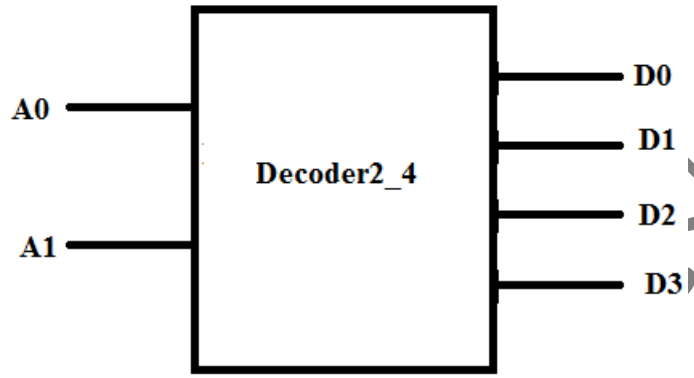
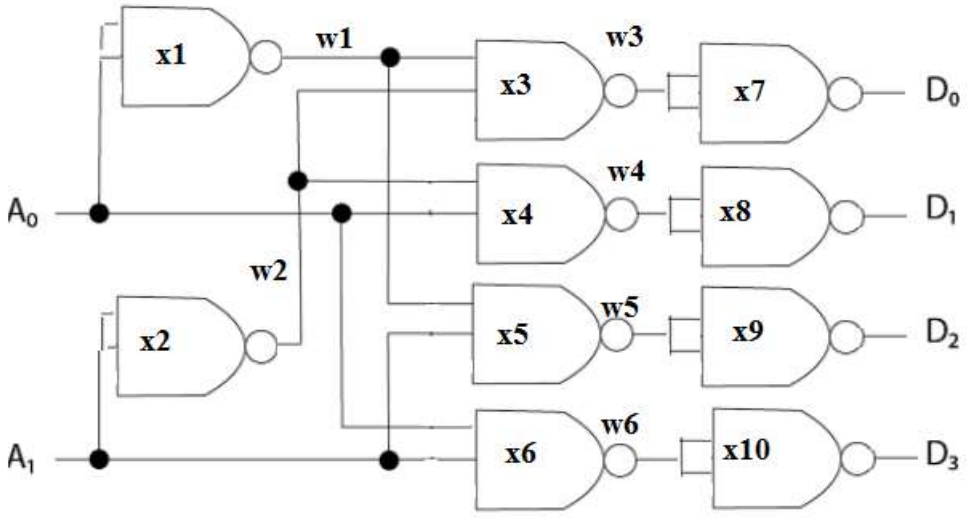
B. E. (EC / TC) Choice Based Credit System (CBCS) and Outcome Based Education (OBE) SEMESTER – V			
HDL LABORATORY			
Laboratory Code	18ECL58	CIE Marks	40
Number of Lecture Hours/Week	02Hr Tutorial (Instructions)+ 02 Hours Laboratory	SEE Marks	60
RBT Level	L1, L2, L3	Exam Hours	03

1. Write Verilog program for the following combinational design along with test bench to verify the design:

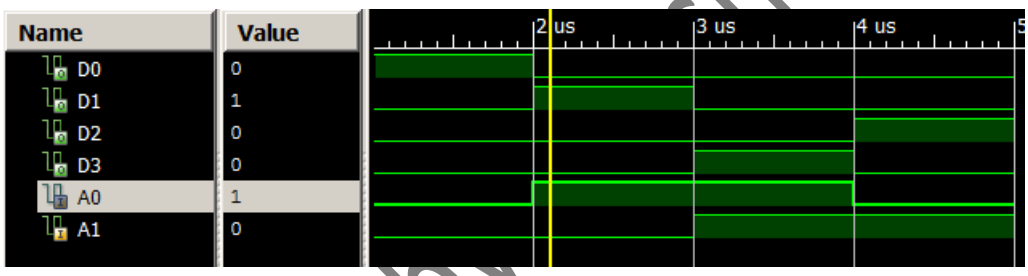
a. 2 to 4 decoder realization using NAND gates only (structural model)



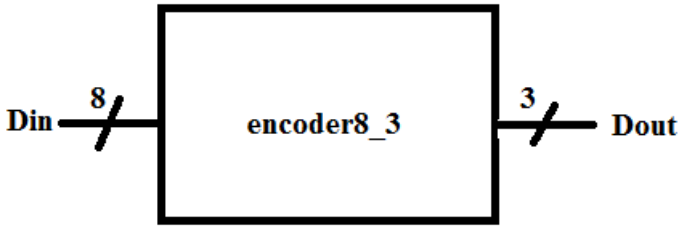



```
module Decoder2_4(D0,D1,D2,D3,A0,A1);

output D0,D1,D2,D3;

input A0,A1;
```

```
wire w1,w2,w3,w4,w5,w6;

nand x1(w1,A0);

nand x2(w2,A1);

nand x3(w3,w1,w2);

nand x4(w4,A0,w2);

nand x5(w5,w1,A1);

nand x6(w6,A0,A1);

nand x7(D0,w3,w3);

nand x8(D1,w4,w4);

nand x9(D2,w5,w5);

nand x10(D3,w6,w6);

endmodule
```

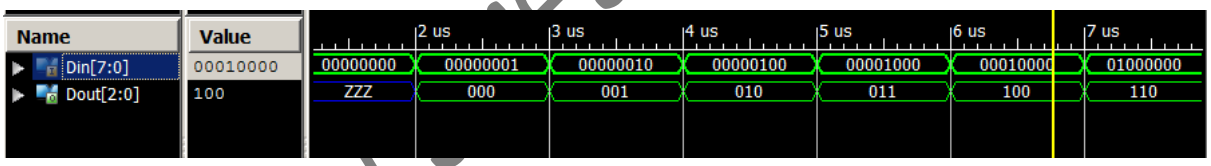


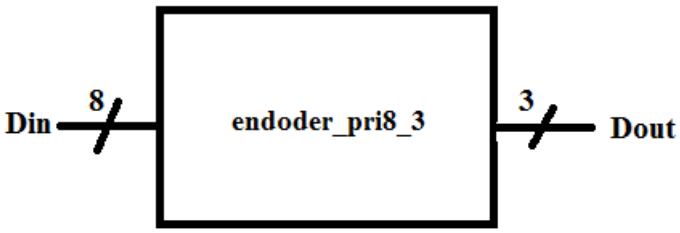
b. 8 to 3 encoder with priority and without priority (behavioural model)




```
module encoder8_3(Dout, Din);

    output [ 2 : 0 ] Dout;

    input [ 7 : 0 ] Din;

    reg [ 2 : 0 ] Dout;

    always@(Din)

    begin
```

```
case (Din)
8'b00000001:Dout = 3'b000;
8'b00000010:Dout = 3'b001;
8'b00000100:Dout = 3'b010;
8'b00001000:Dout = 3'b011;
8'b00010000:Dout = 3'b100;
8'b00100000:Dout = 3'b101;
8'b01000000:Dout = 3'b110;
8'b10000000:Dout = 3'b111;
default: Dout=3'bzzz;
endcase
end
endmodule
```





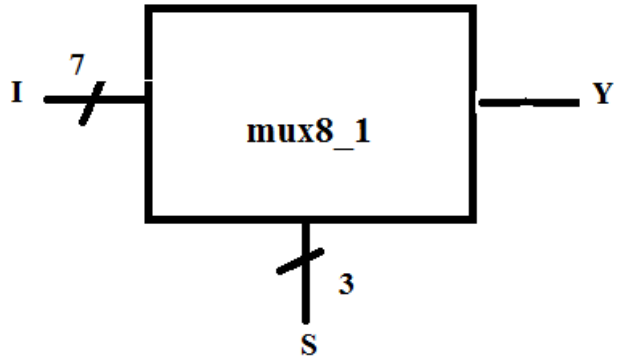

```
module encoder_pri8_3 (Dout,Din);
output [ 2 : 0 ] Dout;
input [ 7 : 0 ] Din;
reg [ 2 : 0 ] Dout;
always@(Din)
begin
casex(Din)
8'b00000001 :Dout = 3'b000;
```

```
8'b0000001X :Dout = 3'b001;

8'b000001XX :Dout = 3'b010;

8'b00001XXX :Dout = 3'b011;

8'b0001XXXX :Dout = 3'b100;

8'b001XXXXX :Dout = 3'b101;

8'b01XXXXXX :Dout = 3'b110;

8'b1XXXXXXX :Dout = 3'b111;

endcase

end

endmodule
```

OR

```
module encoder_pri8_3 (Dout,Din);

input [7:0] Din;

output [2:0] Dout;

reg [2:0] Dout;

always @(Din)

begin

if (Din[7]==1) Dout = 3'd7;

else if(Din[6]==1) Dout=3'd6;

else if(Din[5]==1) Dout=3'd5;

else if(Din[4]==1) Dout=3'd4;

else if(Din[3]==1) Dout=3'd3;

else if(Din[2]==1) Dout=3'd2;

else if(Din[1]==1) Dout=3'd1;

else Dout=3'd0;

end

endmodule
```

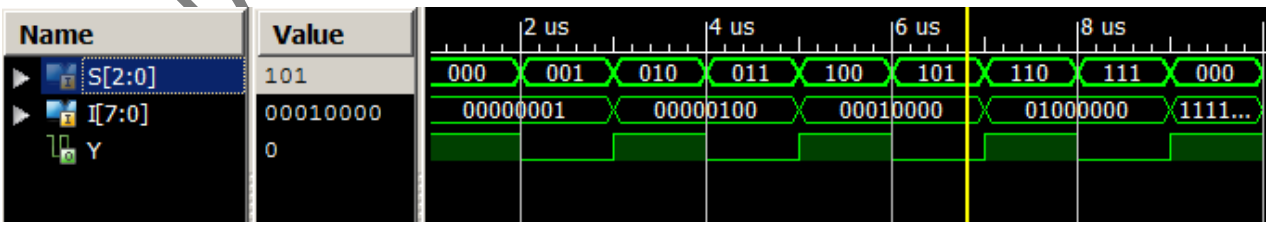
c. 8 to 1 multiplexer using case statement and if statements



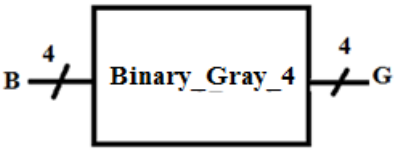

```
module mux8_1(Y,S,I);
output Y;
input [2:0]S;
input[7:0]I;
reg Y;
always@(S,I)
begin
case(S)
3'b000:Y=I[0];
3'b001:Y=I[1];
3'b010:Y=I[2];
3'b011:Y=I[3];
3'b100:Y=I[4];
3'b101:Y=I[5];
3'b110:Y=I[6];
3'b111:Y=I[7];
endcase
end
endmodule
```

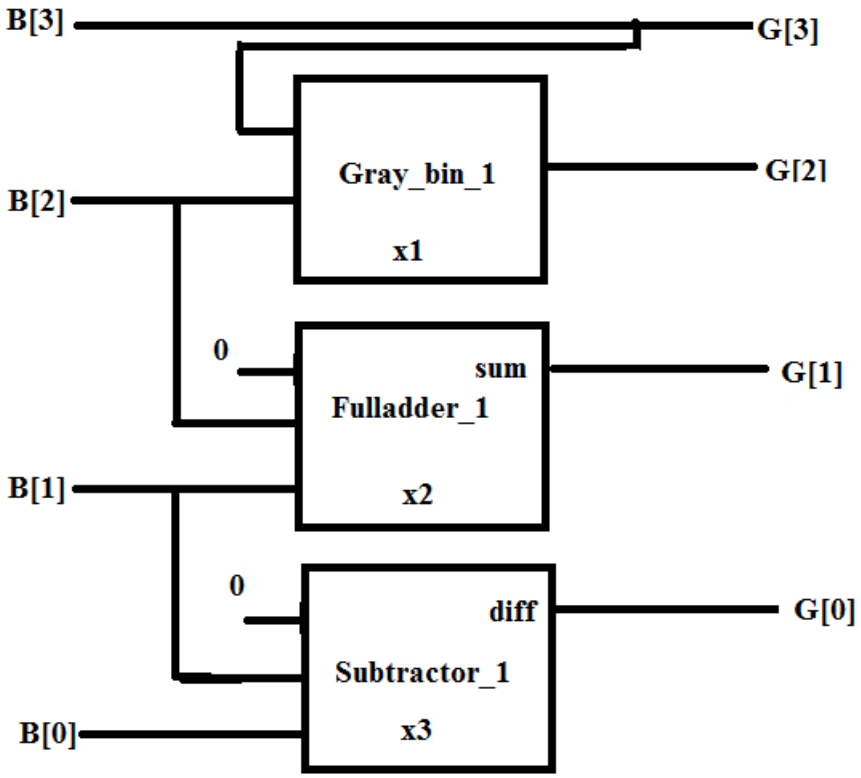
or

```
module mux8_1(Y,S,I);

output Y;

input [2:0]S;

input[7:0]I;

reg Y;

always@(S,I)

begin

if (S==3'b000) Y=I[0];

else if(S==3'b001) Y=I[1];

else if(S==3'b010) Y=I[2];

else if(S==3'b011) Y=I[3];

else if(S==3'b100) Y=I[4];

else if(S==3'b101) Y=I[5];

else if(S==3'b110) Y=I[6];

else if(S==3'b111) Y=I[7];

else Y=3'dz;

end

endmodule
```

d. 4-bit binary to gray converter using 1-bit gray to binary converter 1-bit adder and subtractor

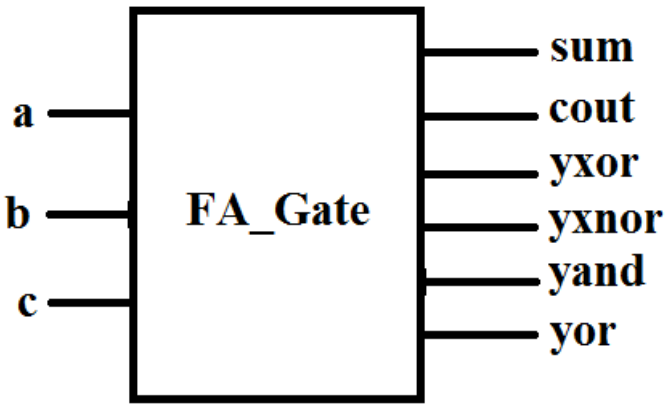




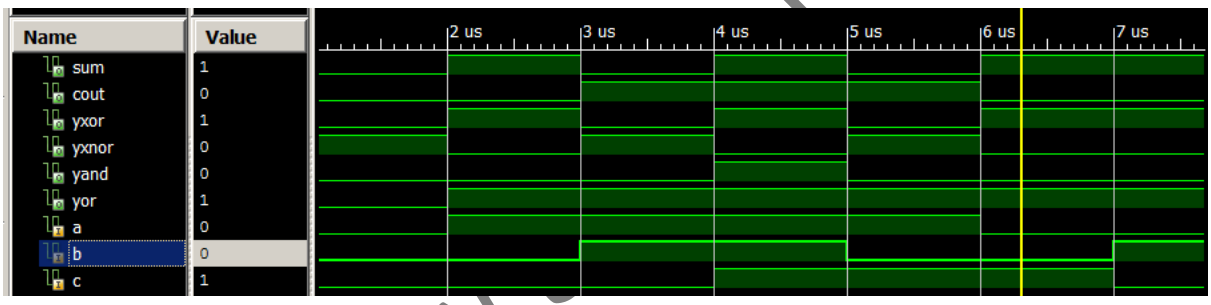

```
module Binary_Gray_4(B,G);
output [3:0]G;
input [3:0]B;
assign G[3]=B[3];
Gray_bin_1 x1(G[2],G[3],B[2]);
Fulladder_1 x2(G[1],  ,1'b0,B[2],B[1]);
Subtractor_1 x3(G[0],  ,1'b0,B[1], B[0]);
endmodule

module Gray_bin_1(b0,g1,g0);
output b0;
input g0,g1;
```

```
assign b0= g0 ^ g1;

endmodule


module Fulladder_1 (sum,cout, a, b, c);

output sum, cout;

input a, b,c;

assign sum= a ^ b ^ c;

assign cout= (a & b) | (b & c) | (c & a);

endmodule


module Subtractor_1 (diff,bout, a, b, c);

output diff, bout;

input a, b,c;

assign diff= a ^ b ^ c;

assign bout= ((~a) & b) | ((((~a) | b)) & c);

endmodule
```

2. Model in Verilog for a full adder and add functionality to perform logical operations of XOR, XNOR, AND and OR gates. Write test bench with appropriate input patterns to verify the modeled behaviour.



```
module fulladder (sum,cout,yxor ,yxnor,yand,yor, a, b, c);

output sum, cout, yxor ,yxnor,yand,yor;

input a, b,c;

assign sum= a ^ b ^ c;

assign cout= (a & b) | (b & c) | (c & a);

assign yxor=a^b^c;

assign yxnor=~(a^b^c);

assign yand=a&b&c;

assign yor=a|b|c;

endmodule
```



3. Verilog 32-bit ALU shown in figure below and verify the functionality of ALU by selecting appropriate test patterns. The functionality of the ALU is presented in Table 1.

a. Write test bench to verify the functionality of the ALU considering all possible input patterns

b. The enable signal will set the output to required functions if enabled, if disabled all the outputs are set to tri-state

c. The acknowledge signal is set high after every operation is complete

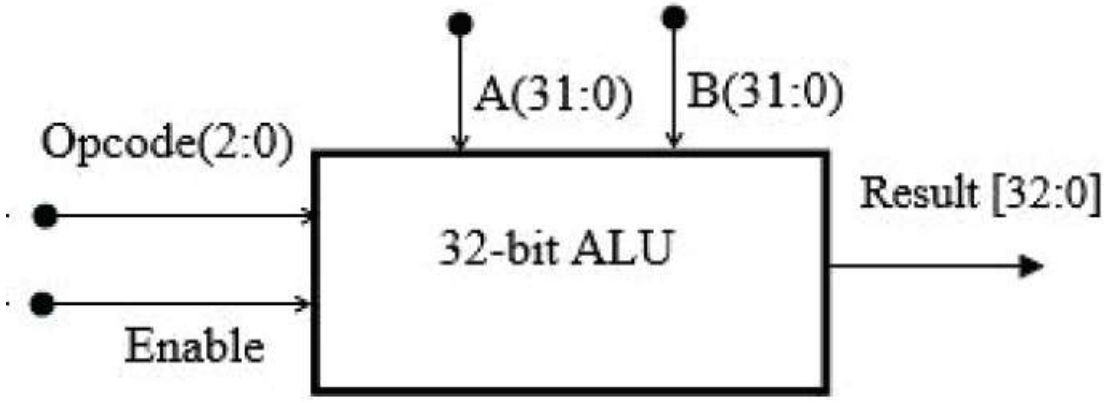


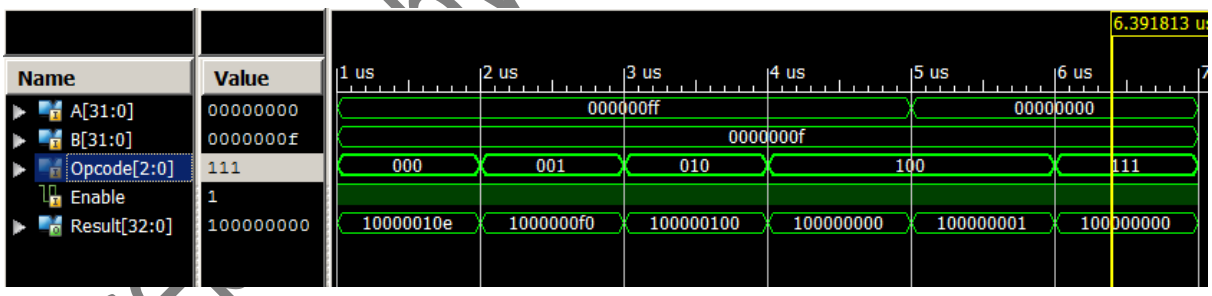
Table 1 ALU Functions

Opcode (2:0)	ALU Operation	Remarks	
000	A + B	Addition of two numbers	Both A and B are in two's complement format
001	A – B	Subtraction of two numbers	
010	A + 1	Increment Accumulator by 1	A is in two's complement format
011	A - 1	Decrement accumulator by 1	
100	A	True	Inputs can be in any format
101	A Complement	Complement	
110	A OR B	Logical OR	
111	A AND B	Logical AND	

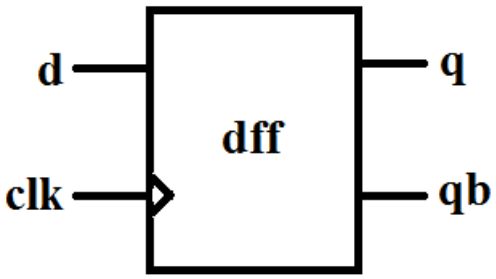
Result[32] is acknowledge signal for completion of operation

```
module alu_3(Result, A, B, Opcode, Enable);
output [32:0] Result;
input signed[31:0] A, B;
input [2:0] Opcode;
input Enable;
reg [32:0] Result;
always@(Opcode,A,B,Enable)
begin
if(Enable==0)
                                        begin
                                        Result=31'bx;
                                        end
```

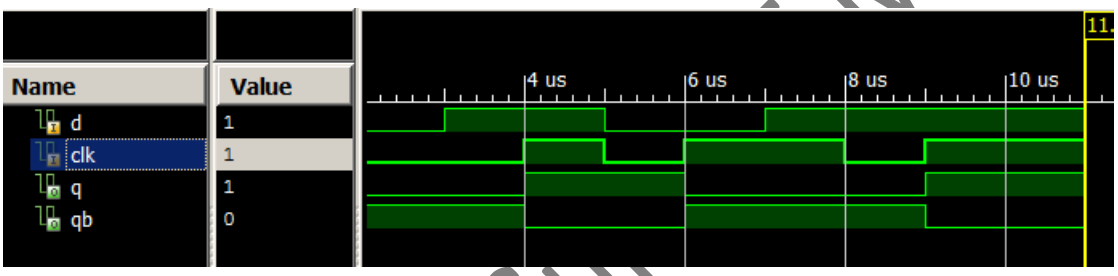
```
else
                                    begin
                                    case(Opcode)
                                    3'b000: begin Result=A+B; end
                                    3'b001: begin Result=A-B; end
                                    3'b010: begin Result=A+1; end
                                    3'b011: begin Result=A-1; end
                                    3'b100: begin Result=!A; end
                                    3'b101: begin Result=~A; end
                                    3'b110: begin Result=A|B; end
                                    3'b111: begin Result=A&B; end
                                    endcase
                                    Result[32]=1'b1;
                                    end
end
endmodule
```

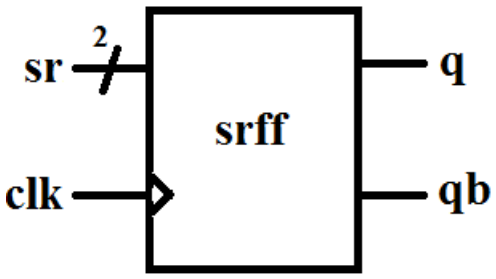


4. Write Verilog code for SR, D and JK and verify the flip flop.



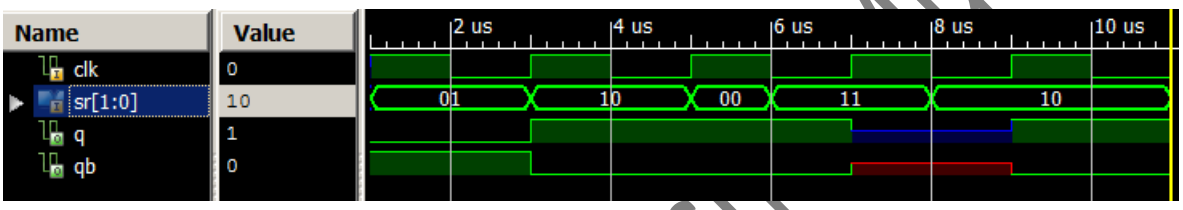
```
module dff(q,qb,d,clk);
output q,qb;
input d,clk;
reg q=0,qb=1;
always@(posedge clk)
begin
q= d;
qb=~q;
end
endmodule
```

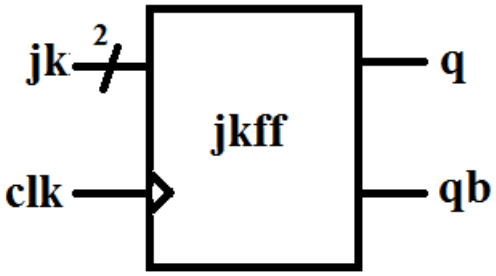





```
module srff(q,qb,sr,clk);
output q,qb;
input clk;
input[1:0]sr;
reg q=0,qb=1;
always@(posedge clk)
begin
```

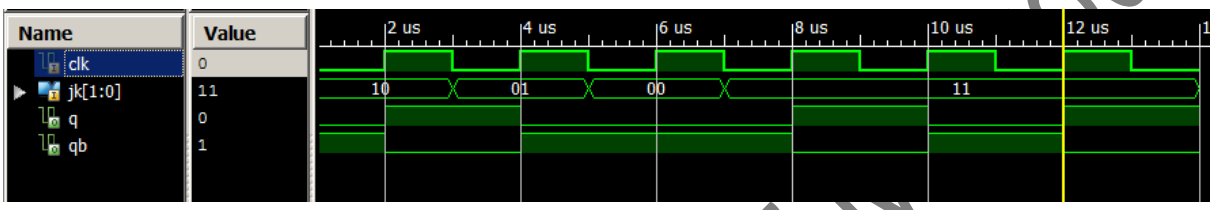
```
case(sr)
2'b00:q=q;
2'b01:q=0;
2'b10:q=1;
2'b11:q=1'bZ;
endcase
qb=~q;
end
endmodule
```



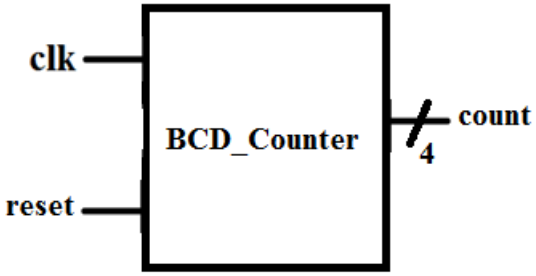



```
module jkff(q,qb,jk,clk);
output q,qb;
input clk;
input [1:0]jk;
reg q=0,qb=1;
always@(posedge clk)
begin
case (jk)
2'b00:q=q;
2'b01:q=0;
```

```
2'b10:q=1;

2'b11:q=~q;

endcase

qb=~q;

end

endmodule
```

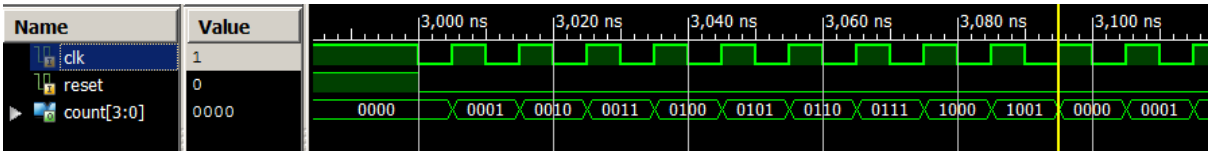


5. Write Verilog code for 4-bit BCD synchronous counter.

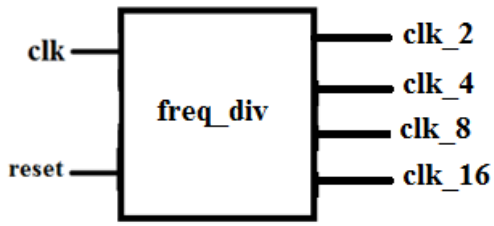



```
module BCD_Counter(count, clk,reset);

output [3:0] count;

input clk,reset;

reg[3:0]count=4'b0000;

always@(posedge clk)

begin

if((reset==1) | (count==4'b1001))

count = 4'b0000;

else

count = count+1;
```

```
end

endmodule
```



6. Write Verilog code for counter with given input clock and check whether it works as clock divider performing division of clock by 2, 4, 8 and 16. Verify the functionality of the code.




```
module freq_div(clk_2,clk_4,clk_8,clk_16, clk,reset);

output clk_2,clk_4,clk_8,clk_16;

input clk,reset;

reg clk_2,clk_4,clk_8,clk_16;

reg[3:0]count=4'b0000;

always@(posedge clk)

begin

if(reset==1)

begin count = 4'b0000; end

else

begin count = count+1; end

clk_2=count[0];

clk_4=count[1];

clk_8=count[2];

clk_16=count[3];
```

```
end

endmodule
```