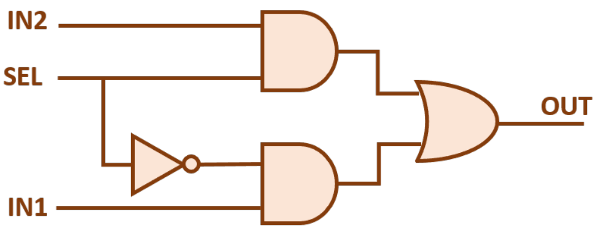
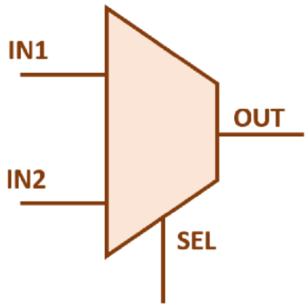
1. Implementing MUX based logic and JK synchronous counter:

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.

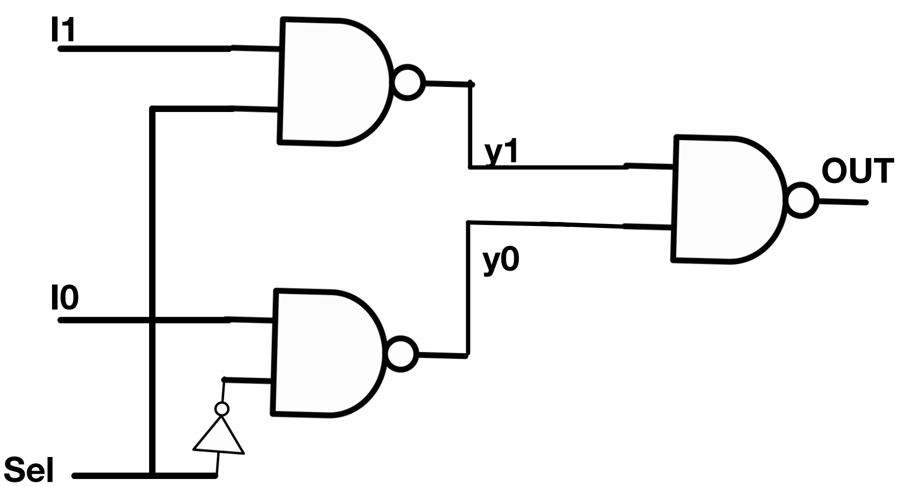
The following figure represents the general 2-to-1 multiplexer(MUX).





SEL	OUT
0	IN1
1	IN2

i) Implementation of MUX using two-input logic gates(NOT, NAND, NOR)

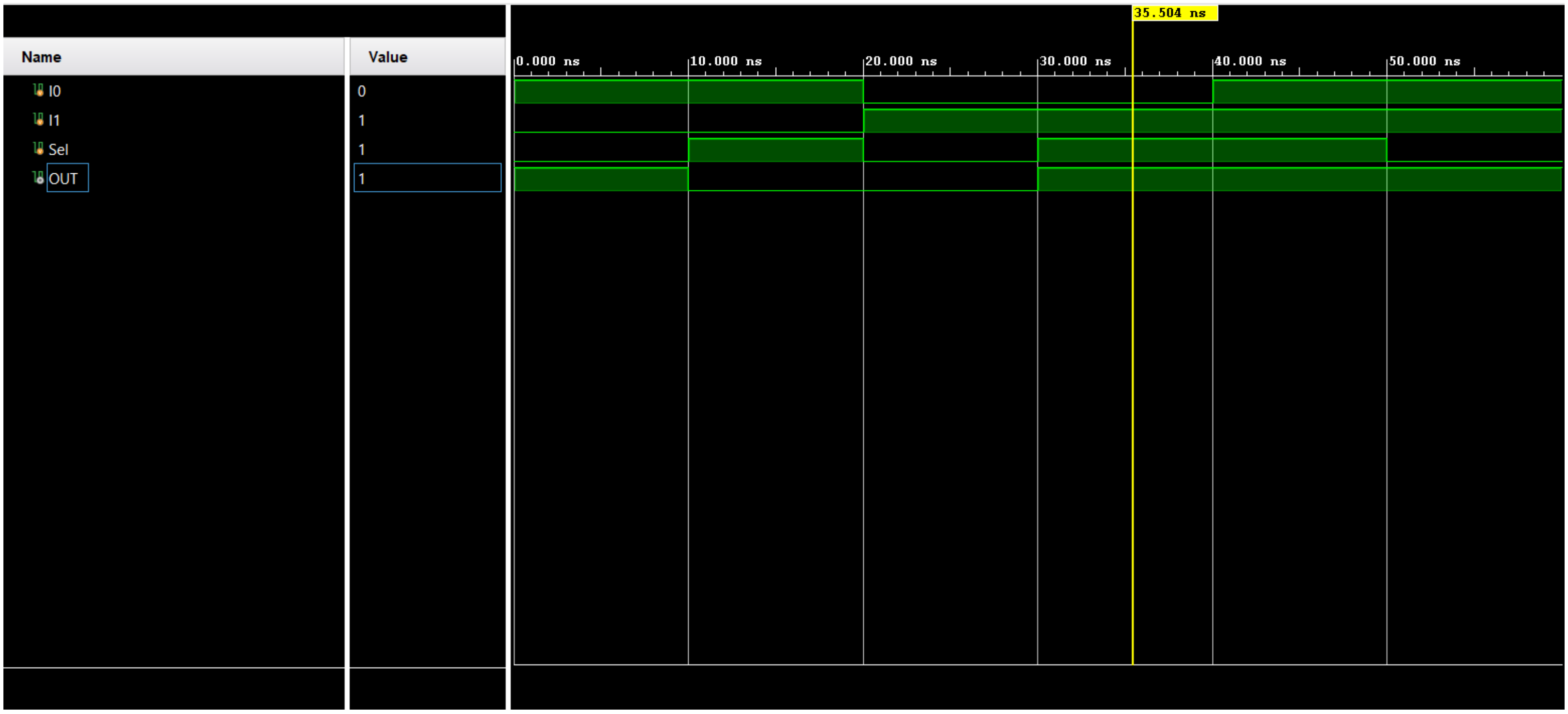


Sel	y0	y1	OUT
0	$\overline{I0}$	1	I0
1	1	$\overline{I1}$	I1

VERILOG CODE:

```verilog
module mux_2to1(OUT,I1,I0,Sel); //Defines a module mux with 4 ports OUT,Iq,I0,Sel
input I0,I1,Sel; //Defines inputs to the module I0,I1, Sel
output OUT; //Defines output for the module OUT
wire y0,y1,Sel_n; //Declared three wires: Sel_n, y0, and y1
not(Sel_n,Sel); //Sel_n is complement of Sel
nand(y0,I0,Sel_n); //Defines y0 as the output nand gate, while I0 and Sel_n are inputs
nand(y1,I1,Sel); //Defines y1 as the output nand gate, while I1 and Sel_n are inputs
nand(OUT,y0,y1); //Defines OUT as the output nand gate, while yo and y1 are inputs
endmodule //Defines end of Module defination
```

Waveform:

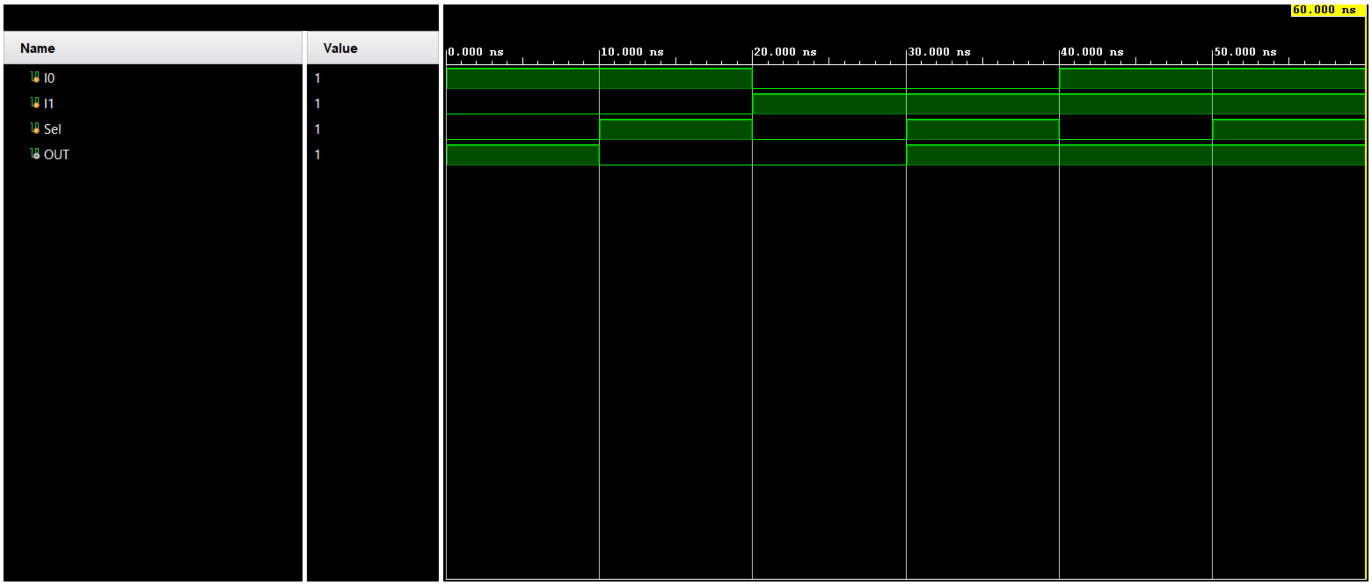


ii) Implementation of MUX in behavioral

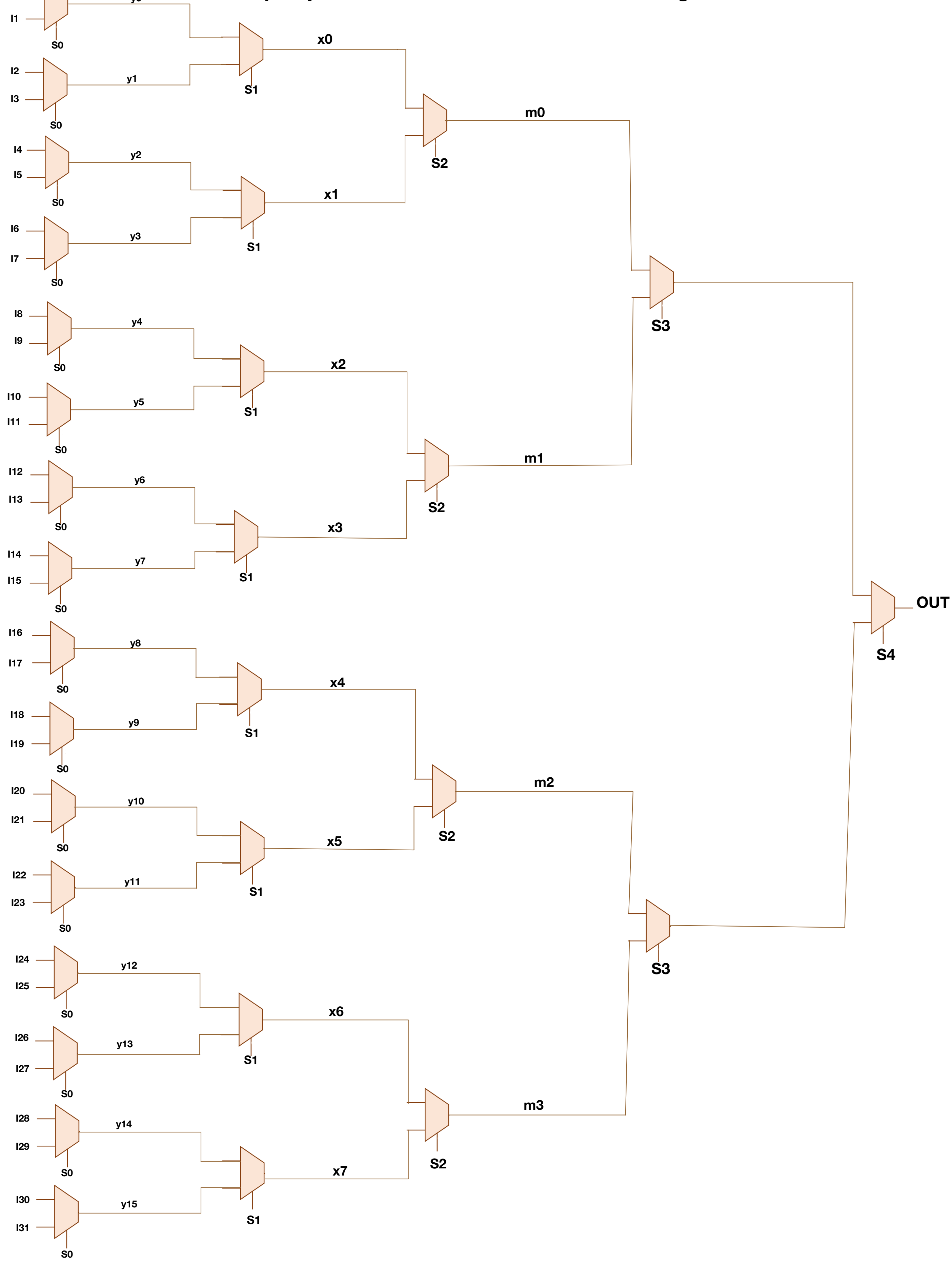
VERILOG CODE:

```
module MUX_2to1_behaviora(OUT,I0,I1,Sel); //Defines module for 2*1 mux with three input ports: ip0, ip1, and s0, and one output port out.
input I0, I1, Sel; // Defines ip0, ip1, and s0 are input ports to the module.
output OUT; // Defines output port of the module
reg OUT; // Defines Out as a register
always@(Sel or I0 or I1) // Define that block should be executed whenever any of the three input signals (s0, ip0, or ip1) change.
begin
if(Sel)
OUT = I1;
else
OUT = I0;// Checks the value of s0 and gives the output respectively
end
endmodule
```

Waveform:



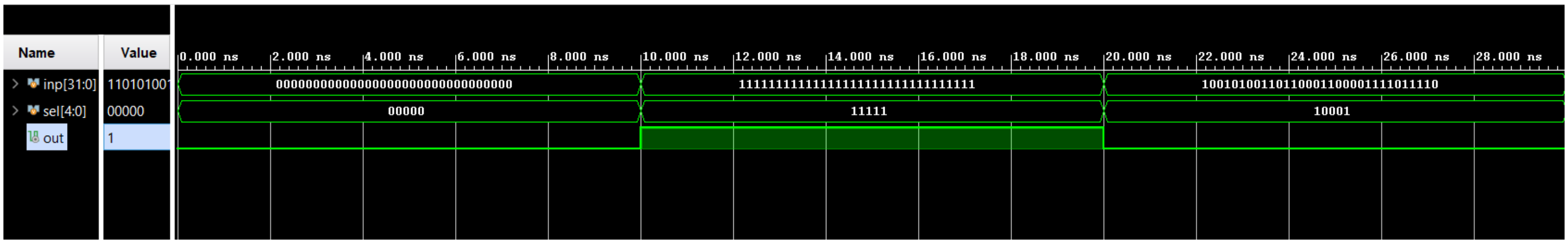
iii) Implementation of 32-to-1MUX using 2-to-1MUX

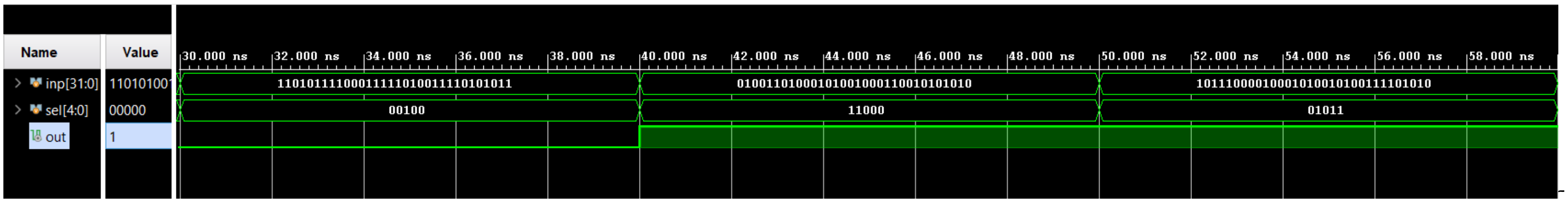


VERILOG CODE:

```
module mux_32to1(out,inp,sel); //Defines module for 32*1 mux with 3 ports
input [31:0] inp;//Defines 32 bit input bus
input [4:0] sel; //5 bit input bus for the select lines
output out; // main output
wire [15:0] y; //Declares a 16-bit wire bus named "y" represents first stage outputs
wire [7:0] x; //declares an 8-bit wire bus named "x" represents second stage outputs
wire [3:0] m; //declares a 4-bit wire bus named "m" represents third stage outputs
wire l0,l1; //declares two single-bit wires named "l0" and "l1" represents fourth stage outputs
genvar i; // generate a variable
//generates 16 instances of a 2-to-1 multiplexer named "inst0"
generate for(i=0;i<16;i=i+1)
   begin
   mux_2to1 inst0(y[i],inp[2*(i)+1],inp[2*i],sel[0]);
   end
   endgenerate
//generates 8 instances of a 2-to-1 multiplexer named "inst1"
generate for(i=0;i<8;i=i+1)
   begin
   mux_2to1 inst1(x[i],y[2*(i)+1],y[2*i],sel[1]);
   end
   endgenerate
//generates 4 instances of a 2-to-1 multiplexer named "inst2"
generate for(i=0;i<4;i=i+1)
   begin
   mux_2to1 inst2(m[i],x[2*(i)+1],x[2*i],sel[2]);
   end
   endgenerate
//instantiates a 2-to-1 multiplexer named m1,m2,m3
mux_2to1 m1(l0,m[1],m[0],sel[3]);
mux_2to1 m2(l1,m[3],m[2],sel[3]);
mux_2to1 m3(out,l1,l0,sel[4]);
endmodule
```

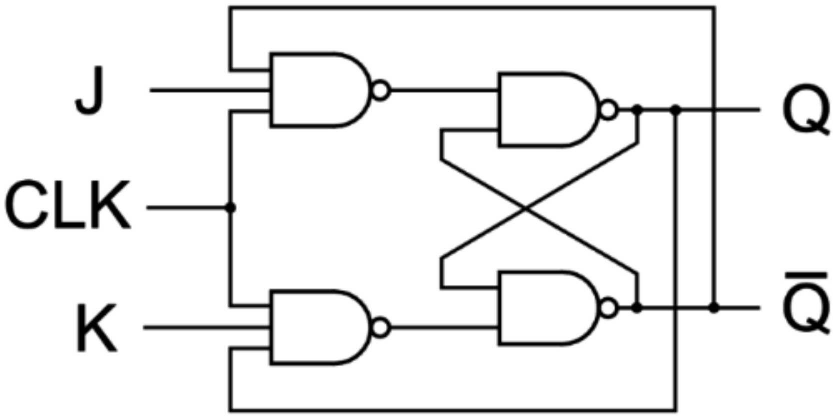
Waveforms:





b) Write a Verilog module to implement a clock-enabled JK flip-flop (Jack Kilby flip-flop) 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.

i) Implementation of JK_flipflop at structural level



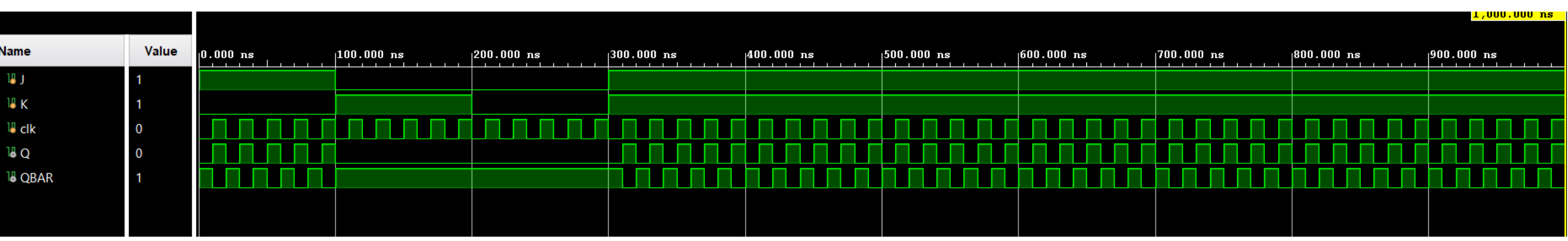
Truth Table

CLK	J	K	Q_{n+1}
↑	0	0	Q_n
↑	0	1	0
↑	1	0	1
↑	1	1	Q_n'

VERILOG CODE:

```verilog
module jkff_structural(q,qbar,clk,j,k);
input j,k,clk;
output q,qbar;
wire nand1_out; // output from nand1
wire nand2_out; // output from nand2
//temporary wires
wire x,xbar,y,ybar;
wire a,b,c,d;
assign a =1'b0;// assumed previous state of q as '0'
assign b=1'b1;// assumed previous state of qbar as '1'
nand(x,clk,b);
not(xbar,x);
nand(nand1_out,j,xbar);//nand1
nand(y,clk,a);
not(ybar,y);
nand(nand2_out,k,ybar);//nand2
nand(c,b,nand1_out);//nand3
nand(d,a,nand2_out);//nand4
assign q = c;
assign qbar =~q;
endmodule
```

Waveform:



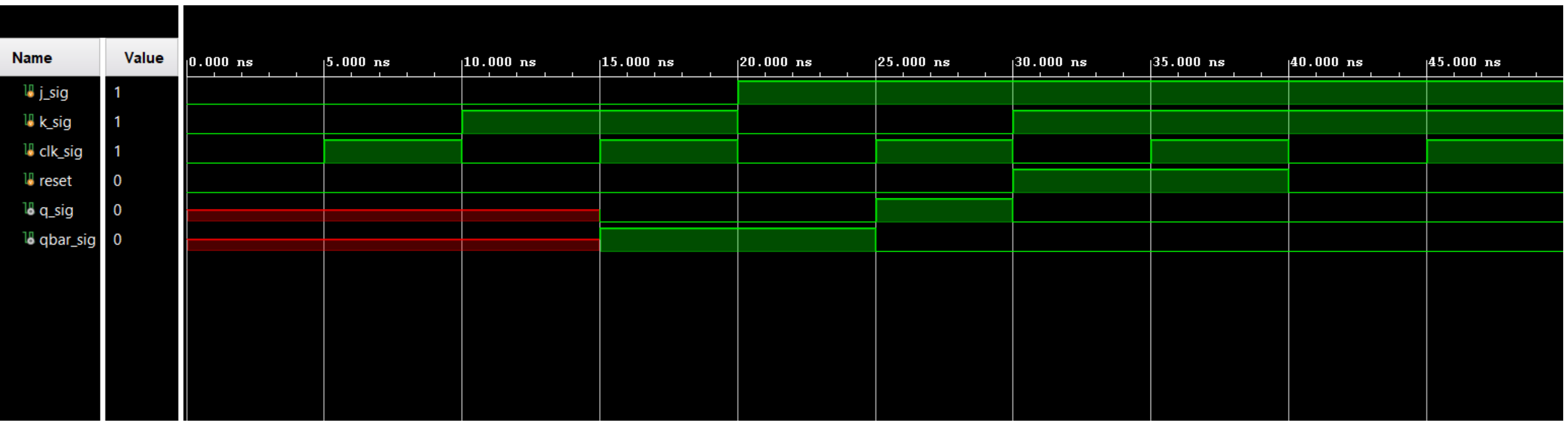
ii) Implementation of JK_flipflop in behavioural model

VERILOG CODE:

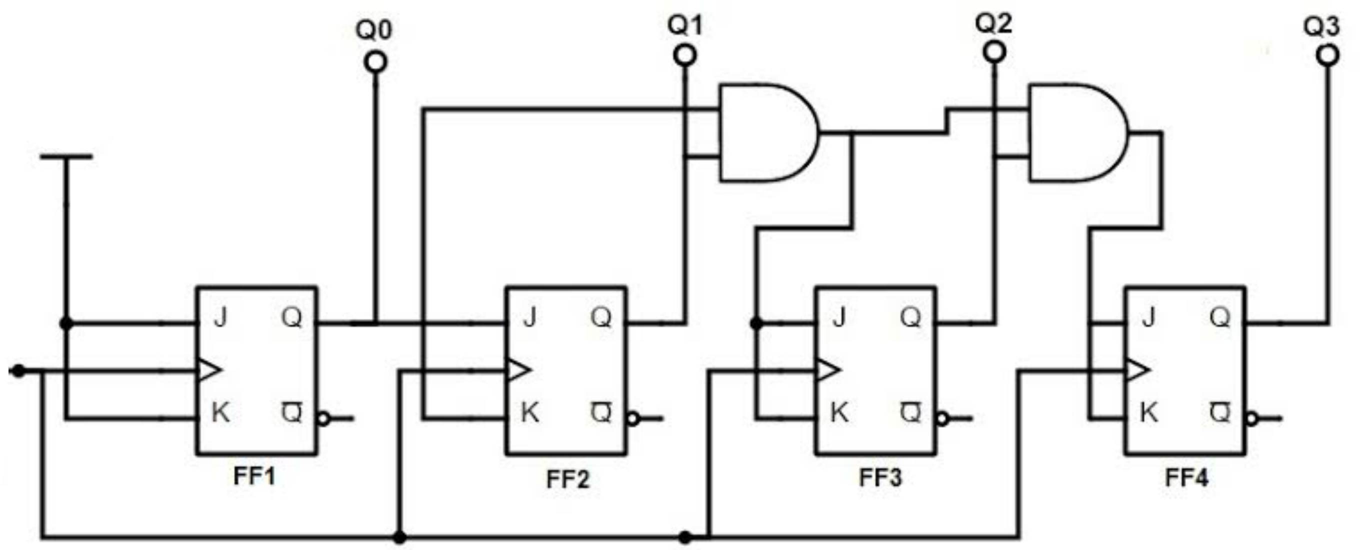
```verilog
module jk_ff_behavioral(
    input j,k, clk, reset,
    output reg q,
    output reg qbar
    //  declares a Verilog module with four input ports: j, k, clk, and reset. It also has two output ports: q for the output signal and qbar for the complemented output signal.
);
// This line starts an always block that is sensitive to the rising edge of the clk signal or the rising edge of the reset signal.
    always @(posedge clk or posedge reset) begin
    // This block contains the logic for the JK flip-flop.
        if (reset) begin
            q <= 0;
            qbar <= 0;
        end else begin
            if (j && k) begin
                q <= qbar;
                qbar <= q;
            end else if (j) begin
                q <= 1;
                qbar <= 0;
            end else if (k) begin
                q <= 0;
                qbar <= 1;
            end
        end
    end

endmodule
```

Waveform:



iii) Implementation of 4_bit_synchronous_counter using JK_flipflop



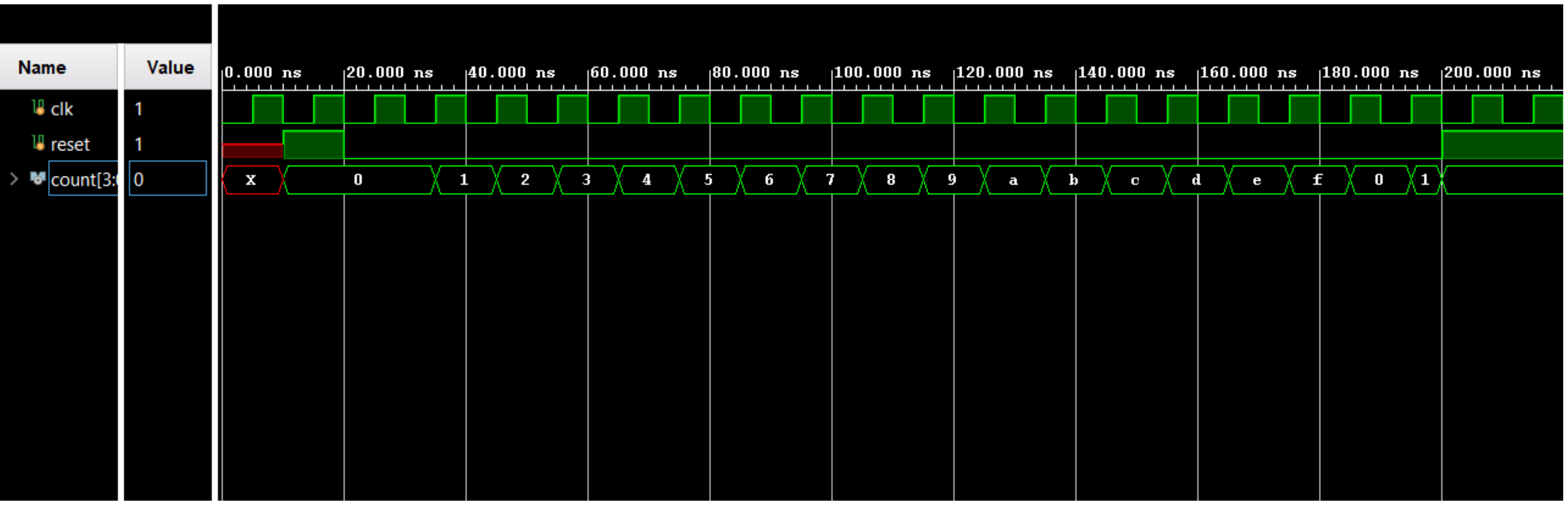
Count	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
0	0	0	0	0

VERILOG CODE:

```
//The inputs are clk, reset, and four JK flip-flop inputs (j and k), and the output is a 4-bit count register.
module up_counter(
    input clk, reset,
    output reg [3:0] count
);
  //Declare four wires (q0, q1, q2, q3) to store the outputs of the four JK flip-flops
    integer i =1;
    wire q0, q1, q2, q3;
    wire qbar0, qbar1, qbar2, qbar3;
    wire x,y;
    and a1(x,q0,q1);
    and a2(y,q0,q1,q2);
    //These four lines declare four instances of the JK flip-flop and connect their inputs and outputs to the wires and signals declared earlier.
    jk_ff_beh u0 (.j(i),.k(i), .clk(clk), .reset(reset), .q(q0), .qbar(qbar0));
    jk_ff_beh u1 (.j(q0), .k(q0), .clk(clk), .reset(reset), .q(q1), .qbar(qbar1));
    jk_ff_beh u2 (.j(x), .k(x), .clk(clk), .reset(reset), .q(q2), .qbar(qbar2));
    jk_ff_beh u3 (.j(y), .k(y), .clk(clk), .reset(reset), .q(q3), .qbar(qbar3));
//This always block is triggered on the rising edge of the clock or when the reset signal goes high
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            count <= 0;
        end else begin
            count <= {q3, q2, q1, q0};
        end
    end

endmodule
```

Waveform:



Name
Value
clk
1
reset
1
count[3:
0
0.000 ns
20.000 ns
40.000 ns
60.000 ns
80.000 ns
100.000 ns
120.000 ns
140.000 ns
160.000 ns
180.000 ns
200.000 ns
X
0
1
2
3
4
5
6
7
8
9
a
b
c
d
e
f
0
1