

Self Project - VERILOG

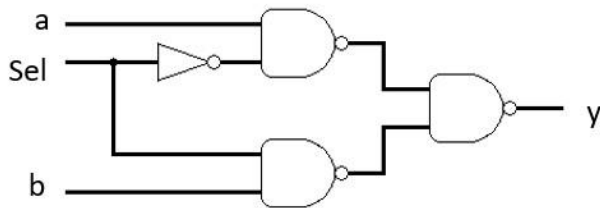
T GOVARDHAN
22104114

1. Implementing MUX based logic and JK synchronous counter:

1.a.

(Que) 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).

Implementation:



Code for 2-to-1 MUX at structural level: (Filename: mux2to1str.v)

```
module mux2to1str (input sel, input a, input b, output y);
    // 2 to 1 mux at structural level
    // sel=select line, a&b=inputs lines
    wire sel_not, a_nand_sel, b_nand_sel, ab_nand;
    // declaring wires
    // NOT gate inverts select line input
    not u_sel_not (sel_not, sel);
    // NAND gates for selection of inputs a or b
    nand u_a_nand_sel (a_nand_sel, a, sel_not);
    //NAND between input a and inverted select line
    nand u_b_nand_sel (b_nand_sel, b, sel);    //NAND
    between input b and select line
    nand u_ab_nand (ab_nand, a_nand_sel, b_nand_sel);
    // Assigning the output of the NOR gate to the output
    port assign y = ab_nand; endmodule
    // test bench file is mux2to1str_tb.v
```

Test bench code: (Filename: mux2to1str_tb.v)

```

`timescale 1ns / 1ns
`include
"mux2to1str.v" module
mux2to1str_tb;    reg
sel;    reg a;    reg b;
wire y;
    //Inputs & Outputs    mux2to1str uut (.sel(sel),
.a(a), .b(b), .y(y));
    initial
begin
    $dumpfile("mux2to1str.vcd");
    $dumpvars(0, mux2to1str_tb);
// Test case - sel=0, a=0, b=0
sel = 0;    a = 0;    b = 0;
    #5;
    $display("Test case - sel=0, a=0, b=0: y=%b", y);

    // Test case - sel=0, a=0,
b=1    sel = 0;    a = 0;    b
= 1;
    #5;
    $display("Test case - sel=0, a=0, b=1: y=%b", y);

    // Test case - sel=0, a=1,
b=0    sel = 0;    a = 1;    b
= 0;
    #5;
    $display("Test case - sel=0, a=1, b=0: y=%b", y);
    // Test case - sel=0, a=1,
b=1    sel = 0;    a = 1;    b
= 1;
    #5;
    $display("Test case - sel=0, a=1, b=1: y=%b", y);

    // Test case - sel=1, a=0, b=0
sel = 1;

```

```
    a = 0;
b = 0;
    #5;
    $display("Test case - sel=1, a=0, b=0: y=%b", y);

    // Test case - sel=1, a=0,
b=1    sel = 1;    a = 0;    b
= 1;
    #5;
    $display("Test case - sel=1, a=0, b=1: y=%b", y);

    // Test case - sel=1, a=1,
b=0    sel = 1;    a = 1;    b
= 0;
    #5;
    $display("Test case - sel=1, a=1, b=0: y=%b", y);
    // Test case - sel=1, a=1,
b=1    sel = 1;    a = 1;    b
= 1;
    #5;
    $display("Test case - sel=1, a=1, b=1: y=%b", y);

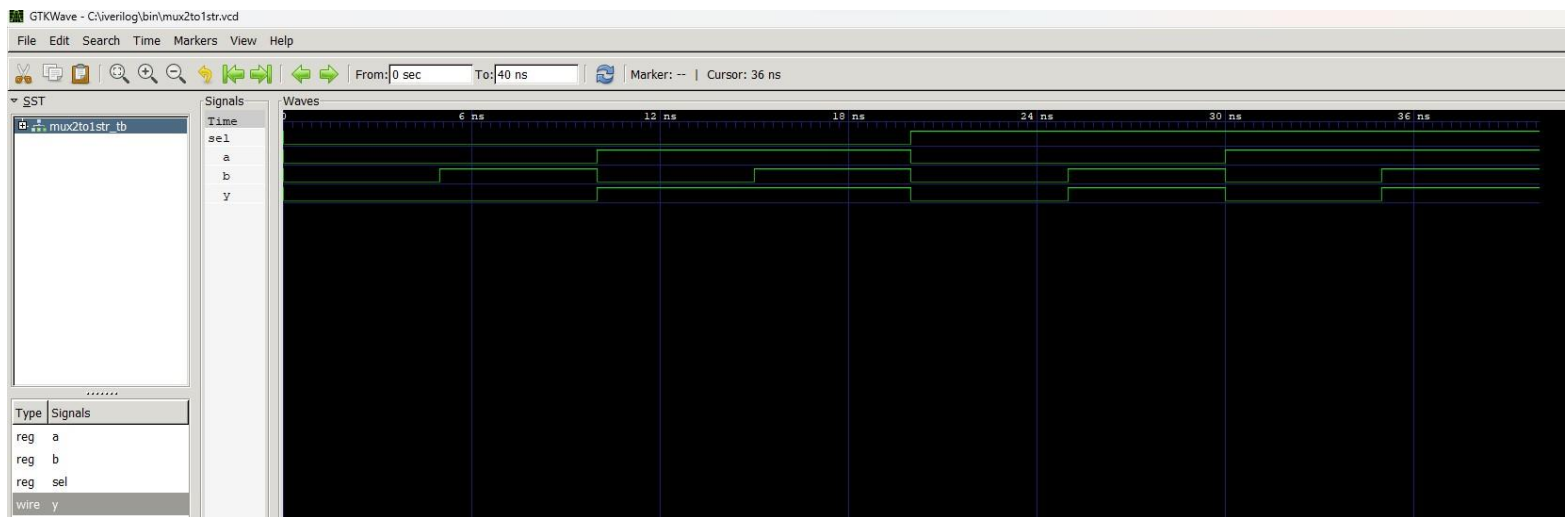
    $finish;
end endmodule
```

Results in terminal:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

VCD info: dumpfile mux2to1str.vcd opened for output.
Test case - sel=0, a=0, b=0: y=0
Test case - sel=0, a=0, b=1: y=0
Test case - sel=0, a=1, b=0: y=1
Test case - sel=0, a=1, b=1: y=1
Test case - sel=1, a=0, b=0: y=0
Test case - sel=1, a=0, b=1: y=1
Test case - sel=1, a=1, b=0: y=0
Test case - sel=1, a=1, b=1: y=1
mux2to1str_tb.v:70: $finish called at 40 (1ns)
```

Results in GTKWAVE: (Filename: mux2to1str.vcd)



1.a. (ii) Behavioral level.

Code for 2-to-1 MUX at behavioral level: (Filename: mux2to1beh.v)

```
module mux2to1beh (input wire sel, input wire a, input wire b, output reg out);  
  // inputs declaration  
  initial begin  
    $dumpfile("mux2to1beh.vcd");    $dumpvars(0,  
mux2to1beh_tb);    end    always @(sel, a, b)  
  //structural logic for 2to1mux    if(sel)  
out = b;    else    out = a; endmodule
```

Test bench code: (Filename: mux2to1beh_tb.v)

```

`timescale 1ns/1ns // Set timescale for simulation
`include "mux2to1beh.v"
module
mux2to1beh_tb;

    // Inputs
reg sel;
reg a;    reg
b;

    // Outputs
wire out;

    mux2to1beh uut(
        .sel(sel),
        .a(a),
        .b(b),
        .out(out)
    );
initial begin
    // Test case - sel=0, a=0,
b=0    sel = 0;    a = 0;    b =
0;

    #5;
    $display("Test case - sel=0, a=0, b=0: out=%b", out);

```

```

    // Test case - sel=0, a=0,
b=1    sel = 0;    a = 0;    b
= 1;
    #5;
    $display("Test case - sel=0, a=0, b=1: out=%b", out);

    // Test case - sel=0, a=1,
b=0    sel = 0;    a = 1;    b
= 0;
    #5;
    $display("Test case - sel=0, a=1, b=0: out=%b", out);
    // Test case - sel=0, a=1,
b=1    sel = 0;    a = 1;    b
= 1;
    #5;
    $display("Test case - sel=0, a=1, b=1: out=%b", out);

    // Test case - sel=1, a=0,
b=0    sel = 1;    a = 0;    b
= 0;
    #5;
    $display("Test case - sel=1, a=0, b=0: out=%b", out);

    // Test case - sel=1, a=0,
b=1    sel = 1;    a = 0;    b
= 1;
    #5;
    $display("Test case - sel=1, a=0, b=1: out=%b", out);

    // Test case - sel=1, a=1,
b=0    sel = 1;    a = 1;    b
= 0;
    #5;
    $display("Test case - sel=1, a=1, b=0: out=%b", out);
    // Test case - sel=1, a=1, b=1
sel = 1;

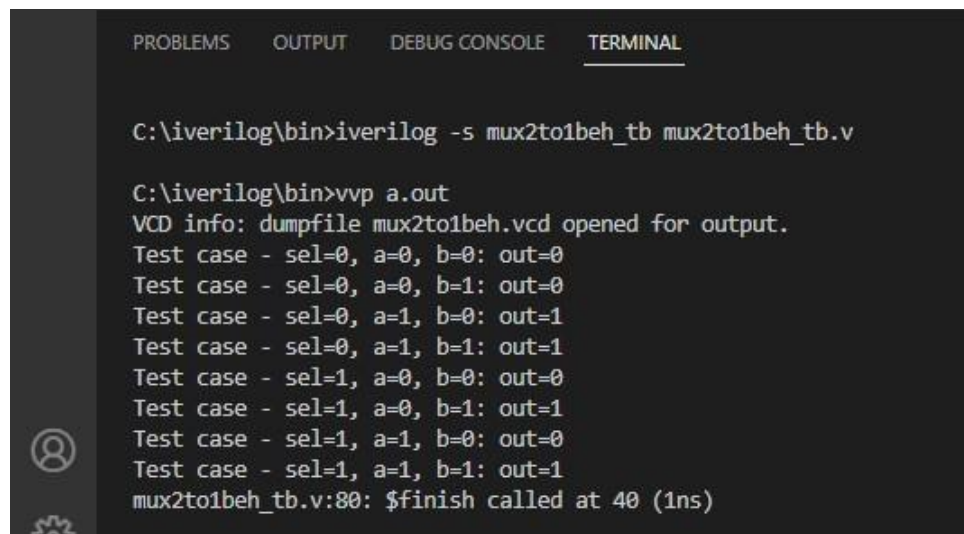
```



```
    a = 1;
b = 1;
    #5;
    $display("Test case - sel=1, a=1, b=1: out=%b", out);

$finish;
end
endmodule
```

Results in terminal:

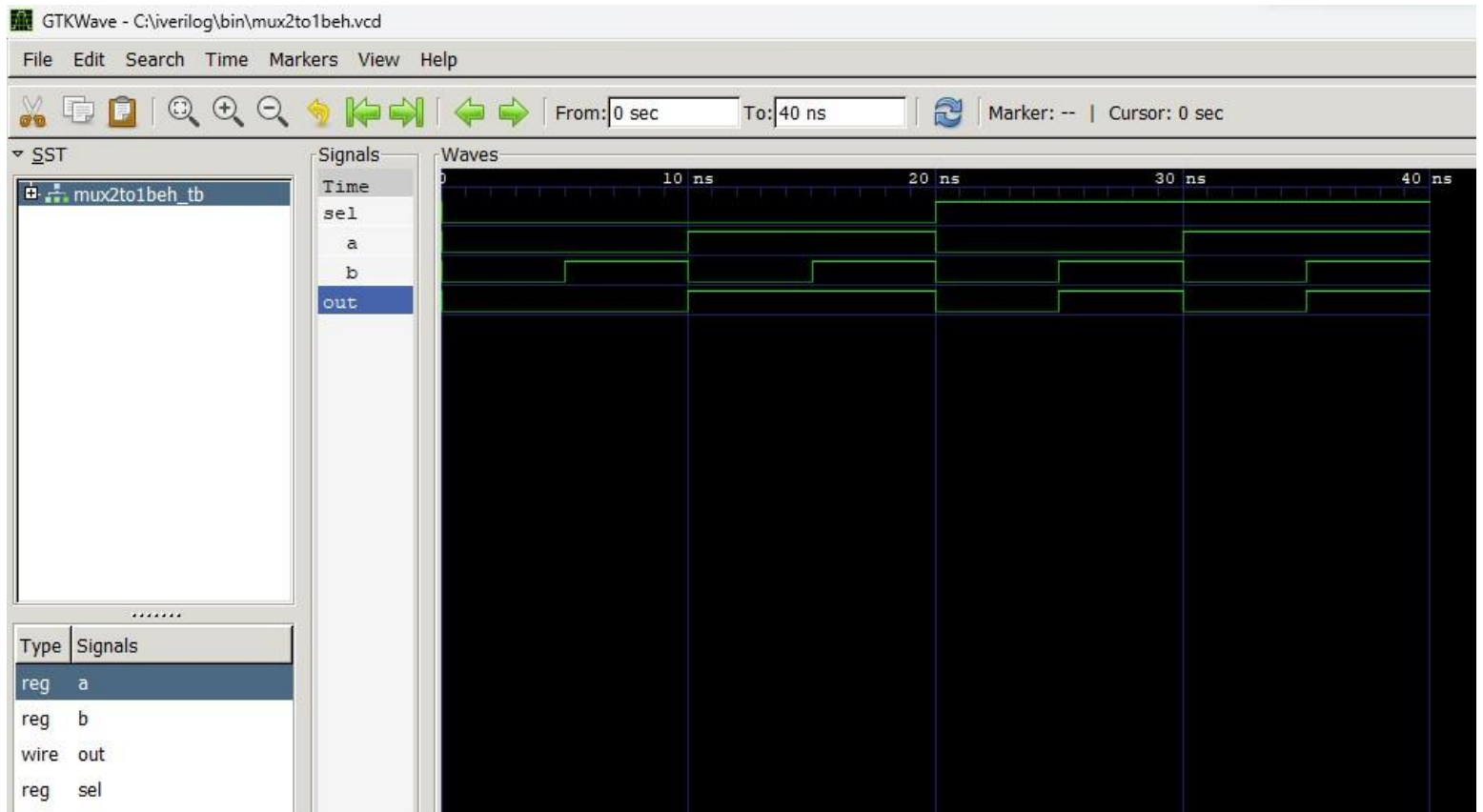


```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

C:\iverilog\bin>iverilog -s mux2to1beh_tb mux2to1beh_tb.v

C:\iverilog\bin>vvp a.out
VCD info: dumpfile mux2to1beh.vcd opened for output.
Test case - sel=0, a=0, b=0: out=0
Test case - sel=0, a=0, b=1: out=0
Test case - sel=0, a=1, b=0: out=1
Test case - sel=0, a=1, b=1: out=1
Test case - sel=1, a=0, b=0: out=0
Test case - sel=1, a=0, b=1: out=1
Test case - sel=1, a=1, b=0: out=0
Test case - sel=1, a=1, b=1: out=1
mux2to1beh_tb.v:80: $finish called at 40 (1ns)
```

Results in GTKWAVE: (Filename: mux2to1beh.vcd)



1.a. (Que) Using this module and other elementary two-input logic gates as necessary, build a 2ⁿ-to-1 MUX that can implement any Boolean function of 5 variables.

To implement any logic function of 5 variables, $2^5 \times 1 = 32 \times 1$ MUX is required. 32x1 MUX is implemented as follows:

1st Level (16-MUX)

2nd Level (8-MUX)

3rd Level (4-MUX) 4th

Level (2-MUX)

5th Level (1-MUX)

Code for 32-to-1 MUX at structural level: (Filename: mux32to1str.v)

```

module mux32to1str (input [4:0] sel, input [31:0] a, output [0:0] y);
    //16 MUX in first level    mux2to1 u_mux0 (.sel(sel[0]),
.a(a[0]), .b(a[1]), .y(w0));    mux2to1 u_mux1 (.sel(sel[0]),
.a(a[2]), .b(a[3]), .y(w1));    mux2to1 u_mux2 (.sel(sel[0]),
.a(a[4]), .b(a[5]), .y(w2));    mux2to1 u_mux3 (.sel(sel[0]),
.a(a[6]), .b(a[7]), .y(w3));    mux2to1 u_mux4 (.sel(sel[0]),
.a(a[8]), .b(a[9]), .y(w4));    mux2to1 u_mux5 (.sel(sel[0]),
.a(a[10]), .b(a[11]), .y(w5));    mux2to1 u_mux6 (.sel(sel[0]),
.a(a[12]), .b(a[13]), .y(w6));    mux2to1 u_mux7 (.sel(sel[0]),
.a(a[14]), .b(a[15]), .y(w7));    mux2to1 u_mux8 (.sel(sel[0]),
.a(a[16]), .b(a[17]), .y(w8));    mux2to1 u_mux9 (.sel(sel[0]),
.a(a[18]), .b(a[19]), .y(w9));    mux2to1 u_mux10 (.sel(sel[0]),
.a(a[20]), .b(a[21]), .y(w10));    mux2to1 u_mux11
(.sel(sel[0]), .a(a[22]), .b(a[23]), .y(w11));    mux2to1
u_mux12 (.sel(sel[0]), .a(a[24]), .b(a[25]), .y(w12));
mux2to1 u_mux13 (.sel(sel[0]), .a(a[26]), .b(a[27]), .y(w13));
mux2to1 u_mux14 (.sel(sel[0]), .a(a[28]), .b(a[29]), .y(w14));
mux2to1 u_mux15 (.sel(sel[0]), .a(a[30]), .b(a[31]), .y(w15));
    // 8 MUX in 2nd level    mux2to1 u_mux16 (.sel(sel[1]),
.a(w0), .b(w1), .y(w16));    mux2to1 u_mux17 (.sel(sel[1]),
.a(w2), .b(w3), .y(w17));    mux2to1 u_mux18 (.sel(sel[1]),
.a(w4), .b(w5), .y(w18));    mux2to1 u_mux19 (.sel(sel[1]),
.a(w6), .b(w7), .y(w19));    mux2to1 u_mux20 (.sel(sel[1]),
.a(w8), .b(w9), .y(w20));    mux2to1 u_mux21 (.sel(sel[1]),
.a(w10), .b(w11), .y(w21));    mux2to1 u_mux22
(.sel(sel[1]), .a(w12), .b(w13), .y(w22));    mux2to1
u_mux23 (.sel(sel[1]), .a(w14), .b(w15), .y(w23));
    // 4 MUX in 3rd level

```

```

    mux2to1 u_mux24 (.sel(sel[2]), .a(w16), .b(w17), .y(w24));
mux2to1 u_mux25 (.sel(sel[2]), .a(w18), .b(w19), .y(w25));
mux2to1 u_mux26 (.sel(sel[2]), .a(w20), .b(w21), .y(w26));
mux2to1 u_mux27 (.sel(sel[2]), .a(w22), .b(w23), .y(w27));
    // 2 MUX in 4th level    mux2to1 u_mux28 (.sel(sel[3]),
    .a(w24), .b(w25), .y(w28));    mux2to1 u_mux29
    (.sel(sel[3]), .a(w26), .b(w27), .y(w29));
    // 1 MUX in 5th level    mux2to1 u_mux30 (.sel(sel[4]),
    .a(w28), .b(w29), .y(y));
    endmodule module mux2to1 (input sel, input a, input
b, output y);    reg y;    always@(*)begin
if(sel==0)        y=a;        else        y=b;
end endmodule

```

Test bench code: (Filename: mux32to1str_tb.v)

(Note: Total 8 cases are tested)

```

`include "mux32to1str.v"
module mux32to1str_tb;
    mux32to1str dut
(
    .sel(sel),
    .a(a),
    .y(y)
);

    // Inputs
reg [4:0] sel;
reg [31:0] a;

    // Output
wire [0:0] y;

```

```
// Initialize inputs  
initial begin
```

```

    $dumpfile("mux32x1str.vcd");
    $dumpvars(0, mux32to1str_tb);
    // Test case 1      sel = 5'b00000;      a
= 32'b00000000000000000000000000000001;
    #5;
    $display("Test case 1: sel=%b, a=%b, y=%b", sel, a, y);
    // Test case 2      sel = 5'b00001;      a
= 32'b00000000000000000000000000000010;
    #5;
    $display("Test case 2: sel=%b, a=%b, y=%b", sel, a, y);
    // Test case 3      sel = 5'b00010;      a
= 32'b000000000000000000000000000000100;
    #5;
    $display("Test case 3: sel=%b, a=%b, y=%b", sel, a, y);
    // Test case 4      sel = 5'b00011;      a
= 32'b0000000000000000000000000000001000;
    #5;
    $display("Test case 4: sel=%b, a=%b, y=%b", sel, a, y);
    // Test case 5      sel = 5'b11111;      a
= 32'b10000000000000000000000000000000;
    #5;
    $display("Test case 5: sel=%b, a=%b, y=%b", sel, a, y);
    // Test case 6
sel = 5'b11110;
    a = 32'b01000000000000000000000000000000;
    #5;
    $display("Test case 6: sel=%b, a=%b, y=%b", sel, a, y);
    // Test case 7
sel = 5'b11101;
    a = 32'b00100000000000000000000000000000;
    #5;
    $display("Test case 7: sel=%b, a=%b, y=%b", sel, a, y);
    // Test case 8

```

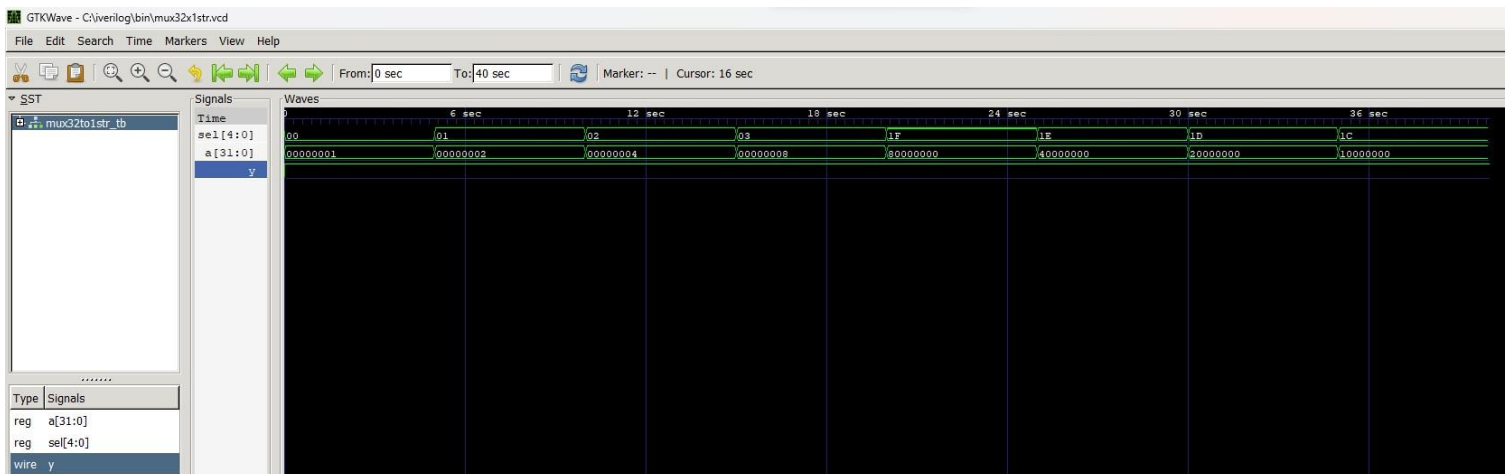
```
        sel = 5'b11100;      a =  
32'b00010000000000000000000000000000;  
#5;  
    $display("Test case 8: sel=%b, a=%b, y=%b", sel, a, y);  
  
$finish;  
end  
endmodule
```


Results in terminal:

```
C:\iverilog\bin>iverilog -s mux32to1str_tb mux32to1str_tb.v

C:\iverilog\bin>vvp a.out
VCD info: dumpfile mux32x1str.vcd opened for output.
Test case 1: sel=00000, a=00000000000000000000000000000001, y=1
Test case 2: sel=00001, a=00000000000000000000000000000010, y=1
Test case 3: sel=00010, a=000000000000000000000000000000100, y=1
Test case 4: sel=00011, a=0000000000000000000000000000001000, y=1
Test case 5: sel=11111, a=1000000000000000000000000000000000, y=1
Test case 6: sel=11110, a=0100000000000000000000000000000000, y=1
Test case 7: sel=11101, a=0010000000000000000000000000000000, y=1
Test case 8: sel=11100, a=0001000000000000000000000000000000, y=1
mux32to1str_tb.v:69: $finish called at 40 (1s)
```

Results in GTKWAVE: (Filename: mux32x1str.vcd)

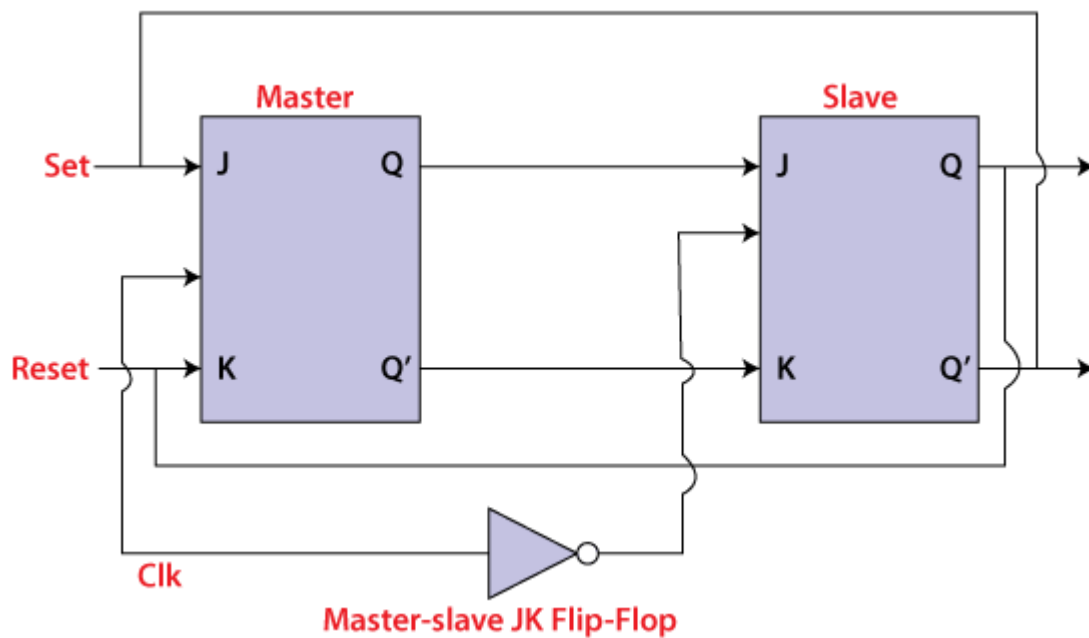


1.b.

(Que) 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).

Implementation:

Master slave topology used to avoid race around condition.



Code for JK Flipflop at structural level: (Filename: jkffstr.v)

```
module jkffstr(q1,qbar1,q2,qbar2,clk,j,k);
//Input output initiation input j,k,clk; output
q1,qbar1,q2,qbar2; wire nand1_out, nand2_out, nand3_out,
nand_out4,clk_not; not(clk_not,clk);//Inverting clock
//Master JK Flipflop
nand(nand1_out, j,clk,qbar2);
nand(nand2_out, k,clk,q2);
nand(q1,qbar1,nand1_out);
nand(qbar1,q1,nand2_out); //Slave
JK Flipflop
nand(nand3_out,q1,clk_not);//slave
nand(nand4_out,qbar1,clk_not);
nand(q2,qbar2,nand3_out);
nand(qbar2,q2,nand4_out);
endmodule
```

Test bench code: (Filename: jkffstr_tb.v)

```

`timescale 1ns/1ns // Set timescale for simulation
`include "jkffstr.v" //Including code file
module jkffstr_tb; reg
J,K, CLK; wire
Q1,QBAR1,Q2,QBAR2;
integer i;
//input output declaration jkffstr uut (.q1(Q1), .qbar1(QBAR1), .q2(Q2),
.qbar2(QBAR2), .clk(CLK), .j(J), .k(K));

initial begin
    CLK=0; //forcing outputs to predefined conditions
    force Q1 =1;    force QBAR1=0;    force Q2 =0;    force
QBAR2=1;
    #1    //releasing outputs
    release Q1;    release Q2;
    release QBAR1;    release
QBAR2; //generating clock

        for(i=0;i<10;i=i+1)
#10 CLK = ~CLK;    end
initial begin
    $dumpfile("jkffstr.vcd"); //generating vcd file
    $dumpvars(0, jkffstr_tb);
    $monitor("  J = %b K = %b Q= %b QBAR = %b clk=\t",J,K,Q2,QBAR2,CLK);
    //testcases
    J= 1; K= 0;
    #20; J= 0; K= 1;
    #20; J= 0; K=
0;    #20; J= 1;
    K=1;    end
endmodule

```

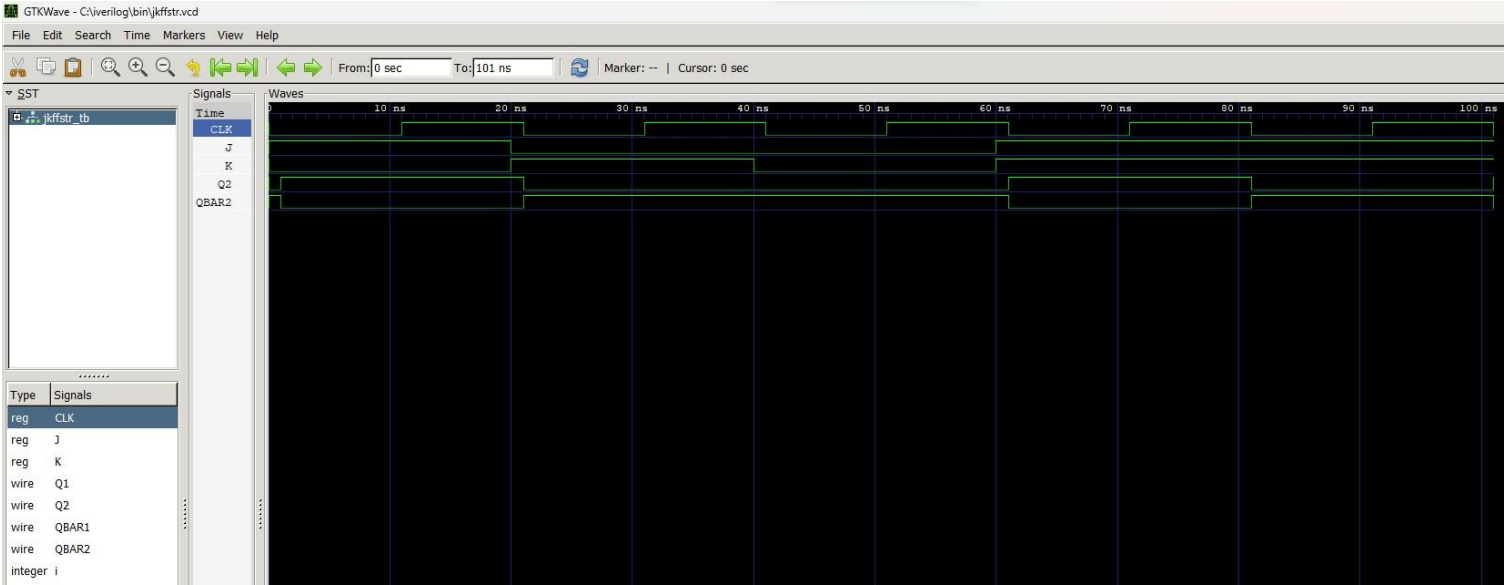
Result in terminal:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
C:\iverilog\bin>iverilog -s jkffstr_tb jkffstr_tb.v

C:\iverilog\bin>vvp a.out
VCD info: dumpfile jkffstr.vcd opened for output.
  J = 1 K = 0 Q= 0 QBAR = 1 clk=      0
  J = 1 K = 0 Q= 1 QBAR = 0 clk=      0
  J = 0 K = 1 Q= 0 QBAR = 1 clk=      1
  J = 0 K = 0 Q= 0 QBAR = 1 clk=      1
  J = 0 K = 0 Q= 0 QBAR = 1 clk=      0
  J = 0 K = 0 Q= 0 QBAR = 1 clk=      1
  J = 1 K = 1 Q= 0 QBAR = 1 clk=      1
  J = 1 K = 1 Q= 1 QBAR = 0 clk=      0
  J = 1 K = 1 Q= 1 QBAR = 0 clk=      1
  J = 1 K = 1 Q= 0 QBAR = 1 clk=      0
  J = 1 K = 1 Q= 0 QBAR = 1 clk=      1
  J = 1 K = 1 Q= 1 QBAR = 0 clk=      0
```

Result in GTKWAVE: (Filename: jkffstr.vcd)



1.b. (ii) Behavioral level.

Code for JK Flipflop at behavioral level: (Filename: jkffbeh.v)

```
module
jkffbeh(jk,clk,q,qbar);
input [1:0] jk;    input
clk;    output q,qbar;
reg q,qbar;
    always@(negedge
clk)    begin
case(jk)        2'b00:
q= q;

        2'b01:
q= 0;

        2'b10:
q=1;

        2'b11:
q= ~q;
endcase
    qbar =
~q;

end
endmodule
```

Testbench Code: (Filename: jkffbeh_tb.vcd)

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

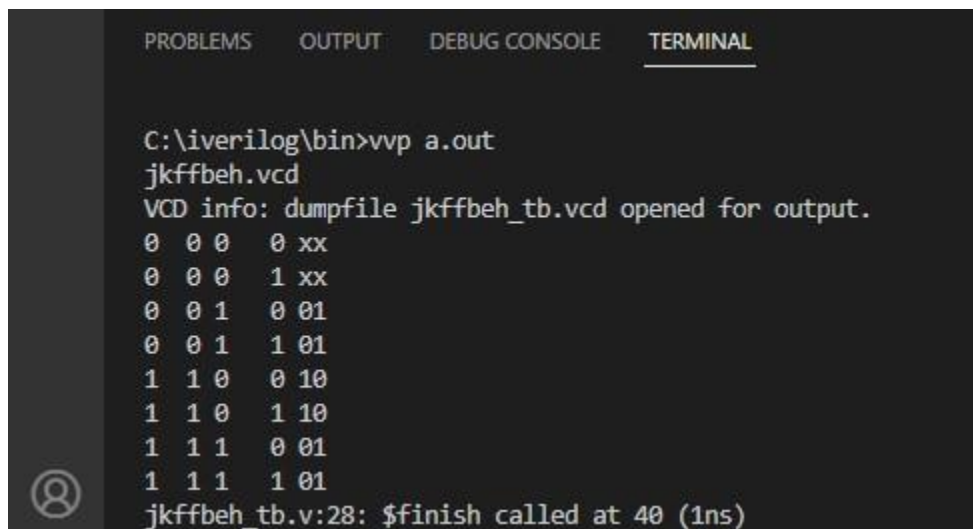
module jkffbeh_tb;
reg [1:0] jk;
reg clk;
integer i;    wire
q,qbar;
    jkffbeh
j1(jk,clk,q,qbar);
initial    begin
    $display("jkffbeh.vcd");
```

```

$monitor("%b %b %b \t %b %b",jk[1],jk[1],jk[0],clk,q,qbar);
$dumpfile("jkffbeh_tb.vcd");
$dumpvars(0,jkffbeh_tb);
    jk =
2'b00;
    #10
    jk =
2'b01;
    #10
    jk =
2'b10;
    #10
    jk =
2'b11;
    #10    $finish;
end    initial    begin
clk = 0;
for(i=0;i<20;i=i+1)
#5 clk = ~clk;    end
endmodule

```

Results in terminal:



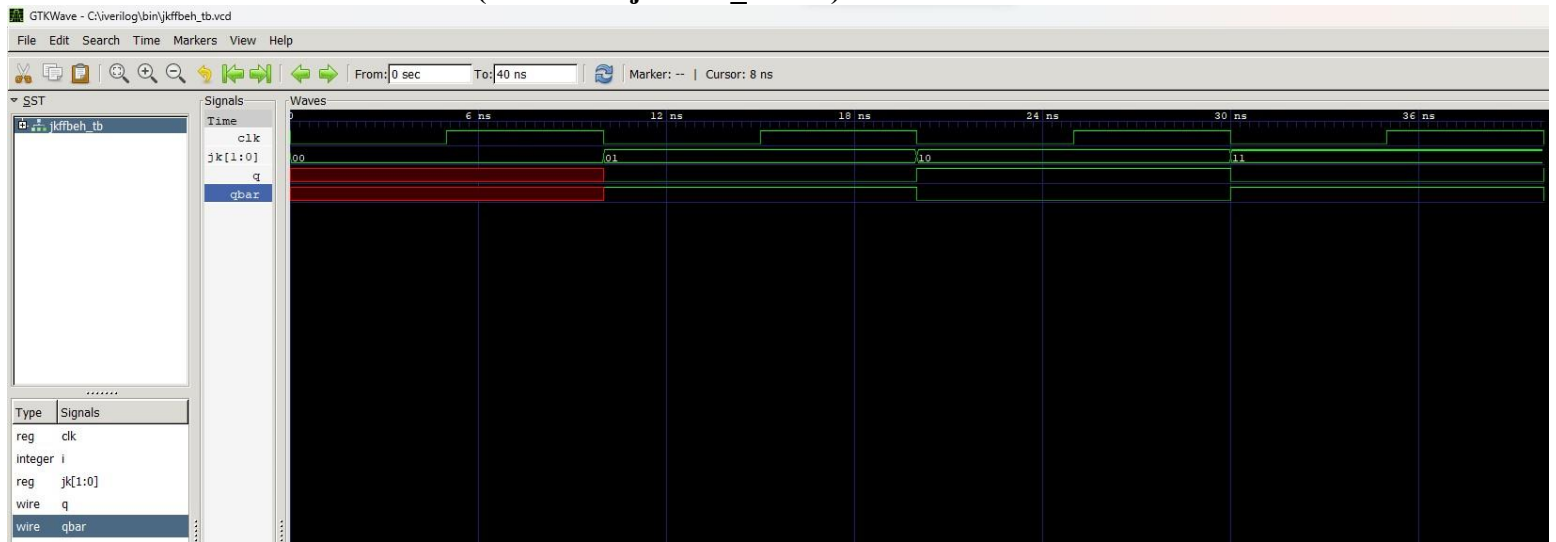
```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

C:\iverilog\bin>vvp a.out
jkffbeh.vcd
VCD info: dumpfile jkffbeh_tb.vcd opened for output.
0 0 0 0 xx
0 0 0 1 xx
0 0 1 0 01
0 0 1 1 01
1 1 0 0 10
1 1 0 1 10
1 1 1 0 01
1 1 1 1 01
jkffbeh_tb.v:28: $finish called at 40 (1ns)

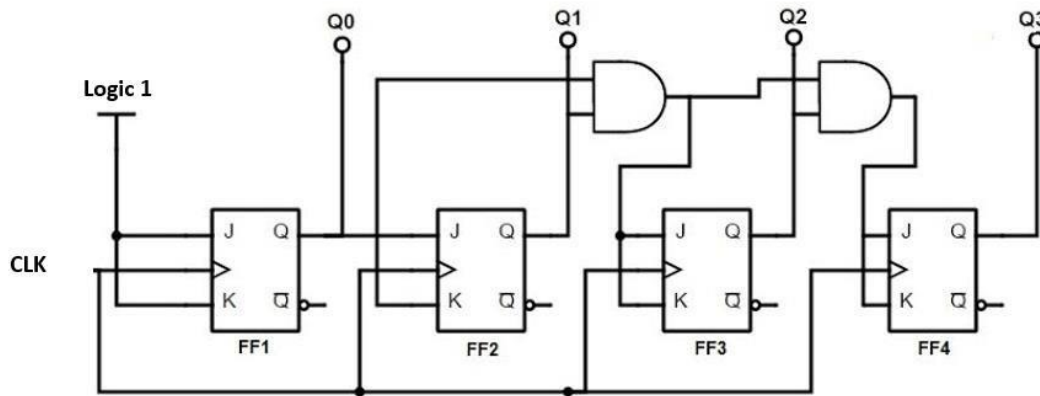
```


Results in GTKWAVE: (Filename: jkffbeh_tb.vcd)



(Que) Using this module and other two-input logic gates as necessary, build a four-bit synchronous binary counter.

Implementation:



Code for JK Flipflop based binary counter at behavioral level:
(Filename: jkff_upcounter.v)

```

module
jkff_upcounter(clk,reset,q);
input clk,reset;  output [3:0] q;
wire j2,j3,q0_bar;
//input & output initialization
    jkff ff0(.j(1'b1), .k(1'b1), .clk(clk), .rst(reset),
.q(q[0]));  jkff ff1(.j(q[0]), .k(q[0]), .clk(clk),
.rst(reset), .q(q[1]));  and(j2,q[1],q[0]);  jkff ff2(.j(j2),
.k(j2), .clk(clk), .rst(reset), .q(q[2]));
and(j3,q[2],q[1],q[0]);  jkff ff3(.j(j3), .k(j3), .clk(clk),
.rst(reset), .q(q[3]));  //interconnecting JK flipflops and
logic gates using wires endmodule
module
jkff(j,k,clk,rst,q);
input j,k,clk,rst;
output reg q;      reg
qbar;

    //positive edge triggered JK flipflop

```

```

        always@(posedge
clk)      begin
if(rst) begin
q<=0;      qbar <=1;
end      else
begin      case({j,k})
2'b00:begin
q<=q;
qbar<=qbar;      end
2'b01:begin
q<= 0;
qbar<=1;      end
2'b10:begin
q<=1;
qbar<=0;      end

2'b11:begin
q<= ~q;
qbar<=~qbar;end
endcase      end
end      endmodule

```

Testbench Code: (Filename: jkff_upcounter_tb.v)

```

`include "jkff_upcounter.v"
module jkff_upcounter_tb;
reg clk, reset;  integer
i;  wire [3:0] q;

jkff_upcounter counter(.clk(clk), .reset(reset), .q(q));
initial begin    clk = 0;
for (i=1;i<66;i=i+1)begin
#2 clk=~clk;
end

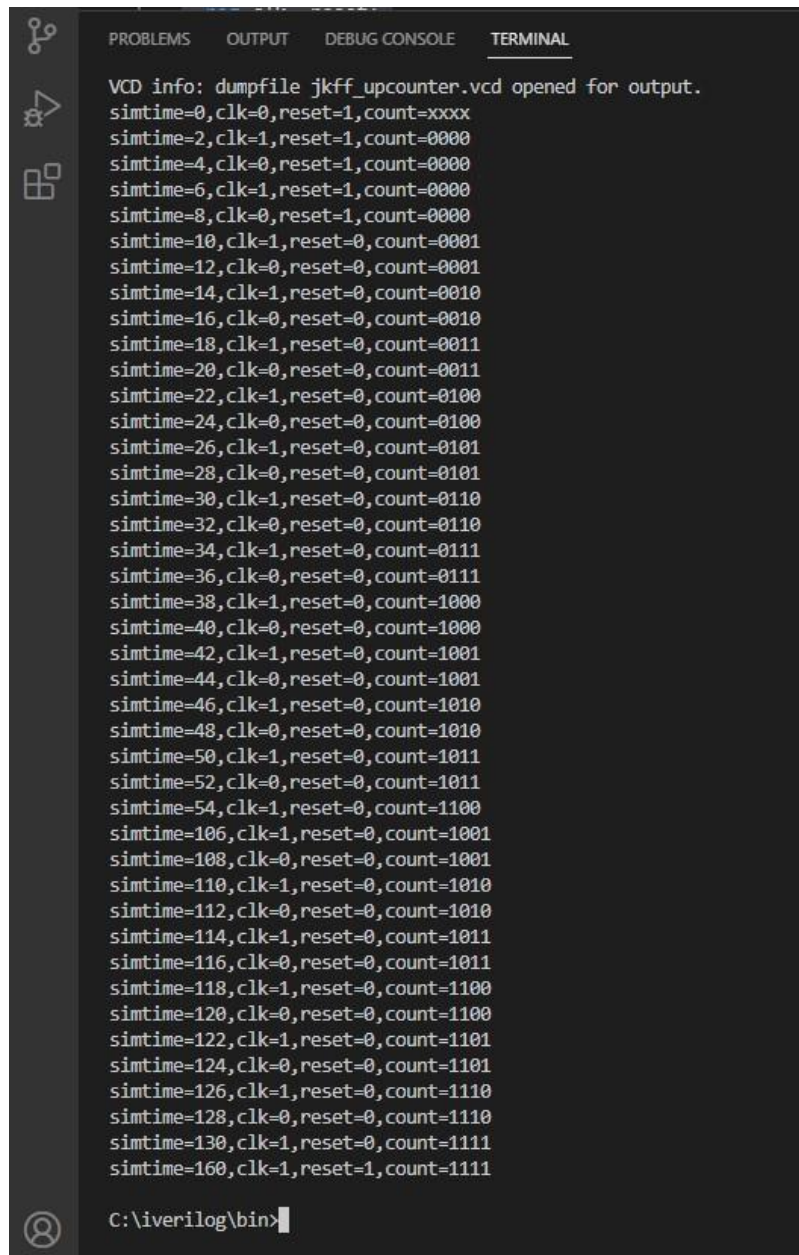
```

```
    end
initial begin

$dumpfile("jkff_upcounter.vcd");
$dumpvars(0, jkff_upcounter_tb);
end
    initial

$monitor("simtime=%g,clk=%b,reset=%b,count=%b%b%b%b",$time,clk,reset,q[3],q[2],q[1],q[0]);
initial begin
reset=1; #10
reset=0;
#150 reset=1;
end
endmodule
```

Result in terminal:



The screenshot shows a terminal window with a dark background and light-colored text. The window has a title bar at the top with the text "VCD info: dumpfile jkff_upcounter.vcd opened for output." and a tab labeled "TERMINAL". The terminal displays a series of simulation results for a counter, showing the state of the clock (clk), reset, and count at various simulation times (simtime). The count starts at 0000 and increments by 1 for each clock cycle, reaching 1111 at the end of the simulation. The reset signal is initially 1, then becomes 0, and finally becomes 1 again at the end of the simulation.

```
VCD info: dumpfile jkff_upcounter.vcd opened for output.
simtime=0,clk=0,reset=1,count=xxxx
simtime=2,clk=1,reset=1,count=0000
simtime=4,clk=0,reset=1,count=0000
simtime=6,clk=1,reset=1,count=0000
simtime=8,clk=0,reset=1,count=0000
simtime=10,clk=1,reset=0,count=0001
simtime=12,clk=0,reset=0,count=0001
simtime=14,clk=1,reset=0,count=0010
simtime=16,clk=0,reset=0,count=0010
simtime=18,clk=1,reset=0,count=0011
simtime=20,clk=0,reset=0,count=0011
simtime=22,clk=1,reset=0,count=0100
simtime=24,clk=0,reset=0,count=0100
simtime=26,clk=1,reset=0,count=0101
simtime=28,clk=0,reset=0,count=0101
simtime=30,clk=1,reset=0,count=0110
simtime=32,clk=0,reset=0,count=0110
simtime=34,clk=1,reset=0,count=0111
simtime=36,clk=0,reset=0,count=0111
simtime=38,clk=1,reset=0,count=1000
simtime=40,clk=0,reset=0,count=1000
simtime=42,clk=1,reset=0,count=1001
simtime=44,clk=0,reset=0,count=1001
simtime=46,clk=1,reset=0,count=1010
simtime=48,clk=0,reset=0,count=1010
simtime=50,clk=1,reset=0,count=1011
simtime=52,clk=0,reset=0,count=1011
simtime=54,clk=1,reset=0,count=1100
simtime=106,clk=1,reset=0,count=1001
simtime=108,clk=0,reset=0,count=1001
simtime=110,clk=1,reset=0,count=1010
simtime=112,clk=0,reset=0,count=1010
simtime=114,clk=1,reset=0,count=1011
simtime=116,clk=0,reset=0,count=1011
simtime=118,clk=1,reset=0,count=1100
simtime=120,clk=0,reset=0,count=1100
simtime=122,clk=1,reset=0,count=1101
simtime=124,clk=0,reset=0,count=1101
simtime=126,clk=1,reset=0,count=1110
simtime=128,clk=0,reset=0,count=1110
simtime=130,clk=1,reset=0,count=1111
simtime=160,clk=1,reset=1,count=1111

C:\iverilog\bin>
```

Result in GTKWAVE: (Filename: jkff_upcounter.vcd)

