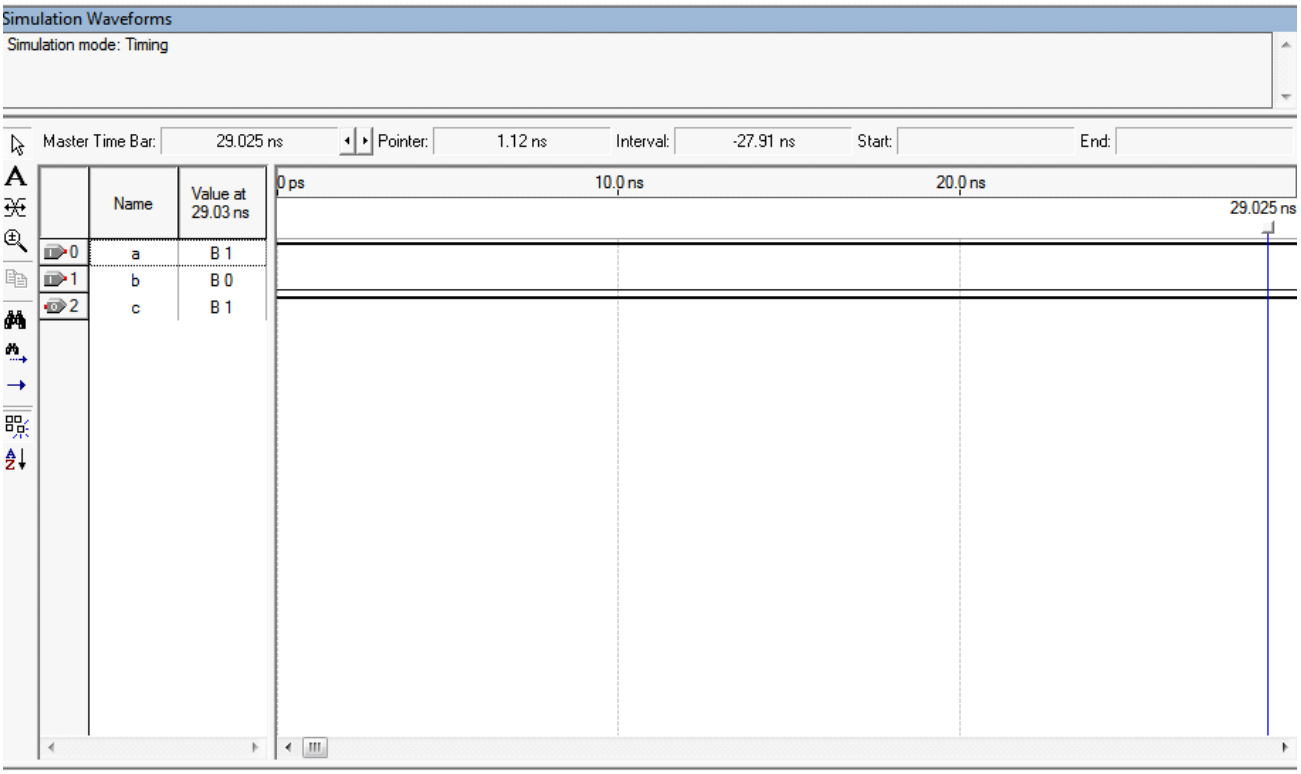
EX NO: 18

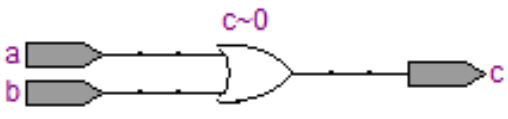
1.OR GATE

```
module or1 (c,a,b);
  output c;
  input a,b;
  assign c=a|b;
endmodule
```

Simulation:



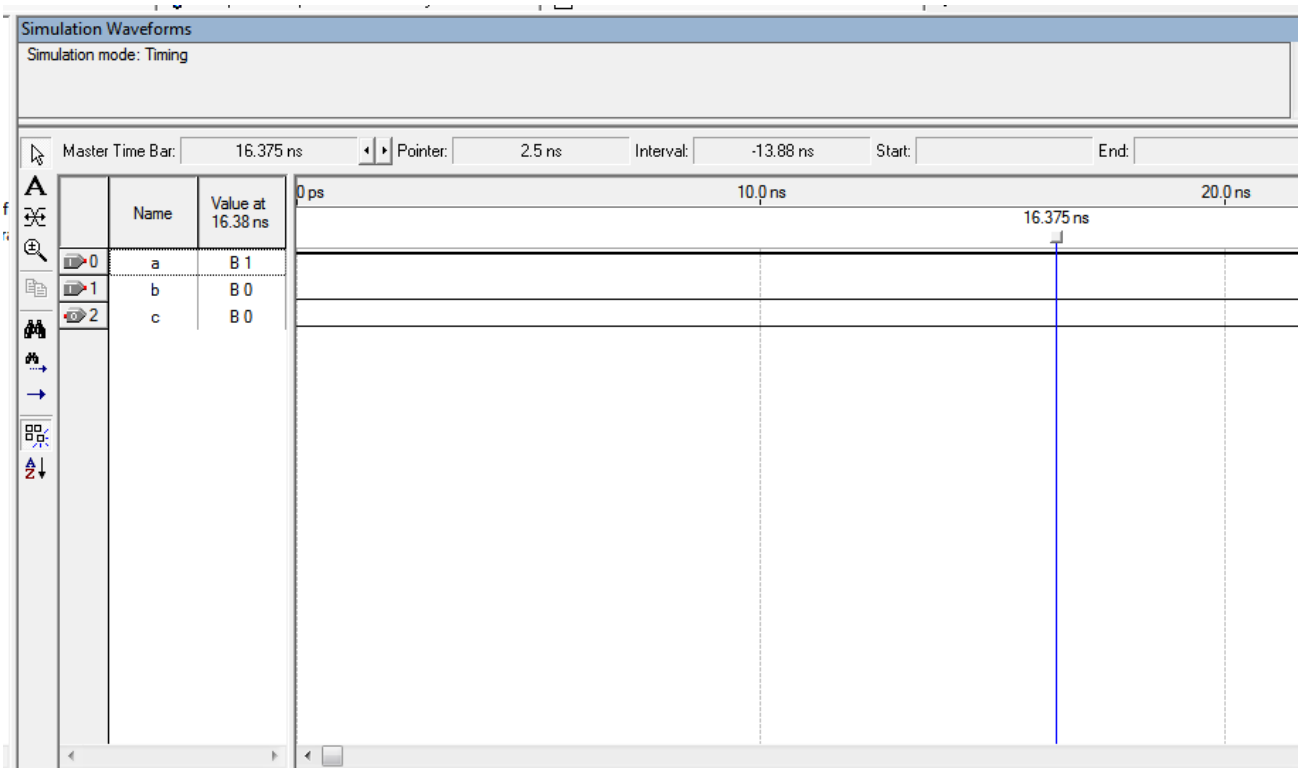
RTL view:



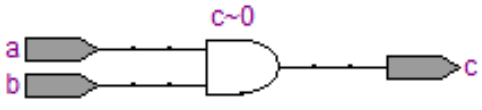
2.AND GATE

```
module and1(c,a,b);
  output c;
  input a,b;
  assign c=a&b;
endmodule
```

Simulation:



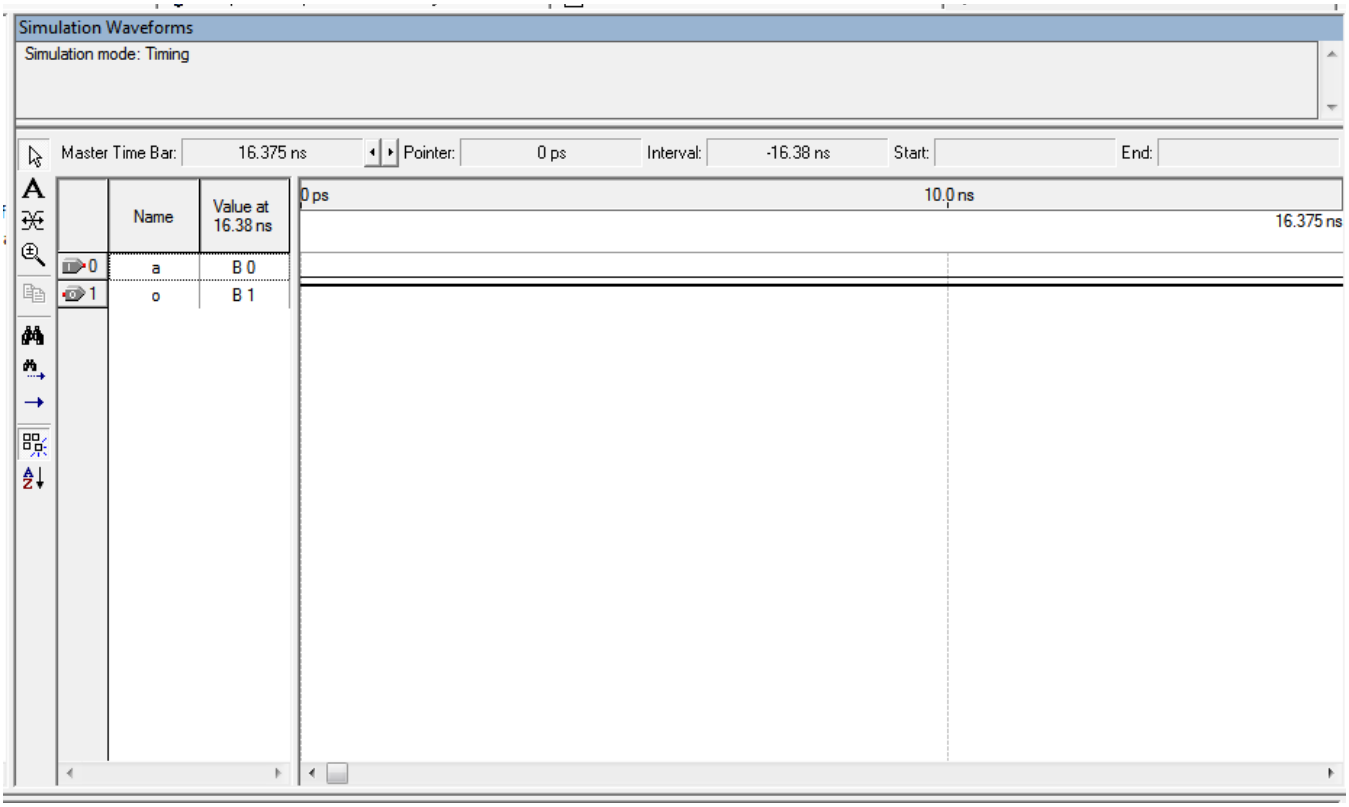
RTL view:

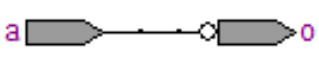


3.NOT GATE

```
module inv(o,a);
  output o;
  input a;
  assign o=~a;
endmodule
```

Simulation:

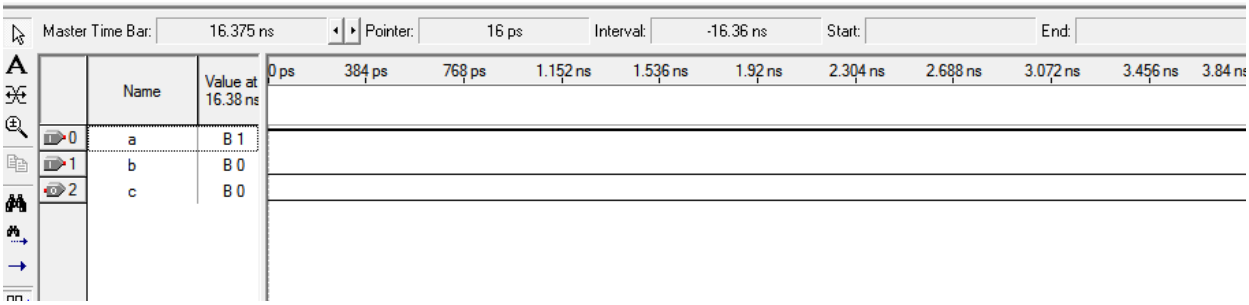




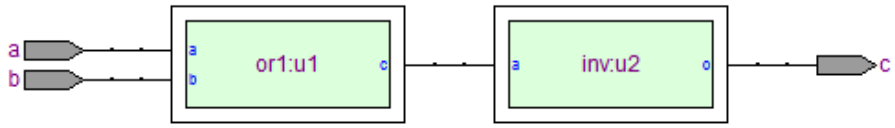
4. NOR GATE

```
module nor1(c,a,b);
  output c;
  input a,b;
  wire d;
  or1 u1(d,a,b);
  inv u2(c,d);
endmodule
module or1(c,a,b);
  output c;
  input a,b;
  assign c=a|b;
endmodule
module inv(o,a);
  output o;
  input a;
  assign o=~a;
endmodule
```

Simulation:



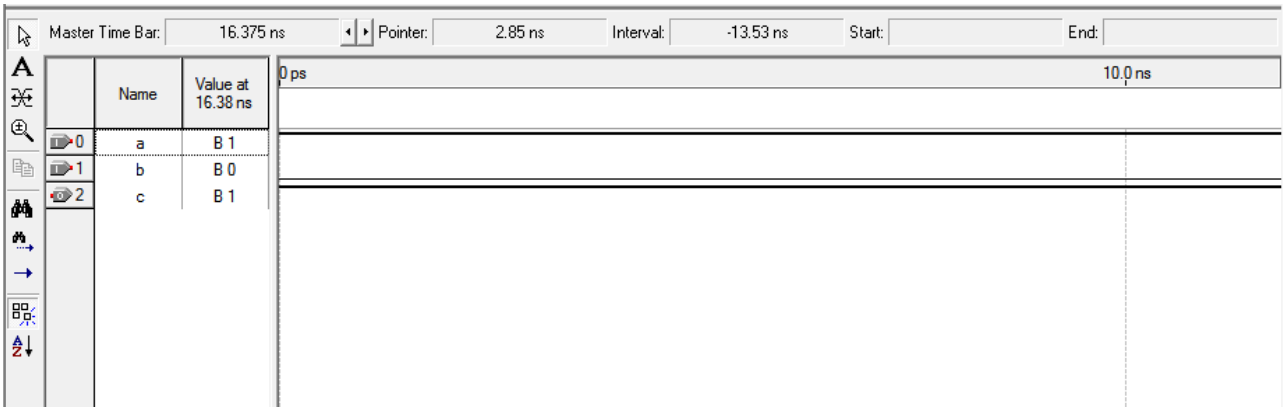
RTL view:



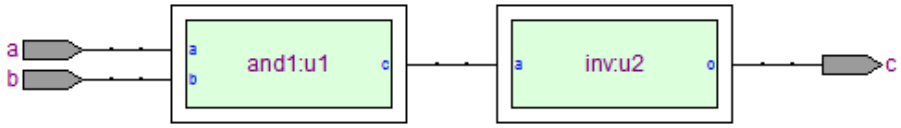
5.NAND GATE

```
module nand1(c,a,b);
  output c;
  input a,b;
  wire d;
  and1 u1(d,a,b);
  inv u2(c,d);
endmodule
module and1(c,a,b);
  output c;
  input a,b;
  assign c=a&b;
endmodule
module inv(o,a);
  output o;
  input a;
  assign o=~a;
endmodule
```

Simulation:



RTL view:

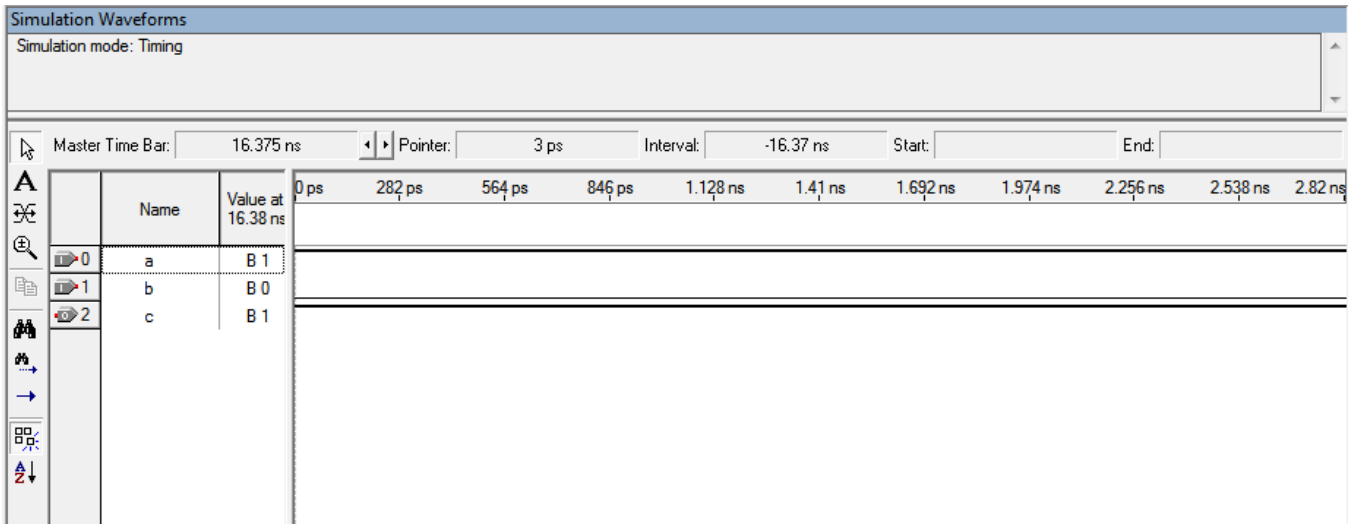


6.EX OR GATE

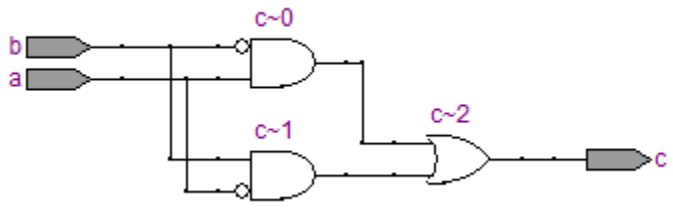
exor

```
module exor1(c,a,b);
input a;
input b;
output c;
assign c=(a&~b)|(~a&b);
endmodule
```

Simulation:



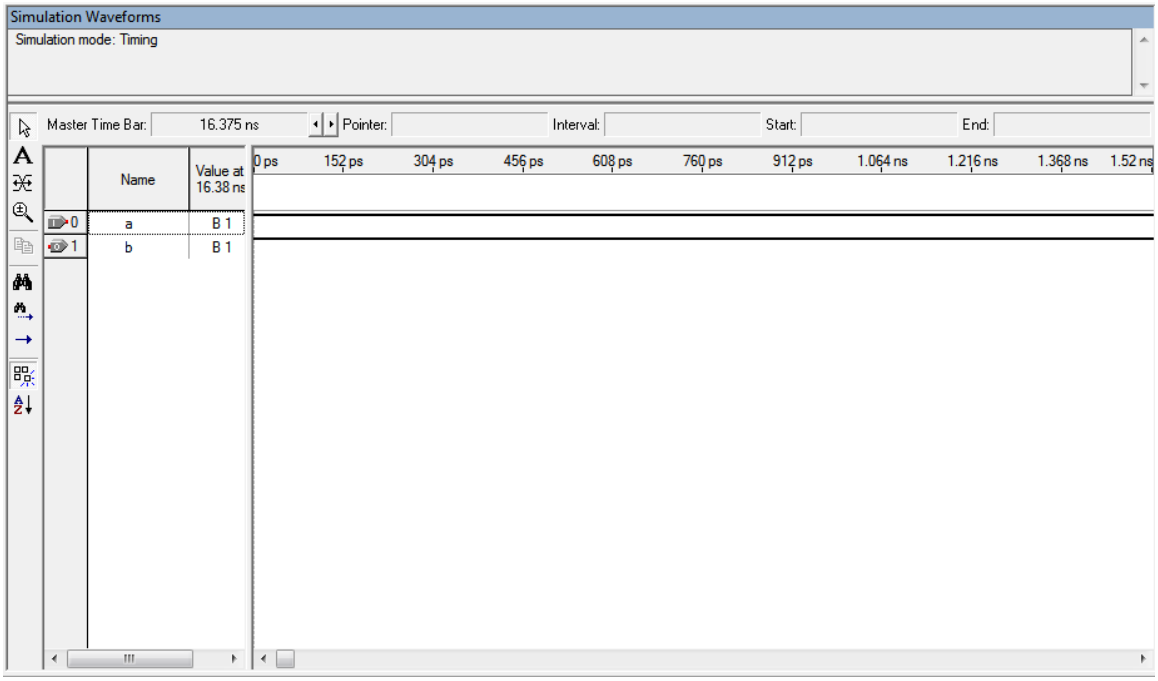
RTL view:



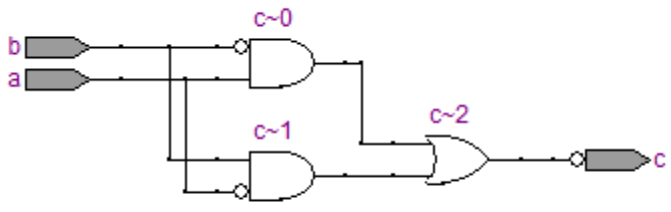
7.EX NOR GATE

```
module exnor1(c,a,b);
input a;
input b;
output c;
assign c=~((a&~b)|(~a&b));
endmodule
```

Simulation:



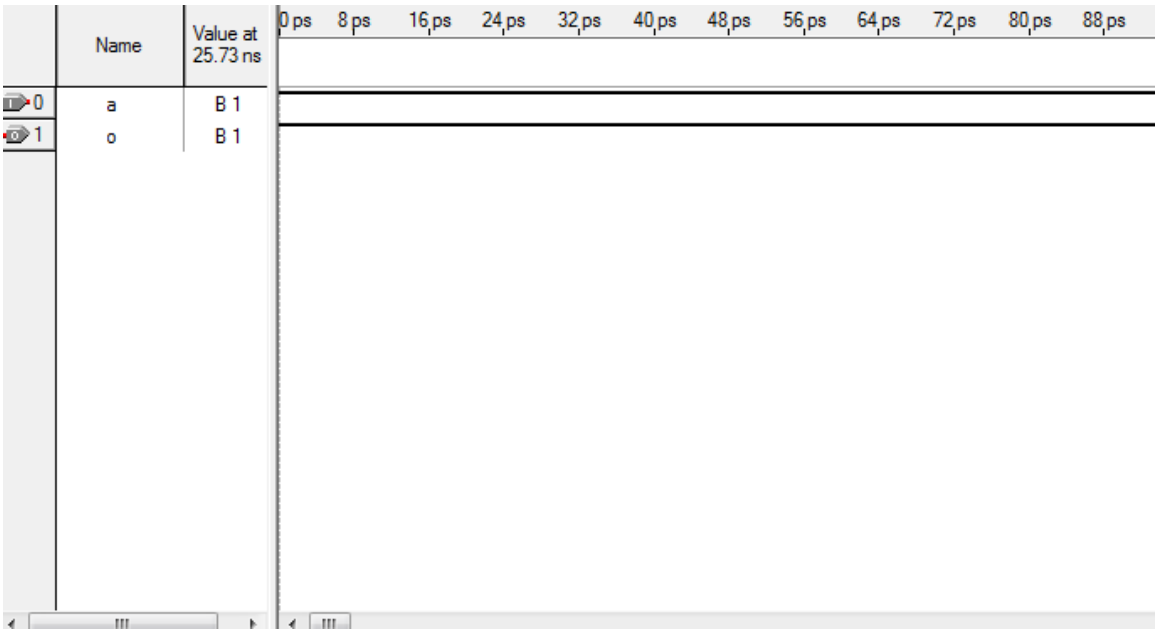
RTL view:



8.BUFFER

```
module buffer1(o,a);
output o;
input a;
assign o=~(~(a));
endmodule
```

Simulation:



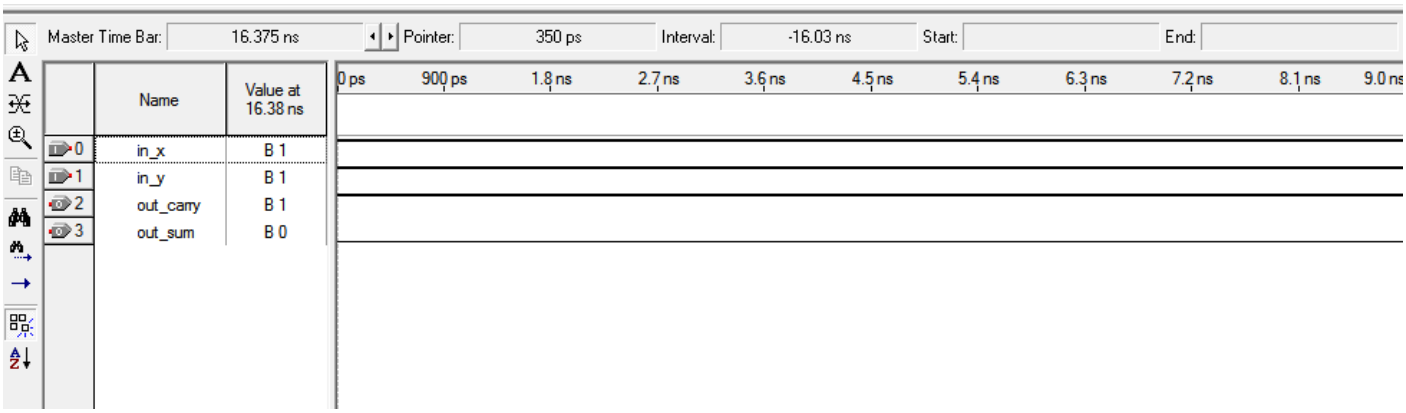
RTL View:

EX NO: 19

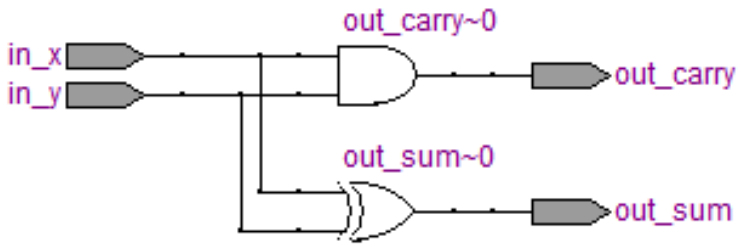
1.HALF ADDER

```
module half_adder(in_x, in_y, out_sum, out_carry);
input in_x;
input in_y;
output out_sum;
output out_carry;
assign out_sum = in_x^in_y;
assign out_carry = in_x&in_y;
endmodule
```

Simulation:



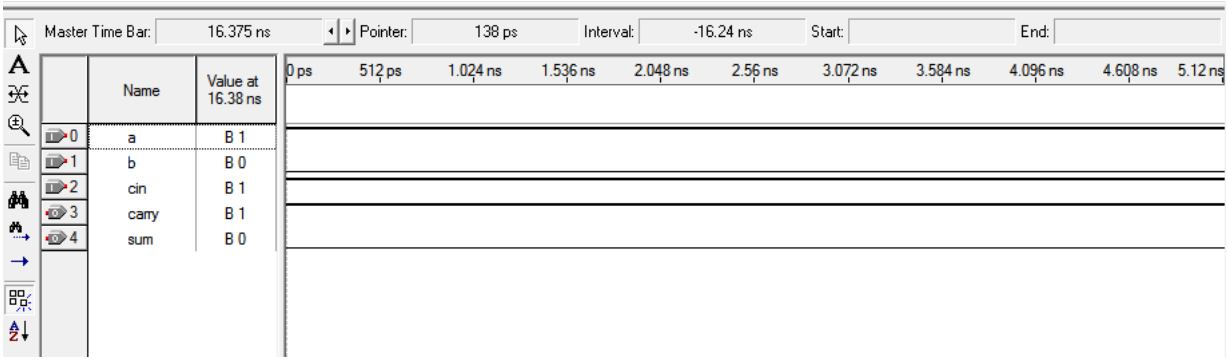
RTL view:



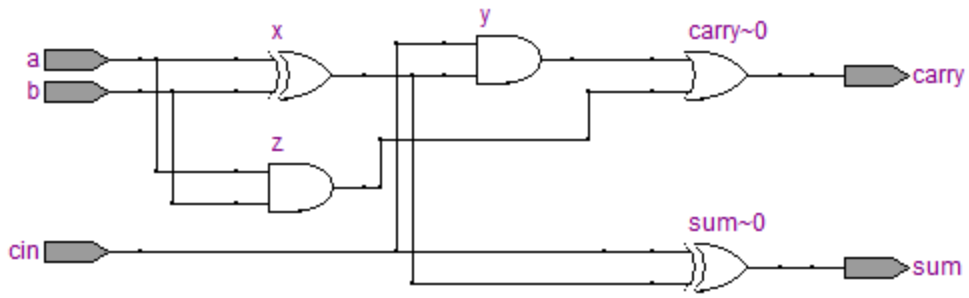
2.FULL ADDER

```
module fulladder(
  input a,
  input b,
  input cin,
  output sum,
  output carry );
assign x=a ^ b;
assign sum=x^cin;
assign y=x & cin;
assign z=a & b;
assign carry= y | z;
endmodule
```

Simulation:



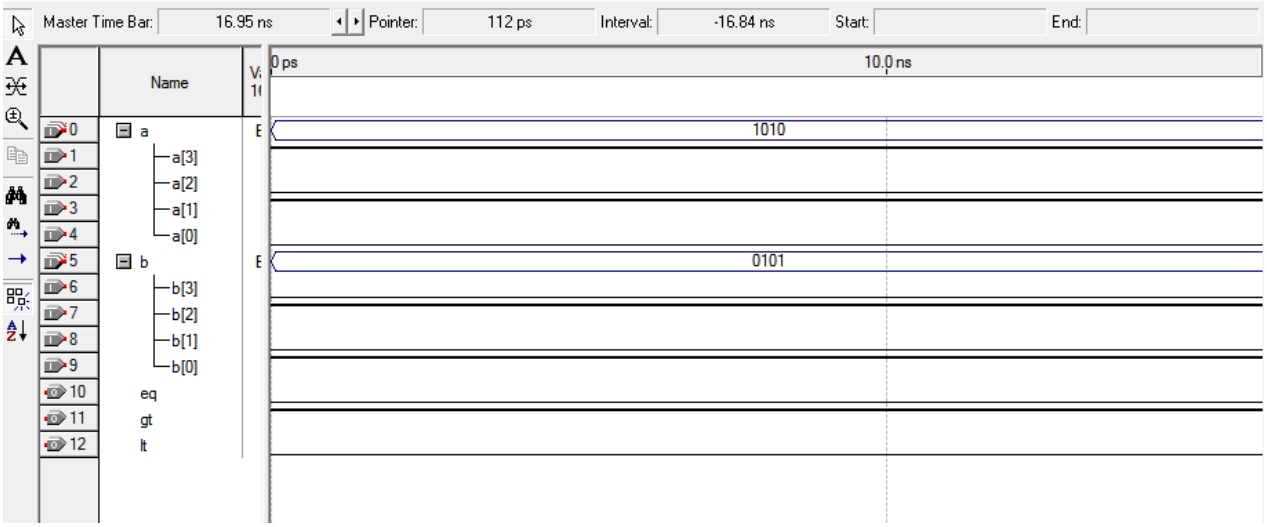
RTL view:



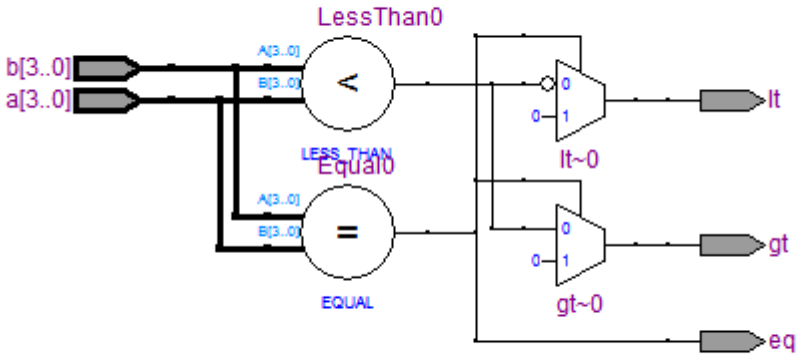
3. 4-BIT MAGNITUDE COMPARATOR

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

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
```

Simulation:



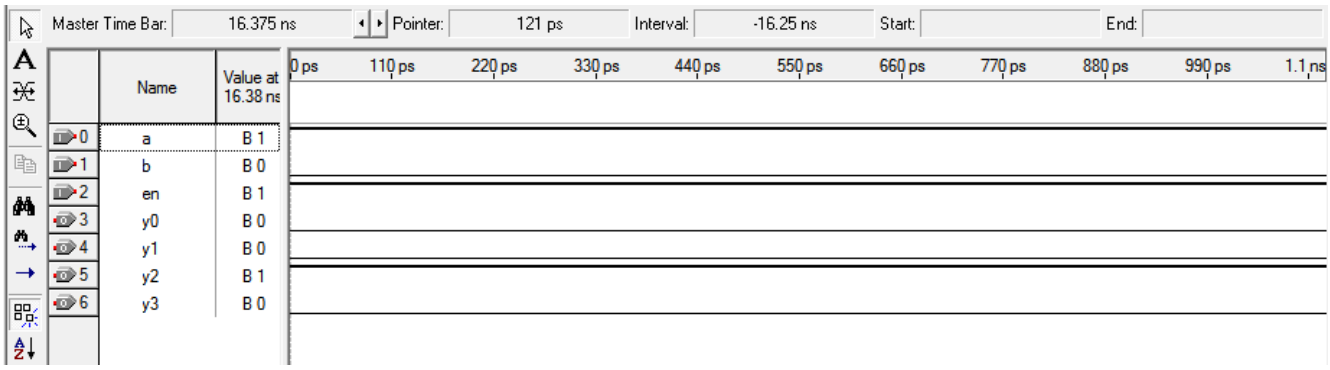
RTL view:



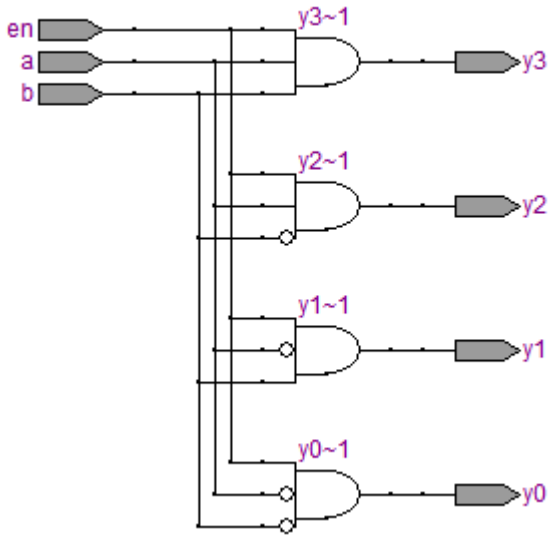
4. 2:4 DECODER

```
module dec2_4(a,b,en,y0,y1,y2,y3);
input a,b,en;
output y0,y1,y2,y3;
assign y0= (~a) & (~b) & en;
assign y1= (~a) & b & en;
assign y2= a & (~ b) & en;
assign y3= a & b & en;
endmodule
```

Simulation:



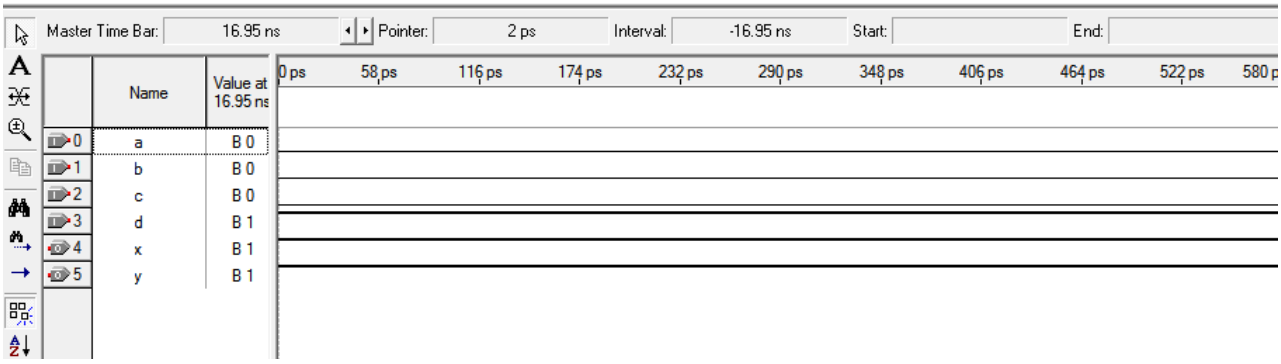
RTL view:



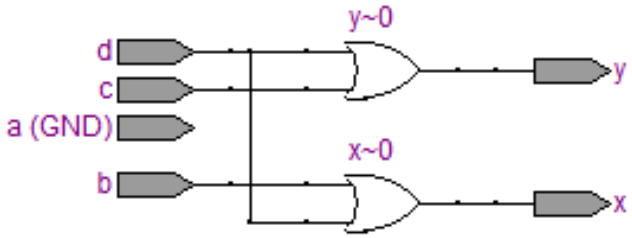
5. 4:2 ENCODER

```
module encoder4_2 ( a ,b ,c ,d ,x ,y );

output x ;
output y ;

input a ;
input b ;
input c ;
input d ;

assign x = b | d;
assign y = c | d;

endmodule
```

Simulation:



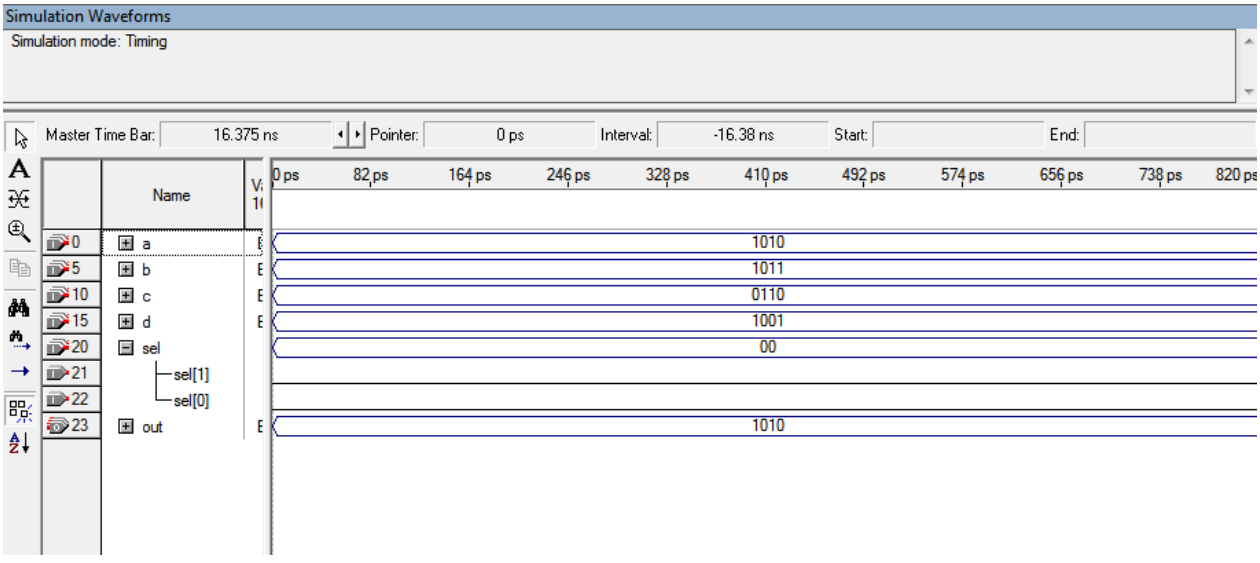
RTL view:

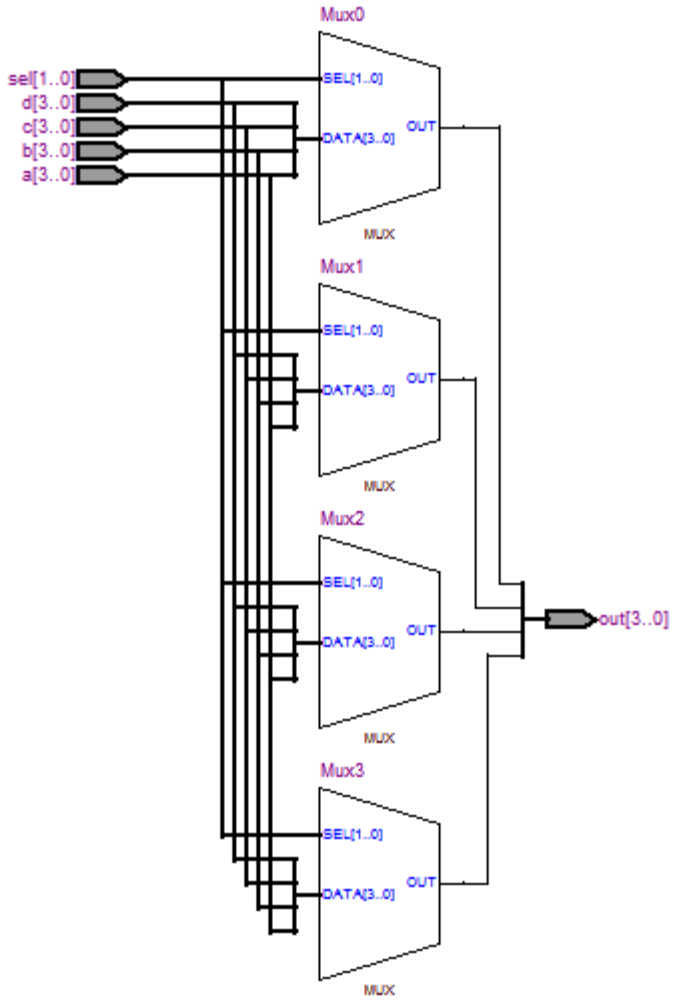


6. 4:1 MUX

```
module mux4to1 ( input [3:0] a, // 4-bit input called a
  input [3:0] b, // 4-bit input called b
  input [3:0] c, // 4-bit input called c
  input [3:0] d, // 4-bit input called d
  input [1:0] sel, // input sel used to select between a,b,c,d
  output reg [3:0] out); // 4-bit output based on input sel
  // This always block gets executed whenever a/b/c/d/sel changes value
  // When that happens, based on value in sel, output is assigned to either a/b/c/d
  always @ (a or b or c or d or sel) begin
  case (sel)
  2'b00 : out <= a;
  2'b01 : out <= b;
  2'b10 : out <= c;
  2'b11 : out <= d;
  endcase
  end
endmodule
```

Simulation:

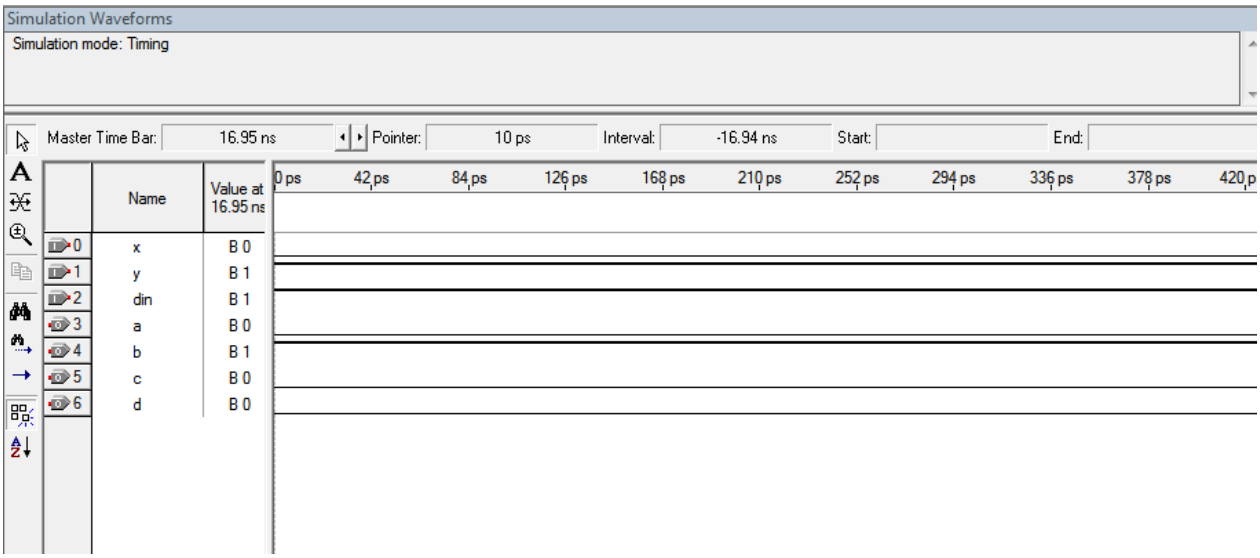



Mux0
sel[1..0]
d[3..0]
c[3..0]
b[3..0]
a[3..0]
SEL[1..0]
DATA[3..0]
OUT
MUX
Mux1
SEL[1..0]
DATA[3..0]
OUT
MUX
Mux2
SEL[1..0]
DATA[3..0]
OUT
MUX
out[3..0]
Mux3
SEL[1..0]
DATA[3..0]
OUT
MUX

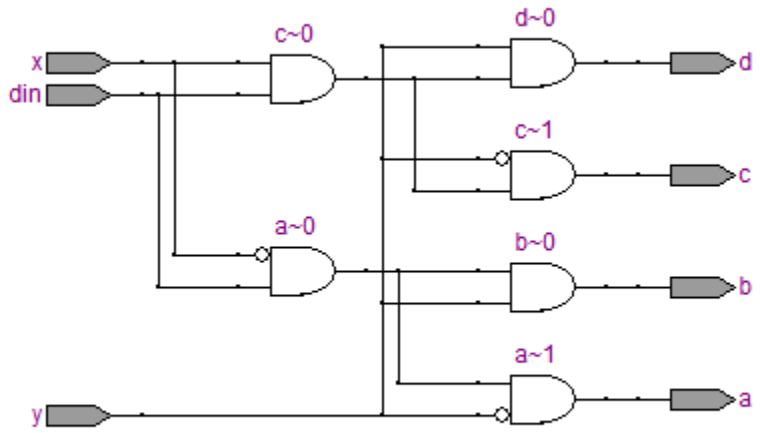
7. 1:4 DEMUX

```
module demux1_4 ( din ,x ,y ,a ,b ,c ,d );
output a ;
output b ;
output c ;
output d ;

input din ;
input x ;
input y ;

assign a = din & (~x) & (~y);
assign b = din & (~x) & y;
assign c = din & x & (~y);
assign d = din & x & y;
endmodule
```

Simulation:



RTL view:

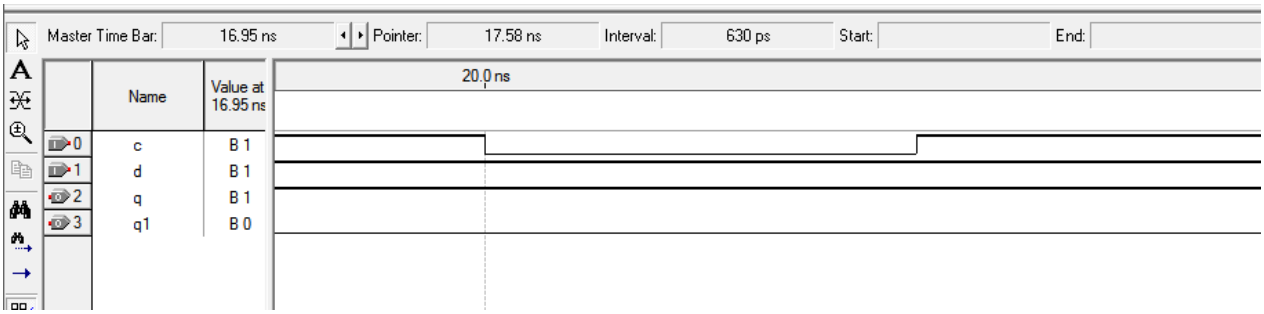


EX NO: 20

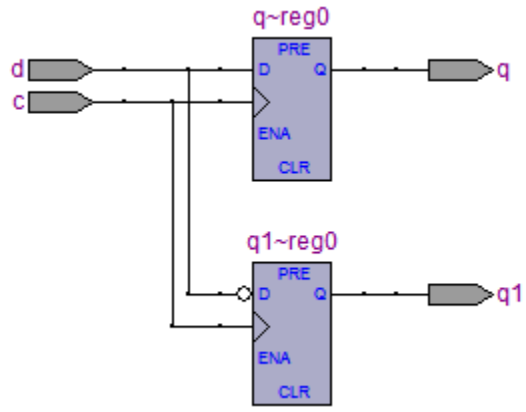
1.D FLIP DLOP

```
module d(q,q1,d,c);
output q,q1;
  input d,c;
  reg q,q1;
        initial
            begin
                    q=1'b0; q1=1'b1;
             end
        always @ (posedge c)
            begin
                q=d;
                q1= ~d;
            end
endmodule
```

Simulation:



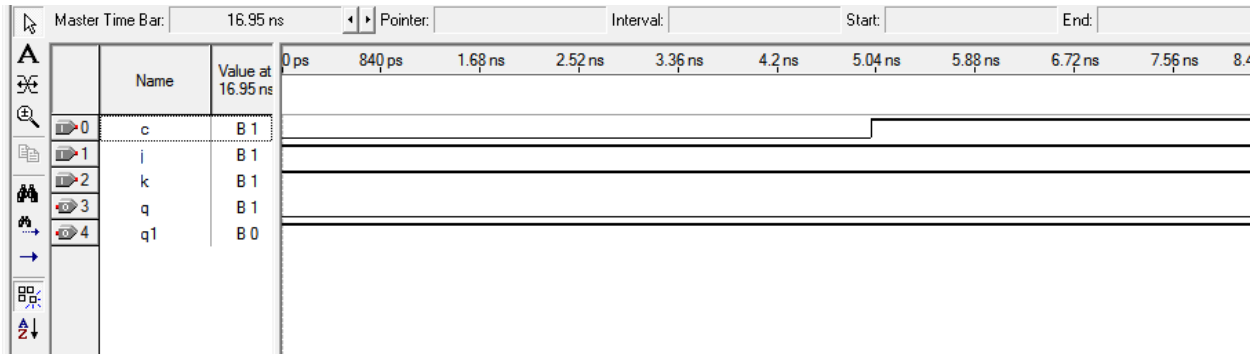
RTL view:



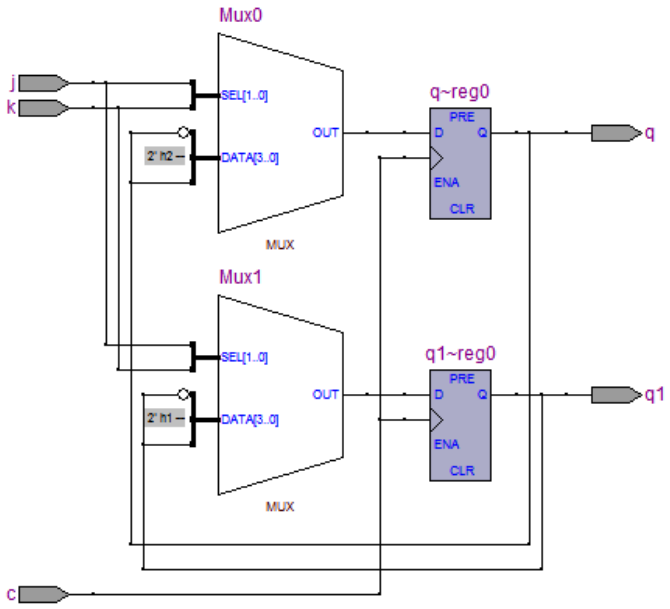
2. JK FLIP FLOP

```
module jk(q,q1,j,k,c);
output q,q1;
input j,k,c;
reg q,q1;
initial begin q=1'b0; q1=1'b1; end
always @ (posedge c)
   begin
       case({j,k})
                {1'b0,1'b0}:begin q=q; q1=q1; end
                {1'b0,1'b1}: begin q=1'b0; q1=1'b1; end
                {1'b1,1'b0}:begin q=1'b1; q1=1'b0; end
                {1'b1,1'b1}: begin q=~q; q1=~q1; end
       endcase
     end
endmodule
```

Simulation:



RTL view:

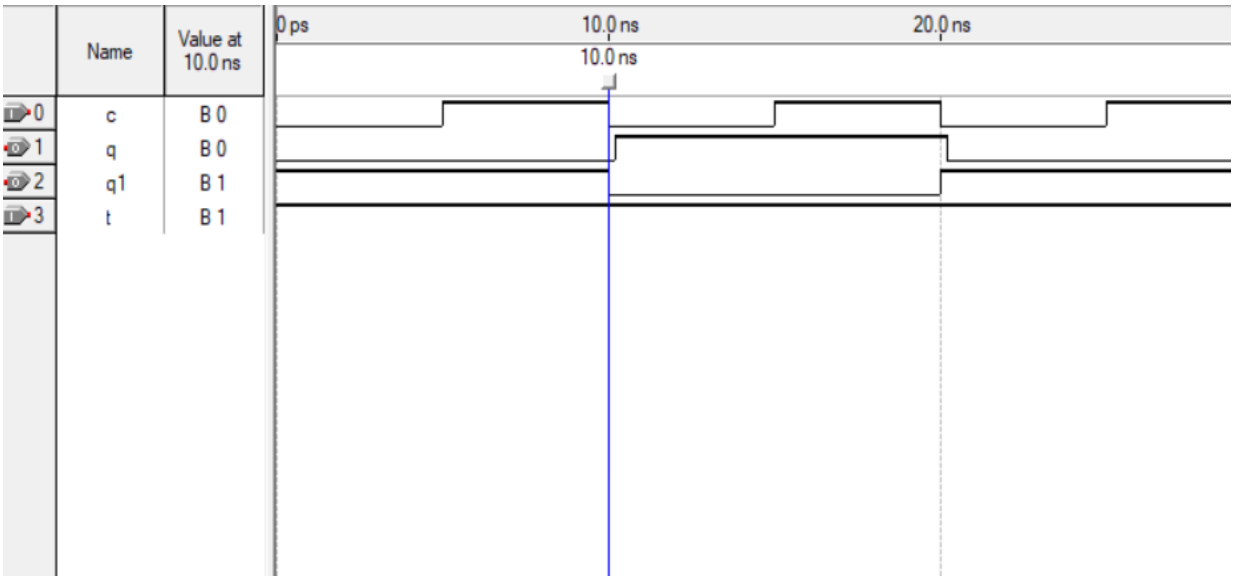


==========

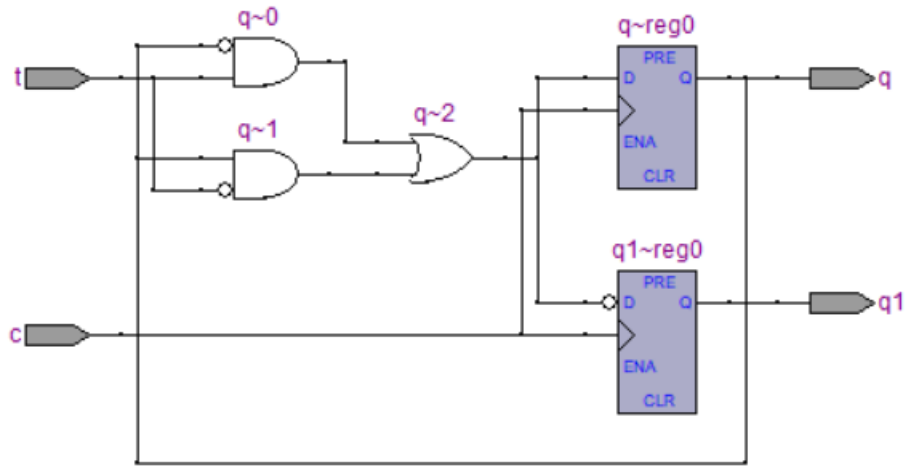
3. T Flip Flop

```
module TFlipFlop1(q,q1,t,c);
output q,q1;
input t,c;
reg q,q1;

initial
begin
        q=1'b0; q1=1'b1;
end
always @ (posedge c)
        begin
                q = (t&(~q))|((~t)&q);
                q1= ~q;
        end
endmodule
```

Simulation:



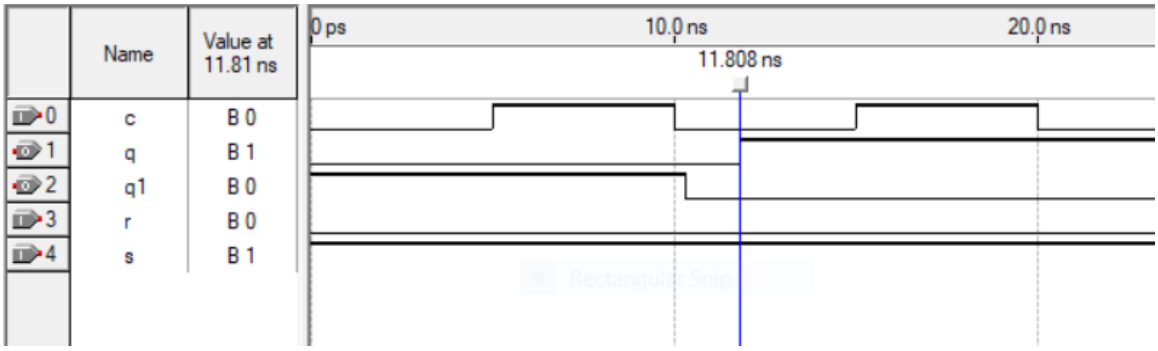
RTL View:



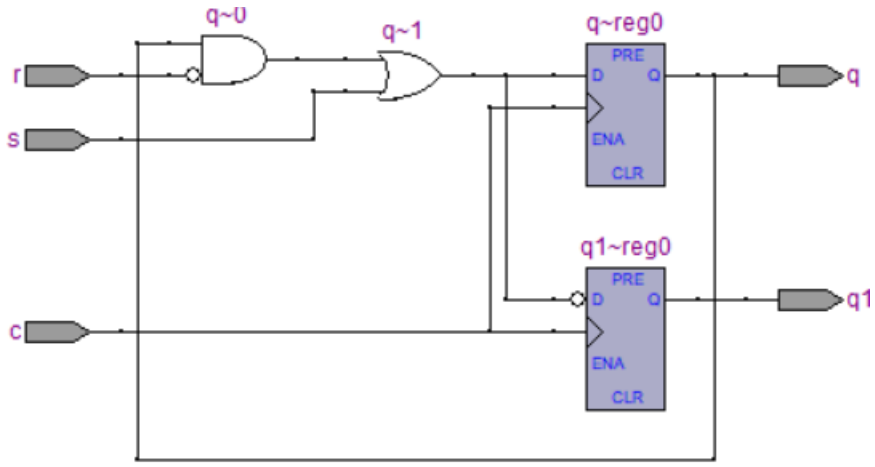
4. SR Flip Flop

```
module SRFlipFlop1(q,q1,s,r,c);
output q,q1;
input s,r,c;
reg q,q1;
initial
        begin
                q=1'b0; q1=1'b1;
        end
always @ (posedge c)
        begin
                q = s |( (~r) & q );
                q1= ~q;
        end
endmodule
```

Simulation:



RTL View:

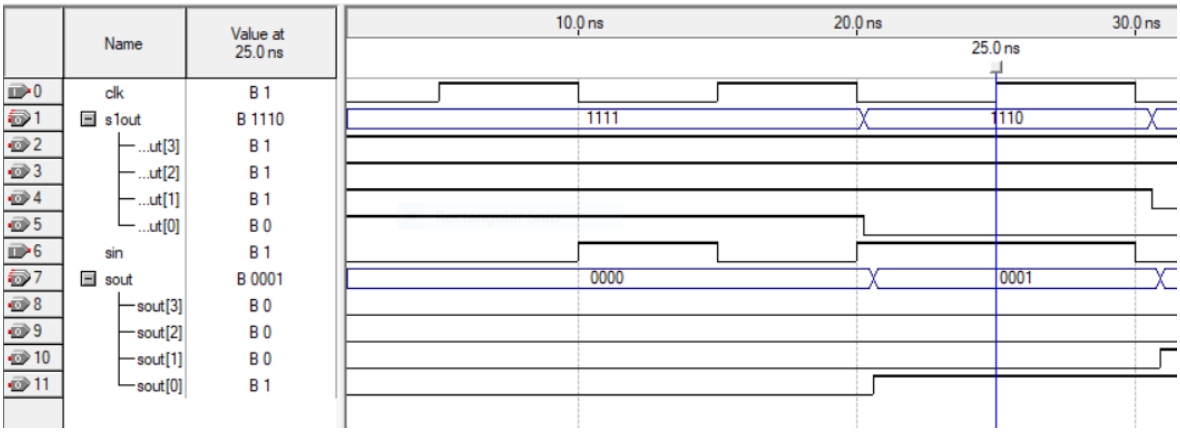


==========

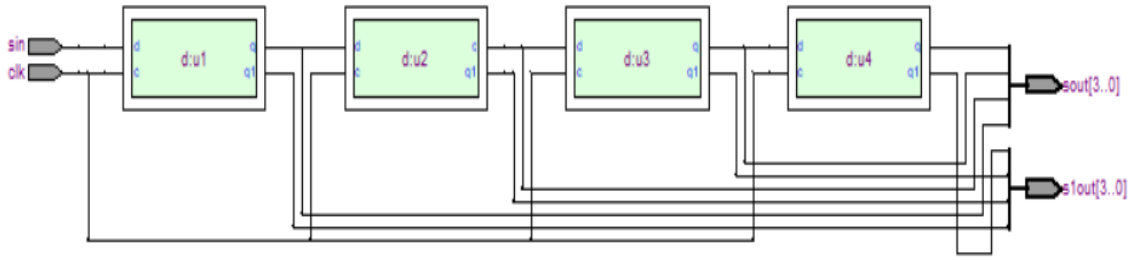
5. 4 Bit Shift Register [SIPO]

```
module sipo(sout,s1out,sin,clk);
output [3:0]sout,s1out;
input sin,clk;
d u1(sout[0],s1out[0],sin,clk);
d u2(sout[1],s1out[1],sout[0],clk);
d u3(sout[2],s1out[2],sout[1],clk);
d u4(sout[3],s1out[3],sout[2],clk);
endmodule
module d(q,q1,d,c);
output q,q1;
input d,c;
reg q,q1;
initial
        begin
                q=1'b0; q1=1'b1;
        end
always @ (posedge c)
        begin
                q=d;
                q1= ~d;
        end
endmodule
```

Simulation:



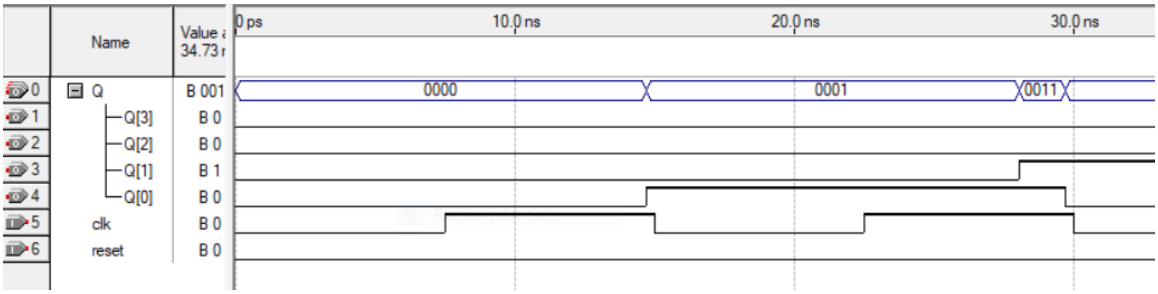
RTL View:



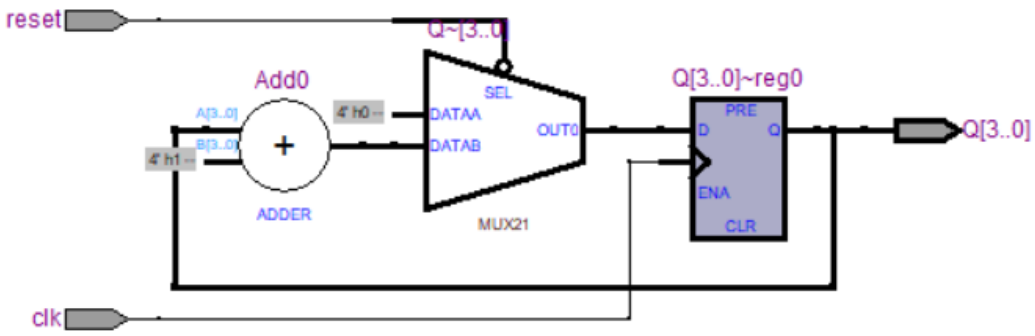
6. 4 Bit Up Counter

```
module UpCounter(clk, reset, Q);
input clk, reset;
output [3:0] Q;
reg [3:0] Q;
always @ (posedge clk)
begin
        if (~reset)
                begin
                        Q <= Q+1;
                end
        else
                Q=0;
end
endmodule
```

Simulation:



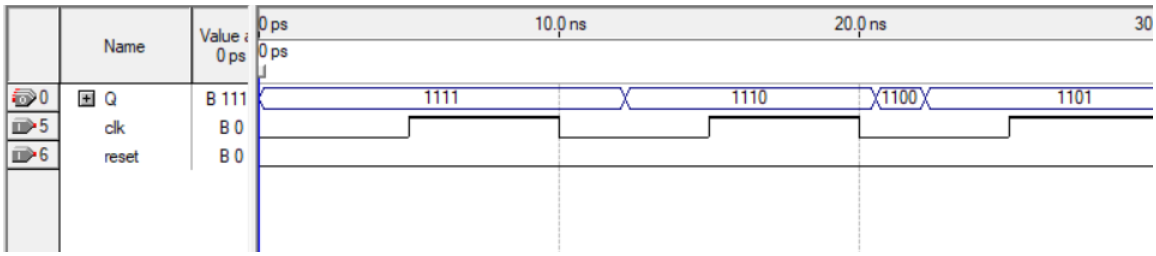
RTL View:



7. 4 Bit Down Counter:

```
module DownCounter(clk, reset, Q);
input clk, reset;
output [3:0] Q;
reg [3:0] Q;
always @ (posedge clk)

begin
        if (~reset)
                begin
                        Q <= Q-1;
                end
        else
                Q <= 15;
end
endmodule
```

Simulation:



RTL View: