Problem Statement:

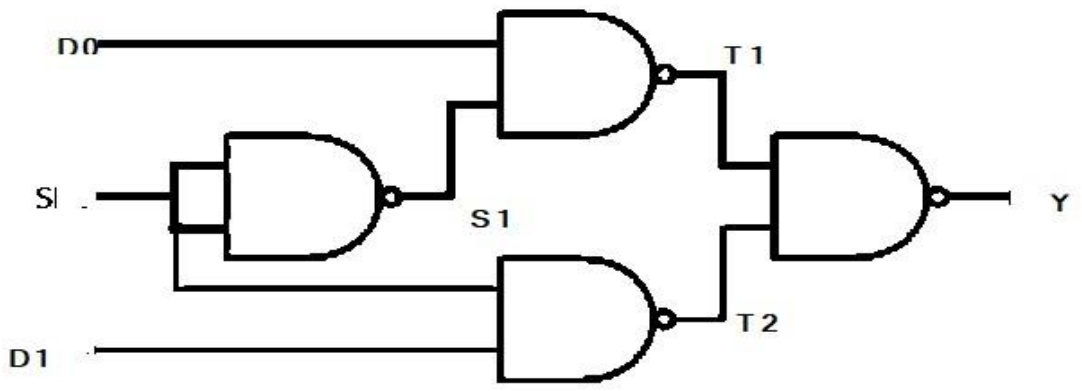
(a) Write a Verilog module to implement a 2-to-1 multiplexer (MUX) at (i) structural level using elementary two-input logic gates (NOT, NAND, NOR), (ii) behavioural level. Using this module and other elementary two-input logic gates as necessary, build a 2 n -to-1 MUX that can implement any Boolean function of 5 variables.

(b) Write a Verilog module to implement a clock-enabled JK flip-flop (Jack Kilby flipflop) at (i) structural level using elementary two-input logic gates (NOT, NAND, NOR), (ii) behavioural level. Using this module and other two-input logic gates as necessary, build a four-bit synchronous binary counter.

Solution:

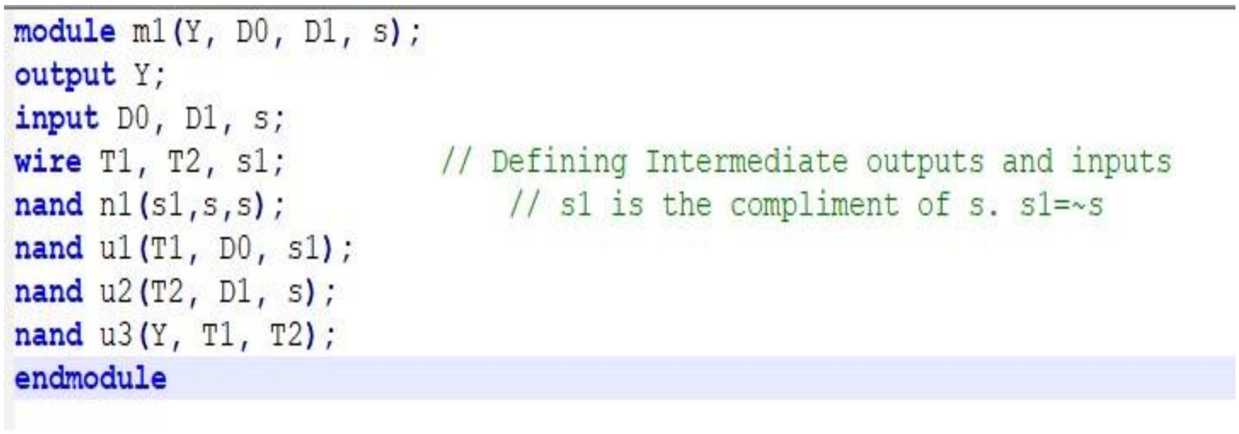
(a) Verilog Module for 2-to-1 multiplexer (MUX)

i. **Structural Level:**



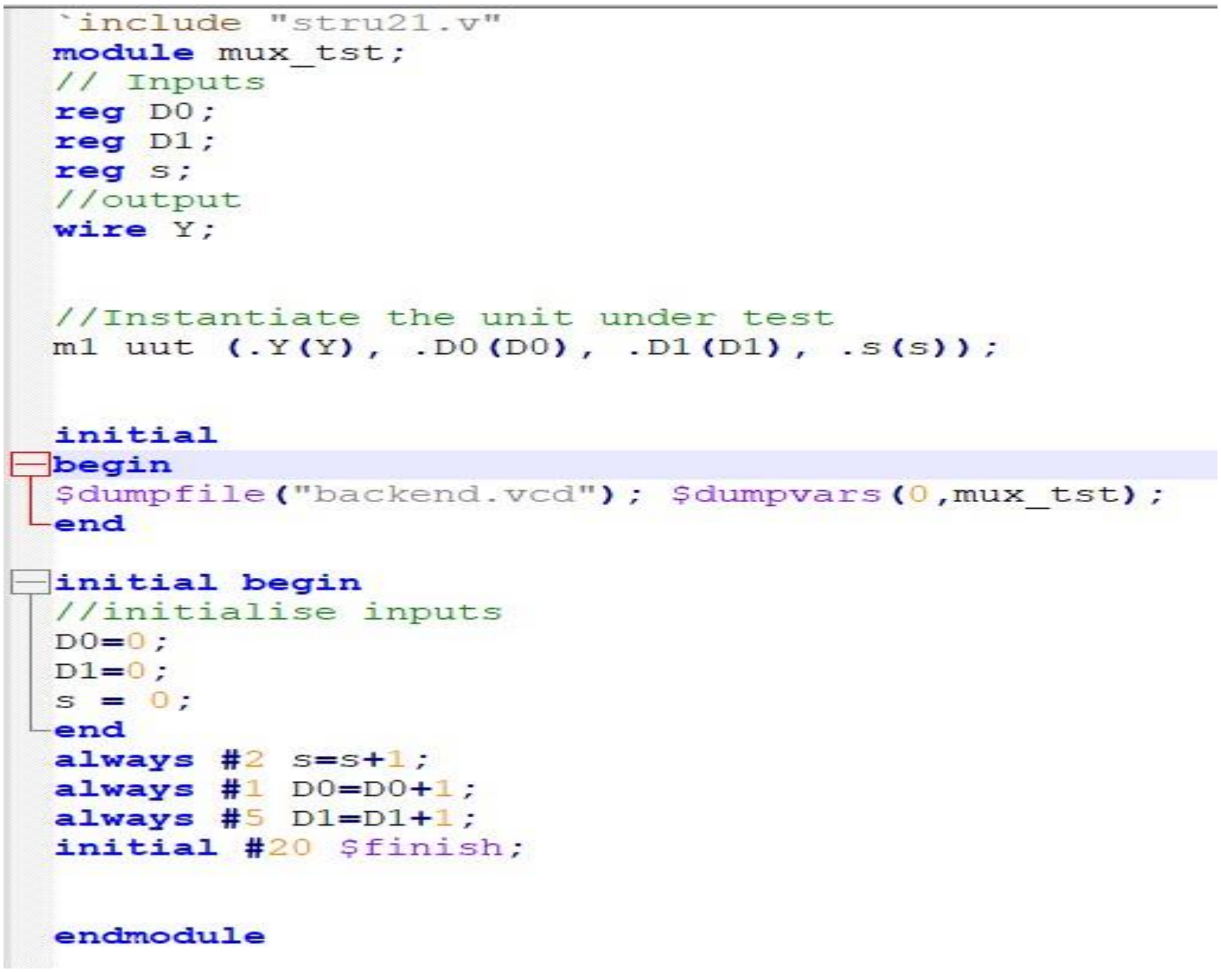
Equation from the circuit: Y = D0.S' + D1.S

- **Module:**



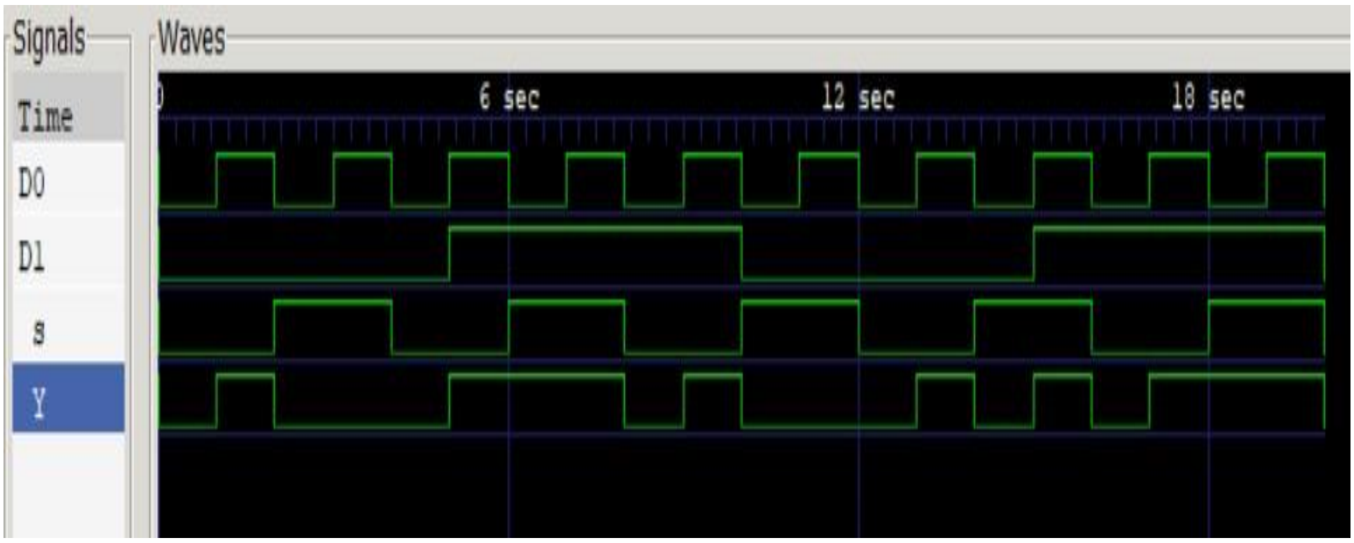
```
module m1(Y, D0, D1, s);
output Y;
input D0, D1, s;
wire T1, T2, s1;          // Defining Intermediate outputs and inputs
nand n1(s1,s,s);              // s1 is the compliment of s. s1=~s
nand u1(T1, D0, s1);
nand u2(T2, D1, s);
nand u3(Y, T1, T2);
endmodule
```

- **Test Bench module:**



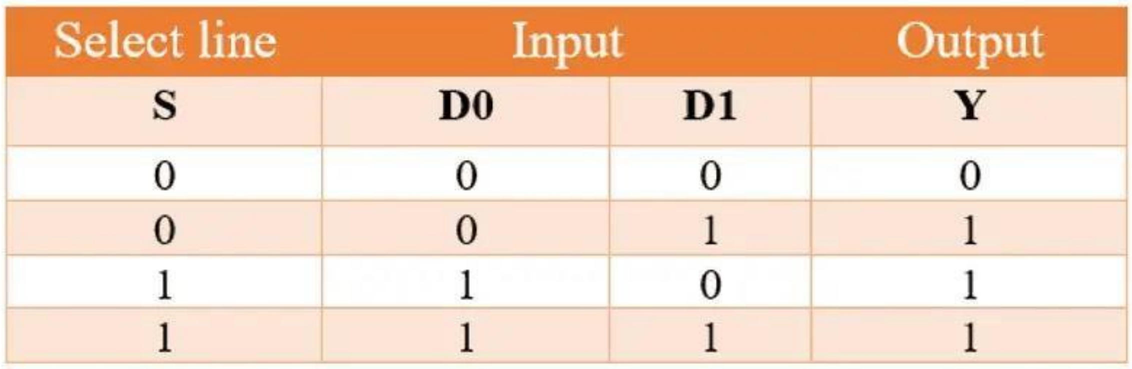
```
`include "stru21.v"
module mux_tst;
// Inputs
reg D0;
reg D1;
reg s;
//output
wire Y;


//Instantiate the unit under test
m1 uut (.Y(Y), .D0(D0), .D1(D1), .s(s));


initial
begin
$dumpfile("backend.vcd"); $dumpvars(0,mux_tst);
end

initial begin
//initialise inputs
D0=0;
D1=0;
s = 0;
end
always #2 s=s+1;
always #1 D0=D0+1;
always #5 D1=D1+1;
initial #20 $finish;


endmodule
```

- **Waveforms:**



ii. **Behavioural Level:**

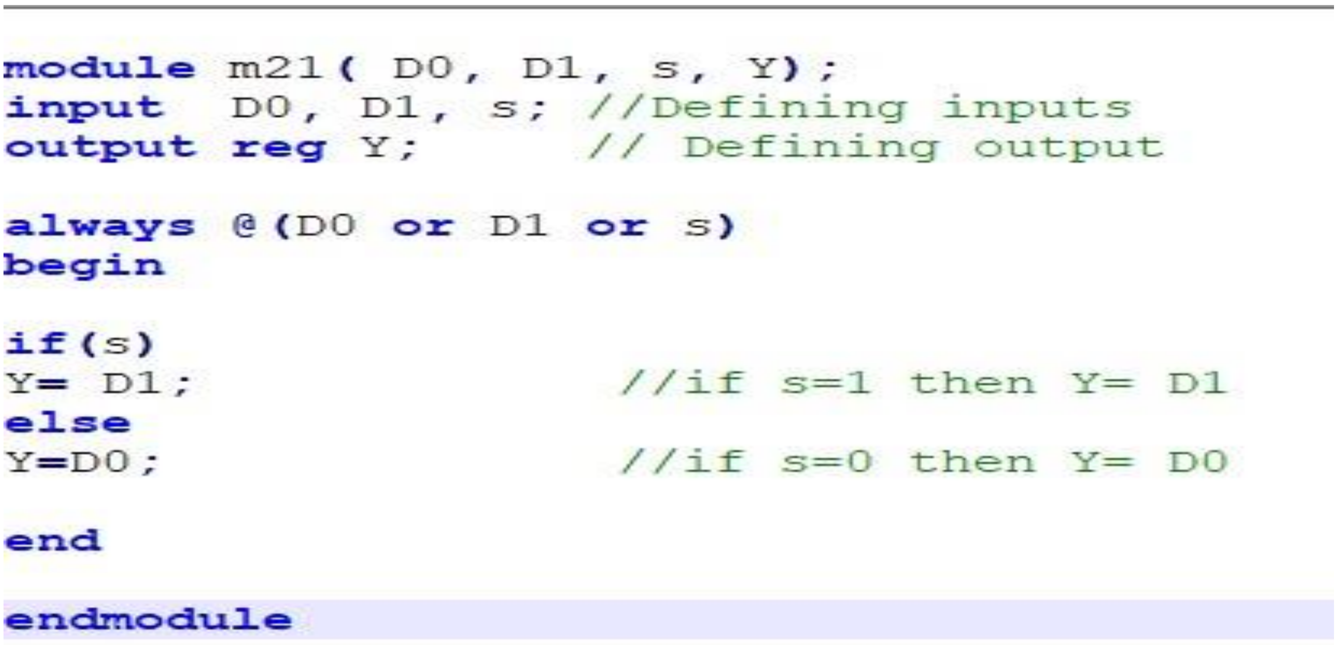
Truth Table:



Select line	Input		Output
S	**D0**	**D1**	**Y**
0	0	0	0
0	0	1	1
1	1	0	1
1	1	1	1

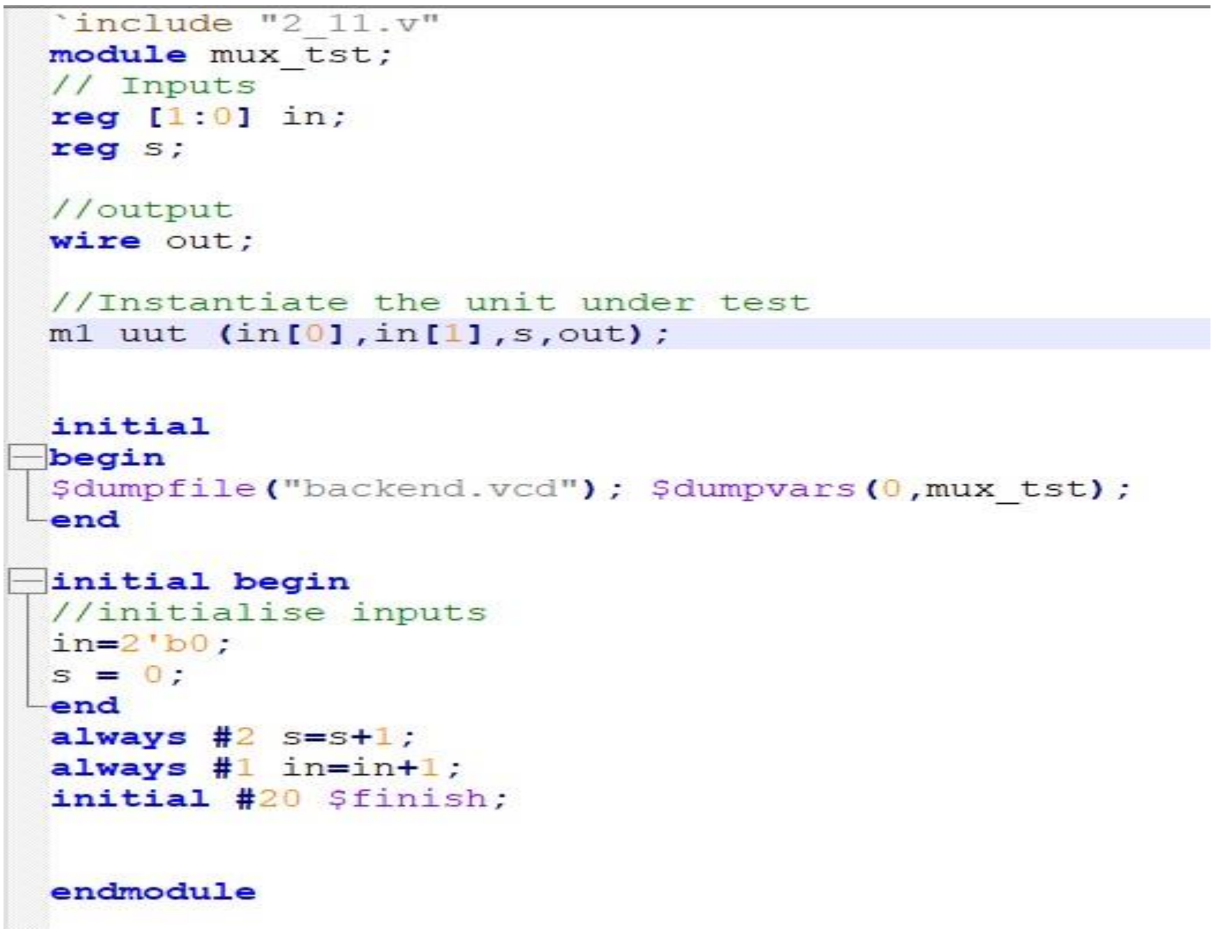
Equation from the truth table: Y = D0.S’ + D1.S

- **Module:**



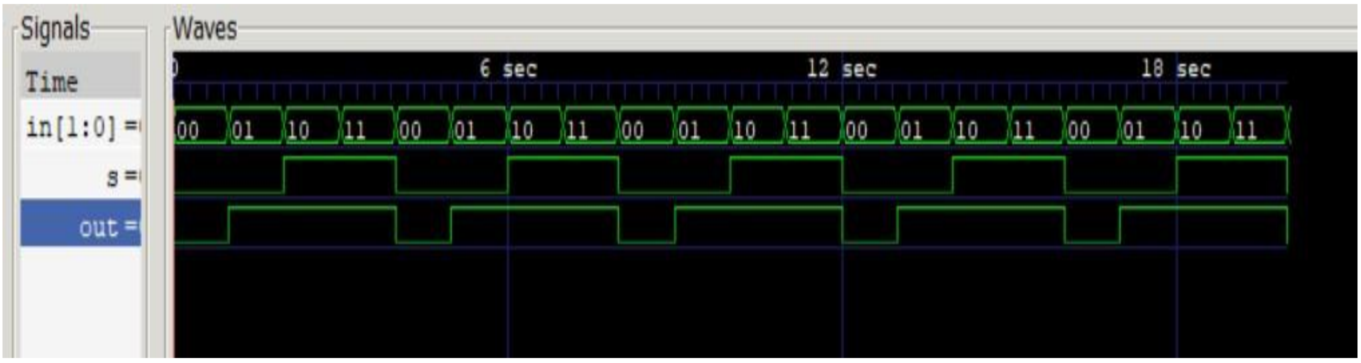
```
module m21( D0, D1, s, Y);
input  D0, D1, s; //Defining inputs
output reg Y;     // Defining output

always @(D0 or D1 or s)
begin

if(s)
Y= D1;                  //if s=1 then Y= D1
else
Y=D0;                   //if s=0 then Y= D0

end

endmodule
```

- **Test Bench:**



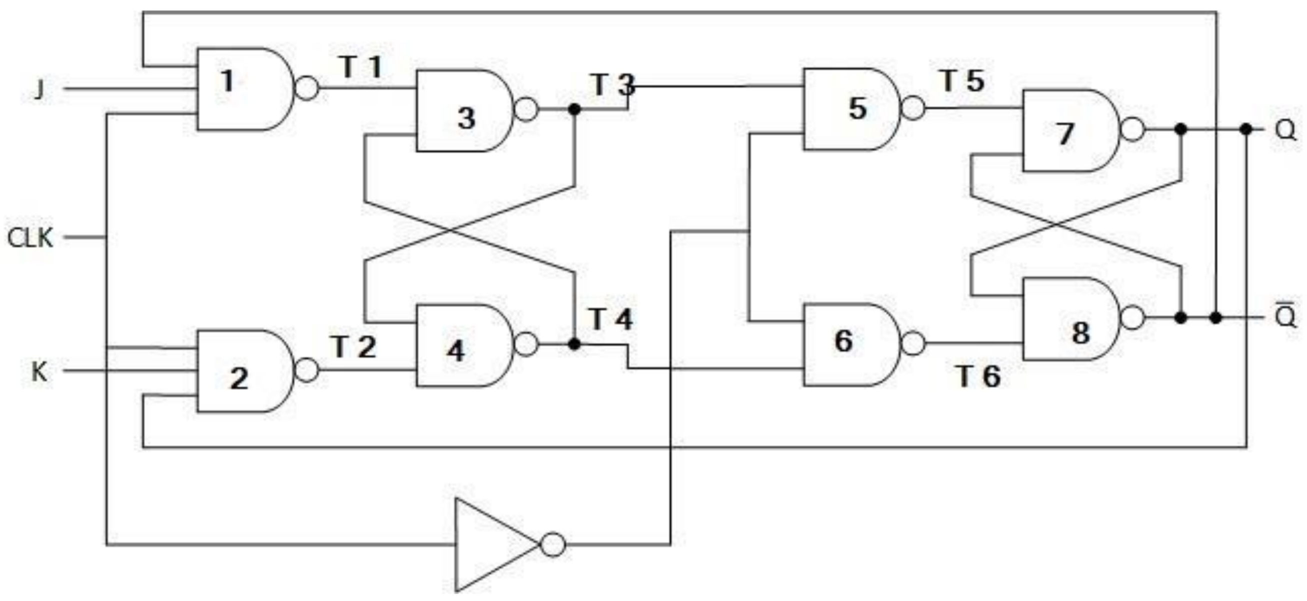
```
`include "2_11.v"
module mux_tst;
// Inputs
reg [1:0] in;
reg s;

//output
wire out;

//Instantiate the unit under test
m1 uut (in[0],in[1],s,out);


initial
begin
$dumpfile("backend.vcd"); $dumpvars(0,mux_tst);
end

initial begin
//initialise inputs
in=2'b0;
s = 0;
end
always #2 s=s+1;
always #1 in=in+1;
initial #20 $finish;


endmodule
```

- **Waveform:**

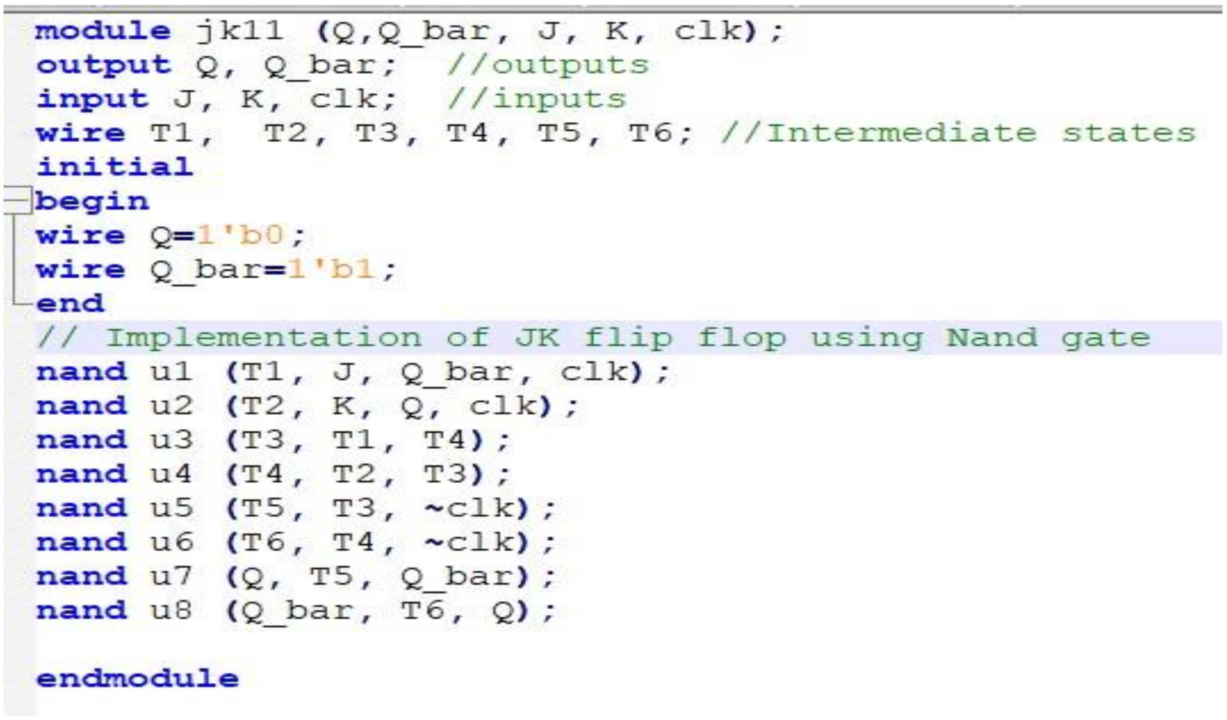


(b) <u>Verilog module to implement a clock-enabled JK flip-flop:</u>

i. Structural Level:



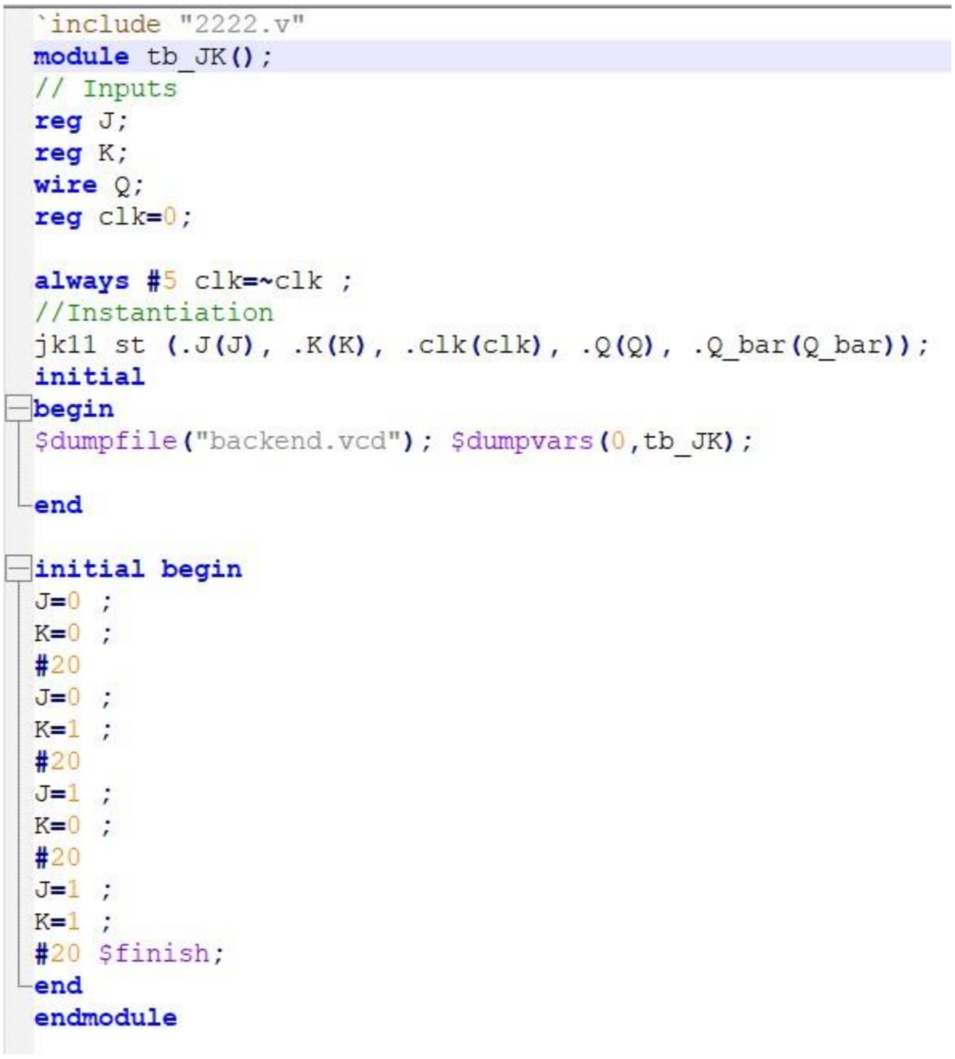
- **Module:**



```
module jk11 (Q,Q_bar, J, K, clk);
output Q, Q_bar;   //outputs
input J, K, clk;   //inputs
wire T1,  T2, T3, T4, T5, T6; //Intermediate states
initial
begin
wire Q=1'b0;
wire Q_bar=1'b1;
end
// Implementation of JK flip flop using Nand gate
nand u1 (T1, J, Q_bar, clk);
nand u2 (T2, K, Q, clk);
nand u3 (T3, T1, T4);
nand u4 (T4, T2, T3);
nand u5 (T5, T3, ~clk);
nand u6 (T6, T4, ~clk);
nand u7 (Q, T5, Q_bar);
nand u8 (Q_bar, T6, Q);

endmodule
```

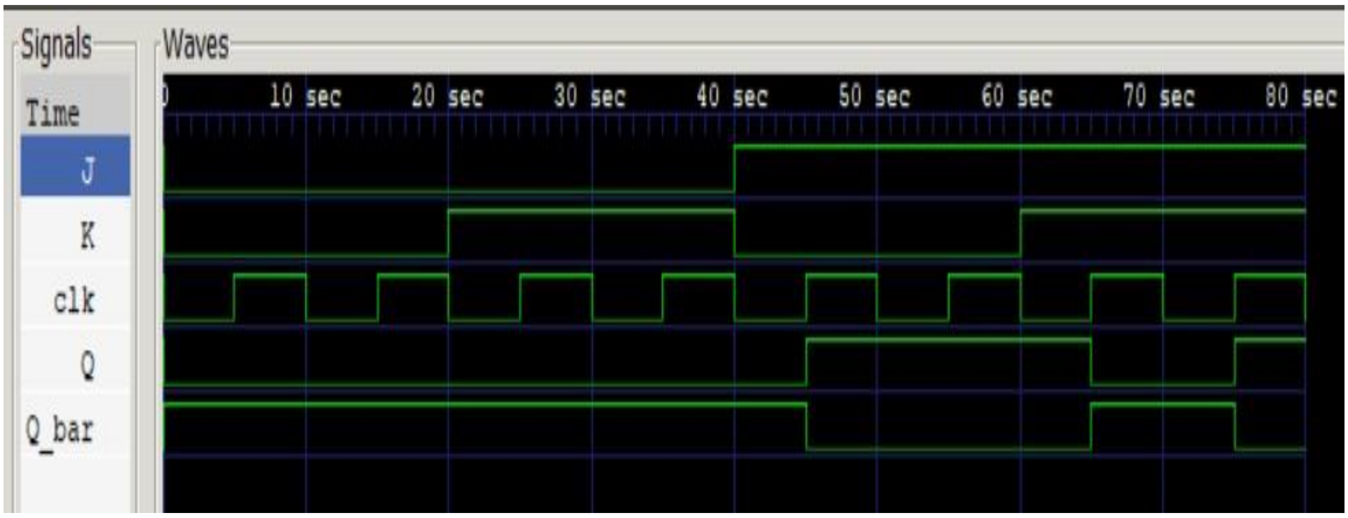
•

Test Bench



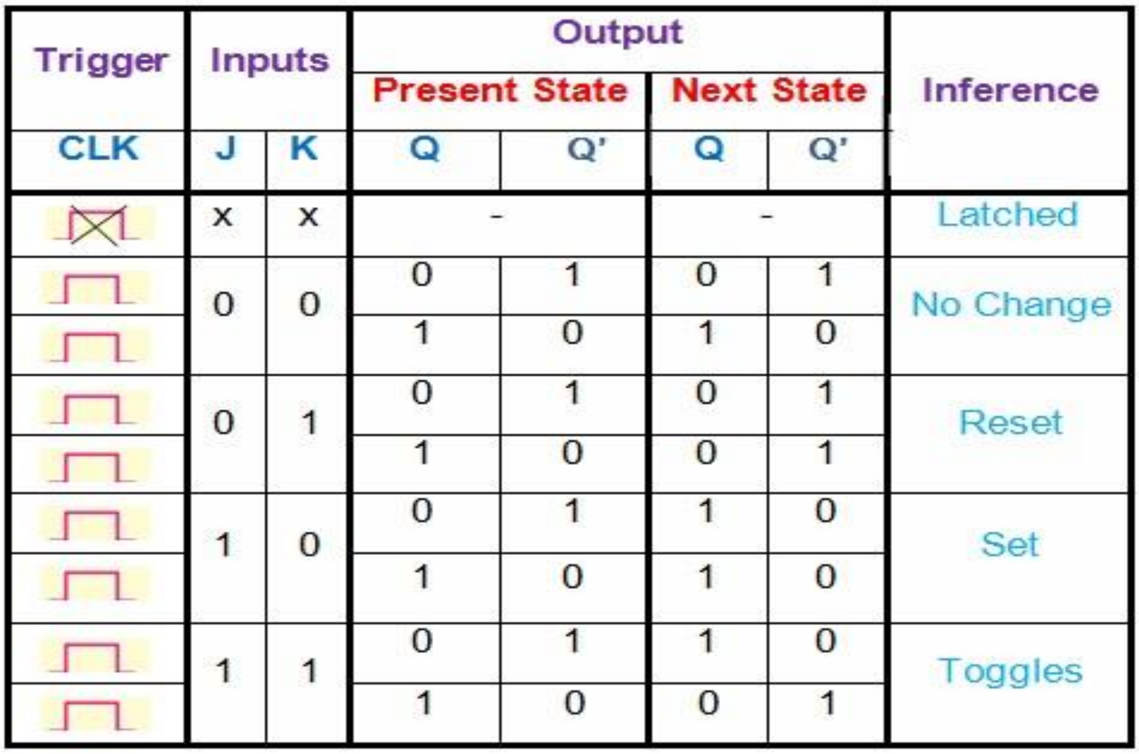
```
`include "2222.v"
module tb_JK();
// Inputs
reg J;
reg K;
wire Q;
reg clk=0;

always #5 clk=~clk ;
//Instantiation
jk11 st (.J(J), .K(K), .clk(clk), .Q(Q), .Q_bar(Q_bar));
initial
begin
$dumpfile("backend.vcd"); $dumpvars(0,tb_JK);

end

initial begin
J=0 ;
K=0 ;
#20
J=0 ;
K=1 ;
#20
J=1 ;
K=0 ;
#20
J=1 ;
K=1 ;
#20 $finish;
end
endmodule
```

• **Waveform:**



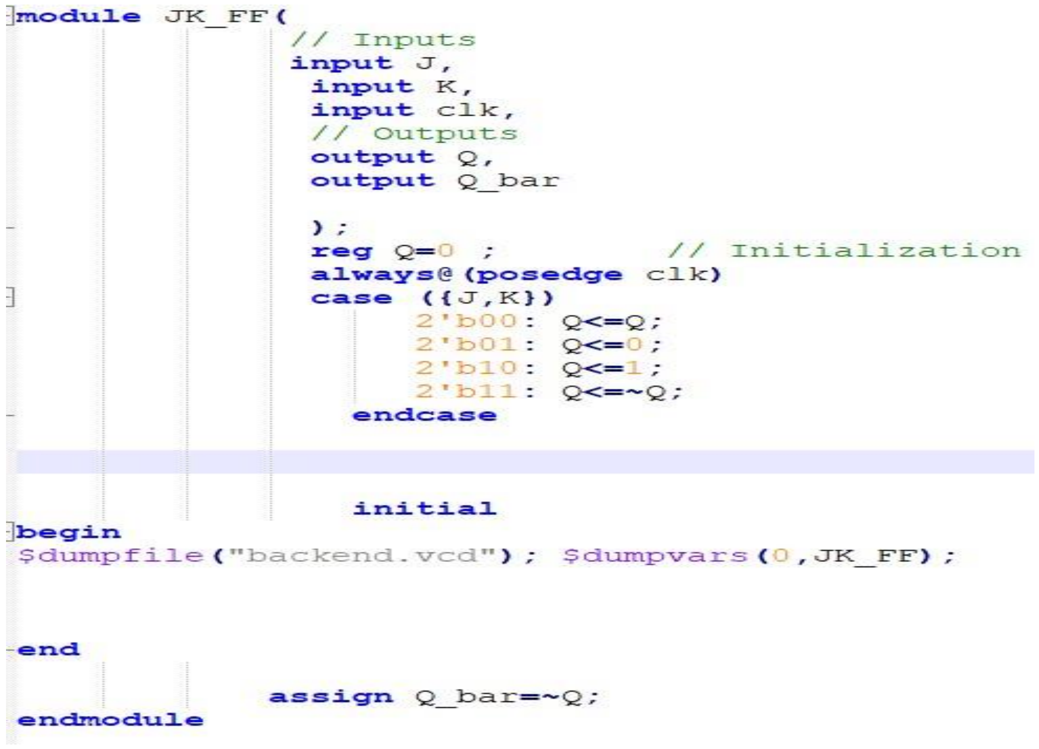
ii. Behavioural Level:

Truth Table:



Trigger	Inputs		Output				Inference
			Present State		Next State		
CLK	J	K	Q	Q'	Q	Q'	
⊠	x	x	-		-		Latched
⎍	0	0	0	1	0	1	No Change
⎍			1	0	1	0	
⎍	0	1	0	1	0	1	Reset
⎍			1	0	0	1	
⎍	1	0	0	1	1	0	Set
⎍			1	0	1	0	
⎍	1	1	0	1	1	0	Toggles
⎍			1	0	0	1	

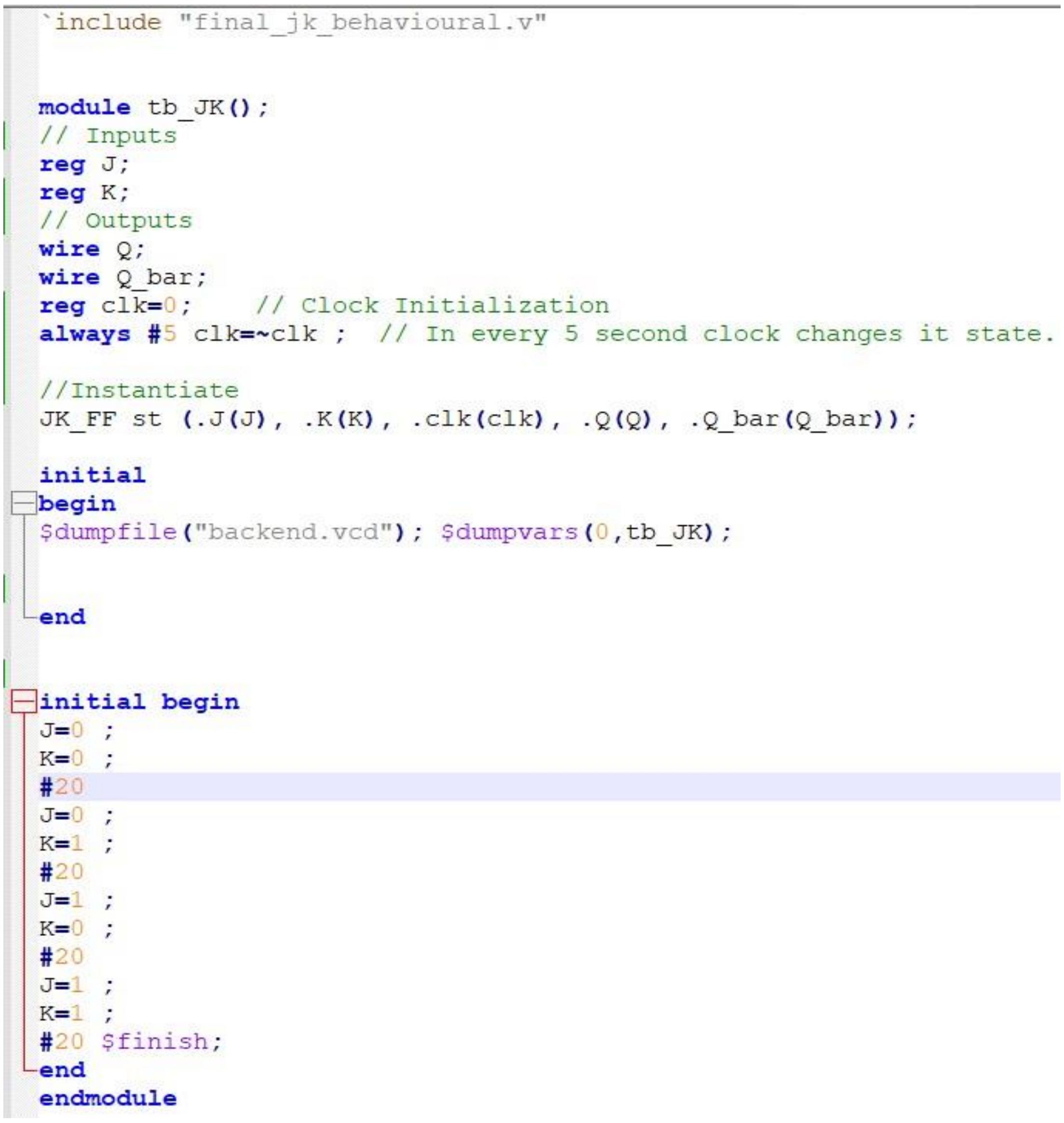
- **Module:**



```
module  JK_FF(
                // Inputs
                input J,
                 input K,
                 input clk,
                 // Outputs
                 output Q,
                 output Q_bar

                 );
                 reg Q=0  ;          // Initialization
                 always@(posedge clk)
                 case  ({J,K})
                      2'b00:  Q<=Q;
                      2'b01:  Q<=0;
                      2'b10:  Q<=1;
                      2'b11:  Q<=~Q;
                   endcase


                   initial
begin
$dumpfile("backend.vcd"); $dumpvars(0,JK_FF);


end
               assign Q_bar=~Q;
endmodule
```

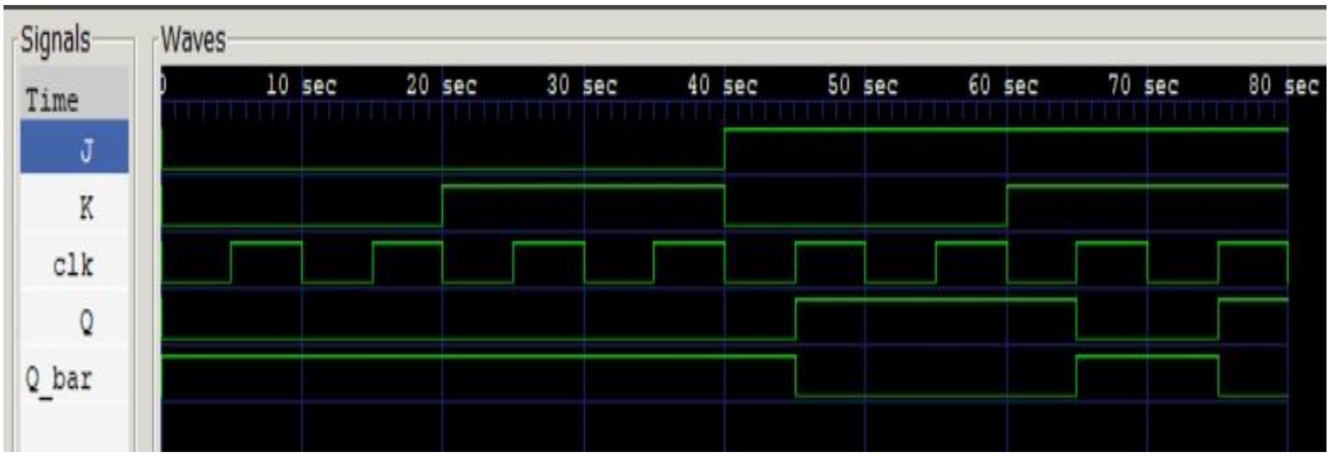
•

Test Bench:

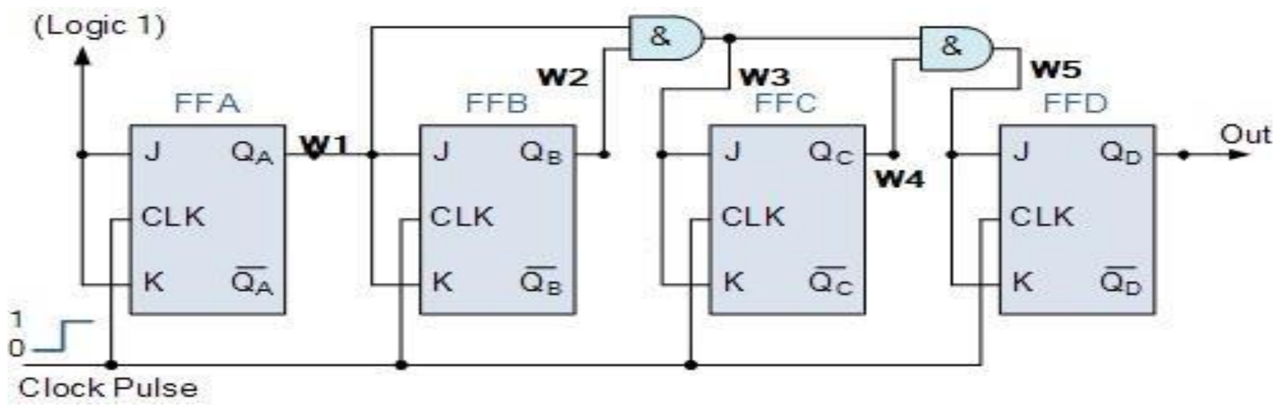


```
`include "final_jk_behavioural.v"


module tb_JK();
// Inputs
reg J;
reg K;
// Outputs
wire Q;
wire Q_bar;
reg clk=0;     // Clock Initialization
always #5 clk=~clk ;  // In every 5 second clock changes it state.

//Instantiate
JK_FF st (.J(J), .K(K), .clk(clk), .Q(Q), .Q_bar(Q_bar));

initial
begin
$dumpfile("backend.vcd"); $dumpvars(0,tb_JK);


end


initial begin
J=0 ;
K=0 ;
#20
J=0 ;
K=1 ;
#20
J=1 ;
K=0 ;
#20
J=1 ;
K=1 ;
#20 $finish;
end
endmodule
```

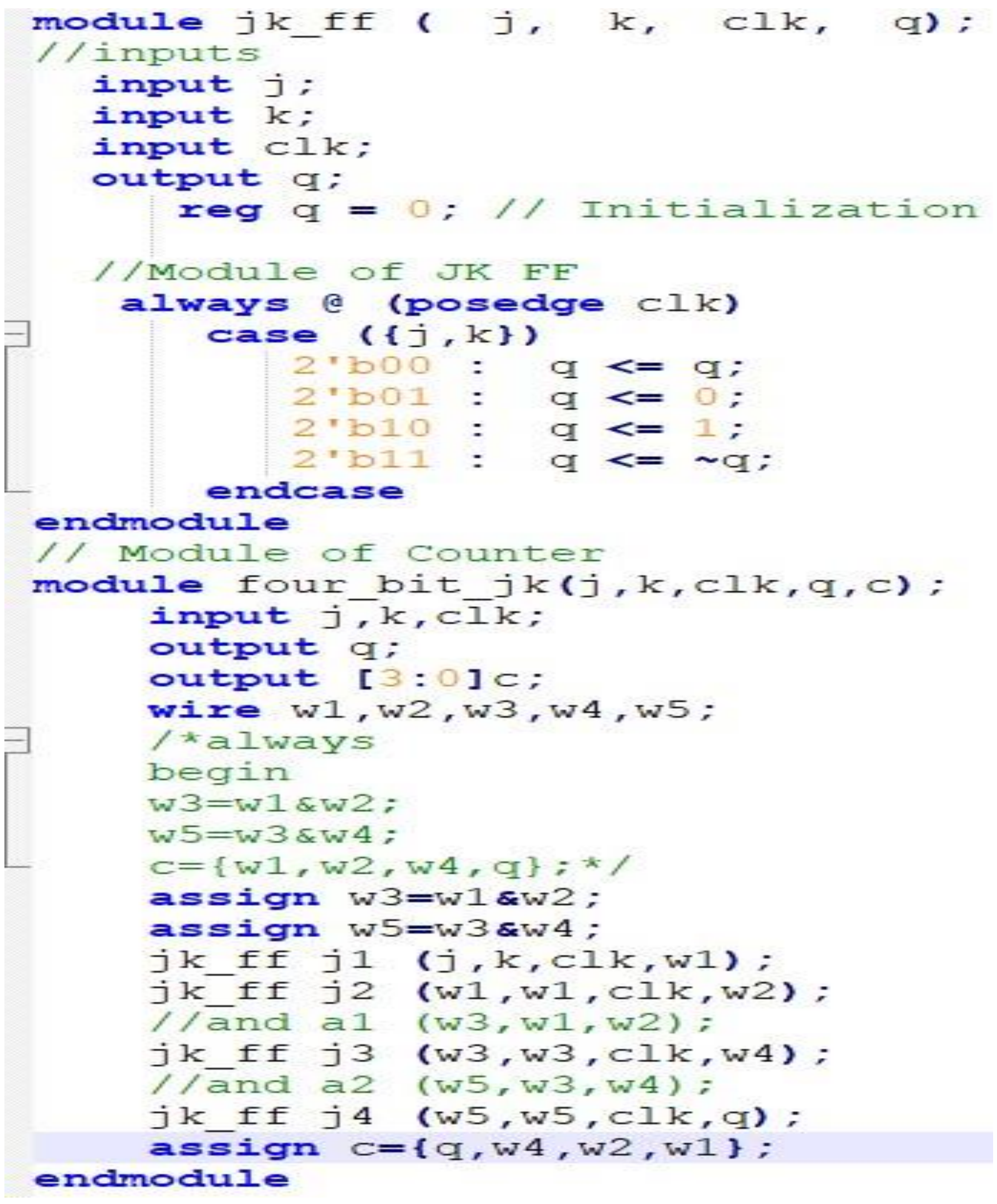
• **Waveforms:**



iii. <u>**4 bit Synchronous Counter using JK Flip Flop:**</u>



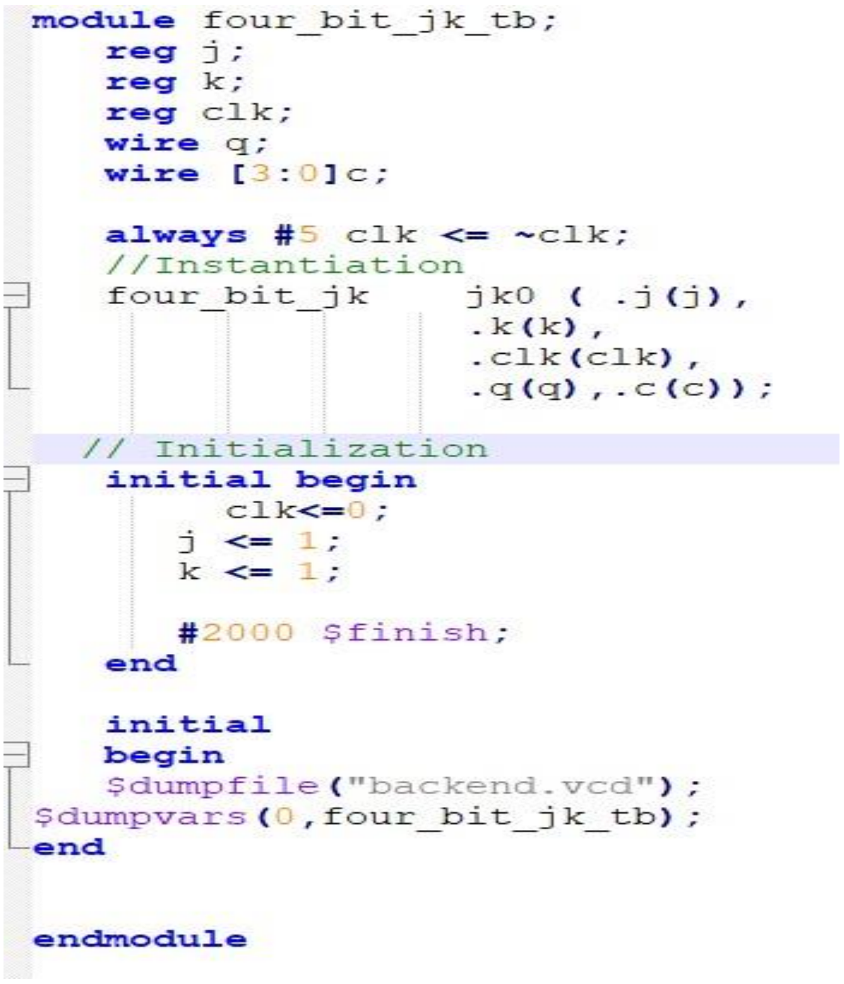
- **Module:**



```
module jk_ff (  j,  k,  clk,  q);
//inputs
  input j;
  input k;
  input clk;
  output q;
     reg q = 0; // Initialization

  //Module of JK FF
   always @ (posedge clk)
      case ({j,k})
         2'b00 :  q <= q;
         2'b01 :  q <= 0;
         2'b10 :  q <= 1;
         2'b11 :  q <= ~q;
      endcase
endmodule
// Module of Counter
module four_bit_jk(j,k,clk,q,c);
    input j,k,clk;
    output q;
    output [3:0]c;
    wire w1,w2,w3,w4,w5;
    /*always
    begin
    w3=w1&w2;
    w5=w3&w4;
    c={w1,w2,w4,q};*/
    assign w3=w1&w2;
    assign w5=w3&w4;
    jk_ff j1 (j,k,clk,w1);
    jk_ff j2 (w1,w1,clk,w2);
    //and a1 (w3,w1,w2);
    jk_ff j3 (w3,w3,clk,w4);
    //and a2 (w5,w3,w4);
    jk_ff j4 (w5,w5,clk,q);
    assign c={q,w4,w2,w1};
endmodule
```

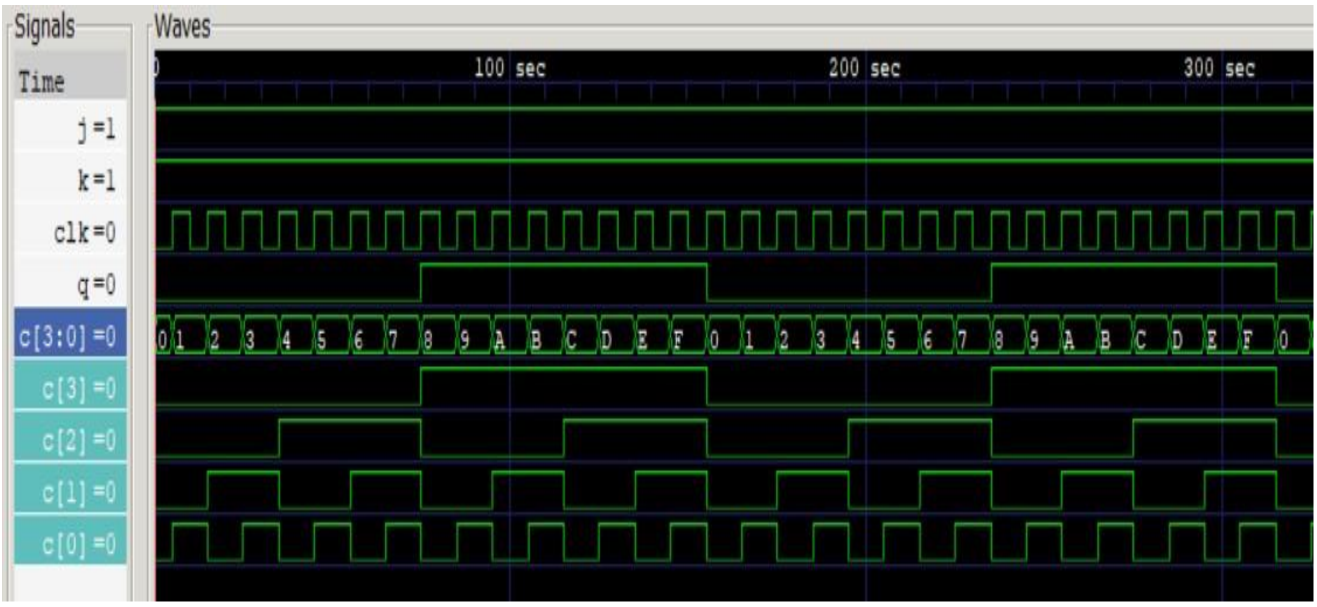
Test Bench:

•



```
module four_bit_jk_tb;
    reg j;
    reg k;
    reg clk;
    wire q;
    wire [3:0]c;

    always #5 clk <= ~clk;
    //Instantiation
    four_bit_jk    jk0 ( .j(j),
                        .k(k),
                        .clk(clk),
                        .q(q),.c(c));

  // Initialization
    initial begin
          clk<=0;
        j <= 1;
        k <= 1;

        #2000 $finish;
    end

    initial
    begin
    $dumpfile("backend.vcd");
$dumpvars(0,four_bit_jk_tb);
end


endmodule
```

• **Waveform:**


Signals
Waves
Time
100 sec
200 sec
300 sec
j=1
k=1
clk=0
q=0
c[3:0]=0
c[3]=0
c[2]=0
c[1]=0
c[0]=0
0 1 2 3 4 5 6 7 8 9 A B C D E F 0 1 2 3 4 5 6 7 8 9 A B C D E F 0