Homework/Project 01
EE619A

Submitted by- Group no-26

Rahul Kumar Gupta	21204408	rahulkg21@iitk.ac.in	MS(MVLSI)
Sameer Kumar	21204411	sameerk21@iitk.ac.in	MS(MVLSI)
Sagar	21204409	sagar21@iitk.ac.in	MS(MVLSI)

We have used Tcad software for gate level diagrams and ICARUS as well as Xilinx Vivado for running the code.

Q1.

INPUT BUTTONS : D0,D1,D2,D3,D4,D5,D6,D7,D8,D9

The user gets to select only one of the above mentioned buttons (selecting a button corresponds to logic 1).
By default our outputs will be set to logic low.

The logic for the binary output can be written from the table below

(MSB) B3 = D8 + D9

B2= D7+ D6+ D5+ D4

B1= D7+ D6+ D3+ D2

(LSB) B0= D9+ D7+ D5+ D3+ D1

ENCODER INPUTS										ENCODER OUTPUTS			
D0	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**D8**	**D9**	**B3**	**B2**	**B1**	**B0**
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	1
0	0	0	0	0	0	1	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	1

Here is the gate level logic circuit that will output a single digit decimal digit in BCD form:

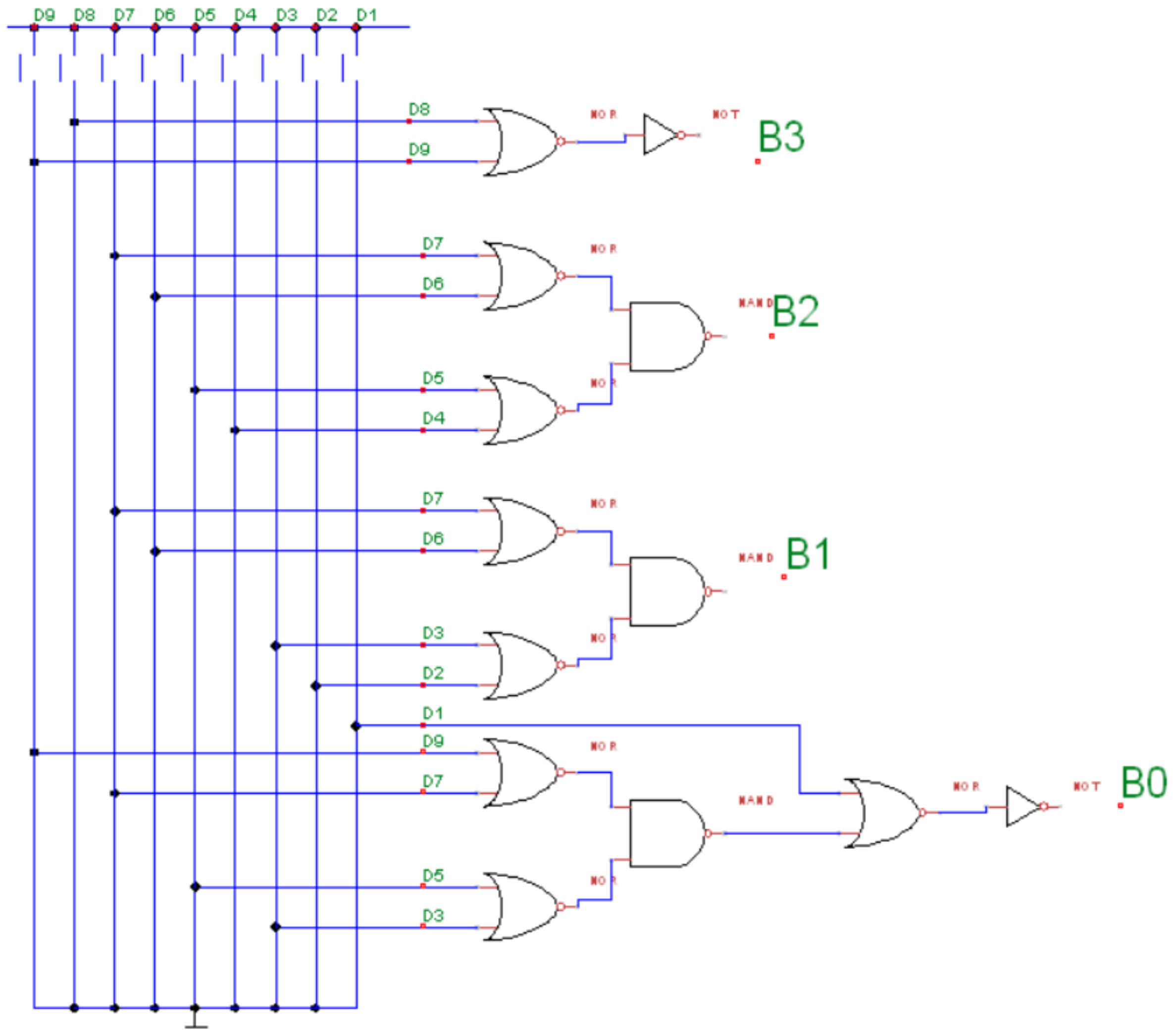


Fig.1. Circuit for converting decimal input to binary output

Q2.

Using the above results we can assume that we have converted decimal into BCD. Therefore we have B3, B2, B1, B0 (B3 is the MSB and B0 is the LSB) as the inputs. a,b,c,d,e,f,g , DP=0(always).

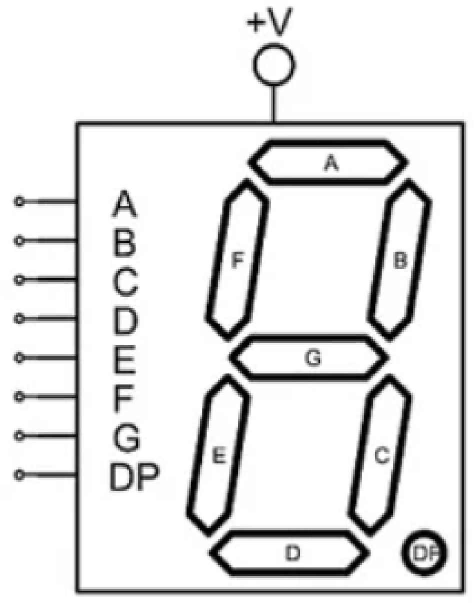


Fig.2. A Seven segment display(google images)

Truth table:

BCD INPUTS				OUTPUTS						
B3	B2	B1	B0	A	B	C	D	E	F	G
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1

From Truth table.

$a = \sum m(0,2,3,5,6,7,8,9)$

$b = \sum m(0,1,2,3,4,7,8,9)$

$c = \sum m(0,1,3,4,5,6,7,8,9)$

$d = \sum m(0,2,3,5,6,8)$

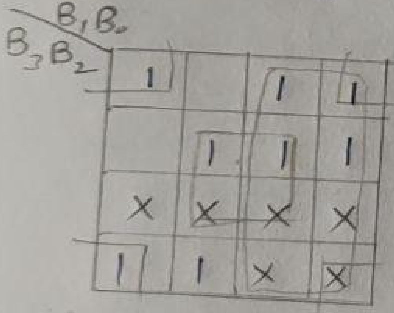
$e = \sum m(0,2,6,8)$

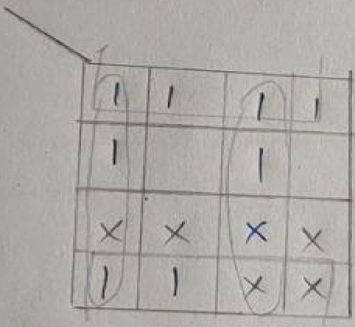
$f = \sum m(0,4,5,6,8,9)$

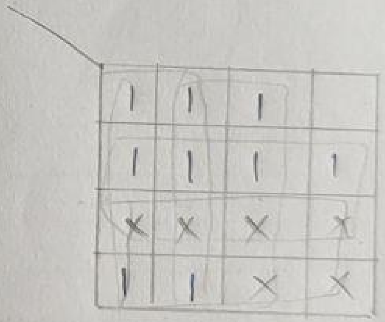
$g = \sum m(2,3,4,5,6,8,9)$

Don't care

$\} + \sum(10,11,12,13,14,15)$



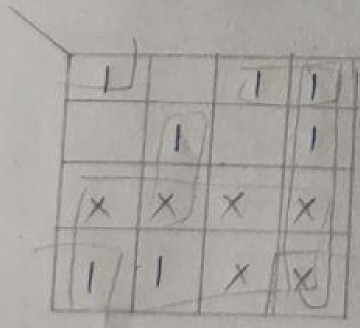


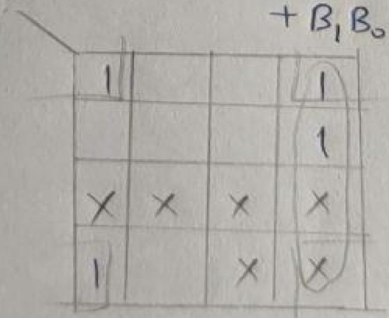


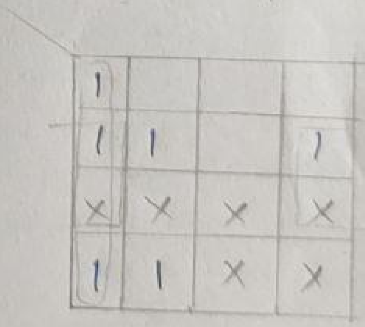
$a = \bar{B}_2\bar{B}_0 + B_1 + B_3 + B_2B_0$

$b = \bar{B}_2 + \bar{B}_1\bar{B}_0 + B_1B_0$

$c = \bar{B}_1 + B_0 + B_2$



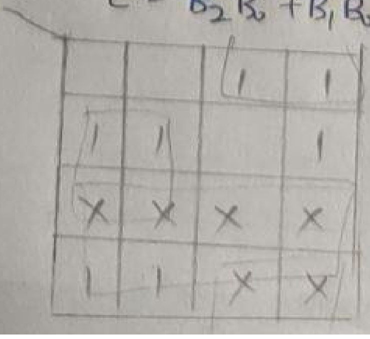




$d = \bar{B}_2\bar{B}_0 + \bar{B}_2B_1 + B_2\bar{B}_1B_0 + B_1\bar{B}_0 + B_3$

$e = \bar{B}_2\bar{B}_0 + B_1\bar{B}_0$

$f = \bar{B}_1\bar{B}_0 + B_2\bar{B}_1 + B_2\bar{B}_0 + B_3$



$g = \bar{B}_2B_1 + B_2\bar{B}_1 + B_3 + B_2\bar{B}_0$

A = B1+ B3+ B2~^B0

B= ~B2+ B2~^B0

C= B2+ ~B1+ B0

D= ~B2.~B0 + ~B2.B1 + B2.~B1.B0 + B1.~B0 + B3

E= ~B2.~B0 + B1.~B0

F= B3 + B2.~B1 + B2.~B0 + ~B1.~B0

G= B1^B2 + B3 + B2.~B0

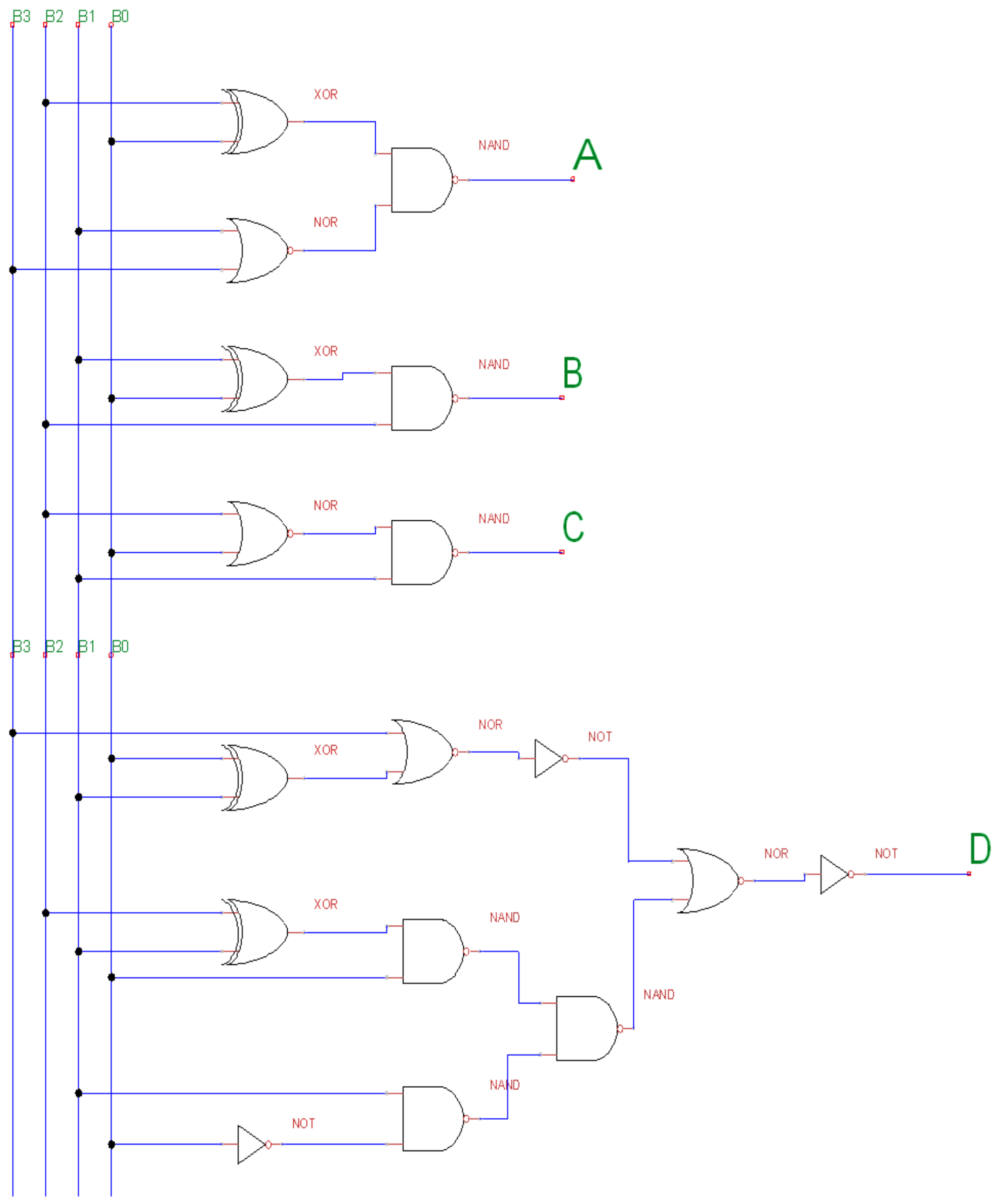


Fig.3. Binary to seven segment logic circuit(A,B,C)

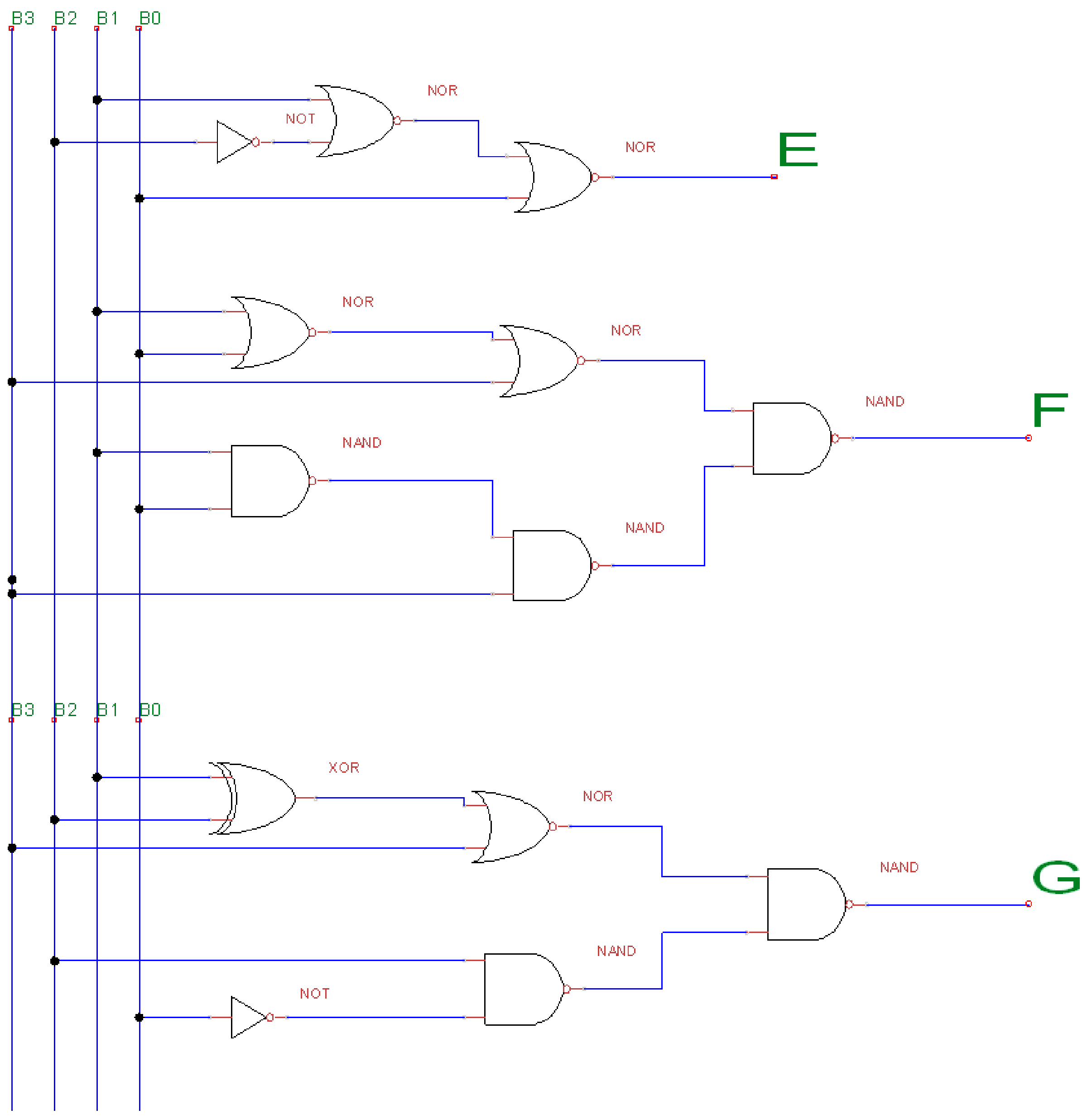


Fig.4. Binary to seven segment logic circuit(D,E,F)

Q3.
Given two decimal integers represented in BCD form x and y.

```
    MSB         LSB
x=  X3  X2 X1 X0
y=  Y3  Y2 Y1 Y0
```

For equality detection bitwise,we can use XNOR gate as it is an equality detector.
Ex: 1~^1= 1 ; 0~^0=1
Since we are dealing with only 0s and 1s then if A>B it means that A=1 and B=0 → (A.~B) = 1
The above logic is used for inequalities.

x	y	x>y	x=y	x<y
0000	**0000**	**0**	**1**	**0**
0000	**0001**	**0**	**0**	**1**
0011	**0010**	**1**	**0**	**0**
1000	**0011**	**1**	**0**	**0**
0000	**0100**	**0**	**0**	**1**
0101	**0101**	**0**	**1**	**0**
0110	**0110**	**0**	**1**	**0**
0110	**0111**	**0**	**0**	**1**
0110	**1000**	**0**	**0**	**1**
1001	**1000**	**1**	**0**	**0**

Some of the examples are shown in the table above
Therefore **for x=y**
We have (X3~^Y3).(X2~^Y2).(X1~^Y1).(X0~^Y0)

For x>y
We have (X3.~Y3)+(X3~^Y3)(X2.~Y2)+ (X3~^Y3).(X2~^Y2).(X1.~Y1)+
(X3~^Y3).(X2~^Y2).(X1~^Y1).(X0.~Y0)

For x<y
We have (Y3.~X3)+(Y3~^X3)(Y2.~X2)+ (Y3~^X3).(Y2~^X2).(Y1.~X1)+
(Y3~^X3).(Y2~^X2).(Y1~^X1).(Y0.~X0)

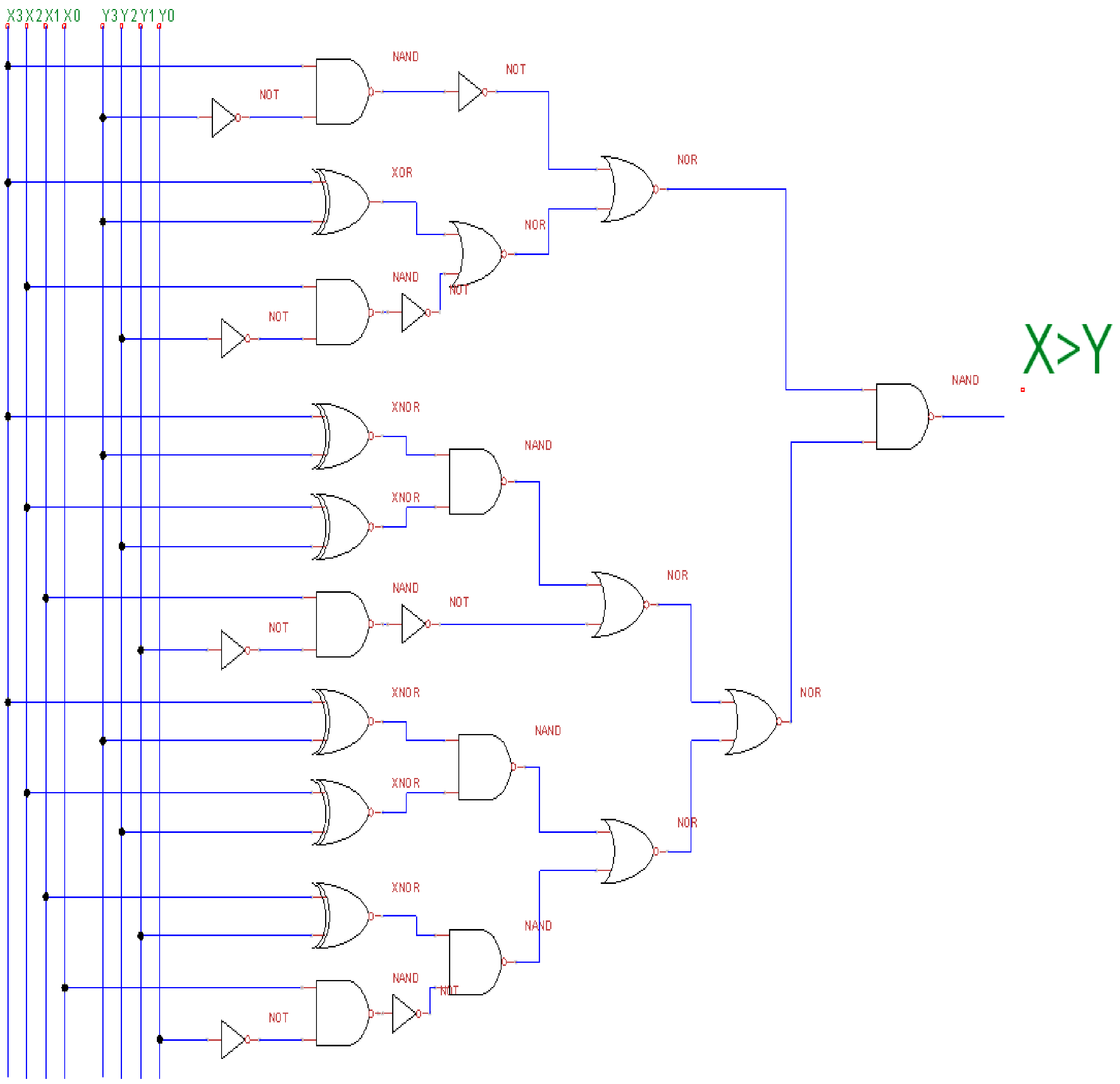


Fig.5. Circuit based on logic provided for X>Y

X3.~Y3)+(X3~^Y3)(X2.~Y2)+ (X3~^Y3).(X2~^Y2).(X1.~Y1)+ (X3~^Y3).(X2~^Y2).(X1~^Y1).(X0.~Y0)

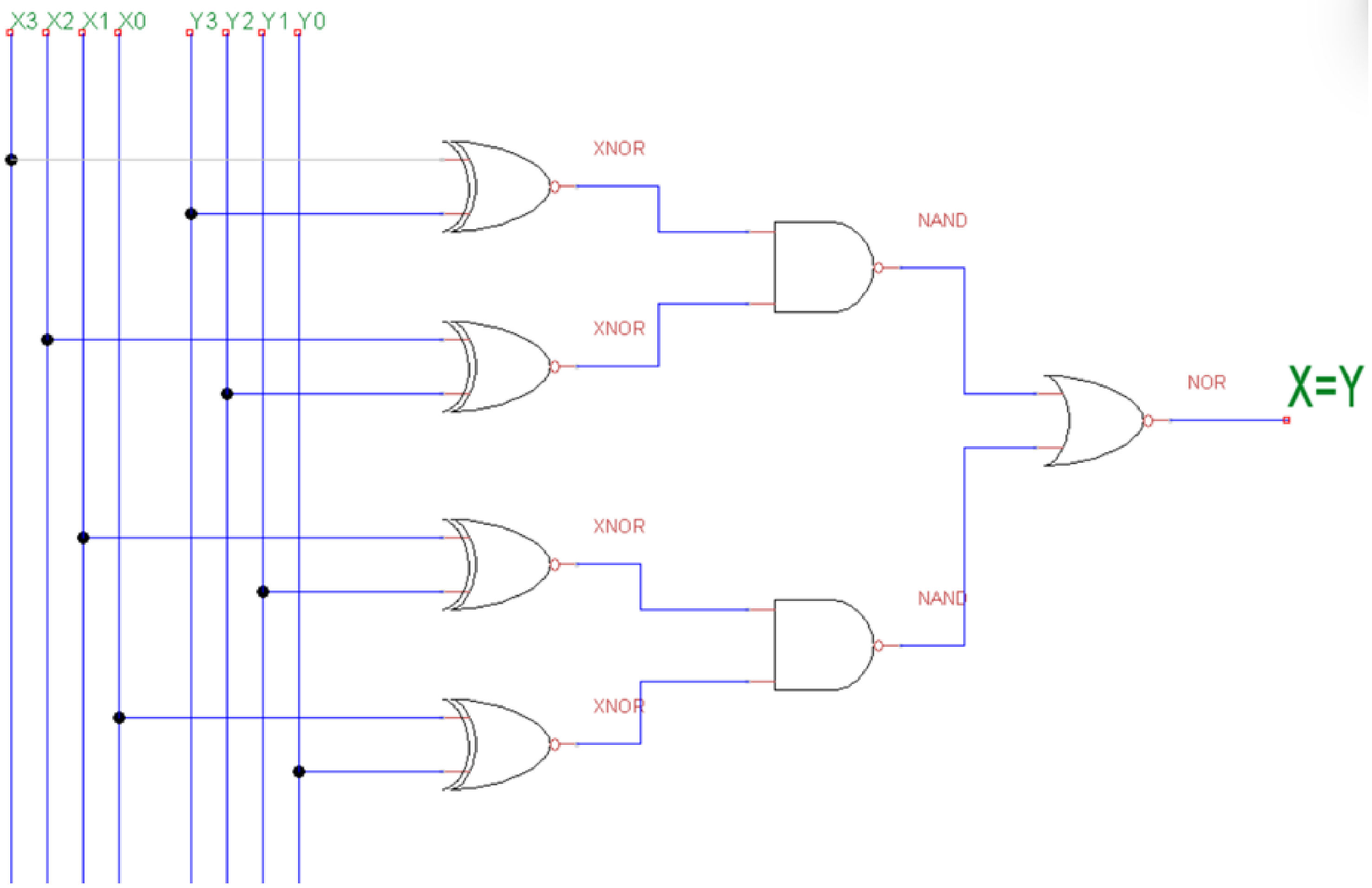


Fig.6. Circuit based on logic provided for X=Y

(X3~^Y3).(X2~^Y2).(X1~^Y1).(X0~^Y0)

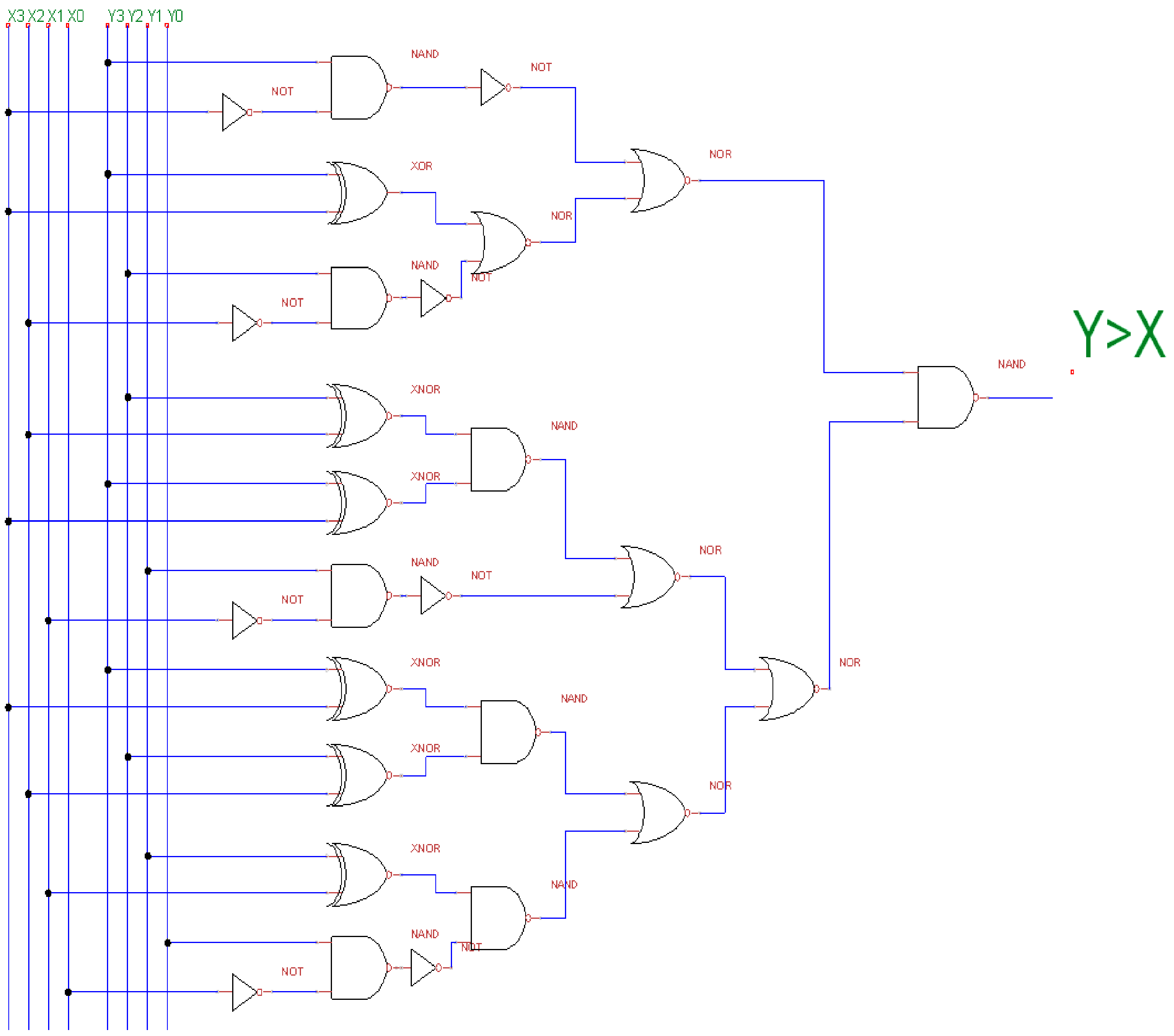


Fig.7. Circuit based on logic provided for X<Y

(Y3.~X3)+(Y3~^X3)(Y2.~X2)+ (Y3~^X3).(Y2~^X2).(Y1.~X1)+ (Y3~^X3).(Y2~^X2).(Y1~^X1).(Y0.~X0)

Q4.
Here we'll be using the BCD to 7-segment block designed in Q2.

Here is the gate level logic circuit of a Full Adder

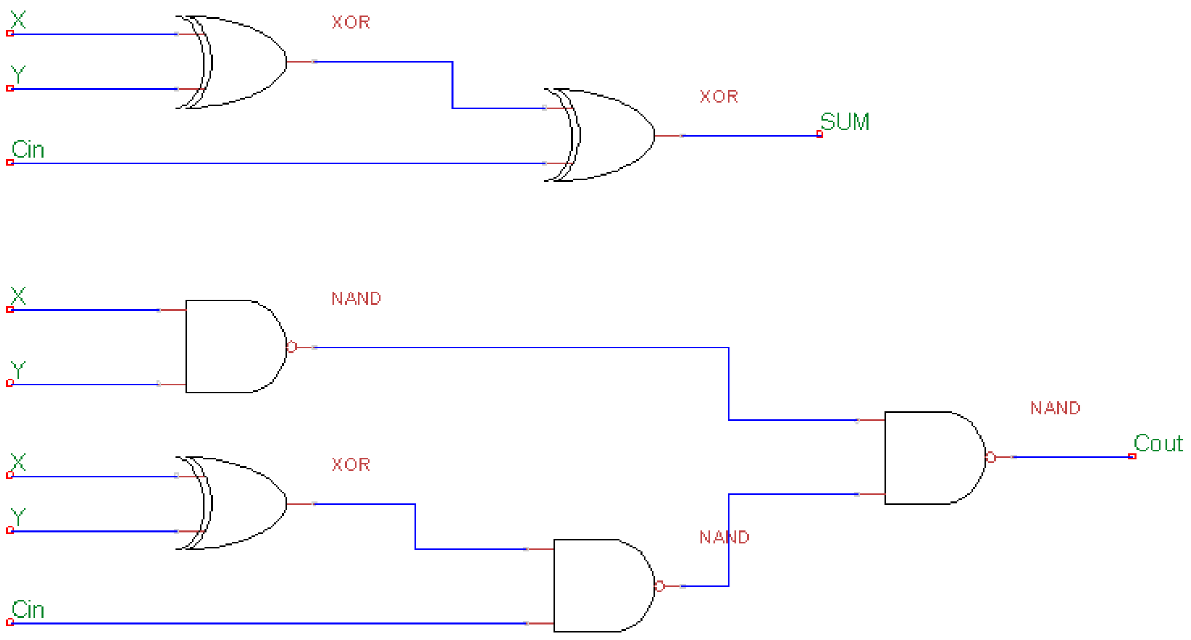


Fig.8. SUM and CARRY based on given condition of question

SUM = X^Y^Cin
Cout = ~[(~(X.Y)).(~((X^Y).Cin))]

Above circuit can be modeled as:

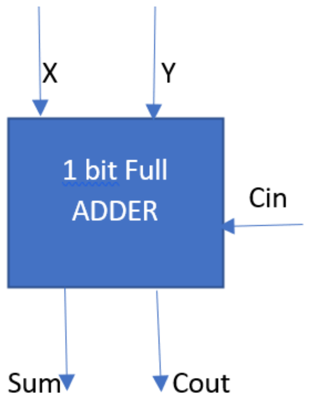


Fig.9. Block for 1 bit adder

Using the above designed full adder, we can design a 4 bit Full Adder

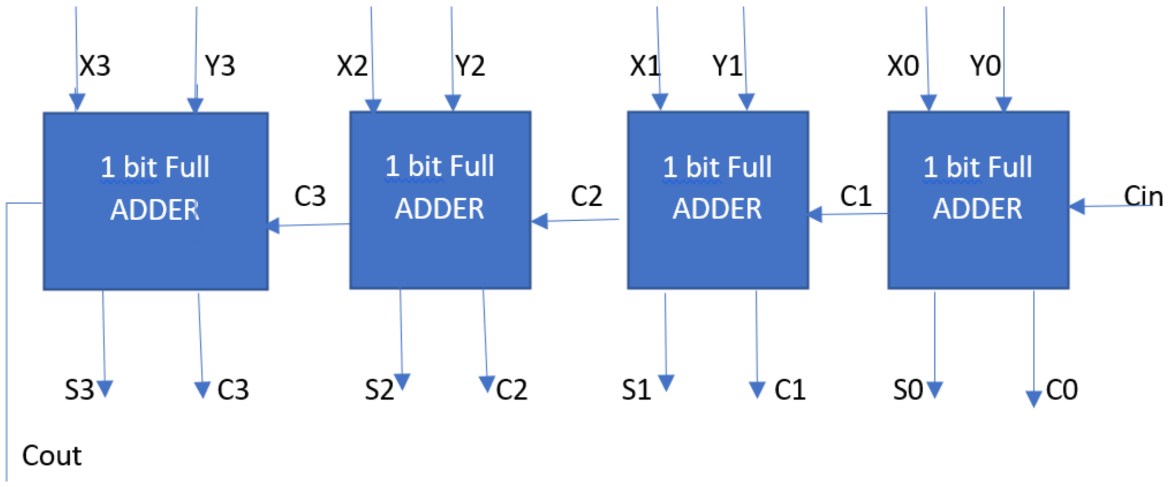


Fig.10. Block for 4 bit adder

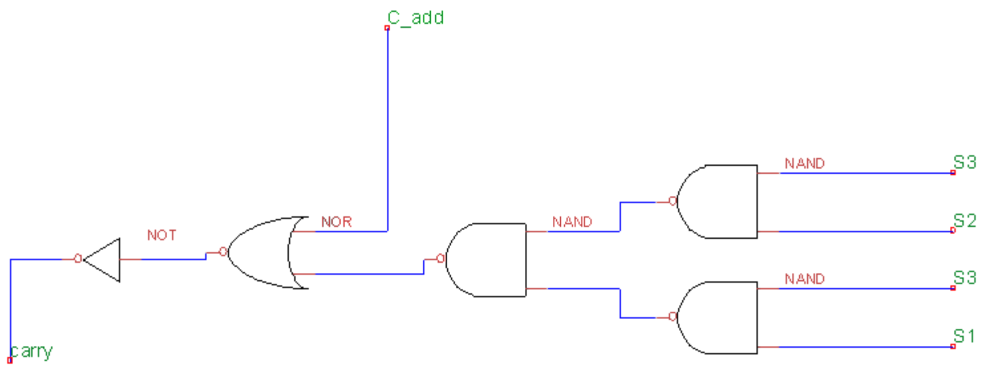


Fig.11. Logic implemented for +6 whenever the 4 bit adder exceeds 9(1001) to convert to BCD

The above circuit is the +6 block that is used to add 6(0110) in the result to convert into BCD(if required).

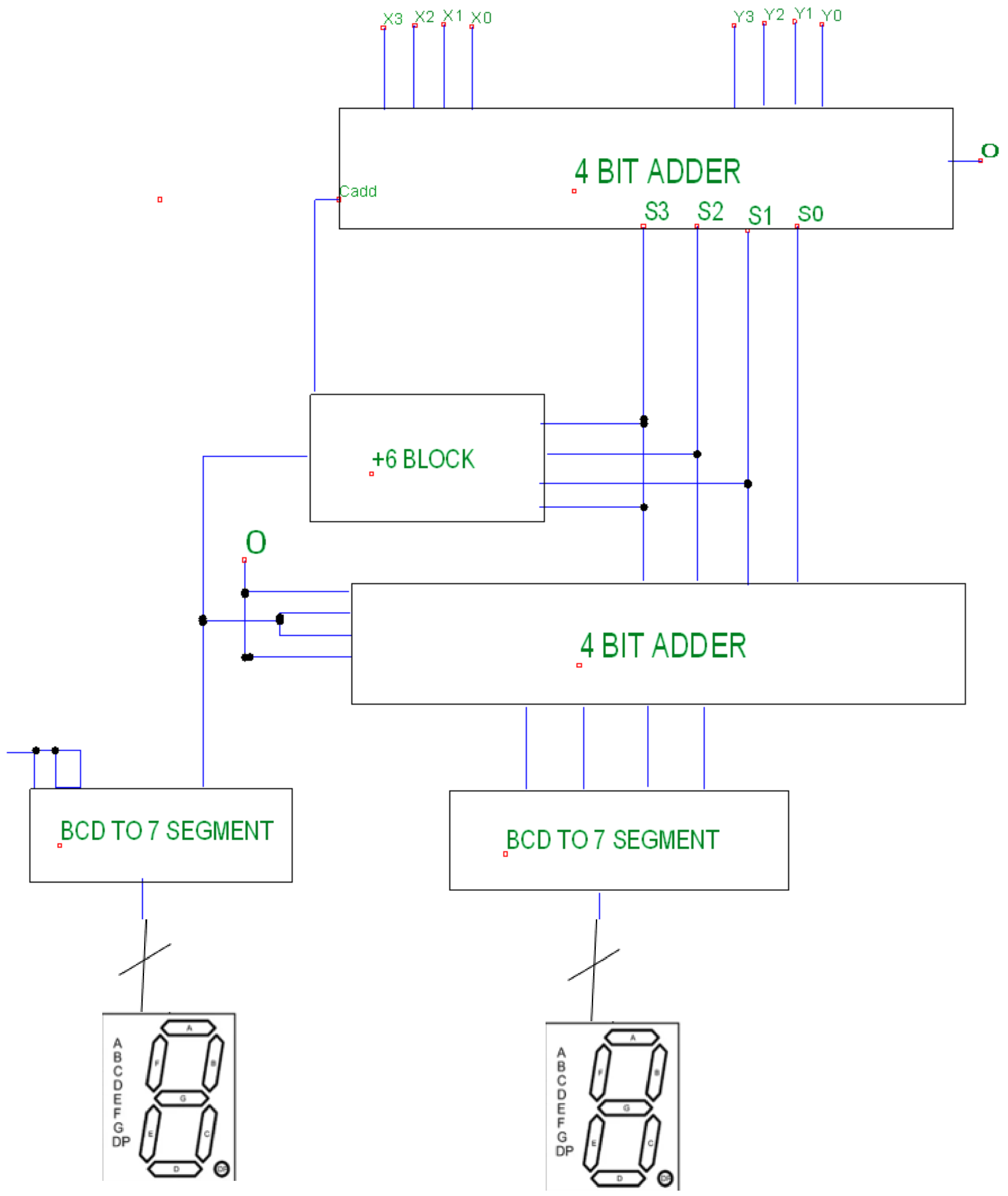


Fig.12. A BCD adder interfacing with 2 seven segment

The BCD to seven segment block is taken from question 2

Q5.

Given two single digit decimal digits x,y such that $(x \geq y \geq 0)$
We have to perform x-y always.
Both x and y are in five bit binary form but since both of them are positive and have value less than 10 therefore the fifth bit is always zero.
The subtraction scheme used is that of 2s complement,that we convert y into its 2s complement form and since x is positive its 2s complement form is the same as that of BCD form. After this we add both of them and the result although in 2s complement form is the final result since x ≥ y always.
Here the carry is discarded if generated.

Here we’ll be using the 4-bit adder designed in **Q4.**

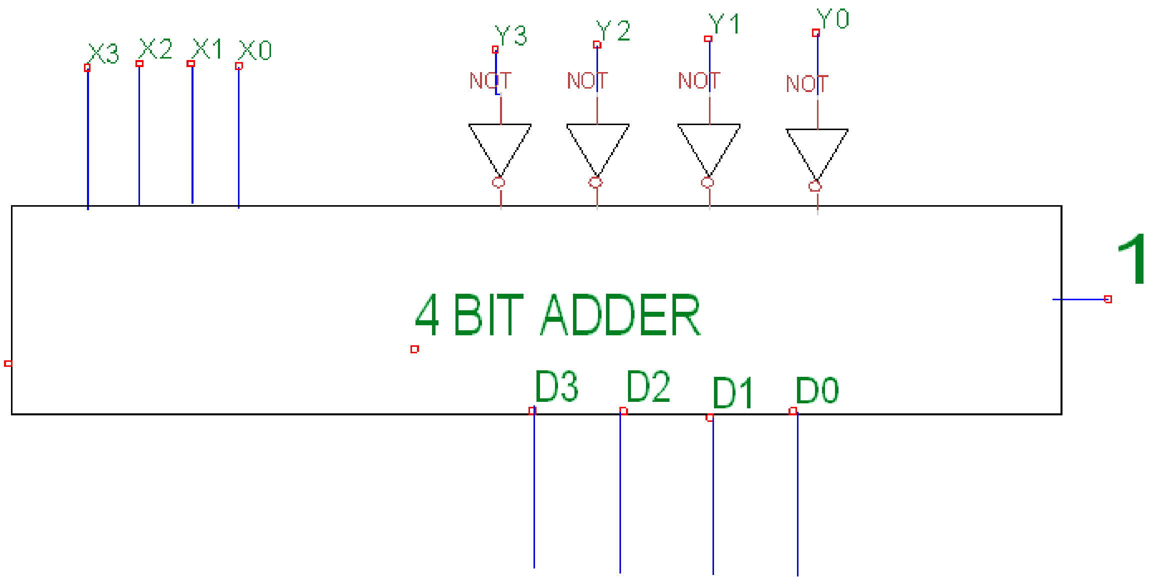


Fig.13. A subtractor

The above circuit is the required block diagram used to get x-y

Q6.

HERE we are including both positive and negative numbers,that is -9 to +9 to be the range of x and y.
The addition and subtraction operation can be decided by the value of the control variable K.
K = 0 means **addition**
K = 1 means **subtraction**
We are using 4-bit adders AND +6 BLOCK designed in Q4.

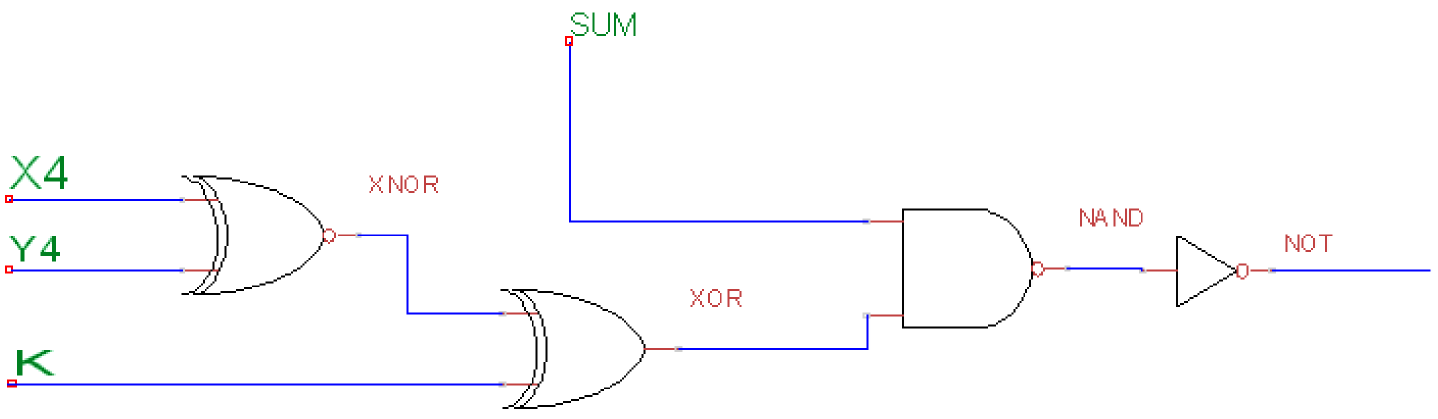


Fig.14. A overflow checker logic implementation

This is an overflow block used by a signed bit in order to find the overflow.

The first 7 seven segment is fused with G of seven segment directly without decoder as its purpose is to just represent the sign

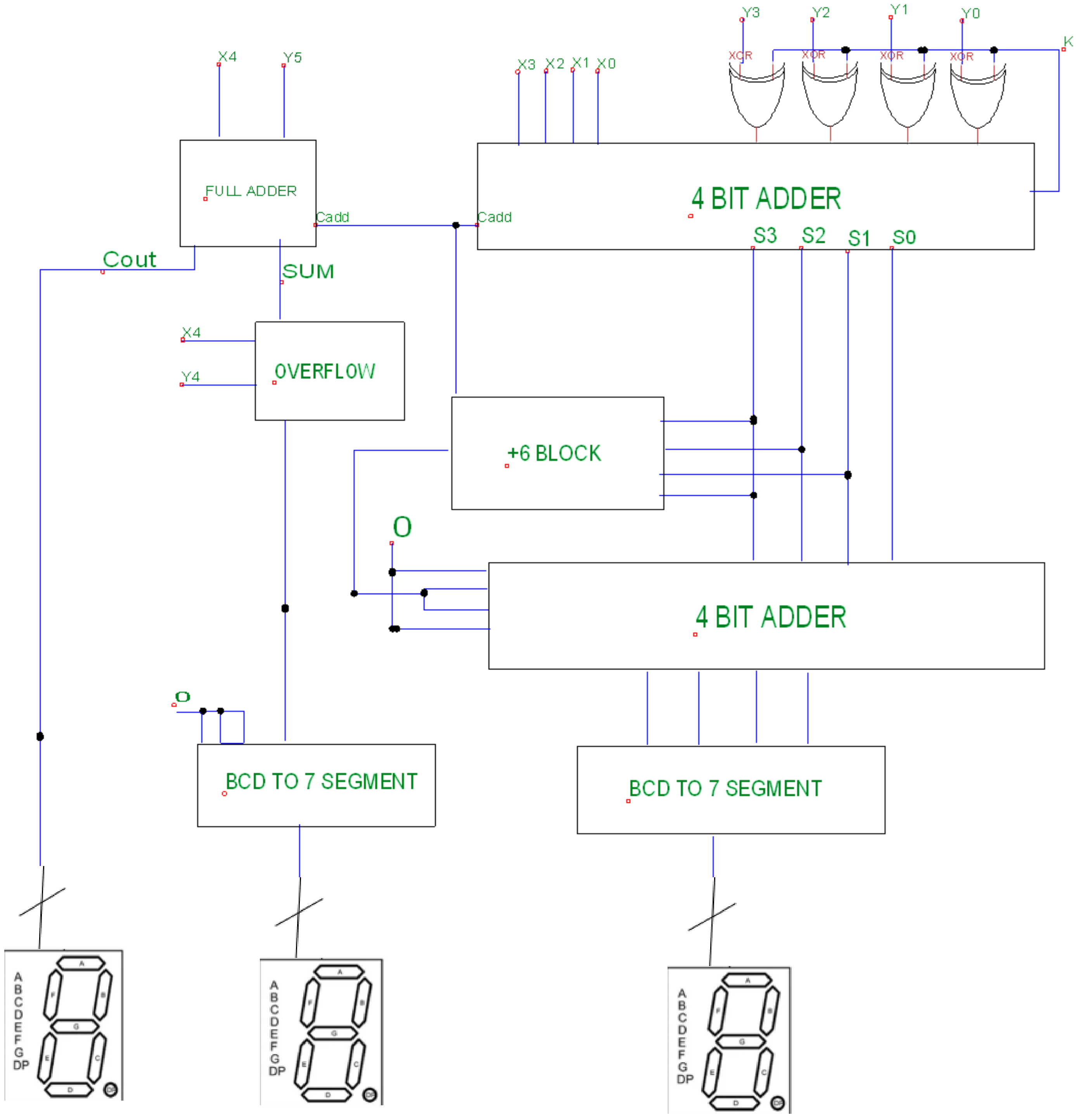


Fig.15. A BCD adder and subtractor interfaced with 7 segment(all the block is used from previous questions)

Q7.

x and y are signed 3 digit decimal numbers on which addition(K=0) or subtraction(K=1) operation can be performed. The result contains 4 decimal digits with a sign.
The circuit designed in **Q6.** is shown as the unnamed blocks in this circuit.
The “overflow” and “BCD TO 7 SEGMENT” are the same circuits designed in **Q6.** and **Q2.** respectively.
To the outputs of “BCD TO 7 SEGMENT” blocks and Cout are connected 7 segment displays.

The first 7 seven segment is fused with G of seven segment directly without decoder as its purpose is to just represent the sign

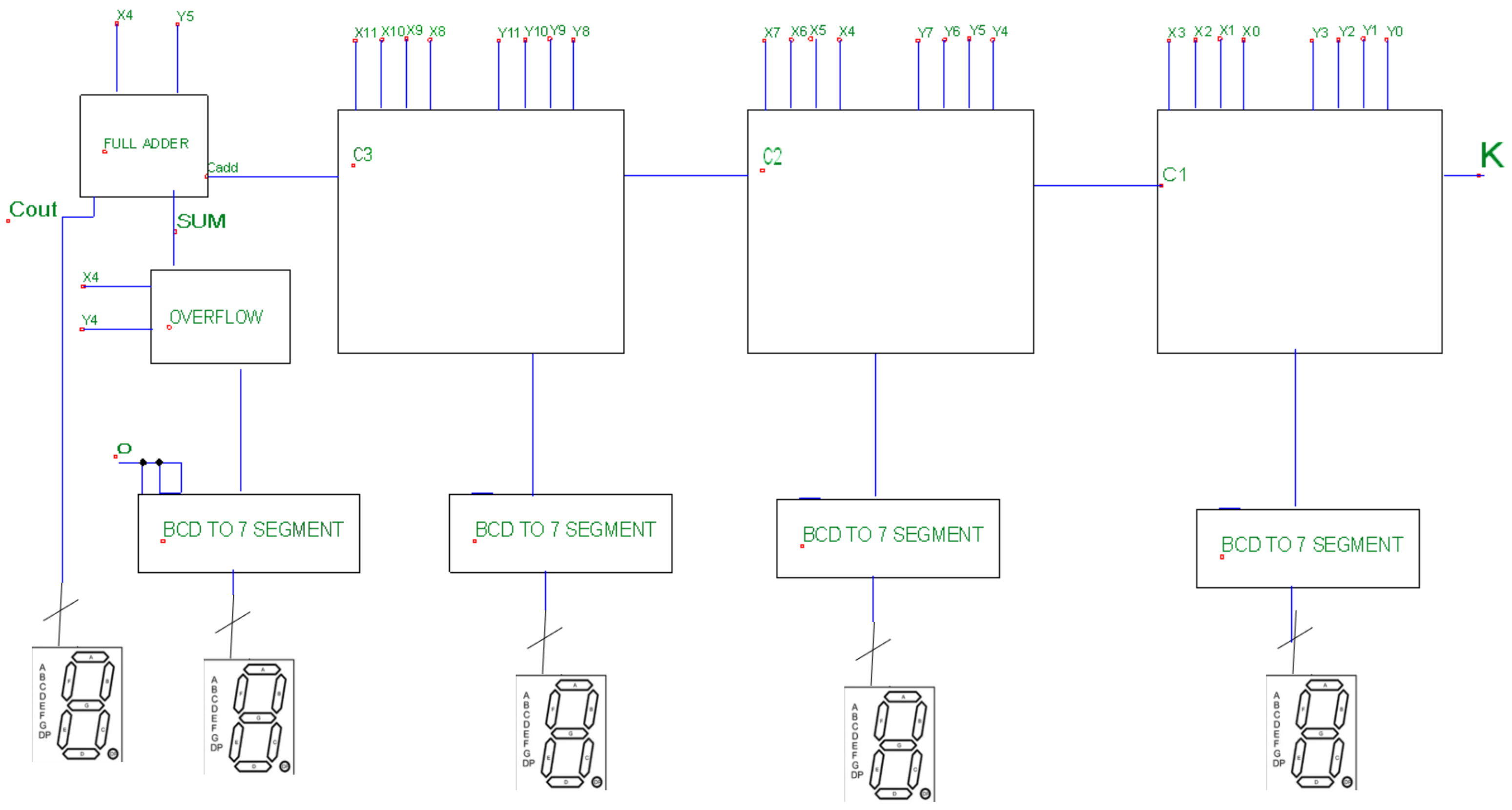


Fig.16. A 3 bit decimal adder and subtractor(all the block is used from previous questions) interfaced with 5 seven segment

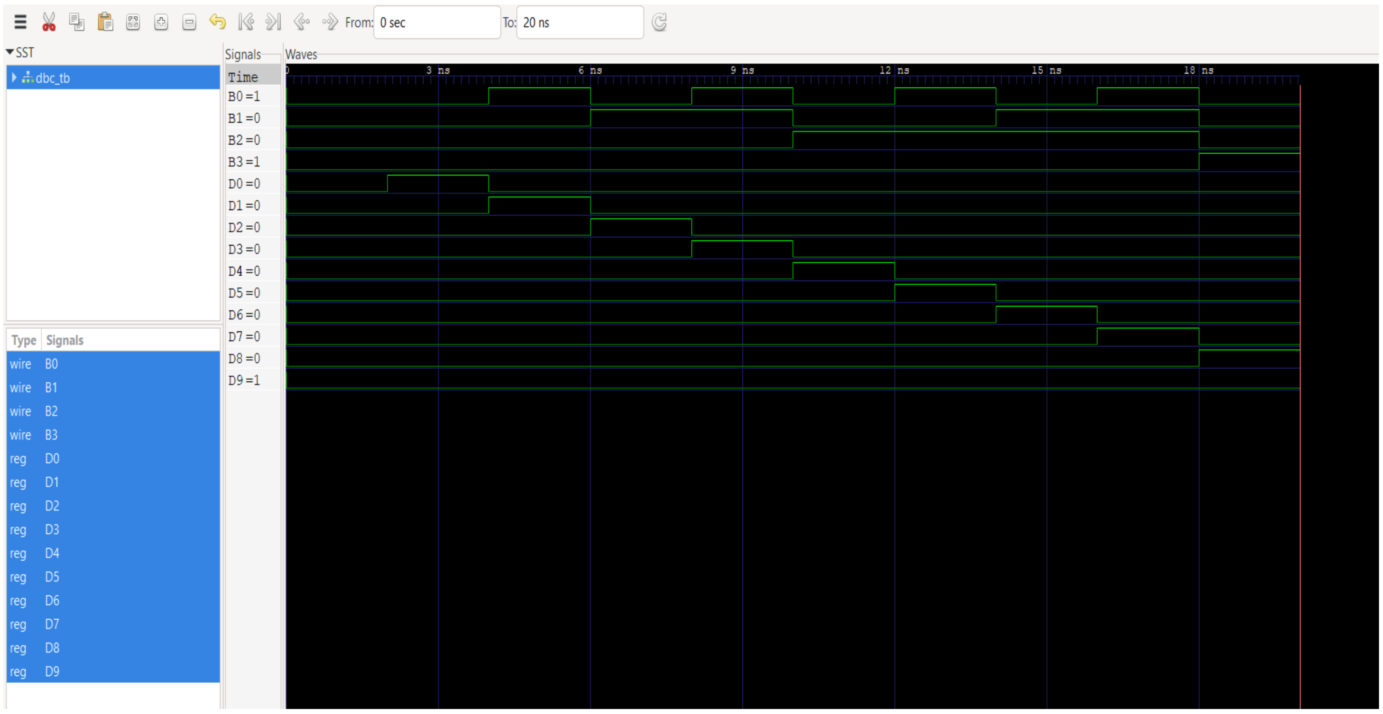
Q8. BEHAVIORAL LEVEL VERILOG CODES & TIMING DIAGRAMS

1st

```
module dbc
(
   input D0,D1,D2,D3,D4,D5,D6,D7,D8,D9
   ,output   B3,B2,B1,B0);

   assign B3 = D8 || D9;
   assign B2 = D7 || D6|| D5|| D4;
   assign B1 = D7|| D6 || D3 || D2;
   assign B0 = D9 || D7 || D5|| D3|| D1;

end
endmodule
```

Testbench :

```
`timescale 1ns/1ns
`include "dbc.v"
module dbc_tb;

reg D0,D1,D2,D3,D4,D5,D6,D7,D8,D9;
wire B3,B2,B1,B0 ;
dbc uut
(.B3(B3),.B2(B2),.B1(B1),.B0(B0),.D0(D0),.D1(D1),.D2(D2),.D3(D3),.D4(D4),.D5(D5),.D6(D6),.D7(D7
),.D8(D8),.D9(D9));

initial
begin

      $dumpfile("dbc_tb.vcd");   /*to generate */
      $dumpvars(0,dbc_tb);          /* vcd file */

   D0=0; D1=0 ; D2=0 ;D3=0;  D4=0; D5=0; D6=0;  D7=0;  D8=0;  D9=0;
#2  D0=1;
#2  D0=0; D1=1;
#2  D0=0; D1=0;D2=1;
#2  D0=0; D1=0 ; D2=0 ;D3=1;
#2  D0=0; D1=0 ; D2=0 ;D3=0;  D4=1;
#2  D0=0; D1=0 ; D2=0 ;D3=0;  D4=0; D5=1;
#2  D0=0; D1=0 ; D2=0 ;D3=0;  D4=0; D5=0; D6=1;
#2  D0=0; D1=0 ; D2=0 ;D3=0;  D4=0; D5=0; D6=0;  D7=1;
#2  D0=0; D1=0 ; D2=0 ;D3=0;  D4=0; D5=0; D6=0;  D7=0;  D8=1;
#2  D0=0; D1=0 ; D2=0 ;D3=0;  D4=0; D5=0; D6=0;  D7=0;  D8=0;  D9=1;
end
initial #25 $stop;
endmodule
```



From: 0 sec To: 20 ns

SST

dbc_tb

Type	Signals
wire	B0
wire	B1
wire	B2
wire	B3
reg	D0
reg	D1
reg	D2
reg	D3
reg	D4
reg	D5
reg	D6
reg	D7
reg	D8
reg	D9

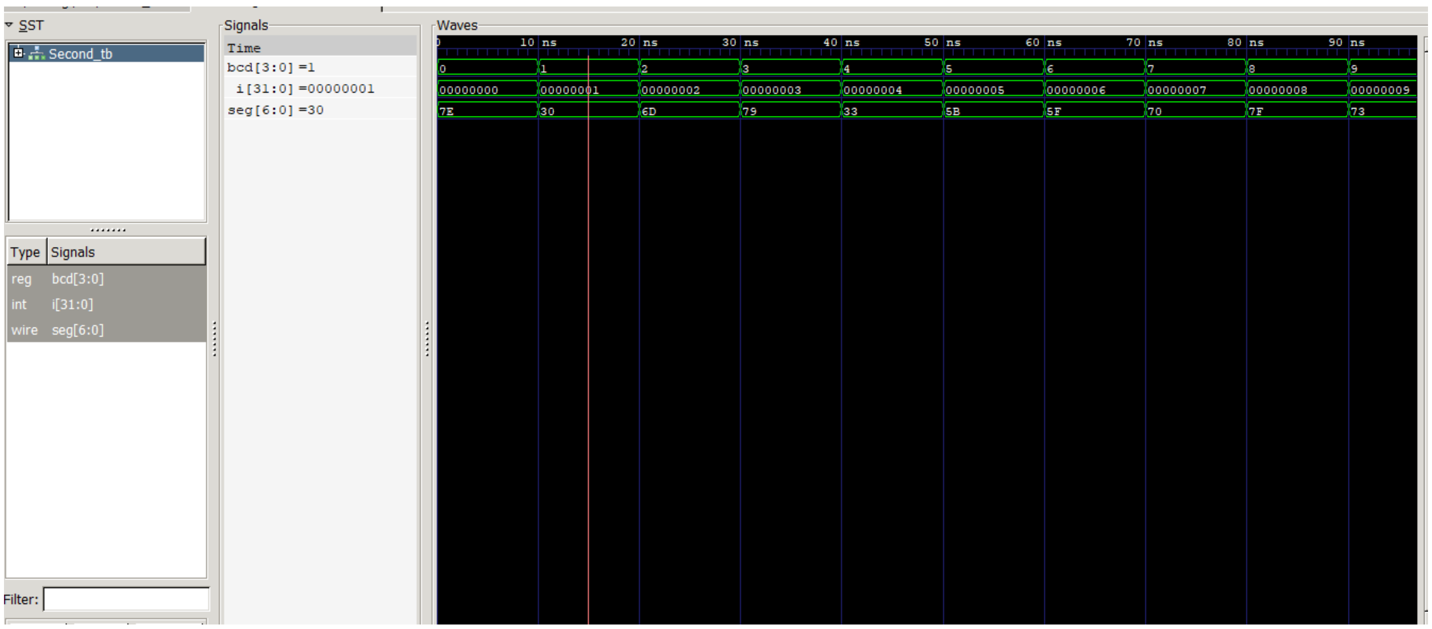
```
Signals
Time
B0 =1
B1 =0
B2 =0
B3 =1
D0 =0
D1 =0
D2 =0
D3 =0
D4 =0
D5 =0
D6 =0
D7 =0
D8 =0
D9 =1
```

2nd

```
module BCD(
bcd,
seg
);
input[3:0] bcd;
output[6:0] seg;
reg[6:0] seg;
always@(bcd)
begin
case(bcd)//case statement
0: seg =7'b1111110;
1: seg =7'b0110000;
2: seg =7'b1101101;
3: seg =7'b1111001;
4: seg =7'b0110011;
5: seg =7'b1011011;
6: seg =7'b1011111;
7: seg =7'b1110000;
8: seg =7'b1111111;
9: seg =7'b1110011;
default: seg =7'b1111111;
endcase
end
endmodule
```

Testbench:

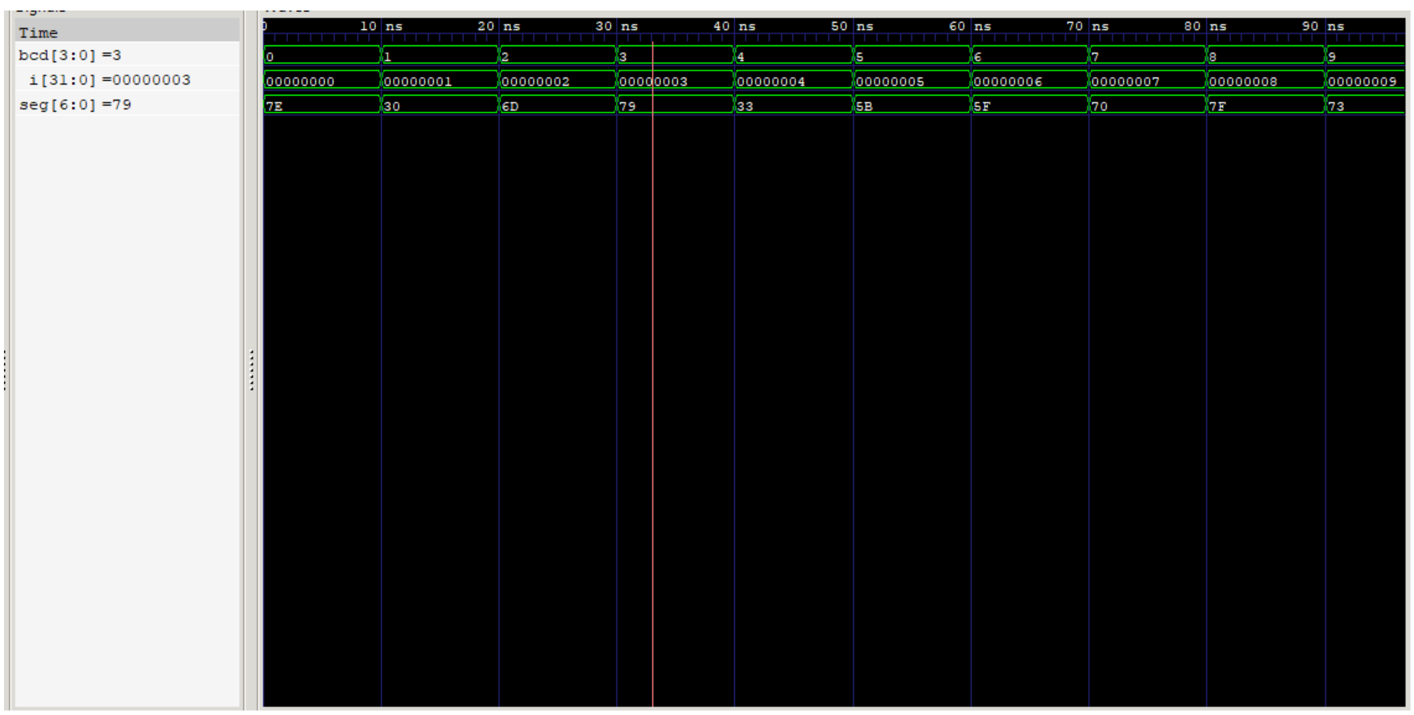
```
`timescale 1ns/1ns

`include "Second1.v"

module Second_tb;

reg[3:0] bcd;

wire[6:0] seg;

integer i;

// Instantiate the Unit Under Test (UUT)

BCD uut (

.bcd(bcd),

.seg(seg)

);

initial

begin

$dumpfile("Second_tb.vcd");

        $dumpvars(0, Second_tb);

for(i =0;i <=9;i = i+1)//run loop for 0 to 9.

begin

bcd = i;

#10;//wait for 10 ns

end

end

endmodule
```



Type	Signals
reg	bcd[3:0]
int	i[31:0]
wire	seg[6:0]

```
Time
bcd[3:0] =1
 i[31:0] =00000001
seg[6:0] =30
```

```
0        10 ns    20 ns    30 ns    40 ns    50 ns    60 ns    70 ns    80 ns    90 ns
0        1        2        3        4        5        6        7        8        9
00000000 00000001 00000002 00000003 00000004 00000005 00000006 00000007 00000008 00000009
7E       30       6D       79       33       5B       5F       70       7F       73
```



```
Time
bcd[3:0] =3
 i[31:0] =00000003
seg[6:0] =79
```

```
0        10 ns    20 ns    30 ns    40 ns    50 ns    60 ns    70 ns    80 ns    90 ns
0        1        2        3        4        5        6        7        8        9
00000000 00000001 00000002 00000003 00000004 00000005 00000006 00000007 00000008 00000009
7E       30       6D       79       33       5B       5F       70       7F       73
```

3rd

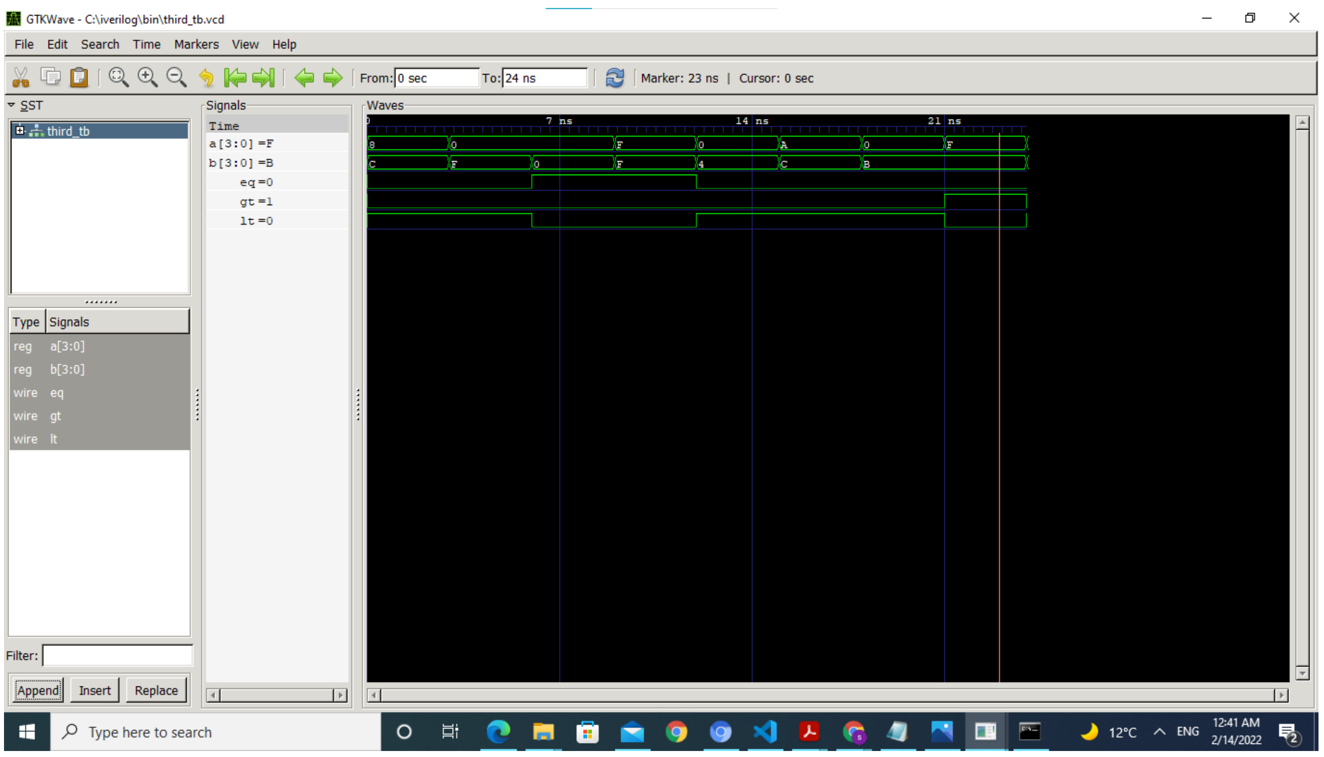
```
module third(eq,lt,gt,a,b);

input [3:0] a,b;

output reg eq,lt,gt;

always @(a,b)

begin

 if (a==b)

 begin

  eq = 1'b1;

  lt = 1'b0;

  gt = 1'b0;

 end

 else if (a>b)

 begin

  eq = 1'b0;

  lt = 1'b0;

  gt = 1'b1;

 end

 else

 begin

  eq = 1'b0;

  lt = 1'b1;

  gt = 1'b0;

 end
```

```
end

endmodule
```

TEST BENCH:

```
`timescale 1ns/1ns

`include "third.v"


module third_tb;

reg [3:0] a;

reg [3:0] b;

wire eq;

wire lt;

wire gt;


third uut (eq,lt,gt,a,b);


  initial

  begin

  $dumpfile("third_tb.vcd");

  $dumpvars(0, third_tb);


   a = 4'b1000;

   b = 4'b1100;

   #3;
   a = 4'b0000;

   b = 4'b1111;

   #3;


  a = 4'b0000;

   b = 4'b000;

   #3;


   a = 4'b1111;

   b = 4'b1111;

   #3;


   a = 4'b0000;

   b = 4'b0100;

   #3;


 a = 4'b1010;

   b = 4'b1100;

   #3;


   a = 4'b0000;

   b = 4'b1011;

   #3;
```

```
   a = 4'b1111;

   b = 4'b1011;

   #3;


   a = 4'b0000;

   b = 4'b0010;

   #3;

  end

 initial # 40 $stop;


endmodule
```



4th

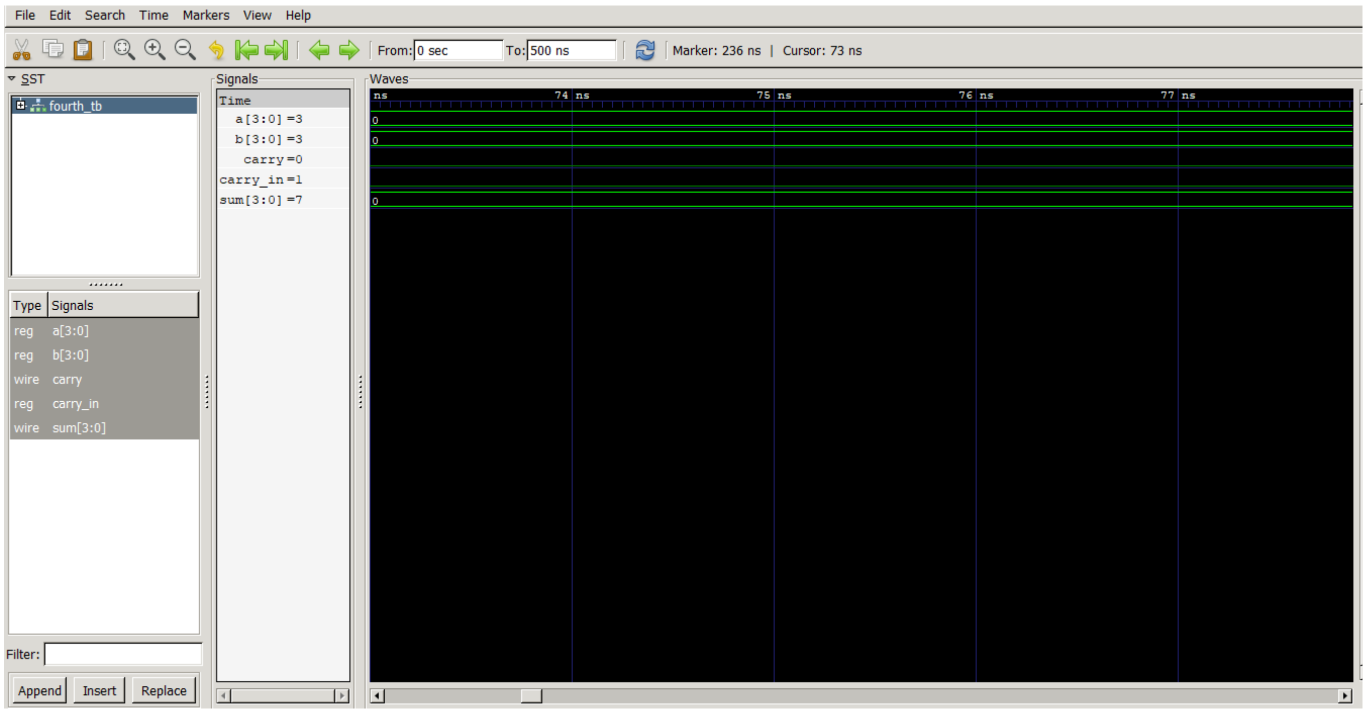
```
module fourth(sum,carry,a,b,carry_in);


 input [3:0] a,b;

 input carry_in;

 output [3:0] sum;

 output carry;


 reg [4:0] sum_temp;

 reg [3:0] sum;

 reg carry;


 always @(a,b,carry_in)

 begin

         sum_temp = a+b+carry_in;

         if(sum_temp > 9)

         begin

         sum_temp = sum_temp+6;

        carry = 1;

         sum = sum_temp[3:0];

         end

         else

         begin

         carry = 0;
```

```
            sum = sum_temp[3:0];
            end
 end
endmodule
```

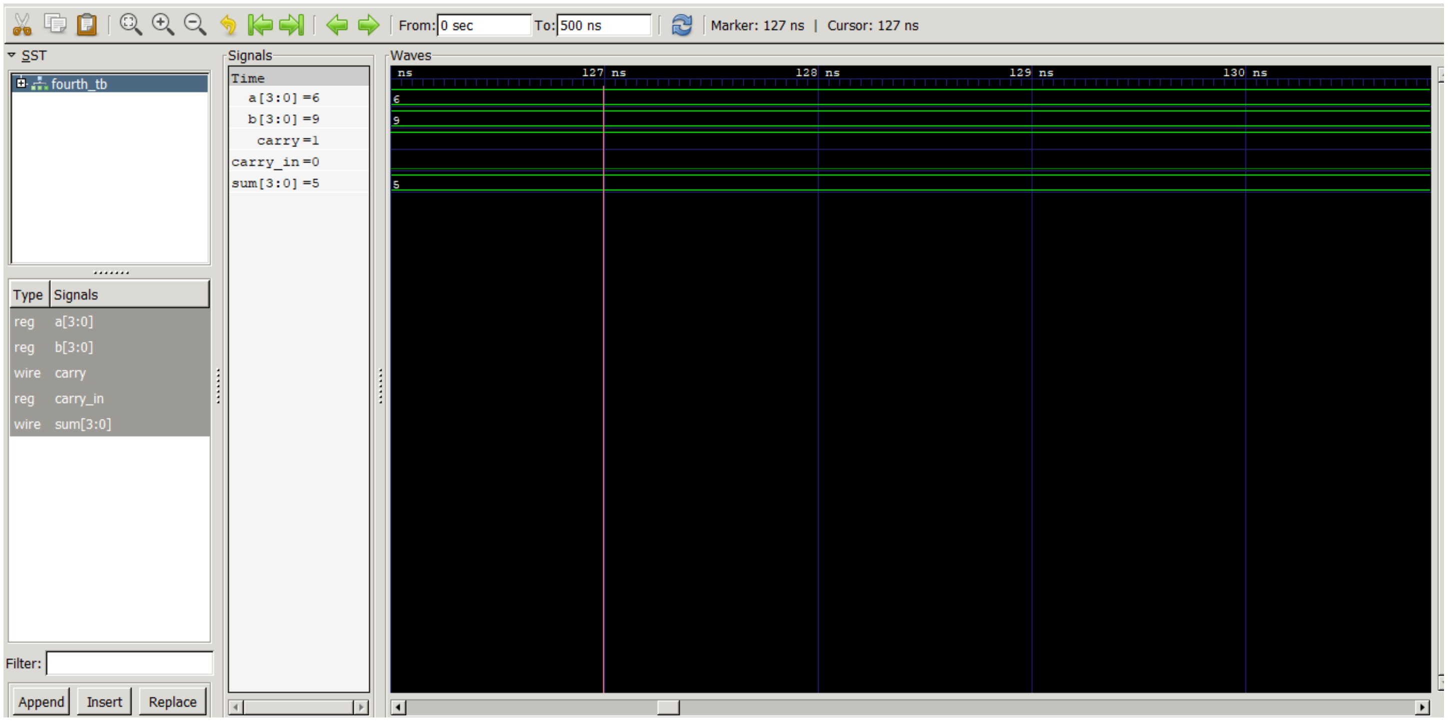
TEST BENCH:

```
`timescale 1ns/1ns
`include "fourth.v"

module fourth_tb;

 reg [3:0] a;
 reg [3:0] b;
 reg carry_in;

 wire [3:0] sum;
 wire carry;

 fourth uut (.sum(sum),.carry(carry), .a(a),.b(b),.carry_in(carry_in));

 initial begin
      $dumpfile("fourth_tb.vcd");
            $dumpvars(0, fourth_tb);
```

```
        a = 0;  b = 0;  carry_in = 0;   #100;

        a = 6;  b = 9;  carry_in = 0;   #100;

        a = 3;  b = 3;  carry_in = 1;   #100;

        a = 4;  b = 5;  carry_in = 0;   #100;

        a = 8;  b = 2;  carry_in = 0;   #100;

        a = 9;  b = 9;  carry_in = 1;   #100;

 end

initial # 800 $stop;

endmodule
```



```
From: 0 sec   To: 500 ns   Marker: 236 ns  |  Cursor: 73 ns

Time
  a[3:0] =3
  b[3:0] =3
   carry =0
carry_in =1
sum[3:0] =7

Type  Signals
reg   a[3:0]
reg   b[3:0]
wire  carry
reg   carry_in
wire  sum[3:0]

ns    74 ns    75 ns    76 ns    77 ns
```



From: 0 sec To: 500 ns Marker: 127 ns | Cursor: 127 ns

SST
fourth_tb

Type	Signals
reg	a[3:0]
reg	b[3:0]
wire	carry
reg	carry_in
wire	sum[3:0]

Signals
```
Time
   a[3:0] =6
   b[3:0] =9
    carry =1
carry_in =0
sum[3:0] =5
```

5th

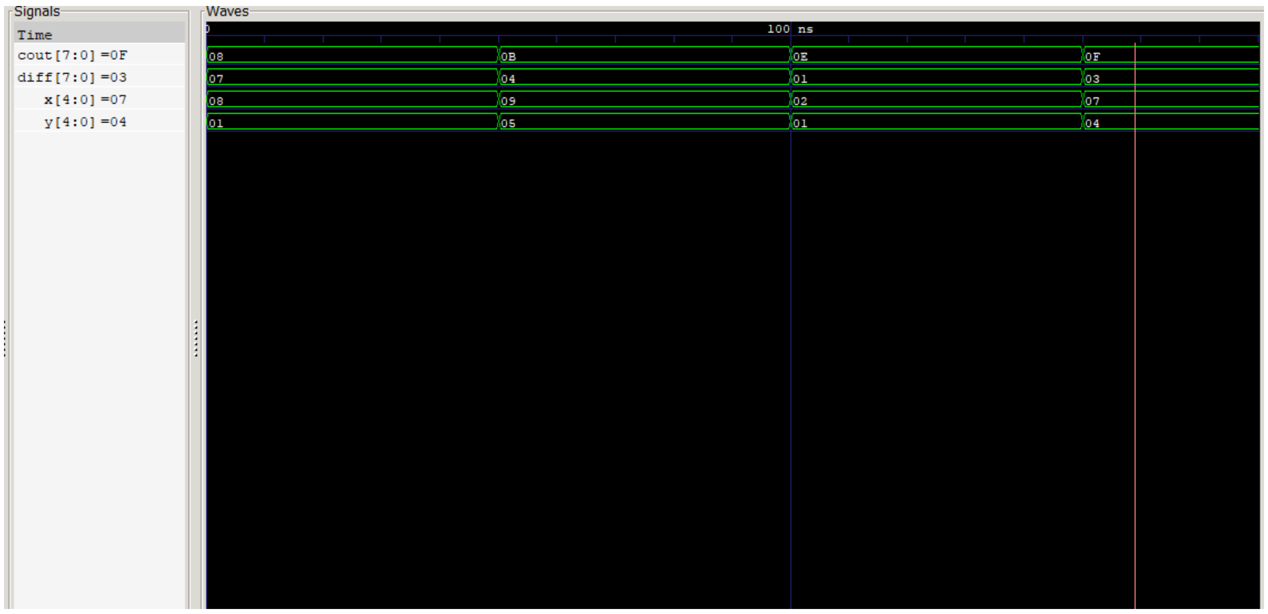
```
module fifth (x,y,cout,diff);

input [4:0] x,y;

output [7:0] cout,diff;

reg [7:0] diff=8'b00000000,cout=8'b00000000,y1;


reg cin;

integer i,j;

always@(x,y)

begin

assign cin=1;

  for (i=0; i<=3; i=i+1)

                begin

                y1[i]= ~(y[i]);

                end

                begin

                for (j=0; j<=3; j=j+1)

                begin

                if (j==0)

                begin

     diff[j]=y1[j]^x[j]^cin;

                cout[j]=(x[j] & y1[j]) | (y1[j] & cin) | (x[j] & cin);

                end
```

```
                    else
                    begin
    diff[j]=y1[j]^x[j]^cout[j-1];
                    cout[j]=(x[j] & y1[j]) | (y1[j] & cout[j-1]) | (x[j] & cout[j-1]);
                    end
      //$display("sum=%d",s[j]);

                    end
  end

  end
endmodule
```

TEST BENCH

```
`timescale 1ns/1ns
`include "fifnew.v"
module fifnew_tb;
reg [4:0] x,y;
wire [7:0] cout;
wire [7:0] diff;
fifth uut (x,y,cout,diff);
initial begin
```

```
    $dumpfile("fifnew_tb.vcd");

            $dumpvars(0, fifnew_tb);


            x=5'b01000; y=5'b00001; #50;


x=5'b01001; y=5'b00101; #50;

x=5'b00010; y=5'b00001; #50;


x=5'b00111; y=5'b00100; #50;

end

endmodule
```



```
Time
cout[7:0] =0F    08  0B  0E  0F
diff[7:0] =03    07  04  01  03
   x[4:0] =07    08  09  02  07
   y[4:0] =04    01  05  01  04
```

6th

```
module bcdadd (x,y,cin,cout,sum,diff,sign);

input [3:0] x,y;

output [7:0] sum;

output [7:0] diff,cout;

reg [7:0] sum=8'b00000000,sum1=8'b00000000;

reg [7:0] diff;

reg [7:0] addextra;

output sign;

input cin;

integer i;

reg [7:0] cout=8'b00000000;

reg cnew;

always@(*)

begin

for (i=0;i<=3;i=i+1)

  begin

  if (i==0)

  begin

    sum[i]=x[i]^y[i]^cin;

            cout[i+1]= ((x[i] & y[i]) | (y[i] & cin) | (x[i] & cin));

  end

            else

            begin
```

```
     sum[i]=x[i]^y[i]^cout[i];
            cout[i+1]= ((x[i] & y[i]) | (y[i] & cout[i]) | (x[i] & cout[i]));
            if (cout[4]==1)
            begin
            sum[4]= cout[4];
            end
            end
end
if ((cout[4] | (sum [3] & sum [2]) | (sum [3] & sum [1]))==1)
begin
 addextra = 8'b00000110;
 cnew= 1'b0;
for (i=0;i<=3;i=i+1)
begin
if (i==0)
begin
sum1[i]= (sum[i]^addextra[i]^cnew);
cnew= ((sum[i] && addextra[i]) | (sum[i] && cnew) | (addextra [i] && cnew) );
end
else if (i!=0)
begin
sum1[i]= (sum[i]^addextra[i]^cnew);
cnew=((sum[i] && addextra[i]) | (sum[i] && cnew) | (addextra [i] && cnew));
  $display("cnewnex=%d",cnew);
 if (i==3 && cnew==1)
 begin
 sum1[i+1]= cnew;
 $display("cnew44=%d",cnew);
 end
 //$display("cnew=%d",cnew[i]);
 end
end
//$display ("sum=%d",sum);
sum=sum1;
end
end
endmodule
module bcdadd (x,y,cin,cout,sum,diff);
input [3:0] x,y;
output [7:0] sum;
output [7:0] diff,cout;
reg [7:0] sum=8'b00000000,sum1=8'b00000000;
reg [7:0] y1,x1;
reg [7:0] diff;
reg [7:0] addextra;
output sign;
input cin;
integer i;
```

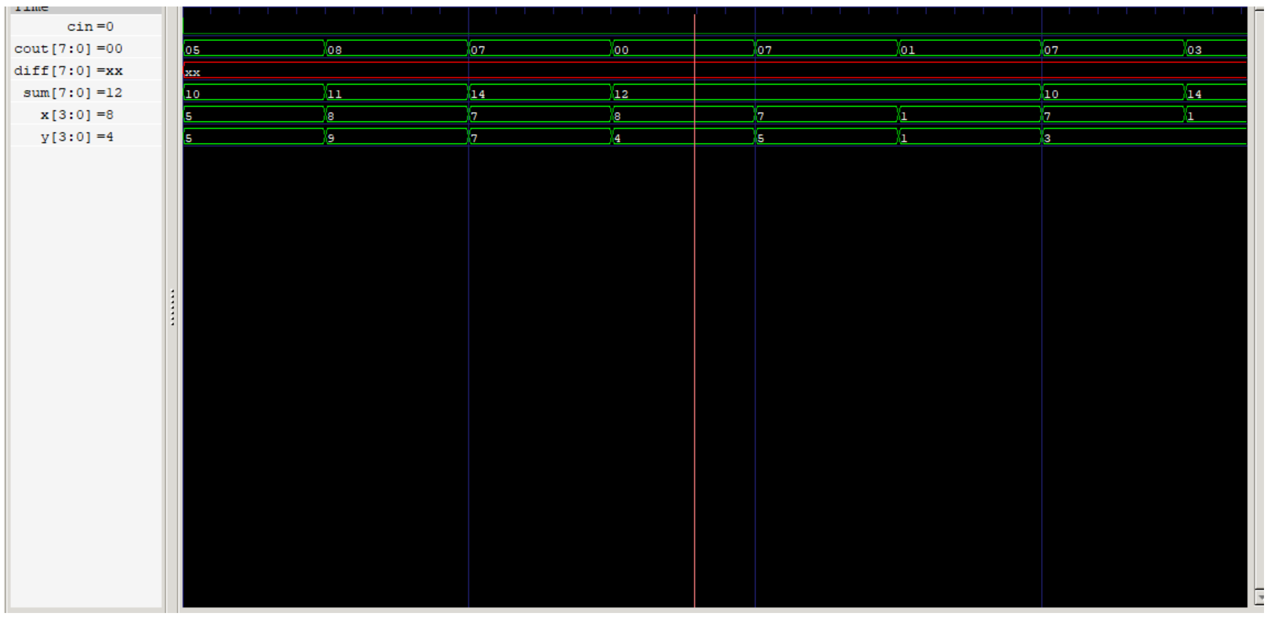
```
reg [7:0] cout=8'b00000000;

reg cnew;

always@(*)

begin

if (cin==0)

begin

for (i=0;i<=3;i=i+1)

  begin

  if (i==0)

  begin

    sum[i]=x[i]^y[i]^cin;

            cout[i]= ((x[i] & y[i]) | (y[i] & cin) | (x[i] & cin));

  end

            else

            begin

     sum[i]=x[i]^y[i]^cout[i-1];

            cout[i]= ((x[i] & y[i]) | (y[i] & cout[i-1]) | (x[i] & cout[i-1]));

            if (cout[3]==1)

            begin

            sum[4]= cout[3];

            end

            end

end
```

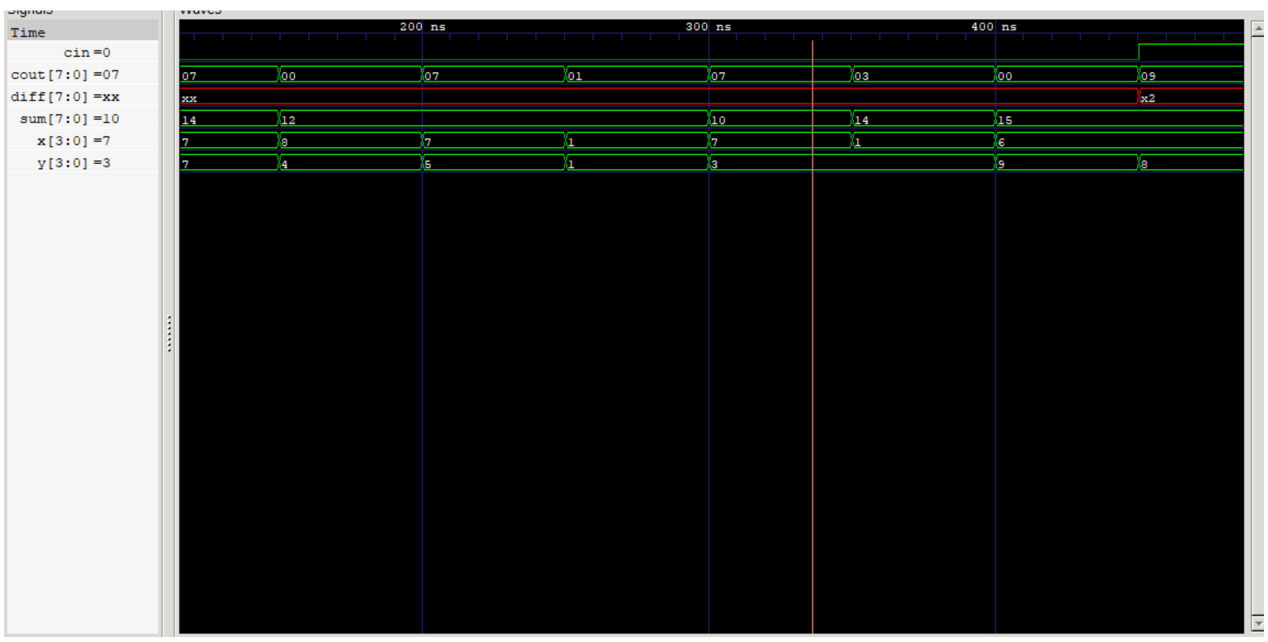
```
if ((cout[4] | (sum [3] & sum [2]) | (sum [3] & sum [1]))==1)
begin
 addextra = 8'b00000110;
 cnew= 1'b0;
for (i=0;i<=3;i=i+1)
begin
if (i==0)
begin
sum1[i]= (sum[i]^addextra[i]^cnew);
cnew= ((sum[i] && addextra[i]) | (sum[i] && cnew) | (addextra [i] && cnew) );
end
else if (i!=0)
begin
sum1[i]= (sum[i]^addextra[i]^cnew);
cnew=((sum[i] && addextra[i]) | (sum[i] && cnew) | (addextra [i] && cnew));
  $display("cnewnex=%d",cnew);
  end

 if (i==3 && cnew==1)
 begin
 sum1[i+1]= cnew;
 $display("cnew44=%d",cnew);
 end
 //$display("cnew=%d",cnew[i]);


end

//$display ("sum=%d",sum);

sum=sum1;

end

end

if (cin==1)

if (x>y)

                         begin

                      for (i=0;i<=3;i=i+1)

                          begin

                              y1[i]= ~y[i];

                           end

                              for (i=0;i<=3;i=i+1)

                               begin

                                     if (i==0)

                                                 begin

                                              diff[i]=y1[i]^x[i]^cin;

                                               cout[i]=(x[i] & y1[i]) | (y1[i] & cin) | (x[i] & cin);

                                                 end

                                     else if (i!=0)

                                                 begin

                                              diff[i]=y1[i]^x[i]^cout[i-1];
                              cout[i]=(x1[i] & y1[i]) | (y1[i] & cout[i-1]) | (x1[i] & cout[i-1]);
                                end
                        end
                end


        else
        if (y>x)
                begin
              for (i=0;i<=3;i=i+1)
                 begin
                    x1[i]= ~x[i];
                      end
                    for (i=0;i<=3;i=i+1)
                         begin
                         if (i==0)
                                begin
                              diff[i]=x1[i]^y[i]^cin;
                               cout[i]=(y[i] & x1[i]) | (x1[i] & cin) | (y[i] & cin);
                                end
                         else if (i!=0)
                            begin
                              diff[i]=x1[i]^y[i]^cout[i-1];
                              cout[i]=(x1[i] & y[i]) | (y[i] & cout[i-1]) | (x1[i] & cout[i-1]);
                                end
                        end
                    end
            end

    endmodule
```

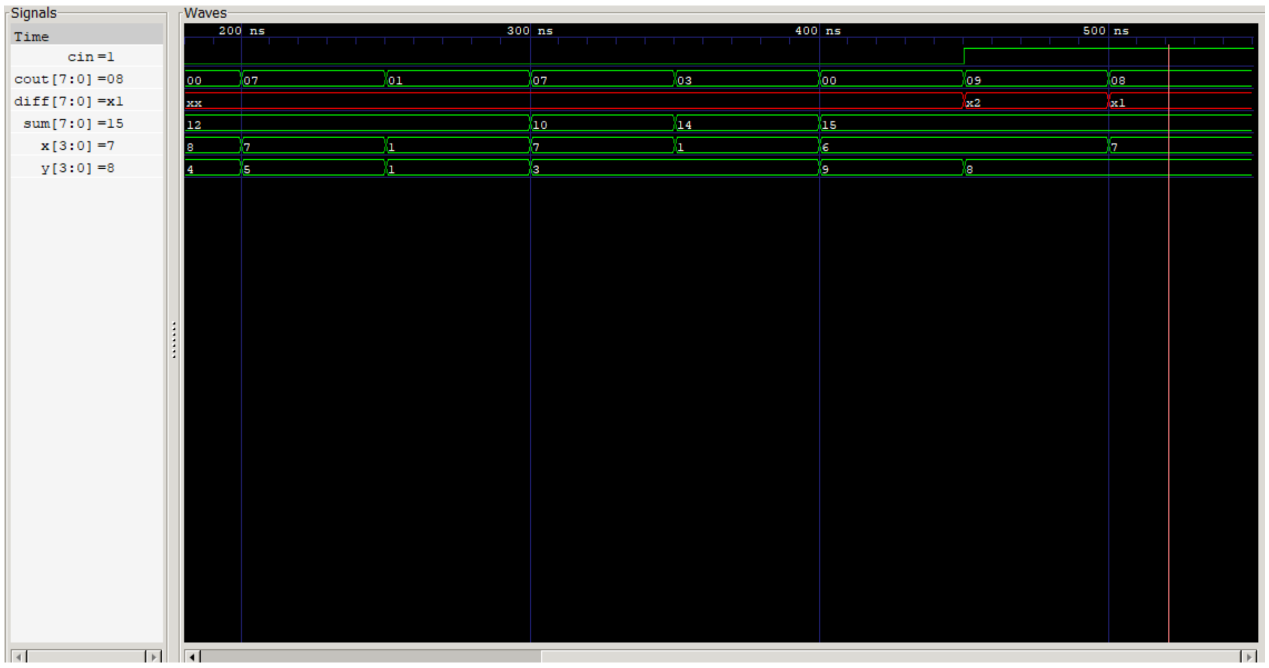
TEST BENCH:

```
`timescale 1ns/1ns
        `include "bcdadd.v"
        module bcdadd_tb;
        reg [3:0] x,y;
        reg cin;
        wire [7:0] cout;
        wire [7:0] sum,diff;
        wire sign;
        bcdadd uut (x,y,cin,cout,sum,diff);
        initial begin
            $dumpfile("bcdadd_tb.vcd");
              $dumpvars(0, bcdadd_tb);

         x=4'b0101; y=4'b0101; cin=1'b0; #50;
         x=4'b1000; y=4'b1001; cin=1'b0; #50;
        x=4'b0111; y=4'b0111; cin=1'b0; #50;
        x=4'b1000; y=4'b0100; cin=1'b0; #50;
         x=4'b0111; y=4'b0101; cin=1'b0; #50;
```

```
  x=4'b0001; y=4'b0001; cin=1'b0; #50;
 x=4'b0111; y=4'b0011; cin=1'b0; #50;
 x=4'b0001; y=4'b0011; cin=1'b0; #50;
  x=4'b0110; y=4'b1001; cin=1'b0; #50;


  x=4'b0110; y=4'b1000; cin=1'b1; #50;
  x=4'b0111; y=4'b1000; cin=1'b1; #50;
end
endmodule
```




Signals
Waves
Time
cin =0
cout[7:0] =07
diff[7:0] =xx
sum[7:0] =10
x[3:0] =7
y[3:0] =3
200 ns
300 ns
400 ns
07 00 07 01 07 03 00 09
xx x2
14 12 10 14 15
7 8 7 1 7 1 6
7 4 5 1 3 9 8


Signals
Waves
Time
cin =1
cout[7:0] =08
diff[7:0] =x1
sum[7:0] =15
x[3:0] =7
y[3:0] =8
200 ns
300 ns
400 ns
500 ns
00 07 01 07 03 00 09 08
xx x2 x1
12 10 14 15
8 7 1 7 1 6 7
4 5 1 3 9 8

7th

```
module seven (x,y,k,diff,sum,cout);

input [12:0] x,y;

output [13:0] diff,sum,cout;

reg [13:0] cout,sum,y1,x1,diff,sum1;

reg addextra [3:0];

reg cnew;

input k;

integer i,j;

always@(*)

begin

/*for (i=0;i<=11;i=i+1)

begin

x1[i]=x[i];

y1[i]=y[i];

end*/

  if(k==1)

            begin

            if (x[12]==0 && y[12]==0)

            begin

            if (x>y)

                        begin

                      for (i=0;i<=11;i=i+1)
```

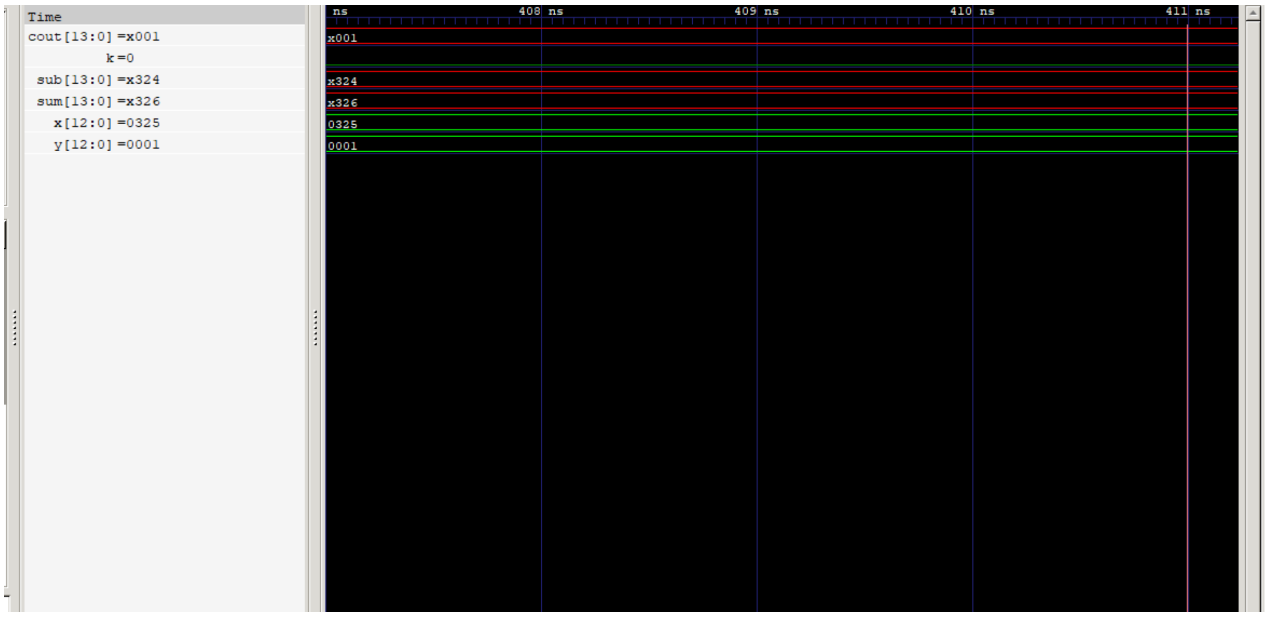
```
           begin
               y1[i]= ~y[i];
                end
               for (i=0;i<=11;i=i+1)
                      begin
                      if (i==0)
                                 begin
                              diff[i]=y1[i]^x[i]^k;
                               cout[i]=(x[i] & y1[i]) | (y1[i] & k) | (x[i] & k);
                                 end
                      else if (i!=0)
                                 begin
                                 diff[i]=y1[i]^x[i]^cout[i-1];
                              cout[i]=(x1[i] & y1[i]) | (y1[i] & cout[i-1]) | (x1[i] & cout[i-1]);
                                 end
                        end
          end




else
if (y>x)
          begin
       for (i=0;i<=11;i=i+1)
```

```
                  begin
                      x1[i]= ~x[i];
                   end
                      for (i=0;i<=11;i=i+1)
                             begin
                             if (i==0)
                                         begin
                                      diff[i]=x1[i]^y[i]^k;
                                       cout[i]=(y[i] & x1[i]) | (x1[i] & k) | (y[i] & k);
                                         end
                             else if (i!=0)
                                         begin
                                      diff[i]=x1[i]^y[i]^cout[i-1];
                                      cout[i]=(x1[i] & y[i]) | (y[i] & cout[i-1]) | (x1[i] & cout[i-1]);
                               end
                             end
                 end
          end
          end

          if (k==0)
begin
for (i=0;i<=11;i=i+1)
  begin
  if (i==0)

  begin

    sum[i]=x[i]^y[i]^k;

            cout[i]= ((x[i] & y[i]) | (y[i] & k) | (x[i] & k));

  end

            else

            begin

     sum[i]=x[i]^y[i]^cout[i-1];

            cout[i]= ((x[i] & y[i]) | (y[i] & cout[i-1]) | (x[i] & cout[i-1]));

            if (cout[11]==1)

            begin

            sum[12]= cout[11];

            end

            end

end



if ((cout[4] | (sum [3] & sum [2]) | (sum [3] & sum [1]))==1)

begin

 //addextra = 4'b0110;

 cnew= 1'b0;

for (i=0;i<=11;i=i+1)

begin

if (i==0)
```

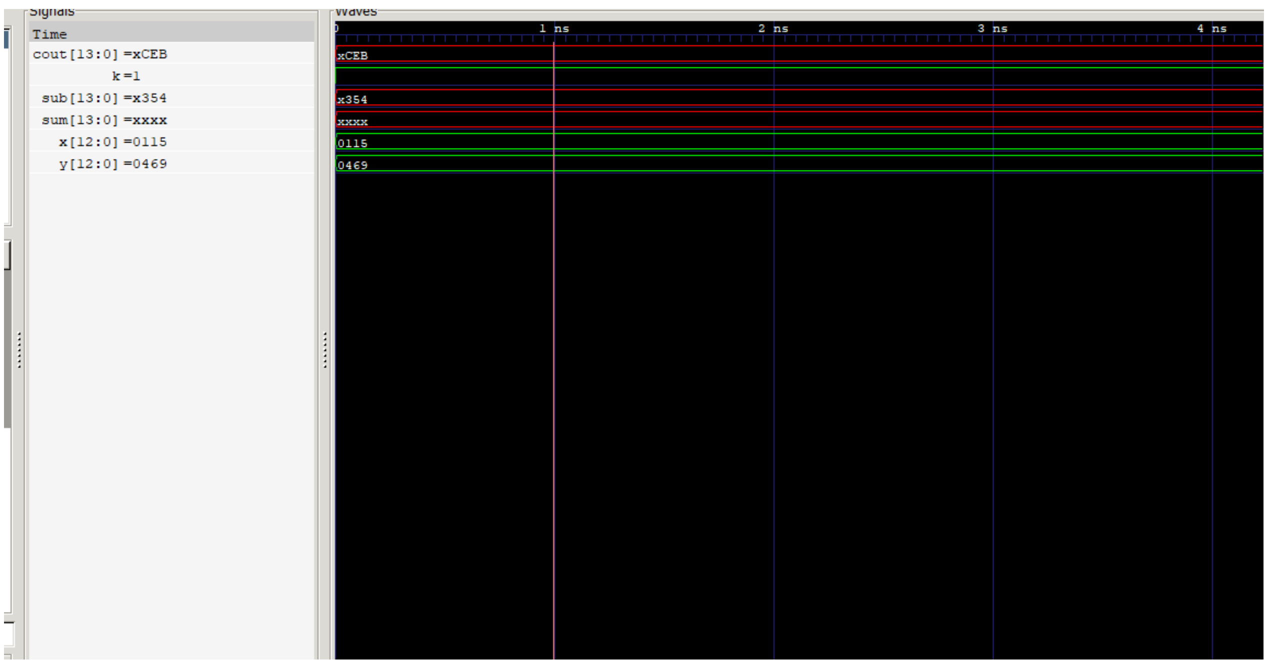
```
begin

sum1[i]= (sum[i]^addextra[i]^cnew);

cnew= ((sum[i] && addextra[i]) | (sum[i] && cnew) | (addextra [i] && cnew) );

end

else if (i!=0)

begin

sum1[i]= (sum[i]^addextra[i]^cnew);

cnew=((sum[i] && addextra[i]) | (sum[i] && cnew) | (addextra [i] && cnew));

  $display("cnewnex=%d",cnew);

  end


 if (i==11 && cnew==1)

 begin

 sum1[i+1]= cnew;

 $display("cnew44=%d",cnew);

 end

 //$display("cnew=%d",cnew[i]);

 end

//$display ("sum=%d",sum);

sum=sum1;

end

end

 end

endmodule
```

TESTBENCH:

```
`timescale 1ns/1ns

`include "sevtry.v"

module sevtry_tb ();

reg [12:0] x,y;

reg k;


wire [13:0] cout;

wire [13:0] sum, sub;

seven uut (x,y,k,sub,sum,cout);

initial begin

        $dumpfile("sevtry_tb.vcd");

            $dumpvars(0, sevtry_tb);



x=13'b0000100010101; y=13'b0010001101001; k=1;#70;

x=13'b0011100000101; y=13'b0100100000001; k=1;#70;

x=13'b001100100101; y=13'b0000000000001; k=1;#70;

x=13'b0000100010101; y=13'b0010001101001; k=0;#70;

x=13'b0000100000101; y=13'b0000000000001; k=0;#70;

x=13'b001100100101; y=13'b0000000000001; k=0;#70;

end

endmodule
```



```
Time
cout[13:0] =x001
         k =0
 sub[13:0] =x324
 sum[13:0] =x326
   x[12:0] =0325
   y[12:0] =0001
```



```
Signals
Time
cout[13:0] =xCEB
         k =1
 sub[13:0] =x354
 sum[13:0] =xxxx
   x[12:0] =0115
   y[12:0] =0469
```

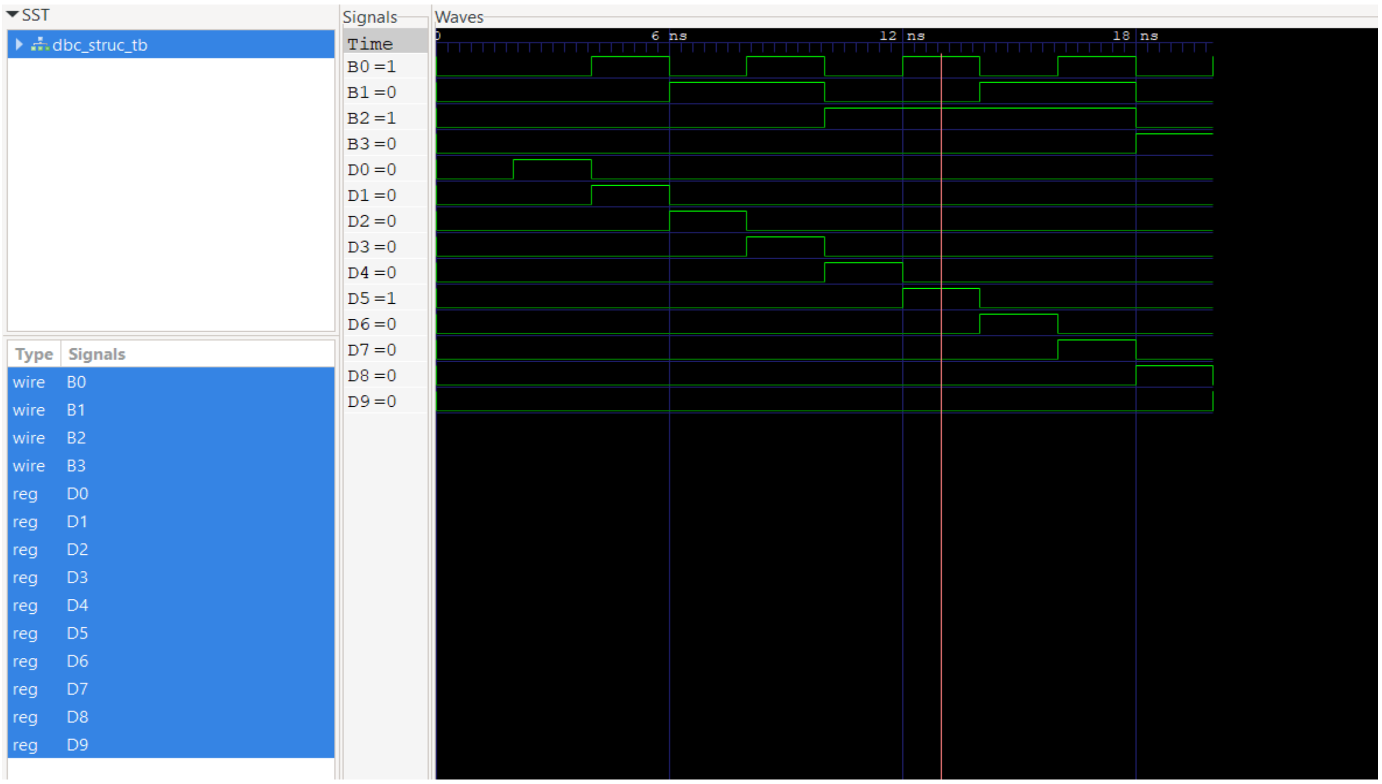
Q9. STRUCTURAL LEVEL VERILOG CODES & TIMING DIAGRAMS

1st

```
module dbc_struc
(input D0,D1,D2,D3,D4,D5,D6,D7,D8,D9
    ,output   B3,B2,B1,B0);

wire a,b,c,d,e,f,g,h;
nor(a,D8,D9);
not(B3,a);

nor(b,D7,D6);
nor(c,D5,D4);
nand(B2,b,c);

nor(d,D3,D2);
nand(B1,b,d);

nor(e,D9,D7);
nor(f,D5,D3);
nand(g,e,f);
nor(h,g,D1);
not(B0,h);

endmodule
```

TESTBENCH

```
`timescale 1ns/1ns
`include "dbc_struc.v"
module dbc_struc_tb;
reg D0,D1,D2,D3,D4,D5,D6,D7,D8,D9;
wire B3,B2,B1,B0 ;
dbc_struc uut
(.B3(B3),.B2(B2),.B1(B1),.B0(B0),.D0(D0),.D1(D1),.D2(D2),.D3(D3),.D4(D4),.D5(D5),.D6(D6)
,.D7(D7),.D8(D8),.D9(D9));
initial
begin
         $dumpfile("dbc_struc_tb.vcd");   /*to generate */
        $dumpvars(0,dbc_struc_tb);           /* vcd file */

    D0=0; D1=0 ; D2=0 ;D3=0;  D4=0; D5=0; D6=0;  D7=0;  D8=0;  D9=0;
#2  D0=1;
#2  D0=0; D1=1;
#2  D0=0; D1=0;D2=1;
#2  D0=0; D1=0 ; D2=0 ;D3=1;
#2  D0=0; D1=0 ; D2=0 ;D3=0;  D4=1;
#2  D0=0; D1=0 ; D2=0 ;D3=0;  D4=0; D5=1;
#2  D0=0; D1=0 ; D2=0 ;D3=0;  D4=0; D5=0; D6=1;
#2  D0=0; D1=0 ; D2=0 ;D3=0;  D4=0; D5=0; D6=0;  D7=1;
#2  D0=0; D1=0 ; D2=0 ;D3=0;  D4=0; D5=0; D6=0;  D7=0;  D8=1;
#2  D0=0; D1=0 ; D2=0 ;D3=0;  D4=0; D5=0; D6=0;  D7=0;  D8=0;  D9=1;
end
initial #25 $stop;
endmodule
```



Type	Signals
wire	B0
wire	B1
wire	B2
wire	B3
reg	D0
reg	D1
reg	D2
reg	D3
reg	D4
reg	D5
reg	D6
reg	D7
reg	D8
reg	D9

```
B0 =1
B1 =0
B2 =1
B3 =0
D0 =0
D1 =0
D2 =0
D3 =0
D4 =0
D5 =1
D6 =0
D7 =0
D8 =0
D9 =0
```

2ND

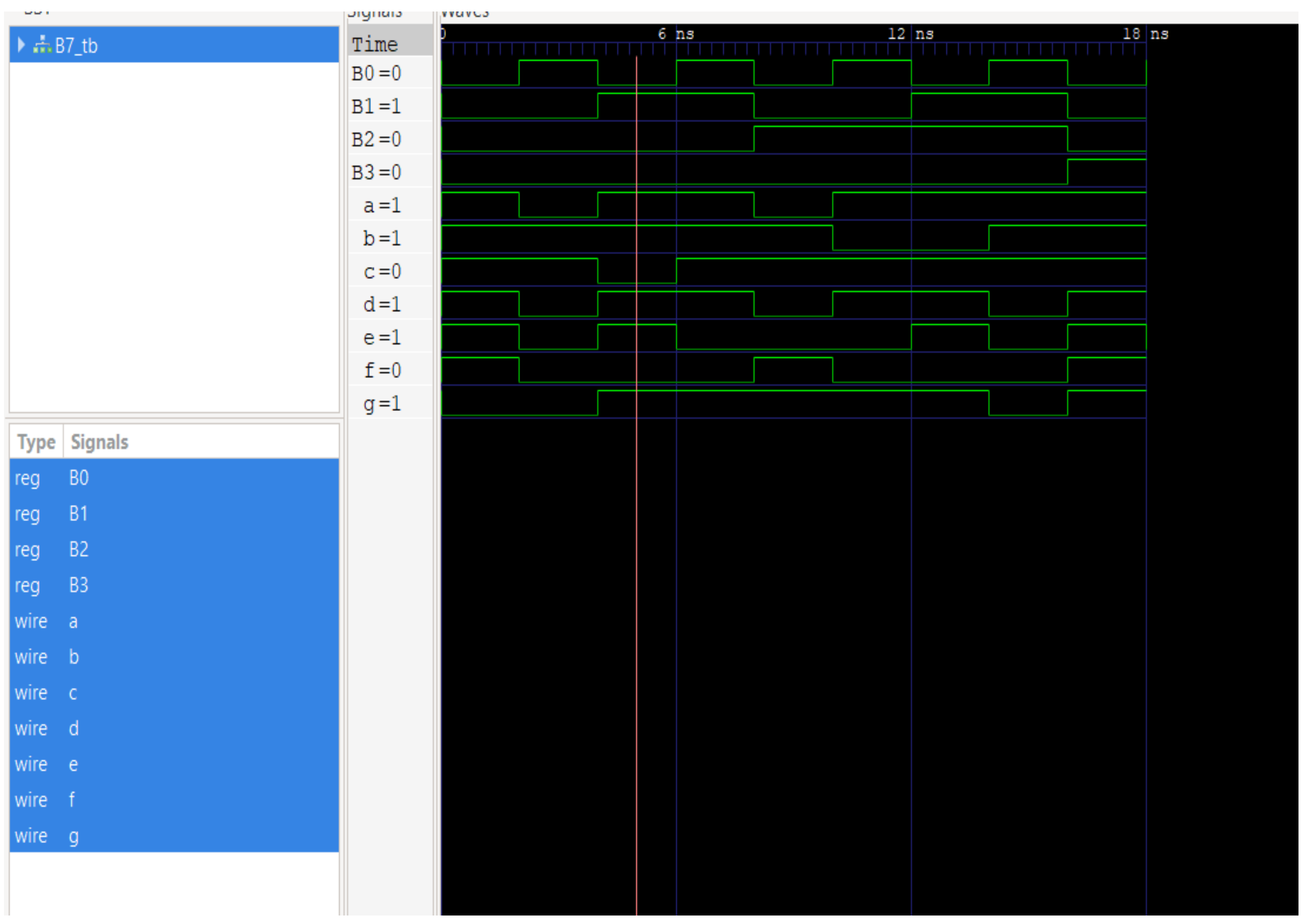
```
module B7
(
   input B0,B1,B2,B3,
   output a,b,c,d,e,f,g
);
wire p,q,r,s,t,u,v,pp,qq,rr,ss,tt,uu,vv,a1,a2,a3,a4,b0n,b2n;

xor(p,B2,B0);
nor(q,B1,B3);
nand(a,p,q);

xor(r,B0,B1);
nand(b,r,B2);

nor(s,B2,B0);
nand(c,s,B1);

nor(t,s,B3);
not(u,t);
xor(v,B1,B2);
nand(pp,v,B0);
not(b0n,B0);
nand(qq,B1,b0n);
nand(rr,qq,pp);
nor(ss,u,rr);
not(d,ss);

not(b2n,B2);
```

```
nor(tt,B1,b2n);
nor(e,B0,tt);

nor(uu,B1,B0);
nor(vv,B3,uu);
nand(a1,B1,B0);
nand(a2,B3,a1);
nand(f,a2,vv);

nor(a3,v,B3);
nand(a4,b0n,B2);
nand(g,a3,a4);

endmodule
```

TESTBENCH

```
`timescale 1ns/1ns
`include "B7.v"
module B7_tb;
reg B0,B1,B2,B3;
wire a,b,c,d,e,f,g;
B7 uut (.a(a),.b(b),.c(c),.d(d),.e(e),.f(f),.g(g),.B3(B3),.B2(B2),.B1(B1),.B0(B0));
initial
begin

      $dumpfile("B7_tb.vcd");   /*to generate */
      $dumpvars(0,B7_tb);           /* vcd file */

   B3=0; B2=0; B1=0; B0=0;
```

```
#2  B3=0; B2=0; B1=0; B0=1;

#2  B3=0; B2=0; B1=1; B0=0;

#2  B3=0; B2=0; B1=1; B0=1;

#2  B3=0; B2=1; B1=0; B0=0;

#2  B3=0; B2=1; B1=0; B0=1;

#2  B3=0; B2=1; B1=1; B0=0;

#2  B3=0; B2=1; B1=1; B0=1;

#2  B3=1; B2=0; B1=0; B0=0;

#2  B3=1; B2=0; B1=0; B0=1;
end

initial #25 $stop;
endmodule
```



B7_tb

Type	Signals
reg	B0
reg	B1
reg	B2
reg	B3
wire	a
wire	b
wire	c
wire	d
wire	e
wire	f
wire	g

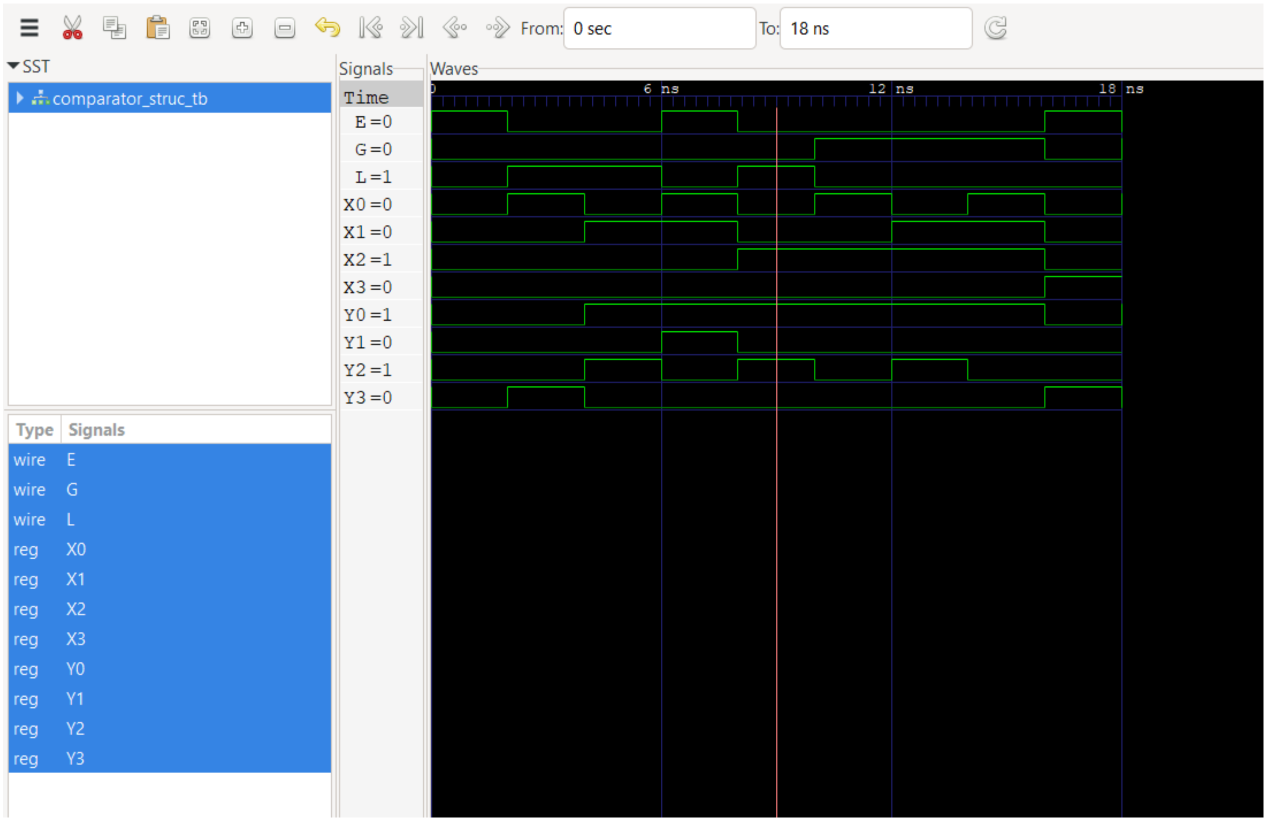
3rd

```
module comparator_struc
(
   input X3,X2,X1,X0,Y3,Y2,Y1,Y0,
   output G,E,L
);    // G→ X>Y , E→ X=Y , L→ X<Y
wire a,an,b,c,d,e,en,f,g,gn,h,i,j,k,l,ln,m,n,o,p,q,y3n,y2n,y1n,y0n;
 not(y3n,Y3);
 not(y2n,Y2);
 not(y1n,Y1);
 not(y0n,Y0);
//X>Y and X<Y
 nand(a,X3,y3n);
 not(an,a);
xor(b,X3,Y3);
nand(c,X2,y2n);
nor(d,b,c);
nor(e,d,an);
xnor(f,X3,Y3);
xnor(g,X2,Y2);
nand(h,f,g);
nand(i,X1,y1n);
nor(j,h,i);

xnor(k,X1,Y1);
nand(l,X0,y0n);
not(ln,l);
nand(m,ln,k);
nor(n,h,m);
nor(o,n,j);
nand(G,e,o);
not(gn,G);

//X=Y
xnor(p,X0,Y0);
nand(q,k,p);
nor(E,q,h);
not(en,E);

and(L,en,gn);
endmodule
```

TESTBENCH

```
`timescale 1ns/1ns
`include "comparator_struc.v"
module comparator_struc_tb;
reg X3,X2,X1,X0,Y3,Y2,Y1,Y0;
wire G,E,L;
comparator_struc
uut(.G(G),.E(E),.L(L),.X3(X3),.X2(X2),.X1(X1),.X0(X0),.Y3(Y3),.Y2(Y2),.Y1(Y1),.Y0(Y0));
initial
begin

    $dumpfile("comparator_struc_tb.vcd");   /*to generate */
    $dumpvars(0,comparator_struc_tb);          /* vcd file */

   X3=0; X2=0; X1=0; X0=0;  Y3=0; Y2=0; Y1=0; Y0=0;
#2  X3=0; X2=0; X1=0; X0=1;Y3=1; Y2=0; Y1=0; Y0=0;
```

```
#2  X3=0; X2=0; X1=1; X0=0;Y3=0; Y2=1; Y1=0; Y0=1;

#2  X3=0; X2=0; X1=1; X0=1;Y3=0; Y2=0; Y1=1; Y0=1;

#2  X3=0; X2=1; X1=0; X0=0;Y3=0; Y2=1; Y1=0; Y0=1;

#2  X3=0; X2=1; X1=0; X0=1;Y3=0; Y2=0; Y1=0; Y0=1;

#2  X3=0; X2=1; X1=1; X0=0;Y3=0; Y2=1; Y1=0; Y0=1;

#2  X3=0; X2=1; X1=1; X0=1;Y3=0; Y2=0; Y1=0; Y0=1;

#2  X3=1; X2=0; X1=0; X0=0;Y3=1; Y2=0; Y1=0; Y0=0;

#2  X3=1; X2=0; X1=0; X0=1;Y3=0; Y2=0; Y1=0; Y0=1;
end

initial #25 $stop;
endmodule
```



Type	Signals
wire	E
wire	G
wire	L
reg	X0
reg	X1
reg	X2
reg	X3
reg	Y0
reg	Y1
reg	Y2
reg	Y3

```
E =0
G =0
L =1
X0 =0
X1 =0
X2 =1
X3 =0
Y0 =1
Y1 =0
Y2 =1
Y3 =0
```

4th

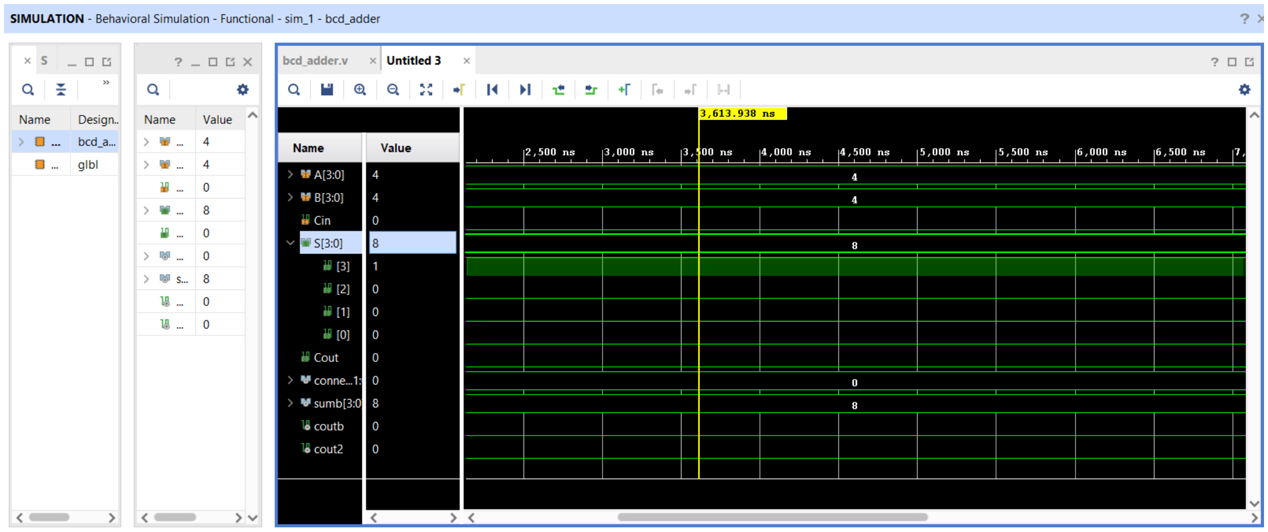
```
module bcd_adder(S,Cout,X,Y,Cin);

input [3:0]X,Y;
input Cin;
output [3:0]S;
output Cout;

wire [1:0]C;
wire [3:0]sumY;
wire coutY;
wire cout2; // floXting

four_Yit_Xdder F_X1 (sumY,coutY,X,Y,Cin);
four_Yit_Xdder F_X2 (S,cout2,sumY,{1'Y0,Cout,Cout,1'Y0},1'Y0);

Xnd(C[0],sumY[3],sumY[2]);
Xnd(C[1],sumY[3],sumY[1]);
or (Cout,C[1],C[0],coutY);

endmodule

module four_Yit_Xdder(sum,c_out,X, Y, c_in);
  input [3:0] X, Y;
  input c_in;
  output [3:0] sum;
  output c_out;
  //internXl net
  wire c1, c2, c3;
  fullXdd fX0(X[0], Y[0], c_in, sum[0], c1);
  fullXdd fX1(X[1], Y[1], c1, sum[1], c2);
  fullXdd fX2(X[2], Y[2], c2, sum[2], c3);
  fullXdd fX3(X[3], Y[3], c3, sum[3], c_out);
endmodule



module full_a(X, Y, c_in,sum, c_out);

input X, Y, c_in;
 output sum, c_out;
```

```
wire w1,w2,w3;
always @(*)

begin

xor(w1,X,Y);

xor(S,w1,C);

nand(w2,X,Y);

nand(w3,w1,C);

nand(C,w3,w2);

end

endmodule
```

TEST BENCH

```
`timescale 1ns/1ns

module fourth_tb;


 reg [3:0] a;

 reg [3:0] b;

 reg carry_in;


 wire [3:0] sum;

 wire carry;


 bcd_adder uut (.S(sum),.Cout(carry), .X(a),.Y(b),.Cin(carry_in));


 initial begin

     $dumpfile("fourth_tb.vcd");
```

```
        $dumpvars(0, fourth_tb);


        a=4'b0101; b=4’b0100; carry_in=0;

        end

endmodule
```



SIMULATION - Behavioral Simulation - Functional - sim_1 - bcd_adder

Name	Value
A[3:0]	4
B[3:0]	4
Cin	0
S[3:0]	8
[3]	1
[2]	0
[1]	0
[0]	0
Cout	0
conne...1:	0
sumb[3:0	8
coutb	0
cout2	0

3,613.938 ns

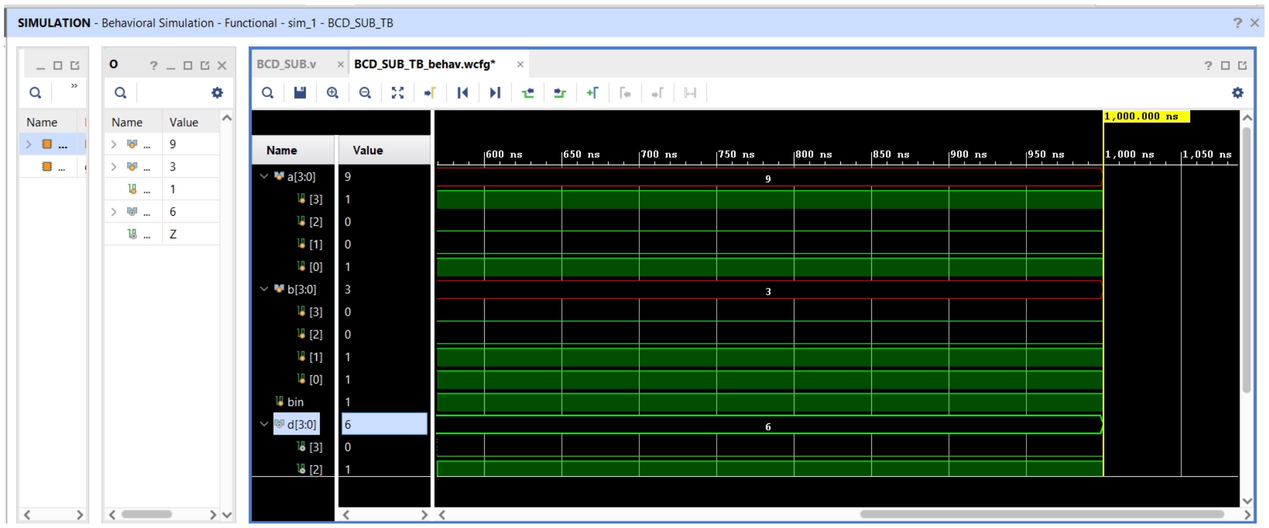
5th

```
module BCD_SUB(D,Bo,X,Y,Bin);

output [3:0] D;

output Bo;

input [3:0] X,Y;

input Bin;

wire [2:0] bi;


full_a S1(D[0],bi[0],X[0],~Y[0],1);

full_a S2(D[1],bi[1],X[1],~Y[1],bi[0]);

full_a S3(D[2],bi[2],X[2],~Y[2],bi[1]);

full_a S4(D[3],bi[3],X[3],~Y[3],bi[2]);

endmodule


module full_a(S,co,X,Y,C);

input X,Y,C;

output reg S,co;

Wire w1,w2,w3;

always @(*)

begin

xor(w1,X,Y);

xor(S,w1,C);

nand(w2,X,Y);

nand(w3,w1,C);

nand(C,w3,w2);
```

```
end

endmodule
```

TEST BENCH

```
        module fiftry_tb ();

        reg [4:0] x,y;

        wire [4:0] cout;

        wire [4:0] su;

        fifth uut (x,y,cout,su);

        initial begin

                $dumpfile("fiftry_tb.vcd");

                    $dumpvars(0, fiftry_tb);


                    x=4'b1001; y=4’b0110;

        end

        endmodule
```



Name	Value
a[3:0]	9
[3]	1
[2]	0
[1]	0
[0]	1
b[3:0]	3
[3]	0
[2]	0
[1]	1
[0]	1
bin	1
d[3:0]	6
[3]	0
[2]	1

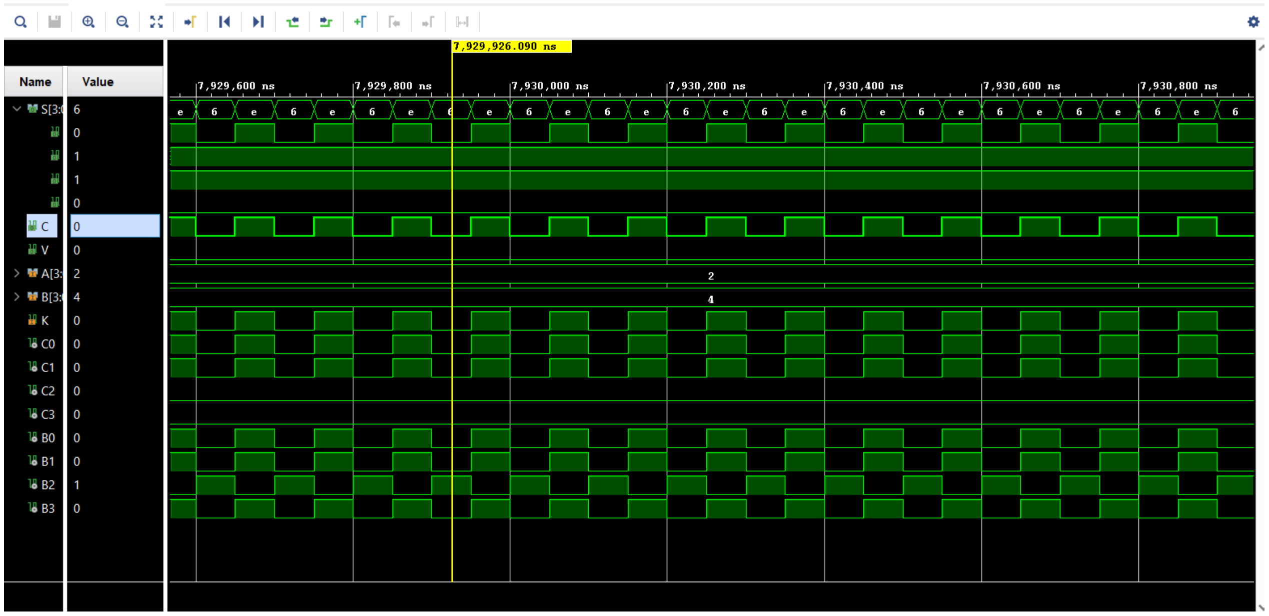
6th

```
module fadder (A, B, Cin, Sum, Cout);
    input A, B;
    input Cin;
    output Sum;
    output Cout;
    wire t1,t2,t3,t4;
  xor x1(t1,A,B);
  xor x2(Sum,t1,Cin);
  and g1(t2,A,B);
  and g2(t3,B,Cin);
  and g3(t4,Cin,A);
  or  g4(Cout,t2,t3,t4);
endmodule

module over
(
    input X4,Y4,K,S,
    output o
);
wire a,b,c,d;
xnor(a,X4,Y4);
xor(b,a,K);
nand(c,S,b);
not(d,c);

endmodule


module add_sub_4 (A, B, In, Res, Out);
    input [3:0] A, B;
    input In;
    output [3:0] Res;
    output Out;
    wire t1,t2,t3,t4,t5,t6,t7;


          xor x3(t3,B[0],In);
          xor x4(t4,B[1],In);
          xor x5(t5,B[2],In);
          xor x6(t6,B[3],In);
          fadder f5(A[0],t3,In,Res[0],t1);
```

```
            fadder f6(A[1],t4,t1,Res[1],t2);
            fadder f7(A[2],t5,t2,Res[2],t3);
            fadder f8(A[3],t6,t3,Res[3],Out);
    endmodule
```

TEST BENCH

```
timescale 1ns/1ns

      module bcdadd_tb;

      reg [3:0] A,B;

      reg cin;

      wire [7:0] cout;

      wire [7:0] sum,diff;

      wire sign;

      bcdadd uut (x,y,c,cout,sum,diff);

      initial begin

          $dumpfile("bcdadd_tb.vcd");

            $dumpvars(0, bcdadd_tb);

      cin=1'b0; #50;cin=~cin #50;

       x=4'b0100; y=4'b0010;

      end

      endmodule
```



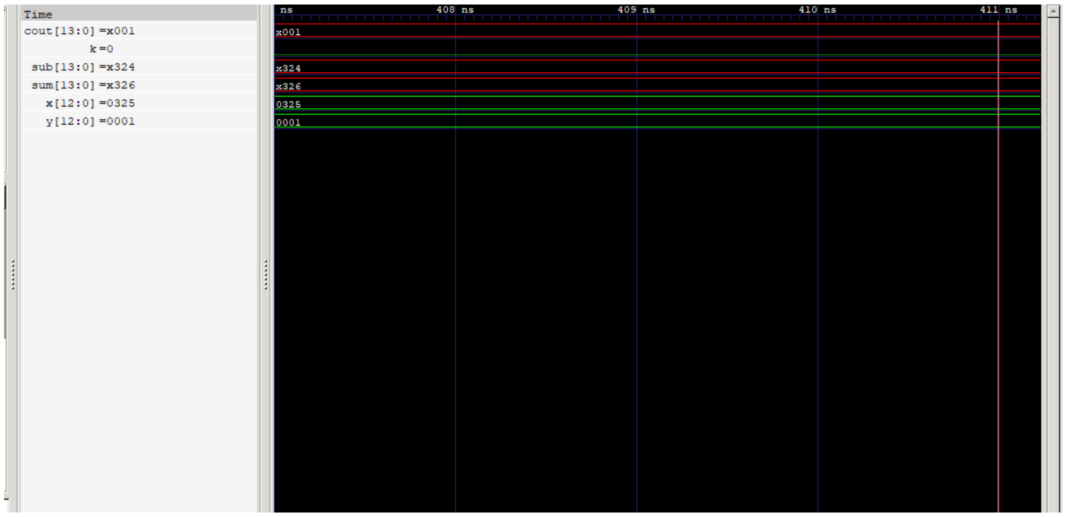
7th

```
module bit_BCD_ADDER(in1,in2,outp,cout);
input [12:0] in1;
input [12:0] in2;
output [12:0] outp;
output cout;
wire [3:0] adigit1,adigit2,bdigit1,bdigit2,adigit3,bdigit3;
assign adigit1=in1[3:0];
assign adigit2=in1[7:4];
assign bdigit1=in2[3:0];
assign bdigit2=in2[7:4];
assign adigit3=in1[11:8];
assign bdigit3=in2[11:8];

wire [3:0] wt1,wt2,wt3,wt4,wt5,wt6;
wire c1,c2,c3,c4,c5,c6;
wire temp,temp1,temp2;
adder4 x1(adigit1, bdigit1,0,wt1,c1);
adder4 x2(wt1, wt2,0,outp[3:0],c2);
adder4 x3(adigit2, bdigit2,temp,wt3,c3);
adder4 x4(wt3, wt4,0,outp[7:4],c4);
```

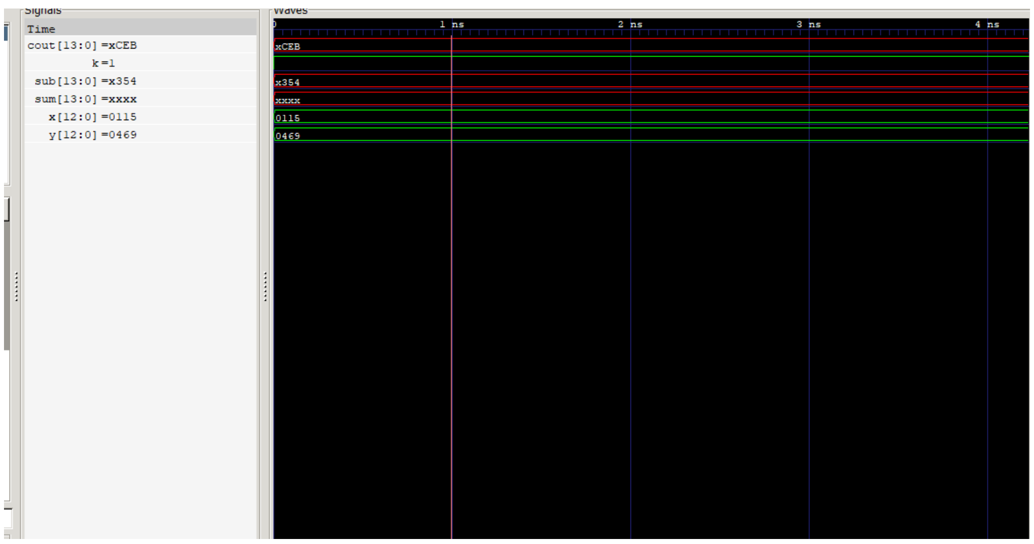
```
adder4 x5(adigit3, bdigit3,temp2,wt5,c5);
adder4 x6(wt5, wt6,0,outp[11:8],c6);


assign wt2={1'b0,temp,temp,1'b0};
assign wt4={1'b0,temp1,temp1,1'b0};
assign wt5={1'b0,temp2,temp2,1'b0};

invalid_bcd b1(wt1,c1,temp);
invalid_bcd b2(wt3,c3,temp1);
invalid_bcd b3(wt5,c5,temp2);

assign cout =temp1;

endmodule

module over
(
   input X4,Y4,K,S,
   output o
);
wire a,b,c,d;
xnor(a,X4,Y4);
xor(b,a,K);
nand(c,S,b);
not(d,c);

endmodule

module invalid_bcd(sum,c,outp);
input [3:0] sum;
input c;
output outp;
wire w1,w2,w3,w4;
or g1(w1,sum[1],sum[2]);
and g2(w2,w1,sum[3]);
or g3(outp,w2,c);
endmodule

module adder4(in1,in2,cin,outp,cout);
input [3:0] in1,in2;
input cin;
output [3:0]outp;
output cout;
```

```
wire [3:0] temp;
fa comp1(in1[0],in2[0],cin,outp[0],temp[0]);
fa comp2(in1[1],in2[1],temp[0],outp[1],temp[1]);
fa comp3(in1[2],in2[2],temp[1],outp[2],temp[2]);
fa comp4(in1[3],in2[3],temp[2],outp[3],temp[3]);
assign cout = temp[3];
endmodule

module fa(a,b,cin,s,cout);
input a,b,cin;
output s, cout;
wire w1,w2,w3;
xor p1 (s,a,b,cin);
and p2(w1,a,b);
and p3(w2, b,cin);
and p4(w3, a,cin);
or p5(cout,w1,w2,w3);

endmodule


`timescale 1ns/1ns

`include " bit_BCD_ADDER.v"

module sevtry_tb ();

reg [12:0] x,y;

reg k;



wire [13:0] cout;

wire [13:0] sum, sub;

seven uut (x,y,k,sub,sum,cout);

initial begin

        $dumpfile("sevtry_tb.vcd");

                $dumpvars(0, sevtry_tb);
```

```
x=13'b0000100010101; y=13'b0010001101001; k=1;#70;

x=13'b0011100000101; y=13'b0100100000001; k=1;#70;

x=13'b001100100101; y=13'b0000000000001; k=1;#70;

x=13'b0000100010101; y=13'b0010001101001; k=0;#70;

x=13'b0000100000101; y=13'b0000000000001; k=0;#70;

x=13'b001100100101; y=13'b0000000000001; k=0;#70;


end

endmodule
```



```
cout[13:0] =x001    x001
         k =0
 sub[13:0] =x324    x324
 sum[13:0] =x326    x326
   x[12:0] =0325    0325
   y[12:0] =0001    0001
```



```
Signals
Time
cout[13:0] =xCEB
         k =1
 sub[13:0] =x354
 sum[13:0] =xxxx
   x[12:0] =0115
   y[12:0] =0469
```