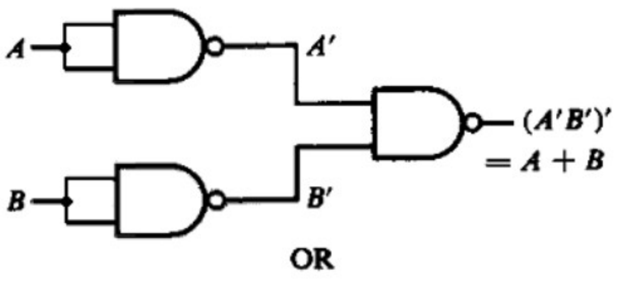
2. AND GATE

And gate is made by just shorting both the inputs of the NAND gate .

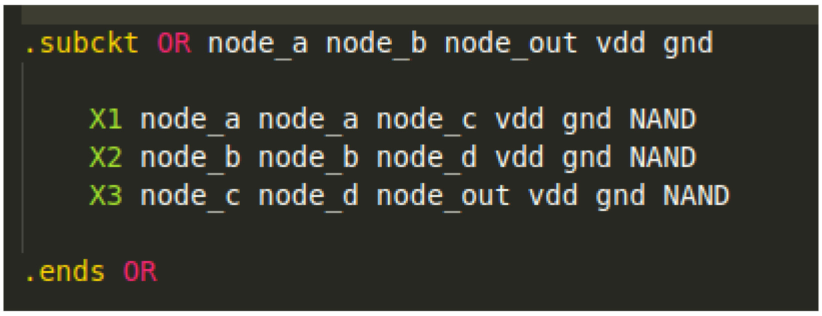


```
.subckt NOT node_a node_out vdd gnd

    X1 node_a node_a node_out vdd gnd NAND

.ends NOT
```

3.OR GATE

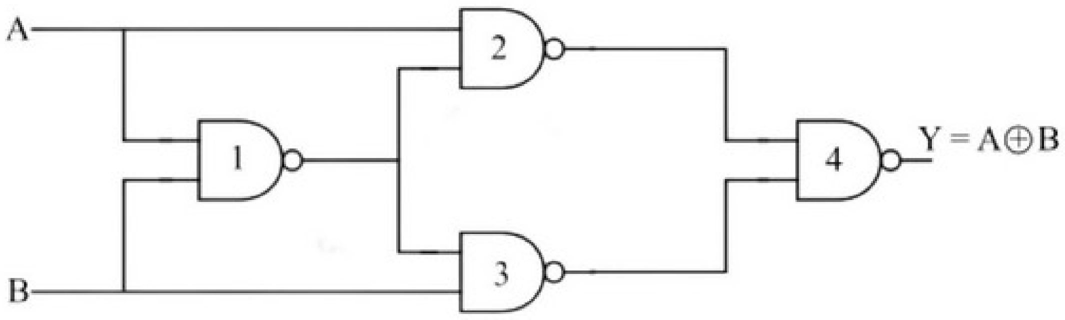


this is implemented in ngspice as shown :

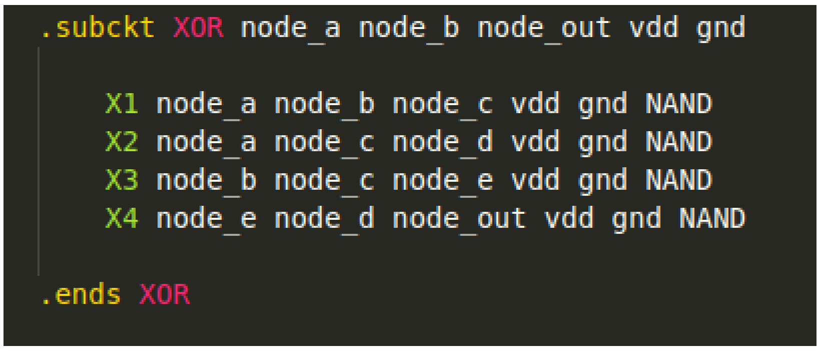


```
.subckt OR node_a node_b node_out vdd gnd

    X1 node_a node_a node_c vdd gnd NAND
    X2 node_b node_b node_d vdd gnd NAND
    X3 node_c node_d node_out vdd gnd NAND

.ends OR
```

4. XOR GATE



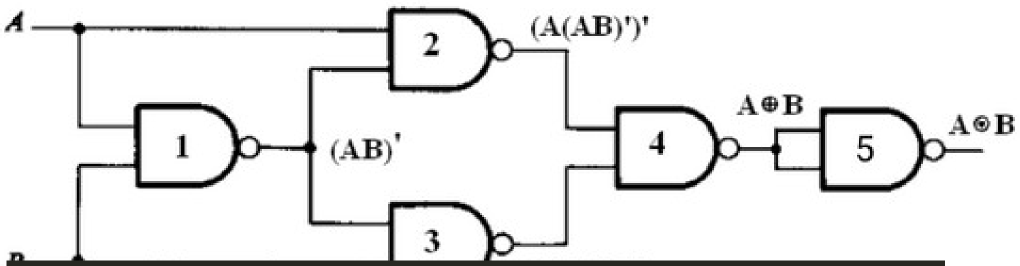
this is implemented in ngspice as shown :

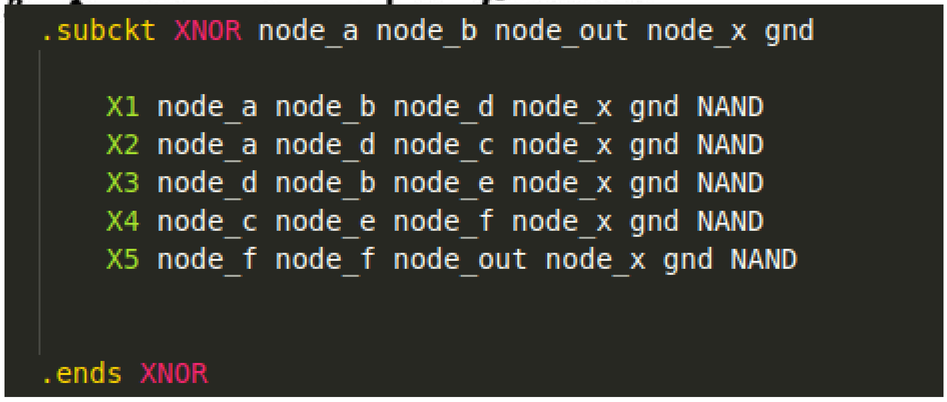


```
.subckt XOR node_a node_b node_out vdd gnd

    X1 node_a node_b node_c vdd gnd NAND
    X2 node_a node_c node_d vdd gnd NAND
    X3 node_b node_c node_e vdd gnd NAND
    X4 node_e node_d node_out vdd gnd NAND

.ends XOR
```

5. XNOR GATE

XNOR is made shorting the output of XOR gate to the NAND gate and hence we get the flipped XOR output = XNOR output .



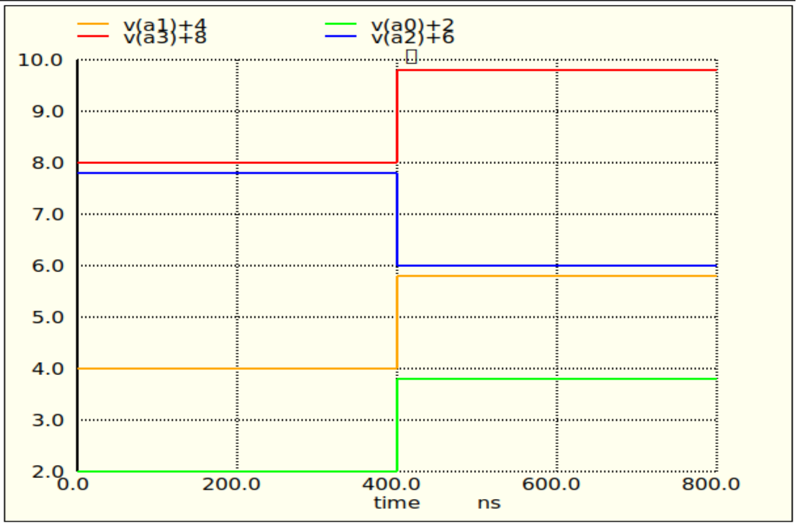


```
.subckt XNOR node_a node_b node_out node_x gnd

    X1 node_a node_b node_d node_x gnd NAND
    X2 node_a node_d node_c node_x gnd NAND
    X3 node_d node_b node_e node_x gnd NAND
    X4 node_c node_e node_f node_x gnd NAND
    X5 node_f node_f node_out node_x gnd NAND


.ends XNOR
```

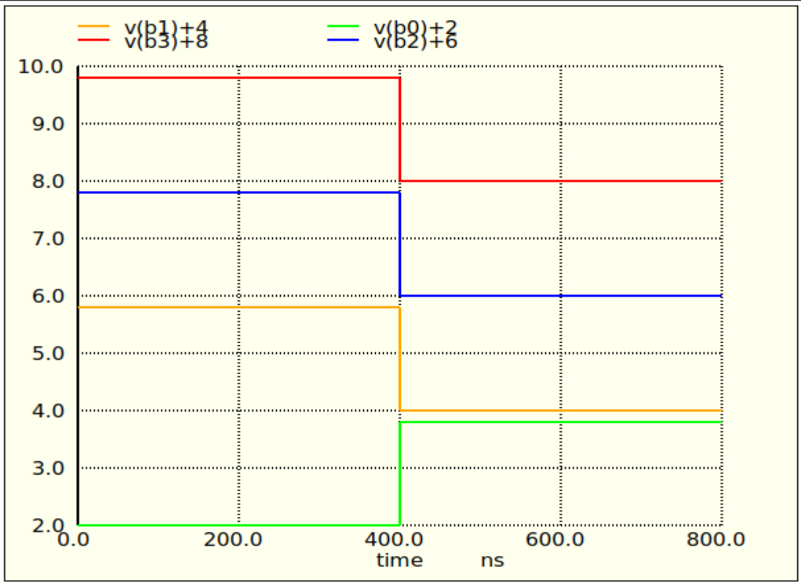
Testing the circuit – NGSPICE

Lets consider input A = A3A2A1A0 and input B = B3B2B1B0 as follows :

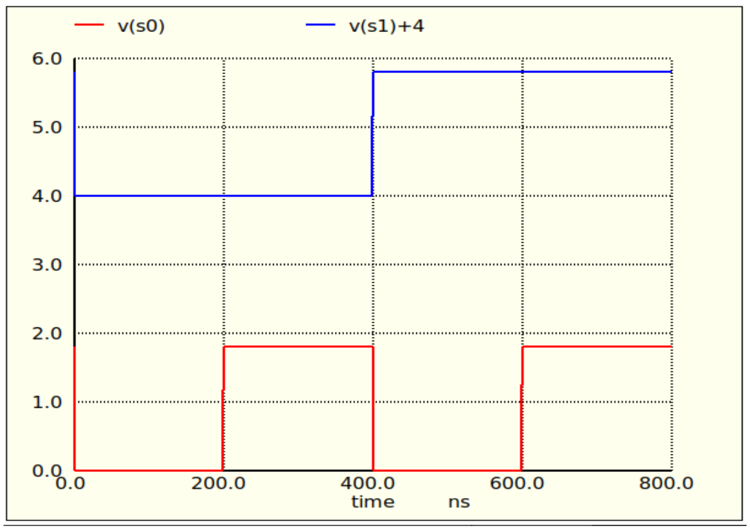
INPUT A :



INPUT B :

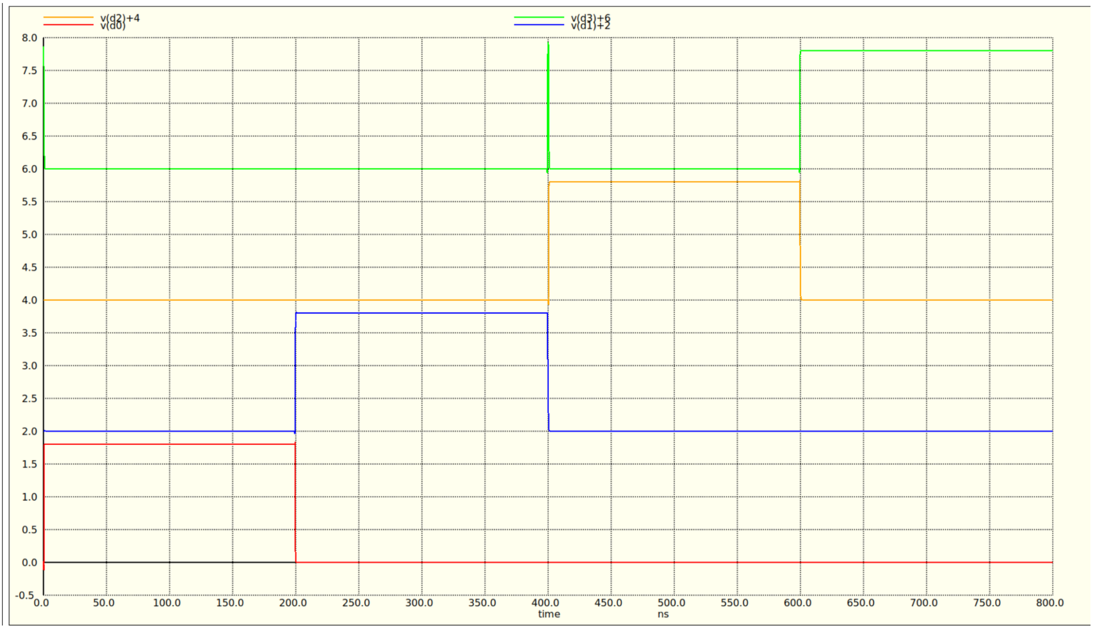


SELECT LINES :



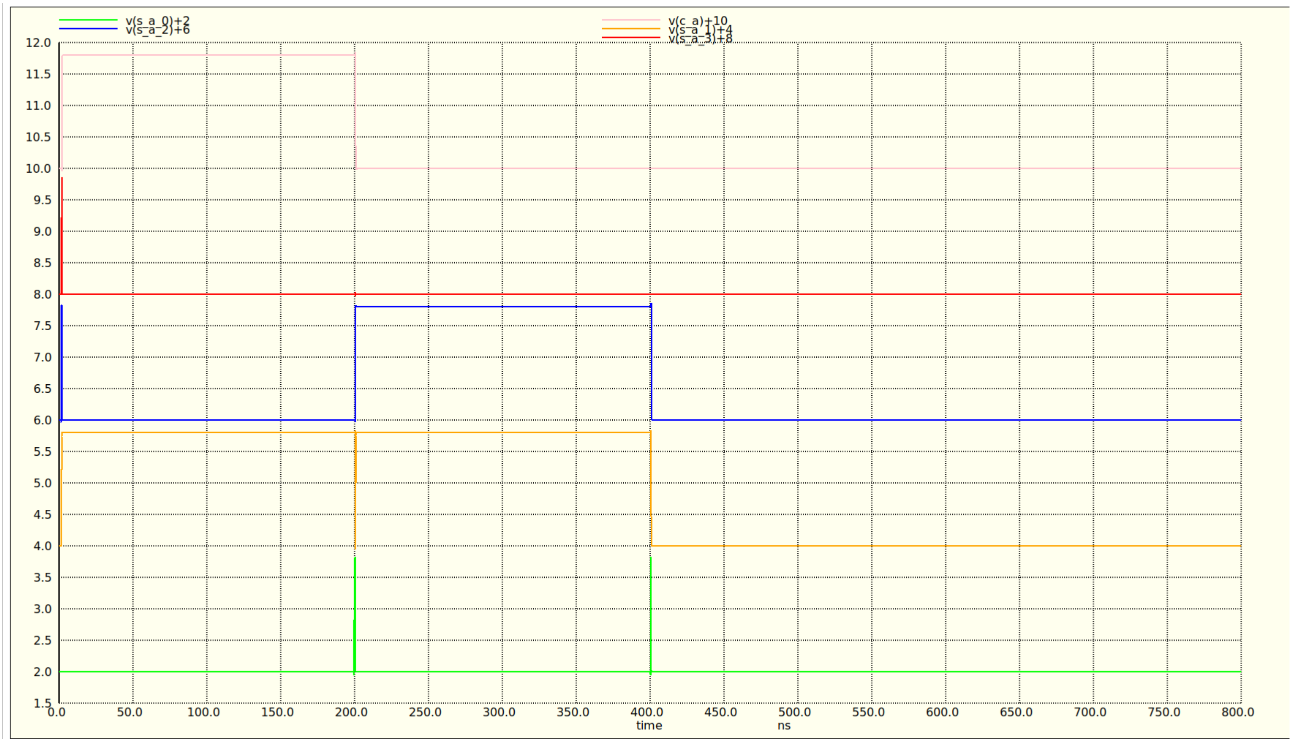
1. For the first 200ns , both S0 and S1 are 0 -> ADDER BLOCK is enabled
2. For the next 200ns , S0 = 1 and S1 = 0 -> SUBTRACTOR BLOCK is enabled .
3. For the next 200ns , S0 = 0 and S1 = 1 -> COMPARATOR BLOCK is enabled
4. For the next 200ns , S0 = 1 and S1 = 1 -> AND BLOCK is enabled .

DECODER :



We see that each range of 200ns only one output is high and the corresponding block will be enabled .

ADDER – SUBTRACTOR :

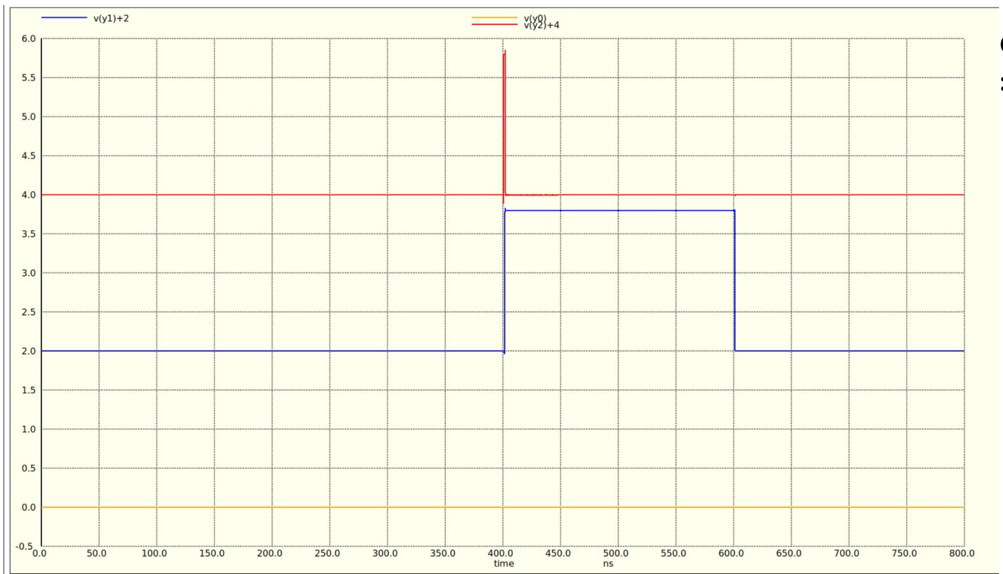


1. For the first 200ns only adder is ON , in this case A = 0100 and B = 1110 and sum = 0010 and carry =1 which is the exact output we obtained as depicted above .

2.For the next 200ns only subtractor is ON , in this case A = 0100 and B = 1110 and sum = 0110 and carry = 0 which is the exact output we obtained as depicted above .

3 . From 400 – 800 ns , both adder and subtractor are OFF and hence the outputs are all 0 .

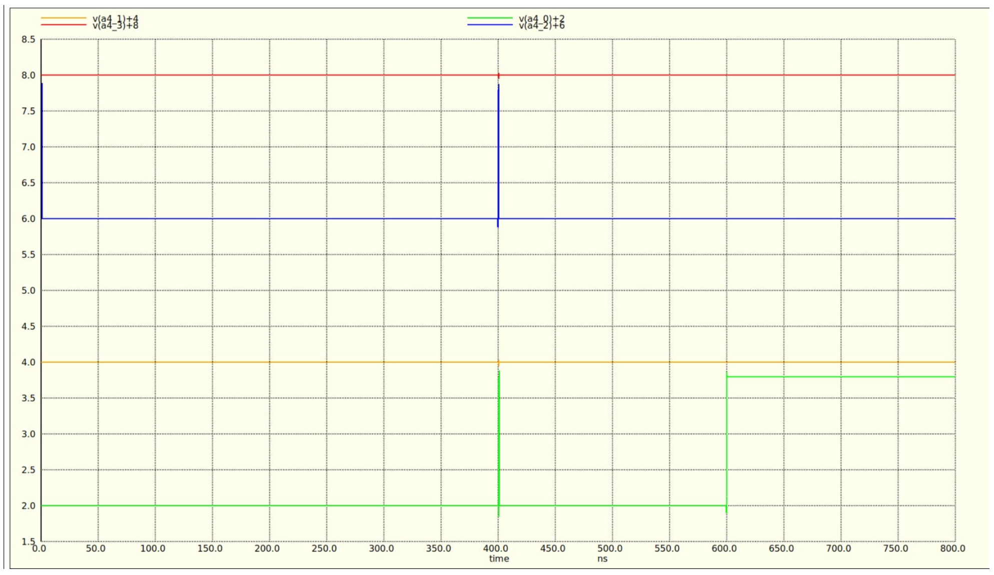
COMPARATOR :



1 .Comparator block is ON only between 400-600 ns and hence the outputs are all 0 , outside that range .

2. In 400 – 600 ns timescale , A = 1011 and B = 0001 and hence A > B so , only y1 is high .

AND BLOCK :



1. AND block is ON only between 600-800 ns and hence the outputs are all 0 , outside that range .

2. In 600 – 800 ns timescale , A = 1011 and B = 0001 and hence output = 0001 and same the depicted above .

AREAS OF CONCERN :

1) The fact that all the lines where the subcircuits are called must start with the letter "X" or "x" was very tricky .

2) Implementing the comparator subcircuit was a bit tedious as it involved a lot of gates and variables making it a bit hard to implement .

3) I kept setting the VDD supply to 1V instead of 1.8V and I kept getting irregular outputs as sometimes the transistor gets switched OFF when the supply was set to 1V .

3 . MAGIC – LAYOUT

Now that our circuit works on both logical (Verilog) and the transistor level (ngspice) , we can proceed to implement the circuit in magic layout

STEPS TO FOLLOW :

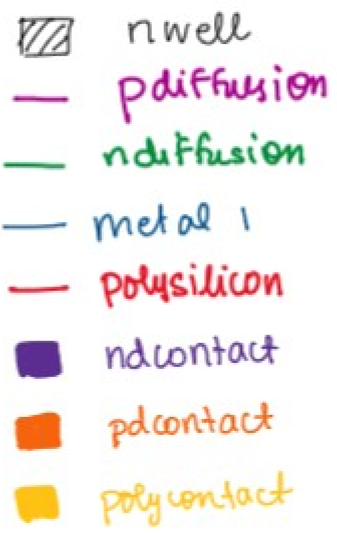
1. Note that default background in MAGIC is pwell . So to implement NMOS , we explicitly need to add nwell , whereas in case of PMOS we dont need to add pwell .

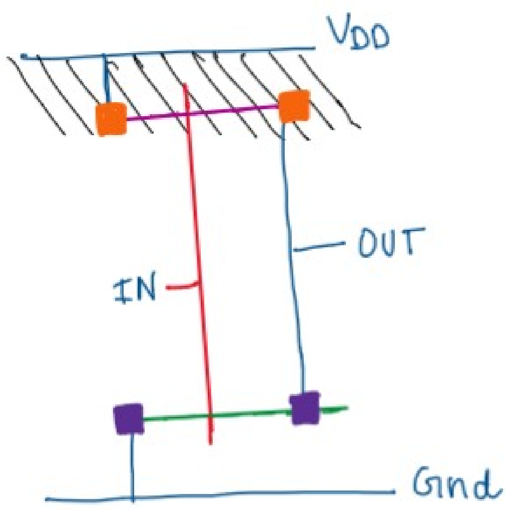
2. While extracting from different files , first copy the same file so that the extracted file can also be edited .i.e, changing labels becomes easier .

3. There are mutliples layers of metal in magic and they run in different layers. So to interconnect them ,we need to use very specific contacts .

STICK DIAGRAM :

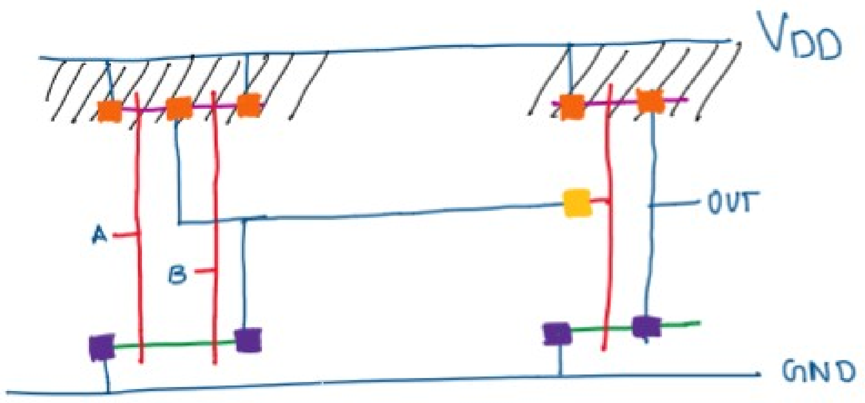
Consider the following colour schematic and we will use the same to construct our gates .



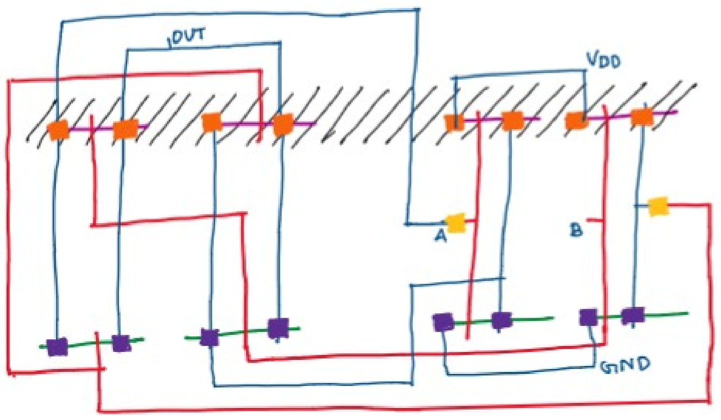


1. NOT GATE :

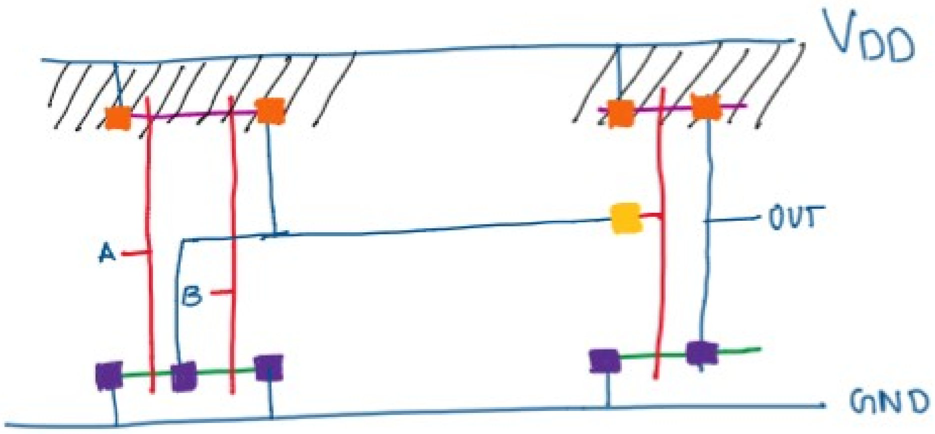
2. AND GATE :



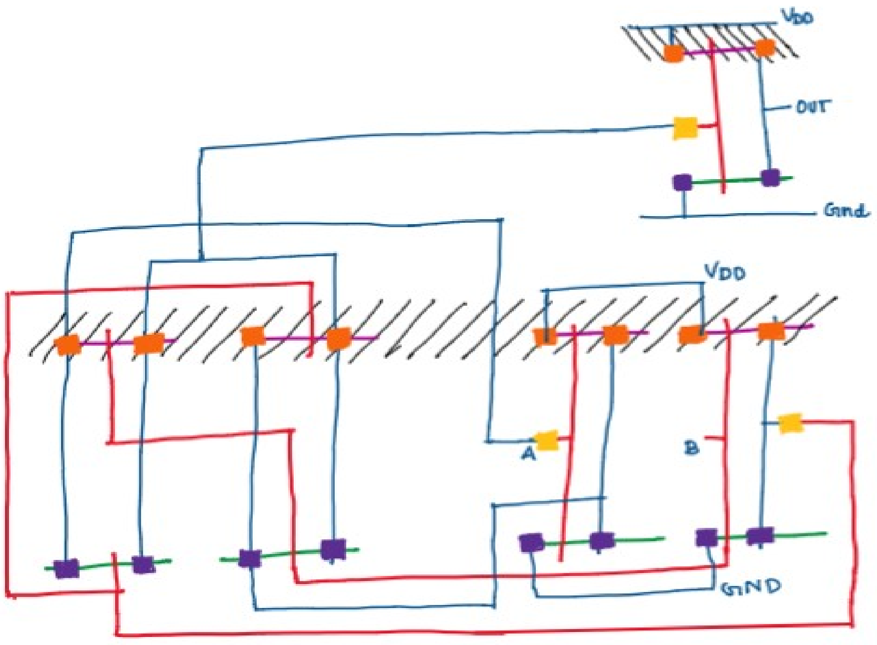
3. XOR GATE :



4. OR GATE :

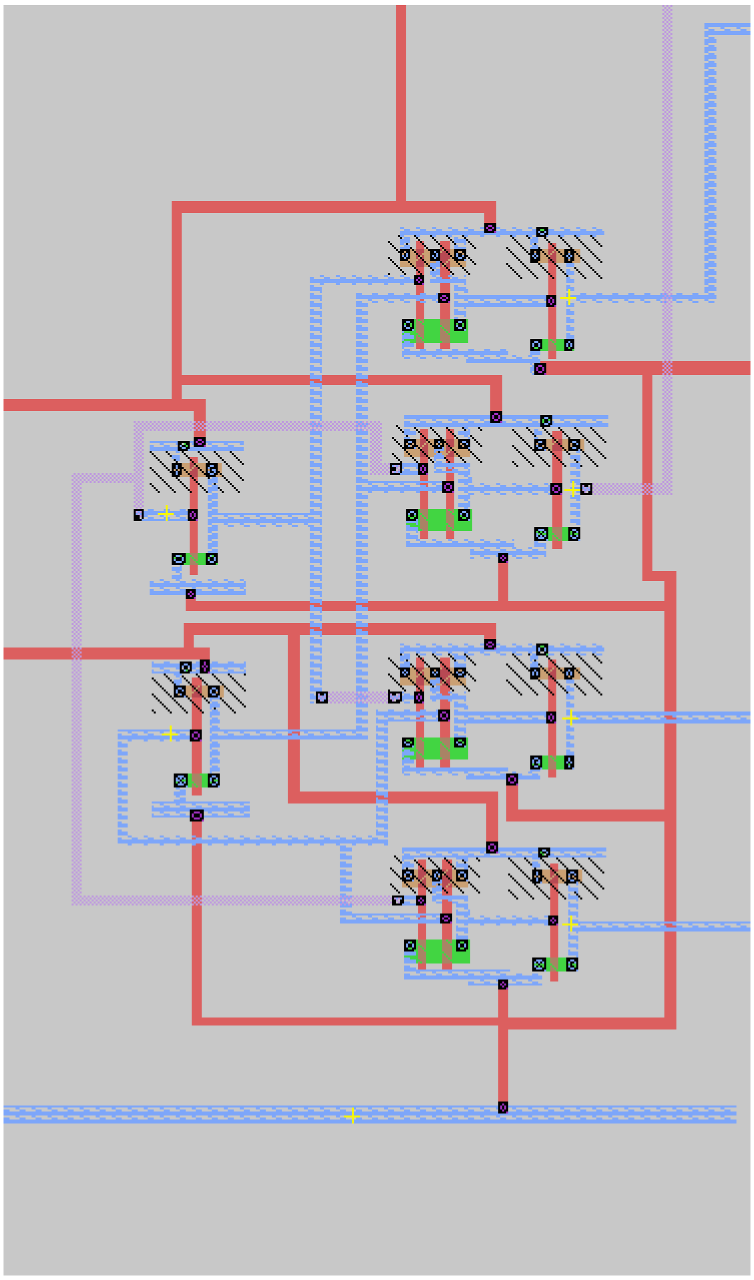


5. XNOR GATE :

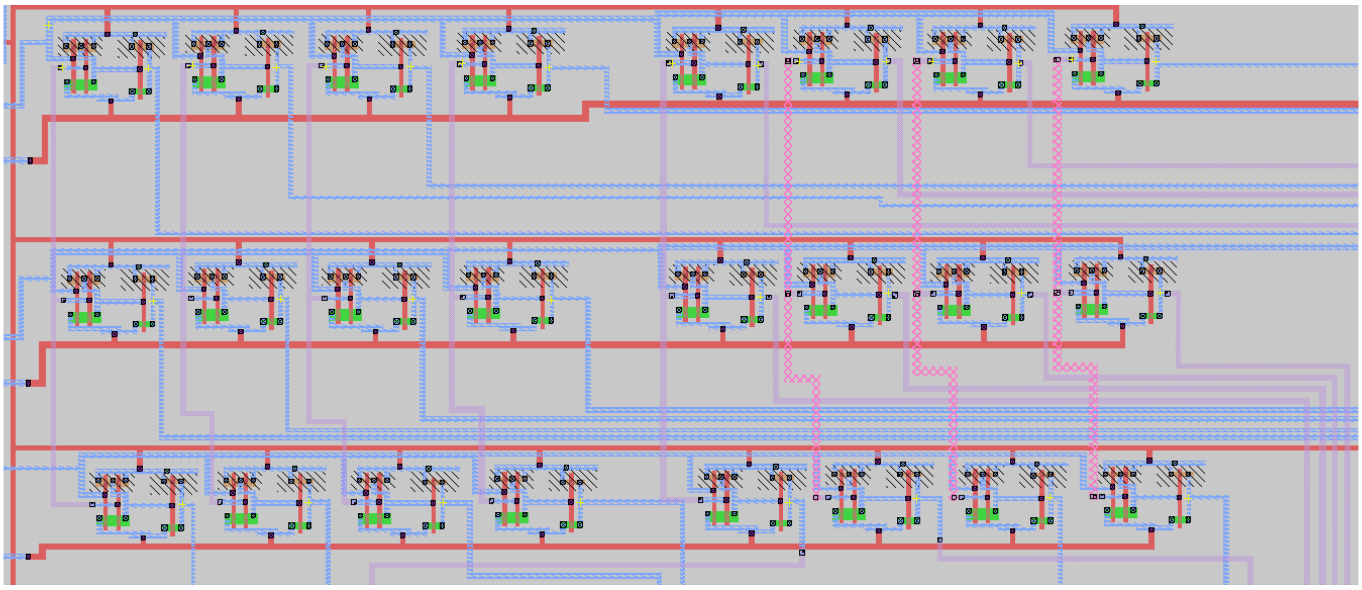


BUILDING THE BLOCKS :

1.DECODER BLOCK

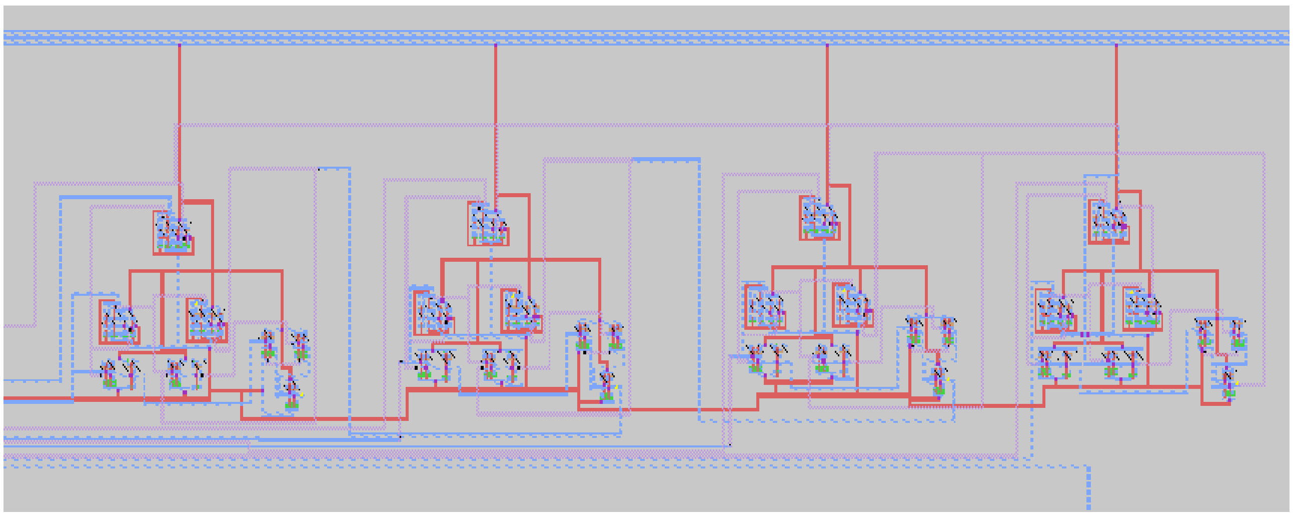


2. ENABLE BLOCKS :



1. The first line of and gates correspond to the inputs send the adder – subtractor .
2. The second line of and gates correspond to the inputs send the comparator block .
3. The third line of and gates correspond to the inputs send the and block .

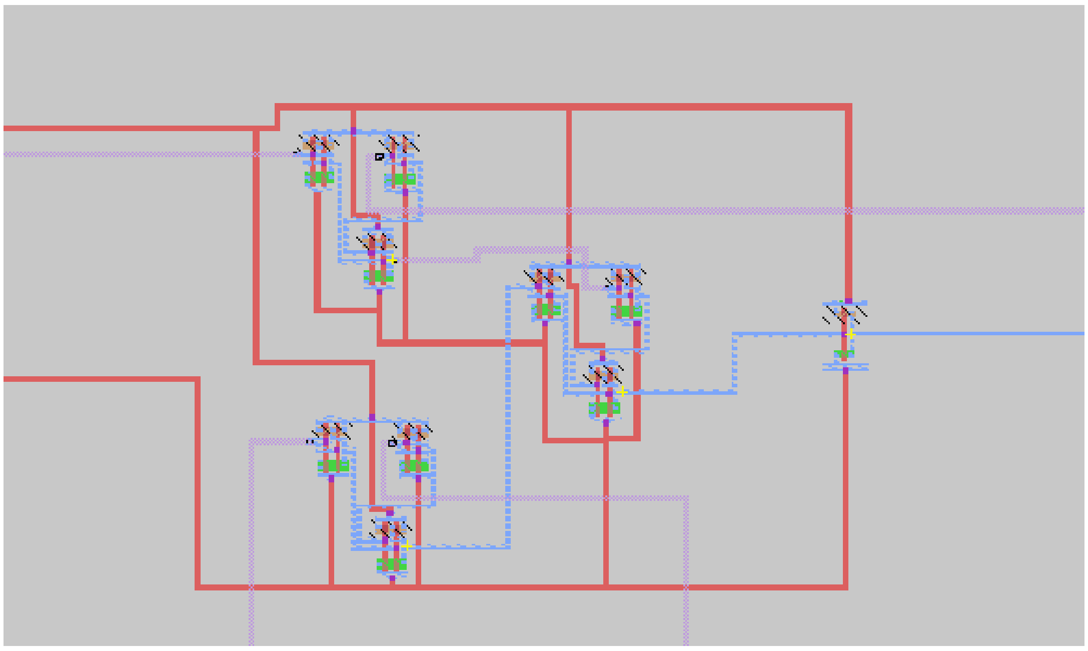
3. ADDER SUBTRACTOR :



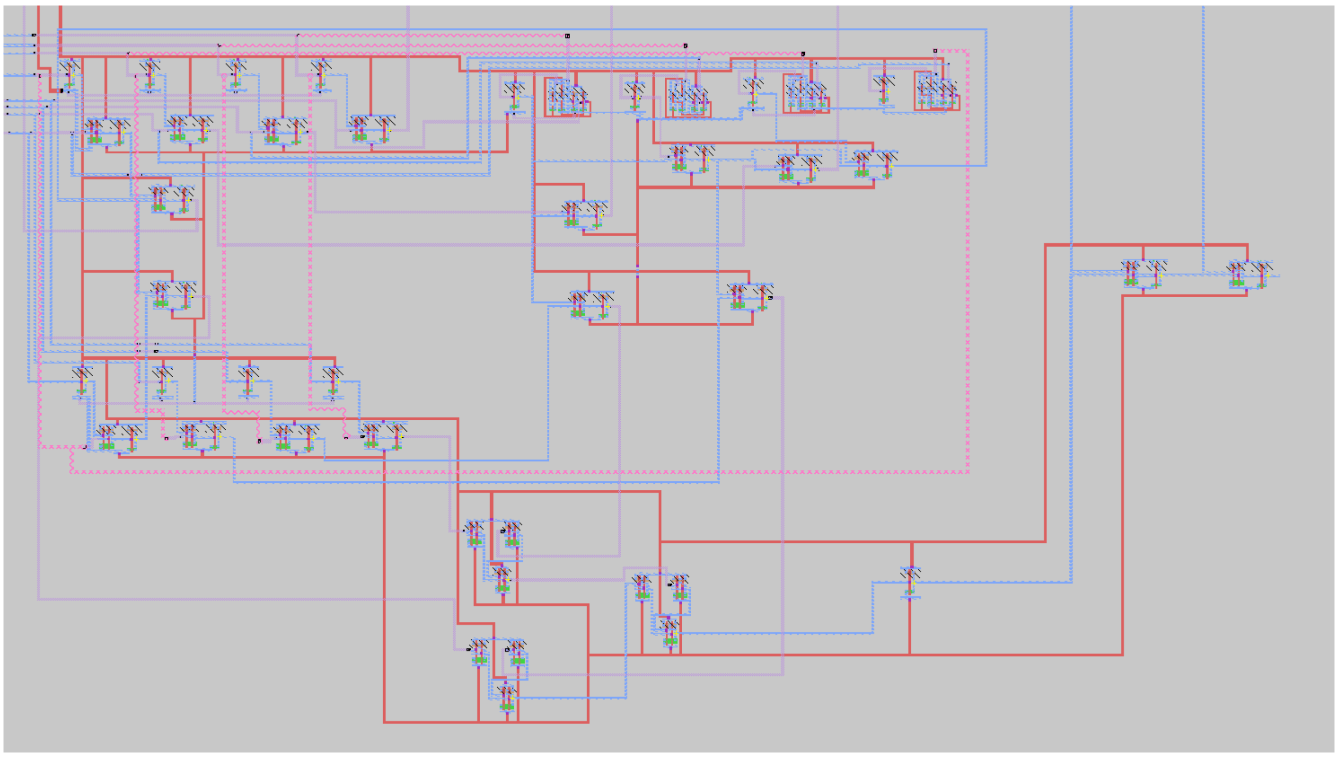
The four units here correspond to the full adder blocks and combining them we get adder subtractor .

4. COMPARATOR BLOCK :

A < B is implemented as shown below :



A > B and A = B :

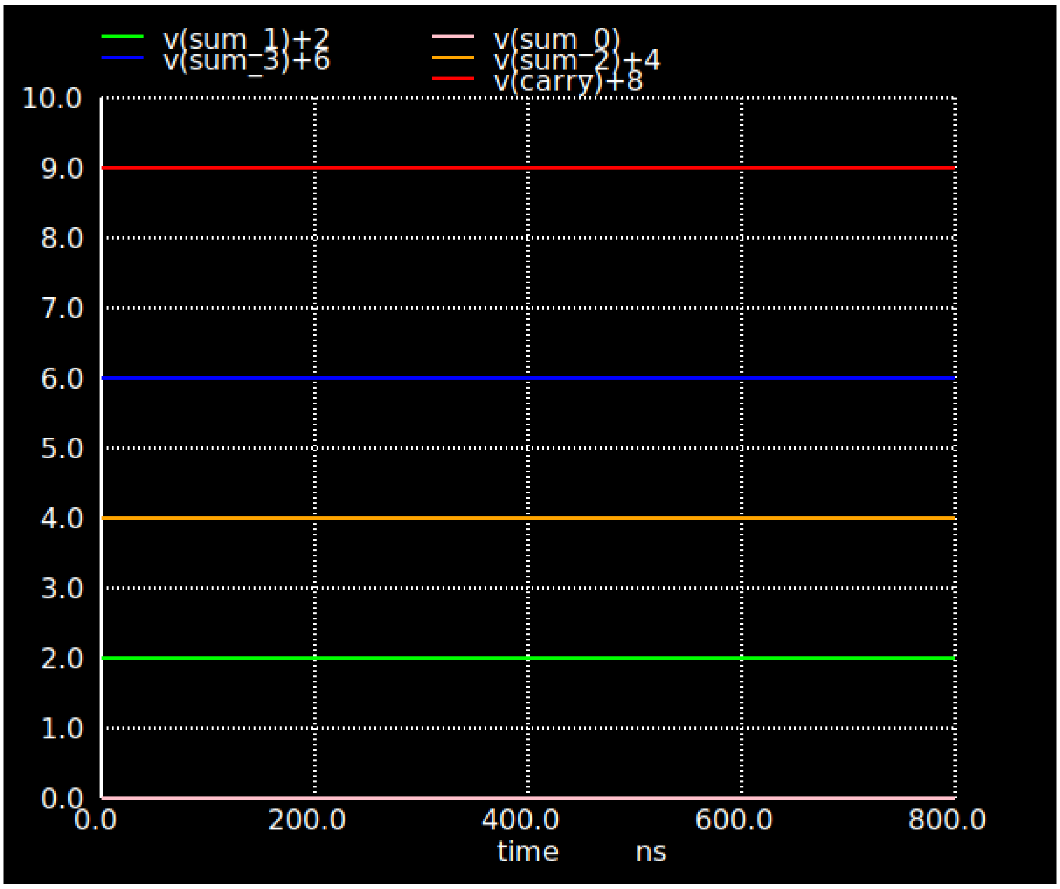


Input and output in magic layout :

After extracting the file from magic , we convert to spice format and run in ngspice to plot and verify the outputs .

1) Adder Block :

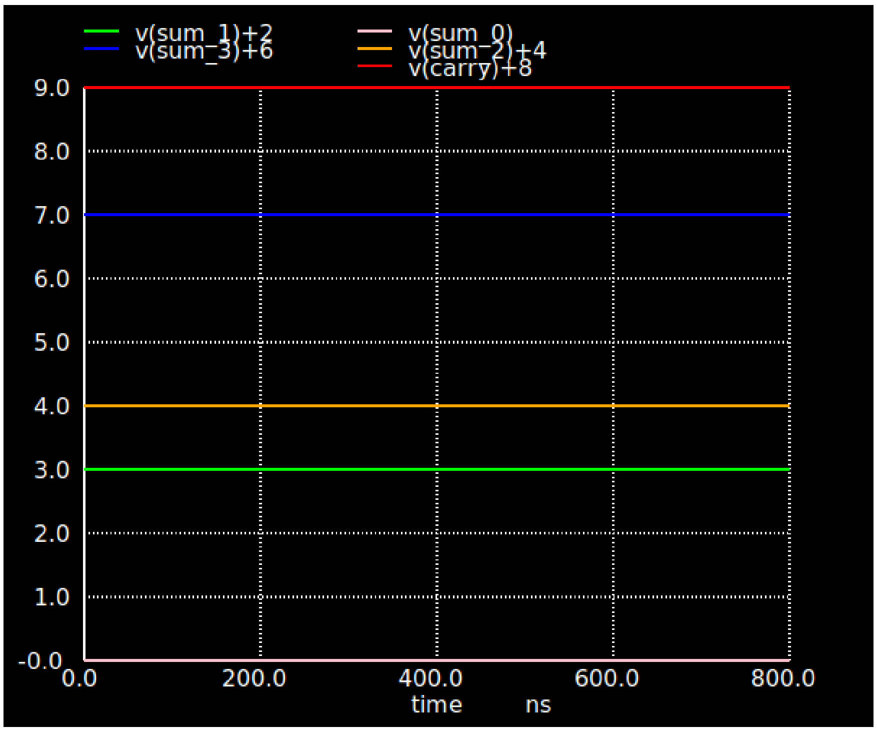
Inputs : A = 0101 , B = 1011 and S0 =0 , S1 = 0



A + B = 5+11 = 16 = 1 0000 which is confirmed by the above plot .

2) Subtractor Block :

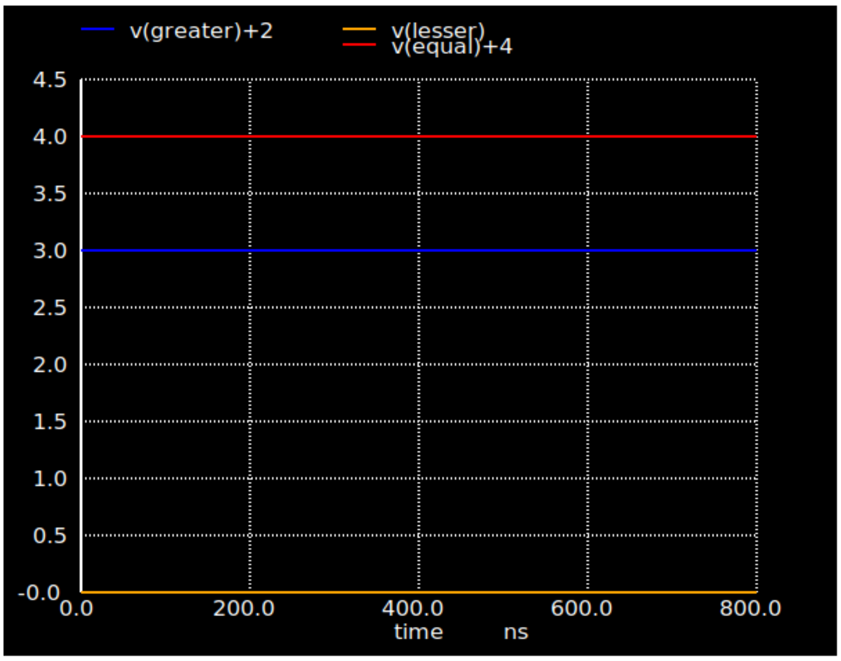
Inputs : A = 1101 , B = 0011 and S0 =1 , S1 = 0



A – B = 1101 – 0011 = 11010 , Carry bit is 1 since A > B

3) Comparator block :

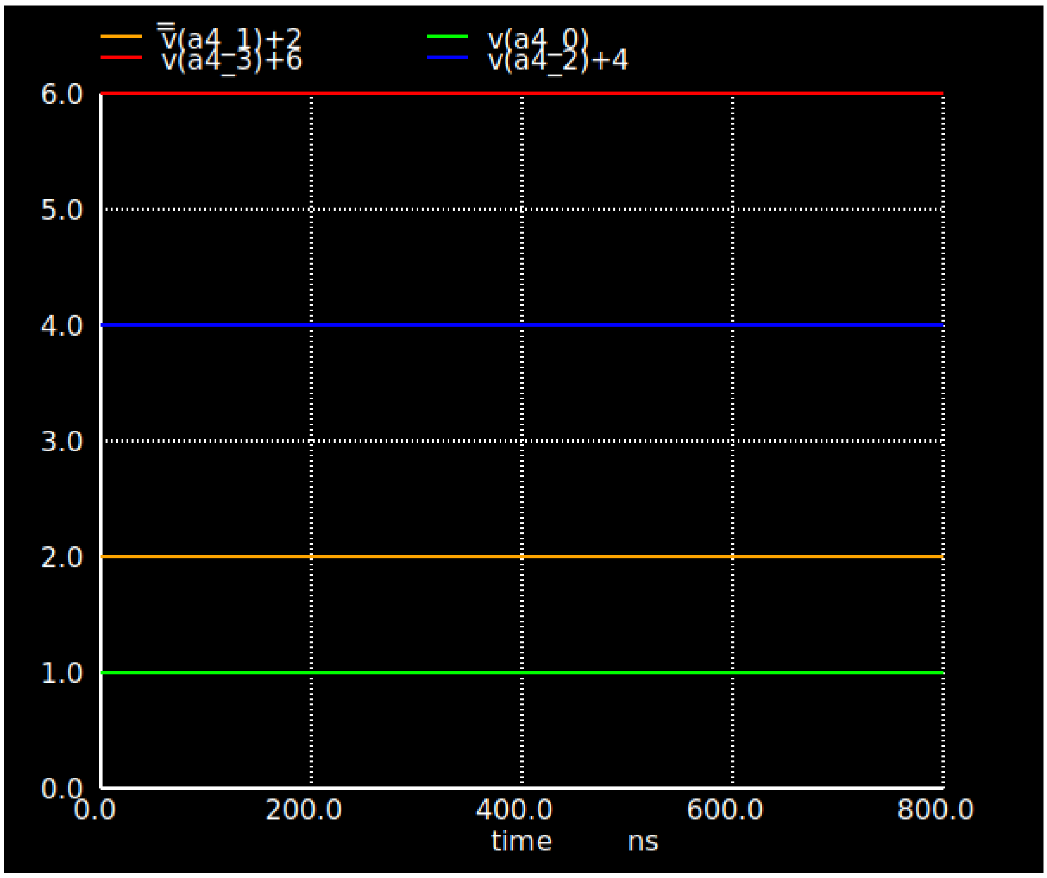
A = 1111 B = 1011 and S0 = 0 , S1 = 1



Clearly A> and only greater than is high .

4. ANDING BLOCK :

INPUTS : A = 1101 , B = 0011 and S0 = 1, S1 =1



Output should be = 0001 which is verified by the above plot .

AREAS OF CONCERN :

1) Major issued I experienced was I kept overlapping different metals which lead of shorting of inputs and eventually wrong output .

2) I didn't initally connect all the vdd and ground to a common supply . I later had to rectify it .

DELAY ANALYSIS :

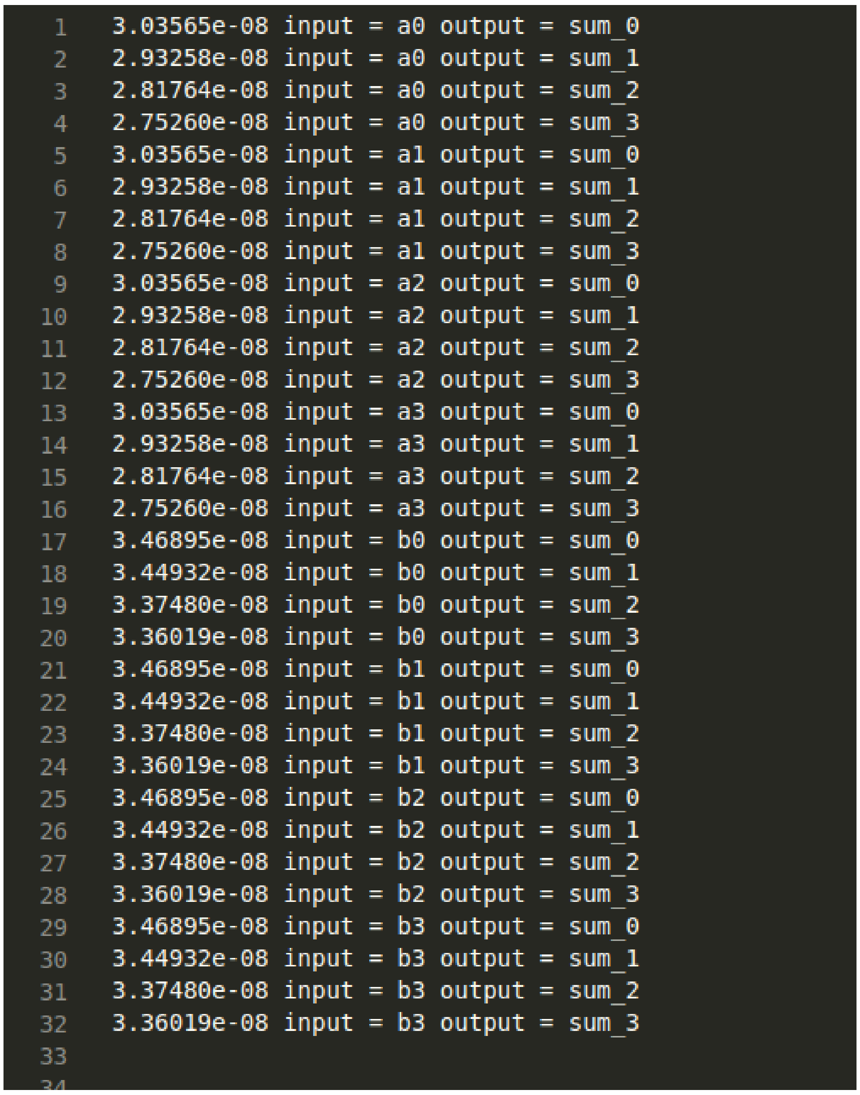
1) ADDER BLOCK

In case of adder , we use 2 different files :

1. File 1 : A is pulse supply and B is fixed at 0000
2. File 2 : B is pulse supply and A is fixed at 0000

In this case , we can calculate the maximum delays i.e 32 by fixing A and B to 0000 .

We calculate **RISE RISE , FALL FALL** for both the inputs.



```
3.03565e-08 input = a0 output = sum_0
2.93258e-08 input = a0 output = sum_1
2.81764e-08 input = a0 output = sum_2
2.75260e-08 input = a0 output = sum_3
3.03565e-08 input = a1 output = sum_0
2.93258e-08 input = a1 output = sum_1
2.81764e-08 input = a1 output = sum_2
2.75260e-08 input = a1 output = sum_3
3.03565e-08 input = a2 output = sum_0
2.93258e-08 input = a2 output = sum_1
2.81764e-08 input = a2 output = sum_2
2.75260e-08 input = a2 output = sum_3
3.03565e-08 input = a3 output = sum_0
2.93258e-08 input = a3 output = sum_1
2.81764e-08 input = a3 output = sum_2
2.75260e-08 input = a3 output = sum_3
3.46895e-08 input = b0 output = sum_0
3.44932e-08 input = b0 output = sum_1
3.37480e-08 input = b0 output = sum_2
3.36019e-08 input = b0 output = sum_3
3.46895e-08 input = b1 output = sum_0
3.44932e-08 input = b1 output = sum_1
3.37480e-08 input = b1 output = sum_2
3.36019e-08 input = b1 output = sum_3
3.46895e-08 input = b2 output = sum_0
3.44932e-08 input = b2 output = sum_1
3.37480e-08 input = b2 output = sum_2
3.36019e-08 input = b2 output = sum_3
3.46895e-08 input = b3 output = sum_0
3.44932e-08 input = b3 output = sum_1
3.37480e-08 input = b3 output = sum_2
3.36019e-08 input = b3 output = sum_3
```

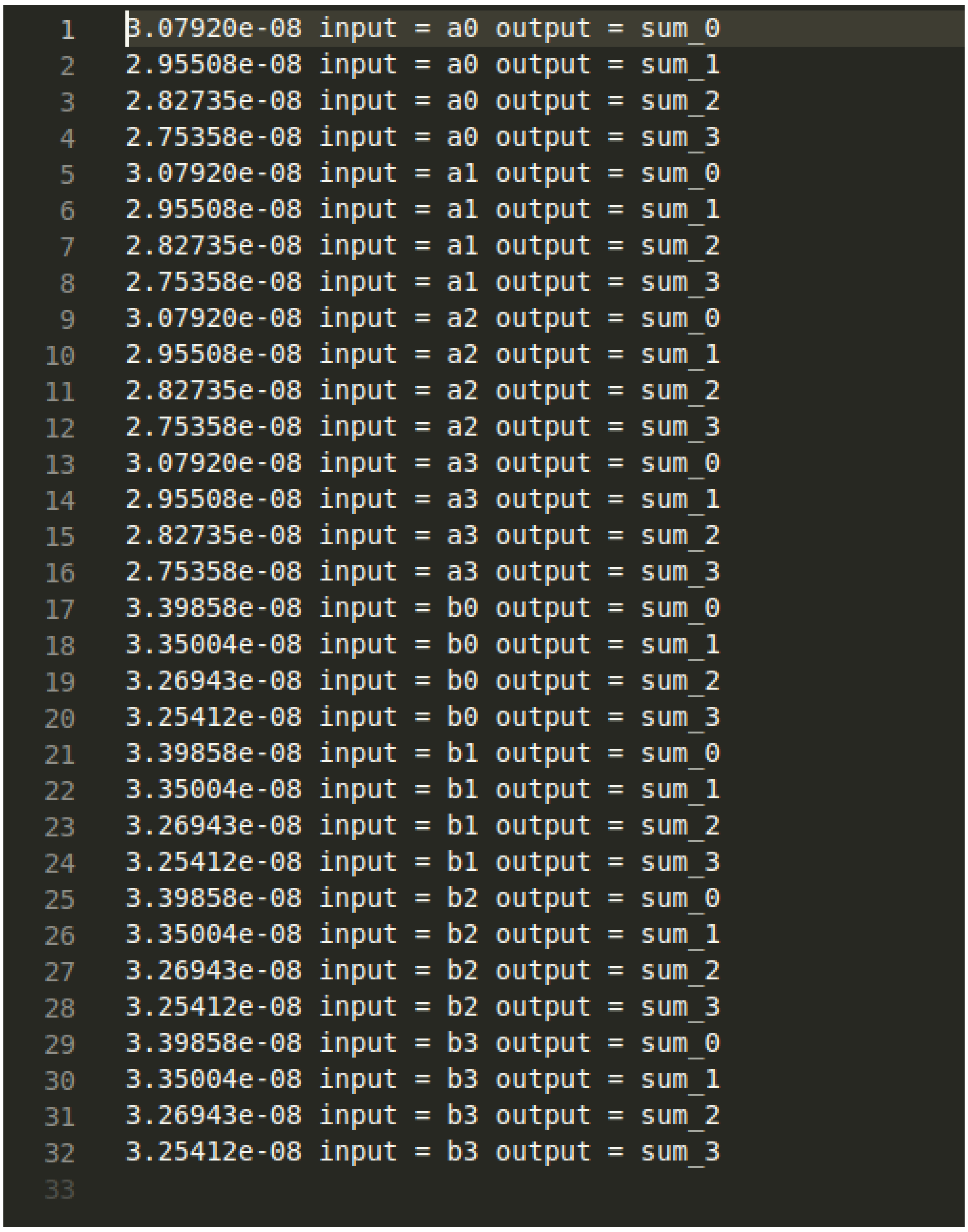
2) SUBTRACTOR BLOCK :

In case of subtractor , we use 2 different files :

1.File 1 : A is pulse supply and B is fixed at 0000
2.File 2 : B is pulse supply and A is fixed at 1111

For file 1 , we run RISE RISE and FALL FALL and calculate the delay .

For file 2 , we calculate RISE FALL and FALL RISE delays . This is because we have fixed A at 1111 and when B increased , the difference reduces and vice – versa .



```
3.07920e-08 input = a0 output = sum_0
2.95508e-08 input = a0 output = sum_1
2.82735e-08 input = a0 output = sum_2
2.75358e-08 input = a0 output = sum_3
3.07920e-08 input = a1 output = sum_0
2.95508e-08 input = a1 output = sum_1
2.82735e-08 input = a1 output = sum_2
2.75358e-08 input = a1 output = sum_3
3.07920e-08 input = a2 output = sum_0
2.95508e-08 input = a2 output = sum_1
2.82735e-08 input = a2 output = sum_2
2.75358e-08 input = a2 output = sum_3
3.07920e-08 input = a3 output = sum_0
2.95508e-08 input = a3 output = sum_1
2.82735e-08 input = a3 output = sum_2
2.75358e-08 input = a3 output = sum_3
3.39858e-08 input = b0 output = sum_0
3.35004e-08 input = b0 output = sum_1
3.26943e-08 input = b0 output = sum_2
3.25412e-08 input = b0 output = sum_3
3.39858e-08 input = b1 output = sum_0
3.35004e-08 input = b1 output = sum_1
3.26943e-08 input = b1 output = sum_2
3.25412e-08 input = b1 output = sum_3
3.39858e-08 input = b2 output = sum_0
3.35004e-08 input = b2 output = sum_1
3.26943e-08 input = b2 output = sum_2
3.25412e-08 input = b2 output = sum_3
3.39858e-08 input = b3 output = sum_0
3.35004e-08 input = b3 output = sum_1
3.26943e-08 input = b3 output = sum_2
3.25412e-08 input = b3 output = sum_3
```

3) COMPARATOR BLOCK :

In case of subtractor , we use 4 different files :

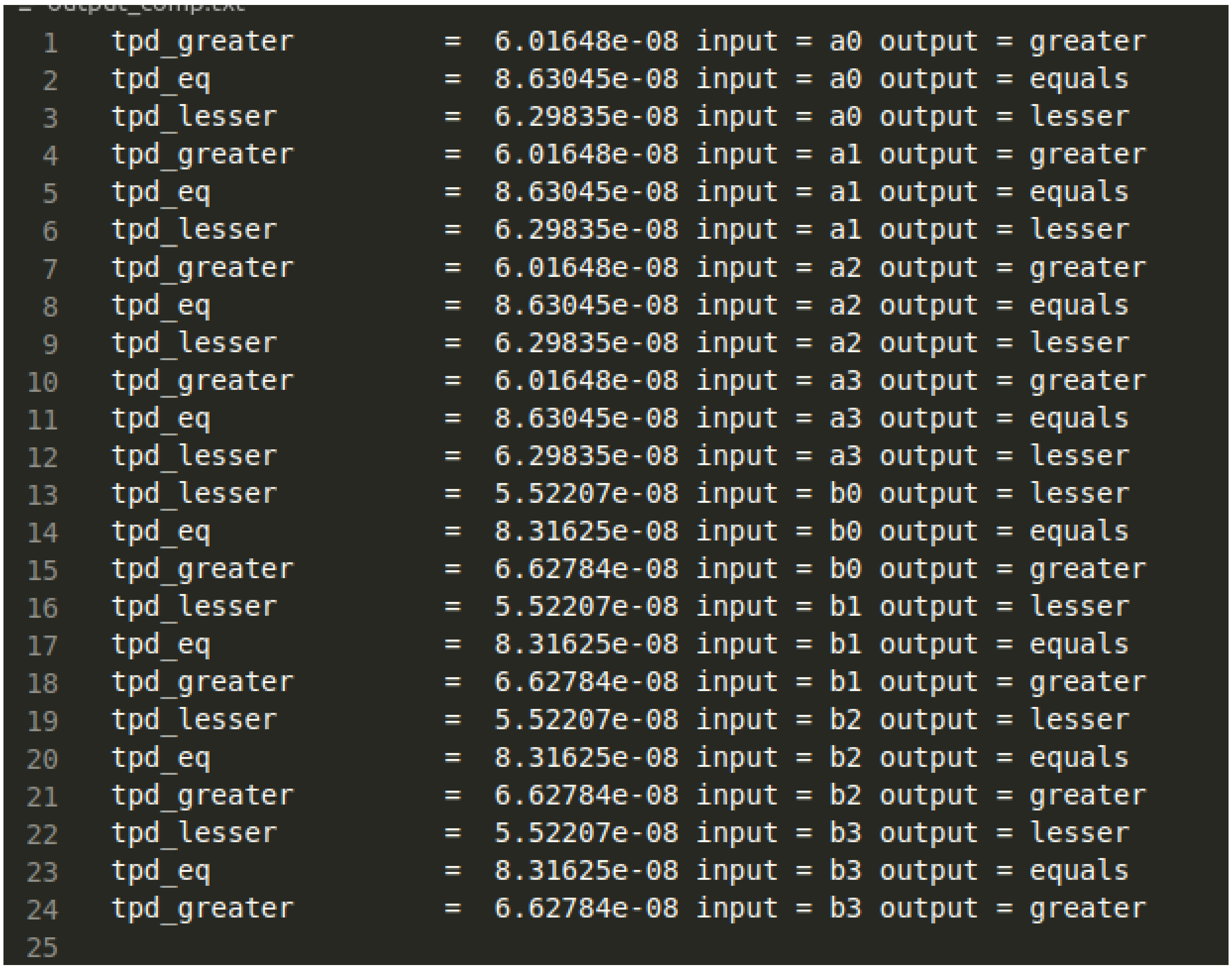
1.File 1 : A is pulse supply and B is fixed at 0000
2.File 2 : A is pulse supply and B is fixed at 1111
3.File 3 : B is pulse supply and A is fixed at 0000
4.File 4 : B is pulse supply and A is fixed at 1111

- For file 1 , we calculate both greater than and equal case for input A and we use RISE RISE FALL FALL for greater than and FALL RISE RISE FALL for the equal to case respectively ..

- For file 2 , we calculate lesser than for input A . We use RISE FALL FALL RISE in this case

- For file 3 , we calculate both lesser than and equal case for inputB and we use RISE RISE FALL FALL for greater than and FALL RISE RISE FALL for the equal to case respectively .

- For file 4 , we calculate greater than for input B . We use RISE FALL FALL RISE in this case .



```
tpd_greater         =  6.01648e-08 input = a0 output = greater
tpd_eq              =  8.63045e-08 input = a0 output = equals
tpd_lesser          =  6.29835e-08 input = a0 output = lesser
tpd_greater         =  6.01648e-08 input = a1 output = greater
tpd_eq              =  8.63045e-08 input = a1 output = equals
tpd_lesser          =  6.29835e-08 input = a1 output = lesser
tpd_greater         =  6.01648e-08 input = a2 output = greater
tpd_eq              =  8.63045e-08 input = a2 output = equals
tpd_lesser          =  6.29835e-08 input = a2 output = lesser
tpd_greater         =  6.01648e-08 input = a3 output = greater
tpd_eq              =  8.63045e-08 input = a3 output = equals
tpd_lesser          =  6.29835e-08 input = a3 output = lesser
tpd_lesser          =  5.52207e-08 input = b0 output = lesser
tpd_eq              =  8.31625e-08 input = b0 output = equals
tpd_greater         =  6.62784e-08 input = b0 output = greater
tpd_lesser          =  5.52207e-08 input = b1 output = lesser
tpd_eq              =  8.31625e-08 input = b1 output = equals
tpd_greater         =  6.62784e-08 input = b1 output = greater
tpd_lesser          =  5.52207e-08 input = b2 output = lesser
tpd_eq              =  8.31625e-08 input = b2 output = equals
tpd_greater         =  6.62784e-08 input = b2 output = greater
tpd_lesser          =  5.52207e-08 input = b3 output = lesser
tpd_eq              =  8.31625e-08 input = b3 output = equals
tpd_greater         =  6.62784e-08 input = b3 output = greater
```

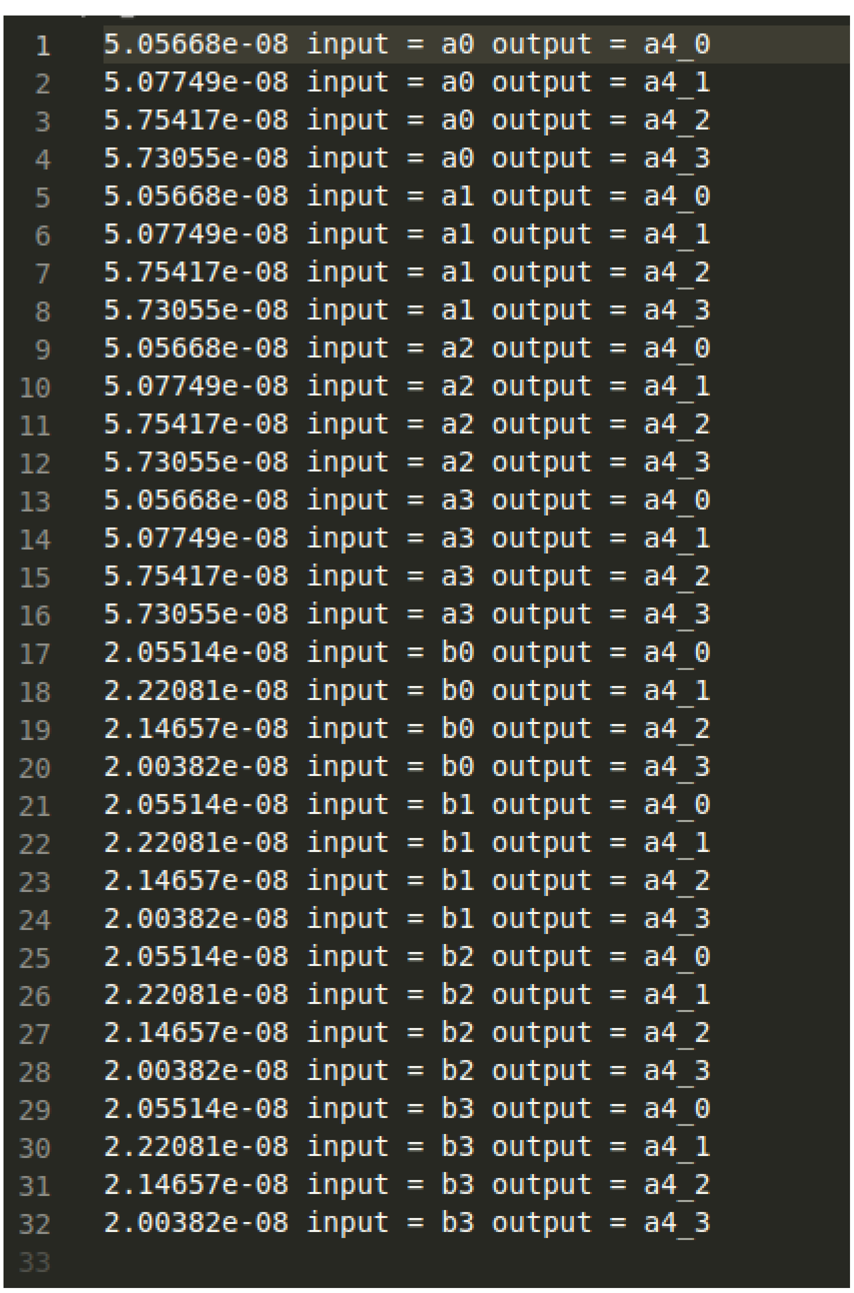
4) AND BLOCK :

In case of and block, we use 2 different files :

3. File 1 : A is pulse supply and B is fixed at 1111
4. File 2 : B is pulse supply and A is fixed at 111

In this case , we can calculate the maximum delays i.e 32 by fixing A and B to 1111 .

We calculate **RISE RISE , FALL FALL** for both the inputs.



```
5.05668e-08 input = a0 output = a4_0
5.07749e-08 input = a0 output = a4_1
5.75417e-08 input = a0 output = a4_2
5.73055e-08 input = a0 output = a4_3
5.05668e-08 input = a1 output = a4_0
5.07749e-08 input = a1 output = a4_1
5.75417e-08 input = a1 output = a4_2
5.73055e-08 input = a1 output = a4_3
5.05668e-08 input = a2 output = a4_0
5.07749e-08 input = a2 output = a4_1
5.75417e-08 input = a2 output = a4_2
5.73055e-08 input = a2 output = a4_3
5.05668e-08 input = a3 output = a4_0
5.07749e-08 input = a3 output = a4_1
5.75417e-08 input = a3 output = a4_2
5.73055e-08 input = a3 output = a4_3
2.05514e-08 input = b0 output = a4_0
2.22081e-08 input = b0 output = a4_1
2.14657e-08 input = b0 output = a4_2
2.00382e-08 input = b0 output = a4_3
2.05514e-08 input = b1 output = a4_0
2.22081e-08 input = b1 output = a4_1
2.14657e-08 input = b1 output = a4_2
2.00382e-08 input = b1 output = a4_3
2.05514e-08 input = b2 output = a4_0
2.22081e-08 input = b2 output = a4_1
2.14657e-08 input = b2 output = a4_2
2.00382e-08 input = b2 output = a4_3
2.05514e-08 input = b3 output = a4_0
2.22081e-08 input = b3 output = a4_1
2.14657e-08 input = b3 output = a4_2
2.00382e-08 input = b3 output = a4_3
```

CRITICAL PATH : The path which has the maximum delay is called critical path

FOR ADDER :

Maximum delay = 3.46895e-08

Critical path of this :

1. input = b0 output = sum_0
2. input = b1 output = sum_0
3. input = b2 output = sum_0

4. input = b3 output = sum_0

All of them have the same delay .

FOR SUBTRACTOR :

Maximum delay = 3.39858e-08

Critical path of this :

1. input = b0 output = sum_0
2. input = b1 output = sum_0
3. input = b2 output = sum_0
4. input = b3 output = sum_0

All of them have the same delay .

FOR COMPARATOR :

Maximum delay = 8.63045e-08

Critical path of this :

1. input = a0 output = equal
2. input = a1 output = equal
3. input = a2 output = equal
4. input = a3 output = equal

All of them have the same delay .

FOR AND BLOCK :

Maximum delay = 3.46895e-08

Critical path of this :

1. input = b1 output = sum_0
2. input = b2 output = sum_0
3. input = b2 output = sum_0
4. input = b2 output = sum_0

All of them have the same delay .