**B.M.S College of Engineering**

***(Autonomous Institution affiliated to VTU, Belagavi)***

**Bengaluru – 19**

**Department of Computer Science and Engineering**



**Report on**

**“Verilog Programs using Structural Modeling, Behavioral Modeling and Data Flow Modeling”**

**LOGIC DESIGN - 19CS3PCLOD**

(Autonomous Scheme 2020-21)

***Submitted by***

Name: NITHIN.C

USN: 1BM19CS106

**B.M.S. College of Engineering**

***(Autonomous Institution affiliated to VTU, Belagavi)***

**Bengaluru - 19**

**Department of Computer Science and Engineering**



**Certificate**

This is to certify that Mr. **\_\_ NITHIN.C\_** has satisfactorily completed the course of Experiments in **LOGIC DESIGN course** prescribed by the Department during the year **2020-2021.**

Name of the Candidate: NITHIN.C

USN No:1BM19CS106 Semester**: 3**

|  |  |  |
| --- | --- | --- |
| Marks | | |
| Max. Marks | Obtained |
| **20** |  |

|  |  |
| --- | --- |
| Marks in Words | |
|  |  |

**Signature of the staff in-charge Head of the Department**

**Date:**

**Verilog Program List**

**19CS3PCLOD**

|  |  |
| --- | --- |
| **Serial No.** | **Title** |
|  | **CYCLE I**  **Structural Modeling** |
|  | Write HDL implementation for the following Logic   * AND/OR/NOT   Simulate the same using structural model and depict the timing diagram for valid inputs. |
|  | Write HDL implementation for the following Logic   * NAND/NOR   Simulate the same using structural model and depict the timing diagram for valid inputs. |
|  | Write HDL implementation for the following Logic  Simulate the same using structural model and depict the timing diagram for valid inputs. |
|  | Write HDL implementation for a 4:1 Multiplexer. Simulate the same using structural model and depict the timing diagram for valid inputs. |
|  | Write HDL implementation for a 2-to-4 decoder. Simulate the same using structural model and depict the timing diagram for valid inputs. |
|  | Write HDL implementation for a 4-to-2 encoder. Simulate the same using structural model and depict the timing diagram for valid inputs. |
|  | **CYCLE II**  **Behavior Modeling** |
|  | Write HDL implementation for a RS flip-flop using behavioral model. Simulate the same using Behavior model and depict the timing diagram for valid inputs. |
|  | Write HDL implementation for a JK flip-flop using behavioral model. Simulate the same using Behavior model and depict the timing diagram for valid inputs. |
|  | Write HDL implementation for a 4-bit right shift register using behavioral model. Simulate the same using Behavior model and depict the timing diagram for valid inputs. |
|  | Write HDL implementation for a 3-bit up-counter using behavioral model. Simulate the same using Behavior model and depict the timing diagram for valid inputs. |
|  | **CYCLE III**  **Dataflow Modeling** |
|  | Write HDL implementation for AND/OR/NOT gates using data flow model. Simulate the same using Dataflow model and depict the timing diagram for valid inputs. |
|  | Write HDL implementation for a 3-bit full adder using data flow model. Simulate the same using Dataflow model and depict the timing diagram for valid inputs. |
|  |  |

**STRUCTURAL MODELING**

**Experiment 1:**

* **Write HDL implementation for the following Logic**
* **AND/OR/NOT**

**Simulate the same using structural model and depict the timing diagram for valid inputs.**

**Main Module:**

module s\_andornot(a,b,y);

input a,b;

output [2:0]y;

and a(y[2],a,b);

or o(y[1],a,b);

not n(y[0],a);

endmodule

**Test Module:**

module tb\_andornot;

reg a,b;

wire [2:0]y;

s\_andornot ob(a,b,y);

initial

begin

$dumpfile("test.vcd");

$dumpvars(0,tb\_andornot);

a=1'b0;b=1'b0;

        #20;

a=1'b0;b=1'b1;

#20;

a=1'b1;b=1'b0;

#20;

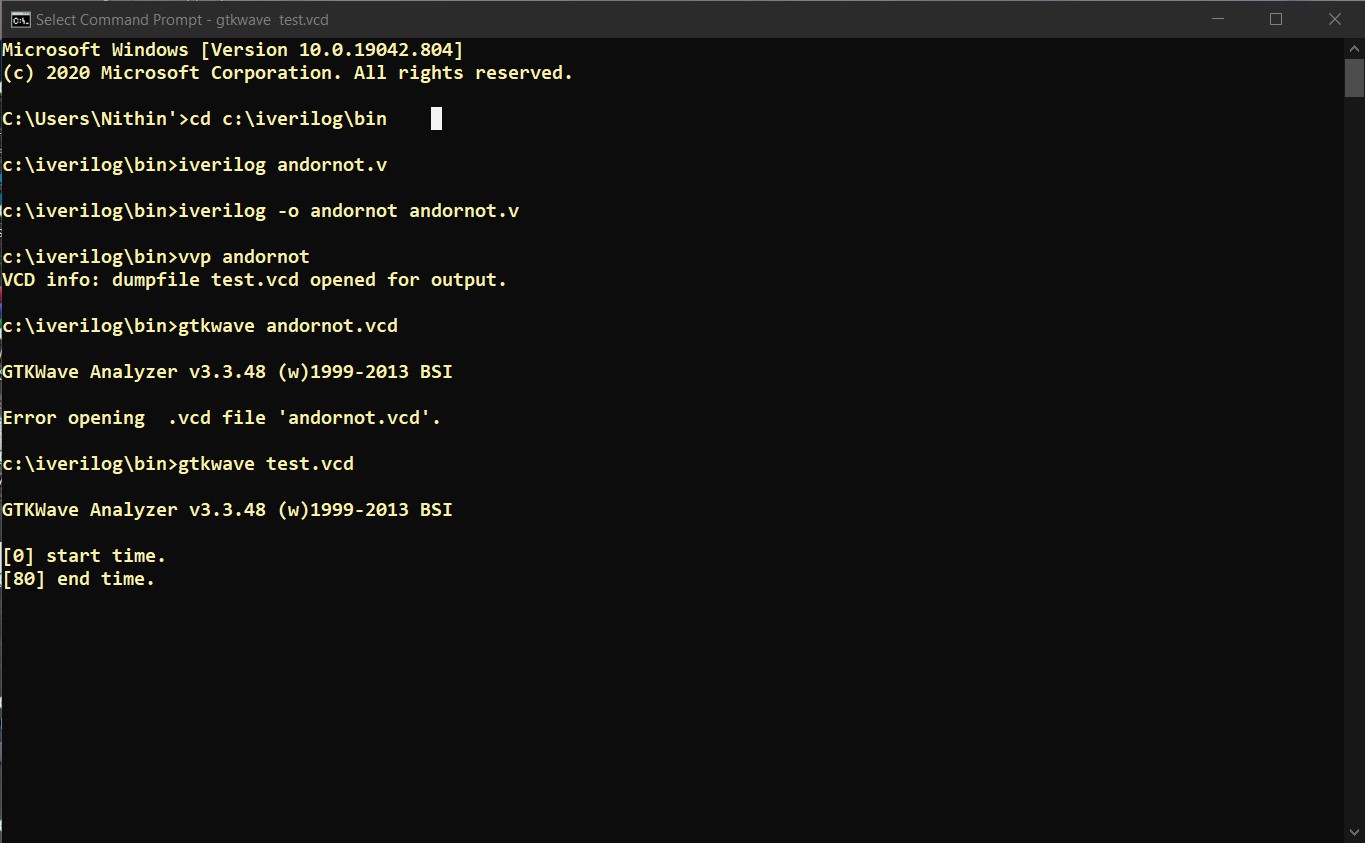
a=1'b1;b=1'b1;

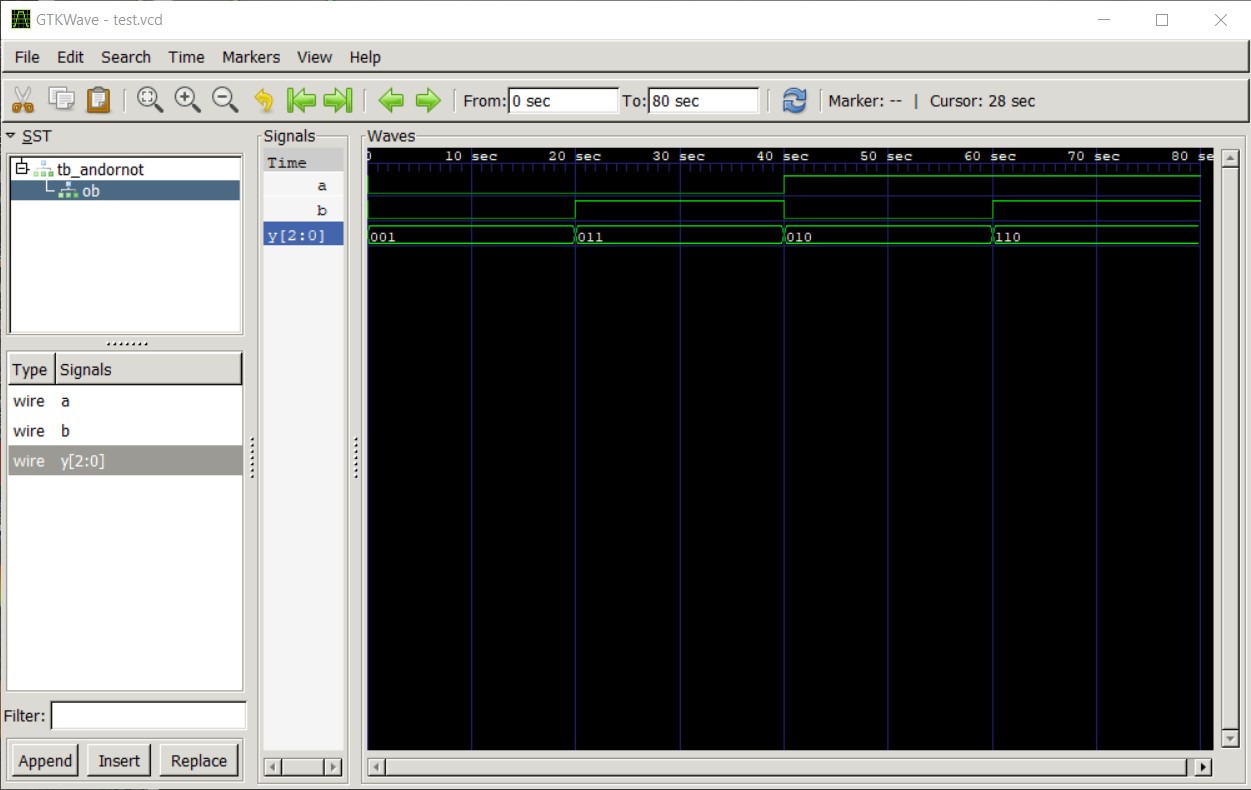
#20;

$finish;

end

endmodule





**Experiment 2:**

**Write HDL implementation for the following Logic**

* **NAND/NOR**

**Simulate the same using structural model and depict the timing diagram for valid inputs.**

**Main Module:**

module s\_nandnor(a,b,y);

    input a,b;

    output [1:0]y;

    nand a(y[1],a,b);

    nor o(y[0],a,b);

endmodule

**Test Module:**

`include "s\_nandnor.v"

module tb\_nandnor;

    reg a,b;

    wire [1:0]y;

    s\_nandnor ob(a,b,y);

    initial

    begin

    $dumpfile("test.vcd");

    $dumpvars(0,tb\_nandnor);

        a=1'b0;b=1'b0;

        #20

        a=1'b0;b=1'b1;

        #20

        a=1'b1;b=1'b0;

        #20

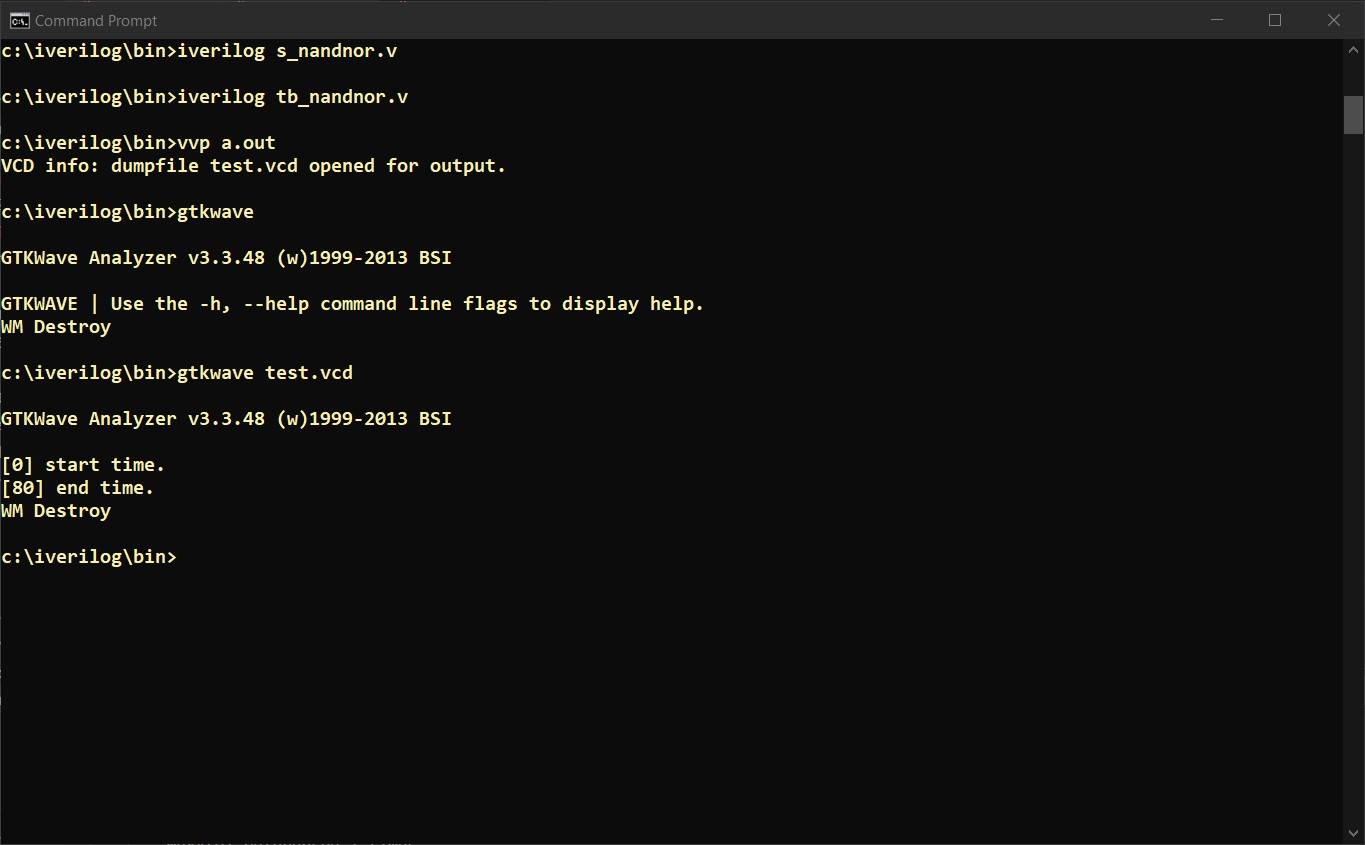
        a=1'b1;b=1'b1;

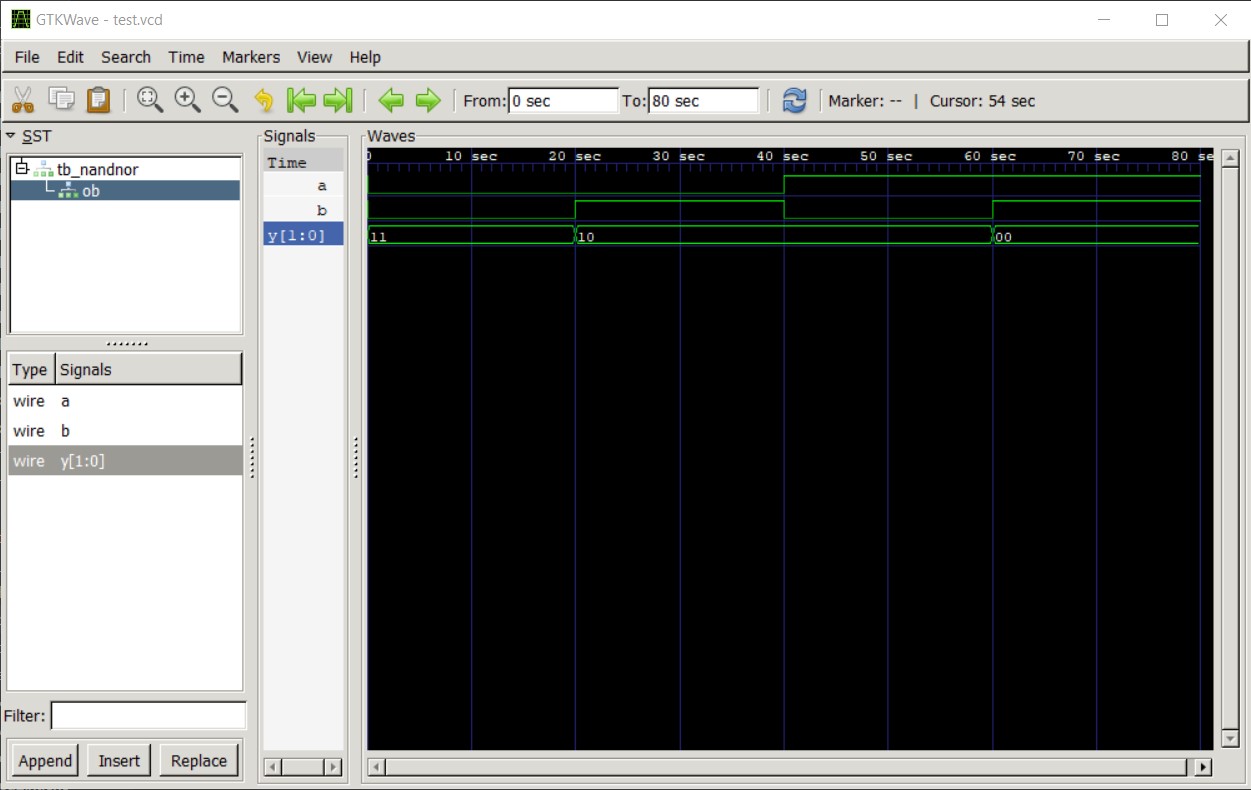
        #20

        $finish;

    End

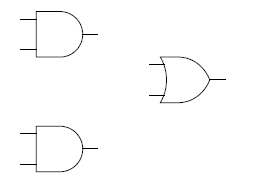
endmodule





**Experiment 3:**

**Write HDL implementation for the following Logic**

****

**Simulate the same using structural model and depict the timing diagram for valid inputs**.

**Main Module:**

module andor(A,B,C,D,Y);

input A,B,C,D;

output Y;

wire al,a2;

and gl(al,A,B);

and g2(a2,C,D);

or g3(Y,al,a2);

endmodule

**Test Module:**

`include "andor.v"

module test\_andor;

reg a,b,c,d;

wire y;

andor ao(a,b,c,d,y);

initial

begin

$dumpfile("andor.vcd");

$dumpvars(0,test\_andor);

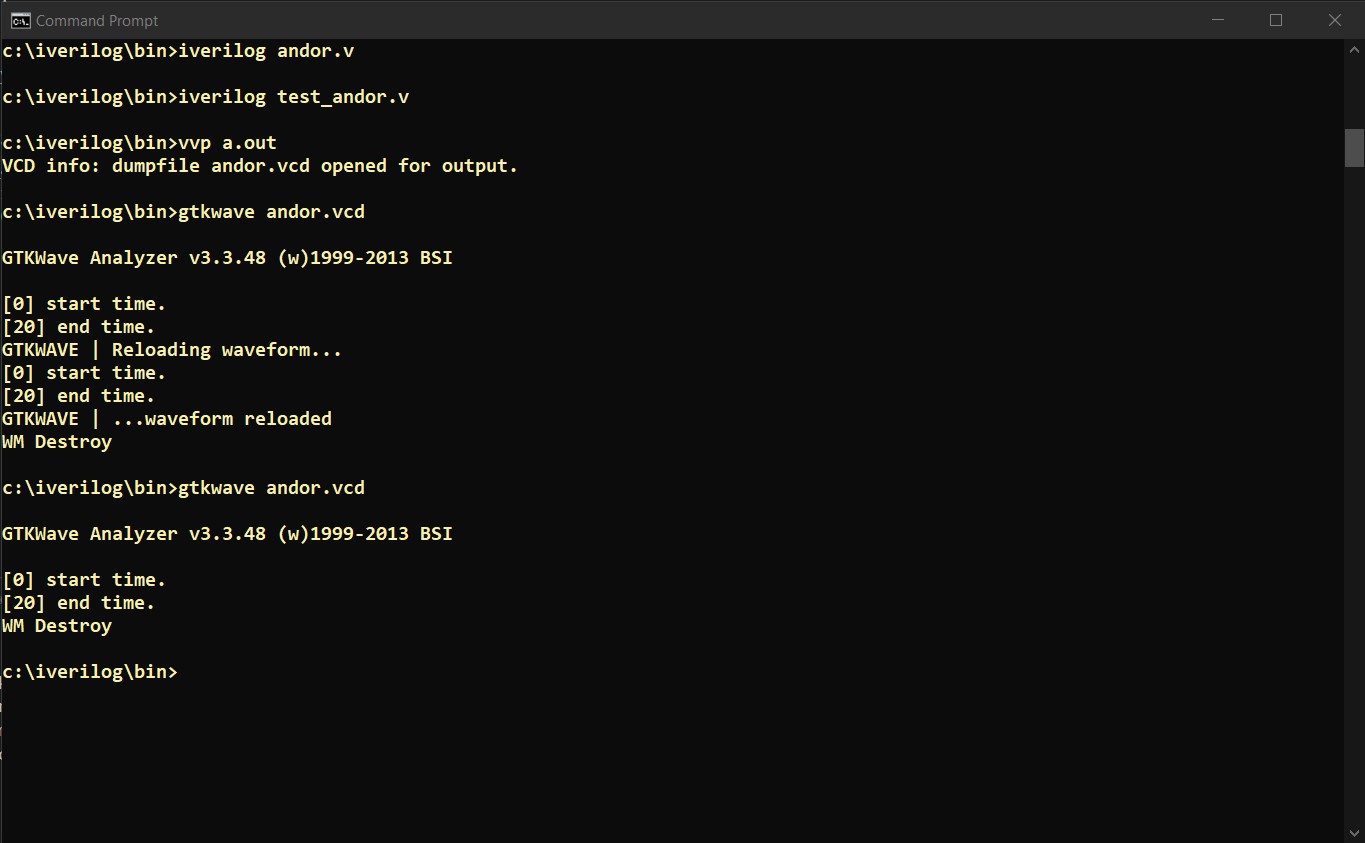
a=0; b=1; c=1; d=1; #10

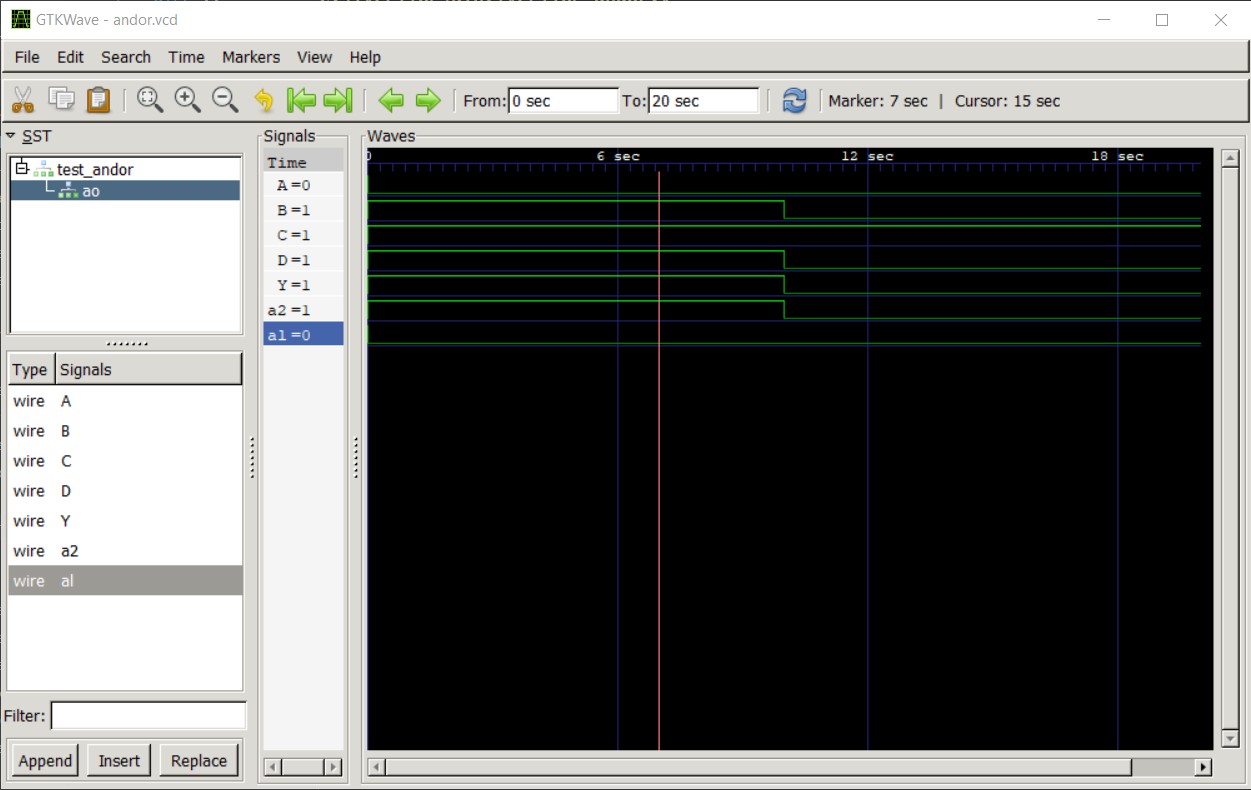
a=0; b=0; c=1; d=0; #10

$finish;

end

endmodule





**Experiment 4:**

**Write HDL implementation for a 4:1 Multiplexer. Simulate the same using structural model and depict the timing diagram for valid inputs.**

**Main Module:**

module mux4to1(a, b, c, d, sel1, sel2, y);

input a, b, c, d, sel1, sel2;

output y;

wire and1, and2, and3, and4;

and g1(and1, a, ~sel1, ~sel2);

and g2(and2, b, ~sel1, sel2);

and g3(and3, c, sel1, ~sel2);

and g4(and4, d, sel1, sel2);

or g5(y, and1, and2, and3, and4);

endmodule

**Test Module:**

module test;

reg a, b, c, d, sel1, sel2;

wire y;

mux4to1 mg(a, b, c, d, sel1, sel2, y);

initial

begin

       $dumpfile("mux4to1.vcd");

    $dumpvars(0, test);

        a = 0; b = 0; c = 0; d = 0; sel1 = 0; sel2 = 0;

        #20

        a = 1; b = 0; c = 0; d = 0; sel1 = 0; sel2 = 0;

        #20

        a = 0; b = 0; c = 0; d = 0; sel1 = 0; sel2 = 1;

        #20

        a = 0; b = 1; c = 0; d = 0; sel1 = 0; sel2 = 1;

        #20

        a = 0; b = 0; c = 0; d = 0; sel1 = 1; sel2 = 0;

        #20

        a = 0; b = 0; c = 1; d = 0; sel1 = 1; sel2 = 0;

        #20

        a = 0; b = 0; c = 0; d = 0; sel1 = 1; sel2 = 1;

        #20

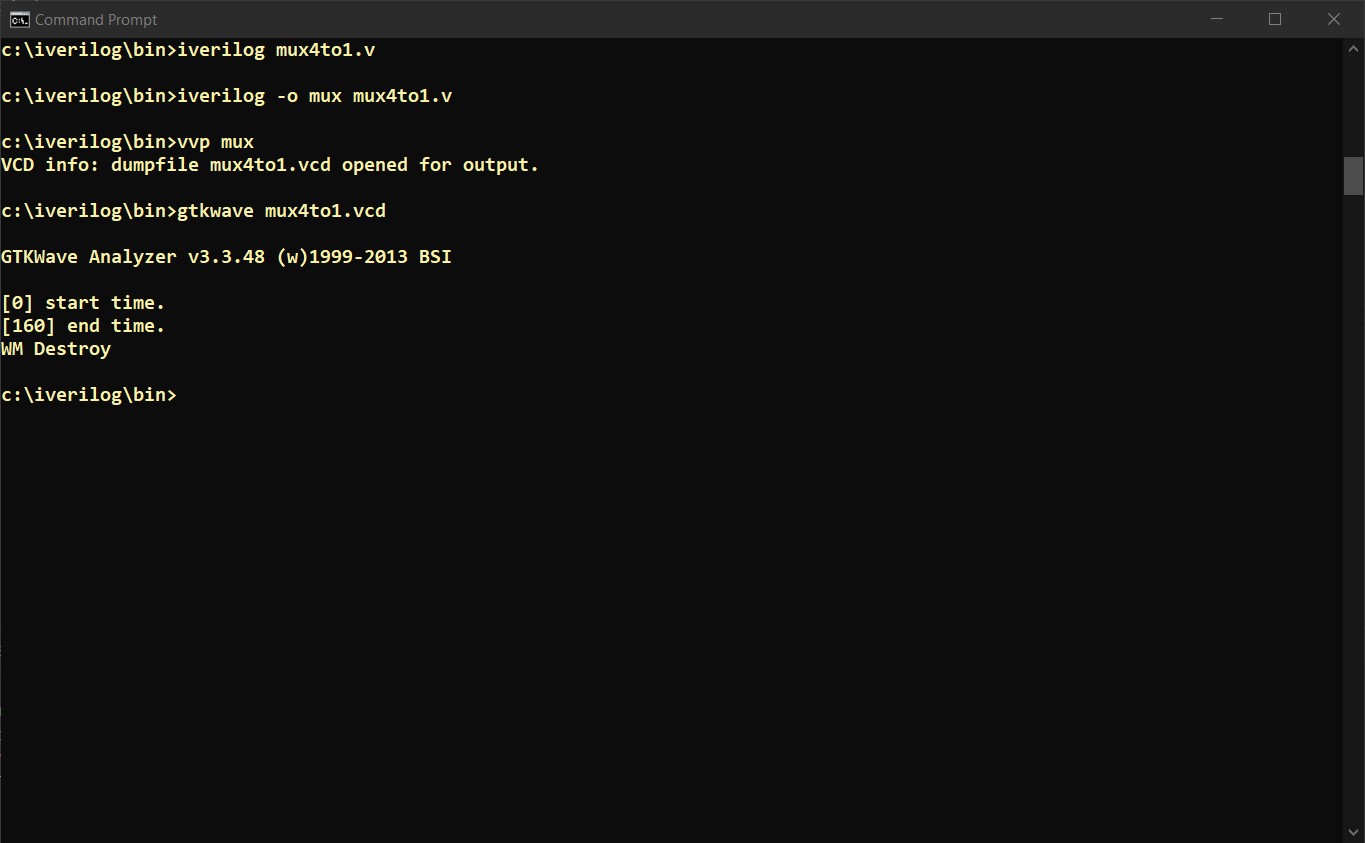
        a = 0; b = 0; c = 0; d = 1; sel1 = 1; sel2 = 1;

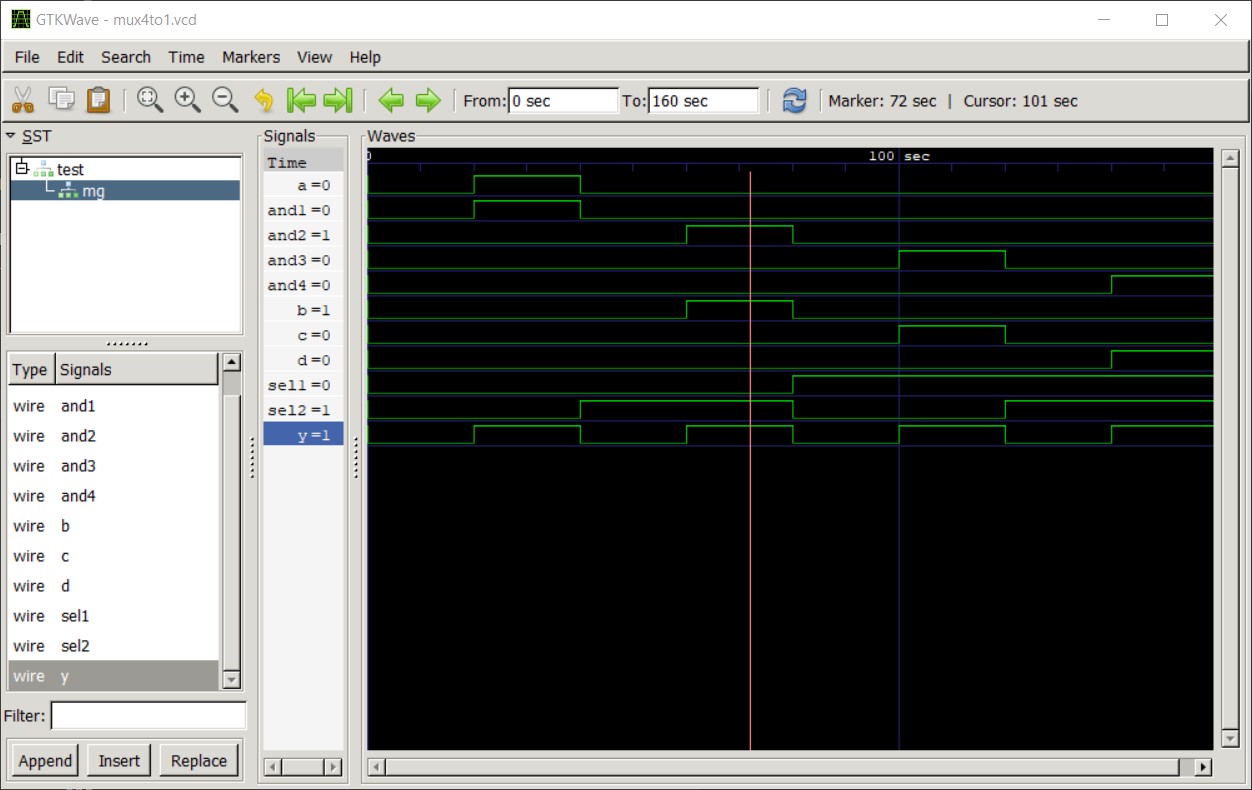
        #20

        $finish;

    end

endmodule





**Experiment 5:**

**Write HDL implementation for a 2-to-4 decoder. Simulate the same using structural model and depict the timing diagram for valid inputs.**

**Main Module:**

module decoder(Do,Din,En);

input En;

input[1:0]Din;

output [3:0]Do;

reg [3:0]Do;

always@(En or Din)

begin

if(En)

begin

case(Din)

    2'b00:Do=4'b0001;

    2'b01:Do=4'b0010;

    2'b10:Do=4'b0100;

    2'b11:Do=4'b1000;

default:Do=4'bzzzz;

endcase

end

end

endmodule

**Test Module:**

`include "decoder.v"

module test\_dec;

 reg [1:0]Din;

 reg En;

wire [3:0]Do;

decoder  udue(.Do(Do),.Din(Din),.En(En));

initial begin

 $dumpfile("decoder.vcd");

$dumpvars(0,test\_dec);

En=1;

Din=2'b00;#100;

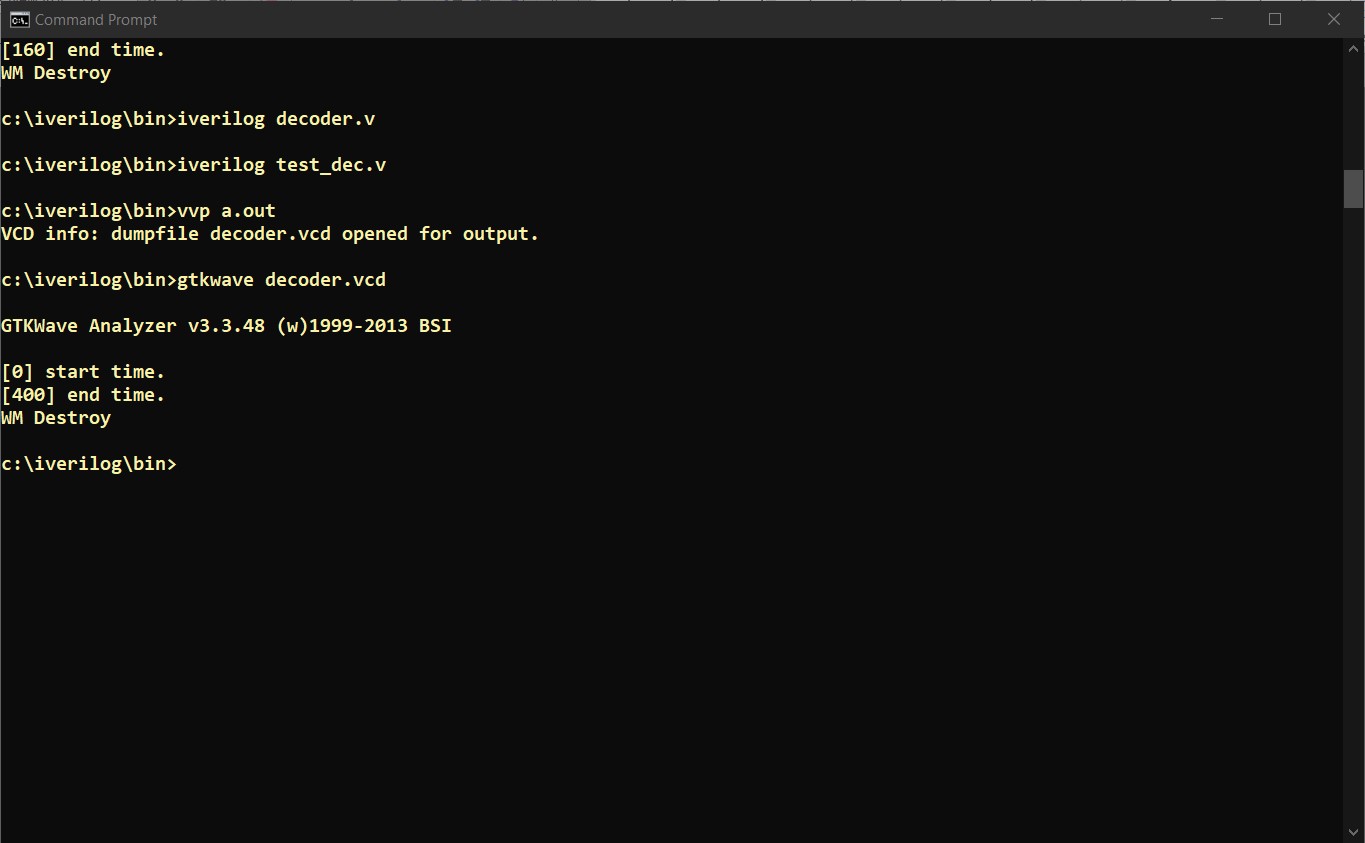
Din=2'b01;#100;

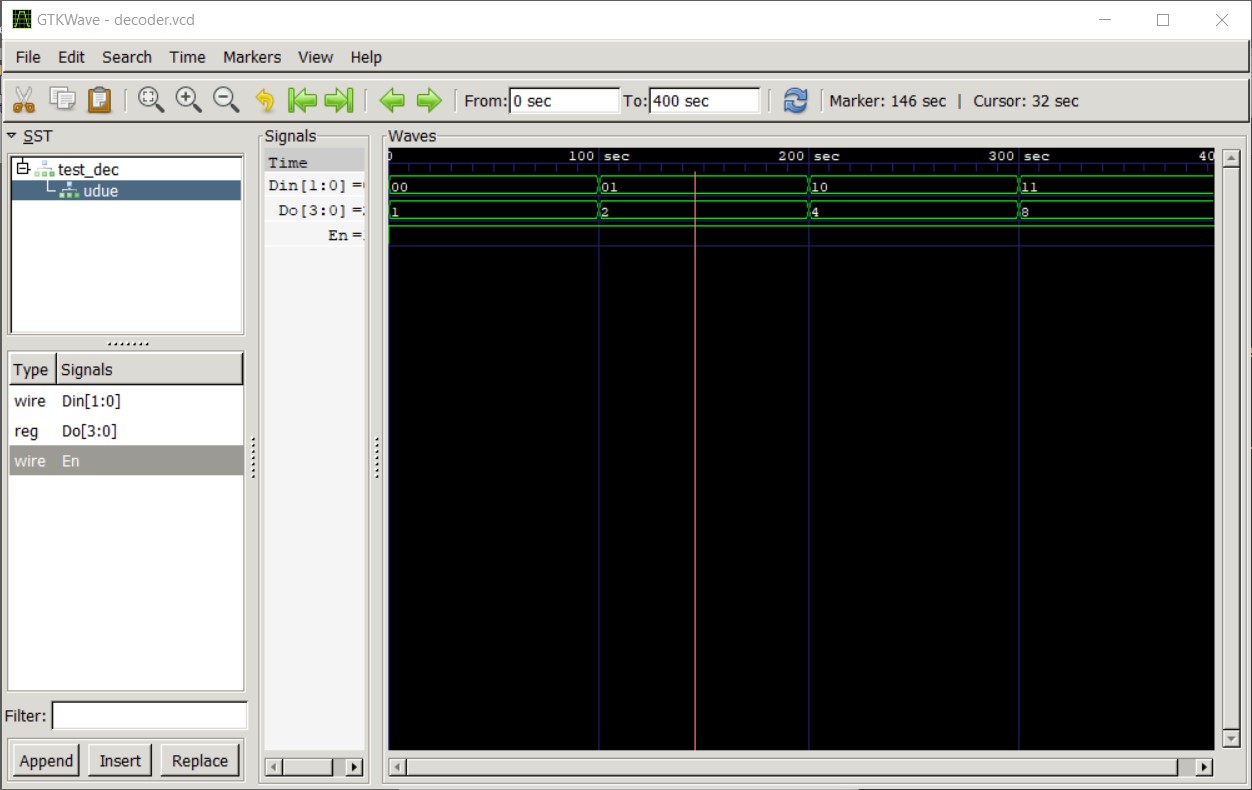
Din=2'b10;#100;

Din=2'b11;#100;

end

endmodule





**Experiment 6:**

**Write HDL implementation for a 4-to-2 encoder. Simulate the same using structural model and depict the timing diagram for valid inputs.**

**Main Module:**

module encoder(Do,Din,En);

input En;

input[3:0]Din;

output [1:0]Do;

reg [1:0]Do;

always@(En or Din)

begin

if(En)

begin

case(Din)

    4'b0001:Do=2'b00;

    4'b0010:Do=2'b01;

    4'b0100:Do=2'b10;

    4'b1000:Do=2'b11;

default:Do=2'bzz;

endcase

end

end

endmodule

**Test Module:**

`include "encoder.v"

module test\_encoder;

 reg [3:0]Din;

 reg En;

wire [1:0]Do;

encoder  mux1(.Do(Do),.Din(Din),.En(En));

initial begin

 $dumpfile("encoder.vcd");

$dumpvars(0,test\_encoder);

En=1;

Din=4'b0001;#100;

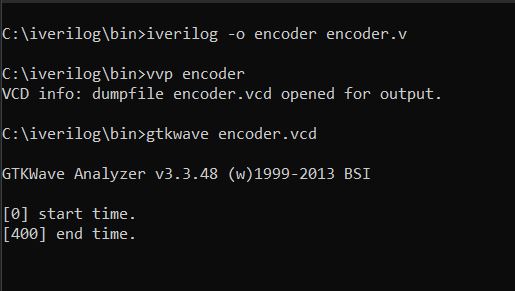
Din=4'b0010;#100;

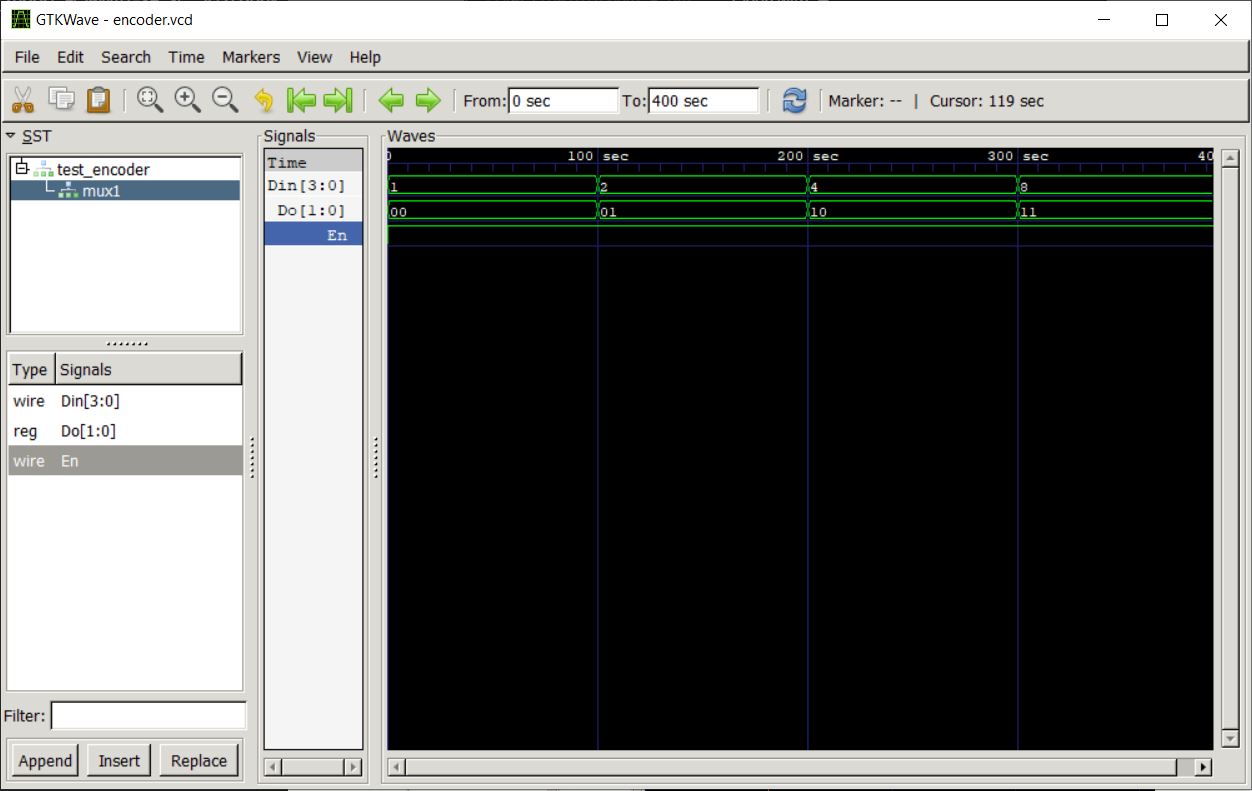
Din=4'b0100;#100;

Din=4'b1000;#100;

end

endmodule





**BEHAVIOUR MODELING**

**Experiment 7:**

**Write HDL implementation for a SR flip-flop using behavioral model. Simulate the same using behavioral model and depict the timing diagram for valid inputs**.

**Main Module:**

module SR\_FF (sr, clk, q, qb);

input [1:0] sr;

input clk;

output reg q=1'b0;

output reg qb;

always @ (posedge clk)

begin

       case (sr)

                2'b00 : q = q ;

                2'b01 : q = 1'b0 ;

                2'b10 : q = 1'b1 ;

                2'b11 : q = 1'bz ;

       endcase

            qb =~ q;

       end

endmodule

**Test Module:**

module testsrflipf;

   reg [1:0] A;

   reg c;

   wire x, xb;

  SR\_FF srff(A,c,x,xb);

  initial c=1'b0;

  always #5 c=~c;

  initial

    begin

    A=2'b00; #10

    A=2'b01;#10

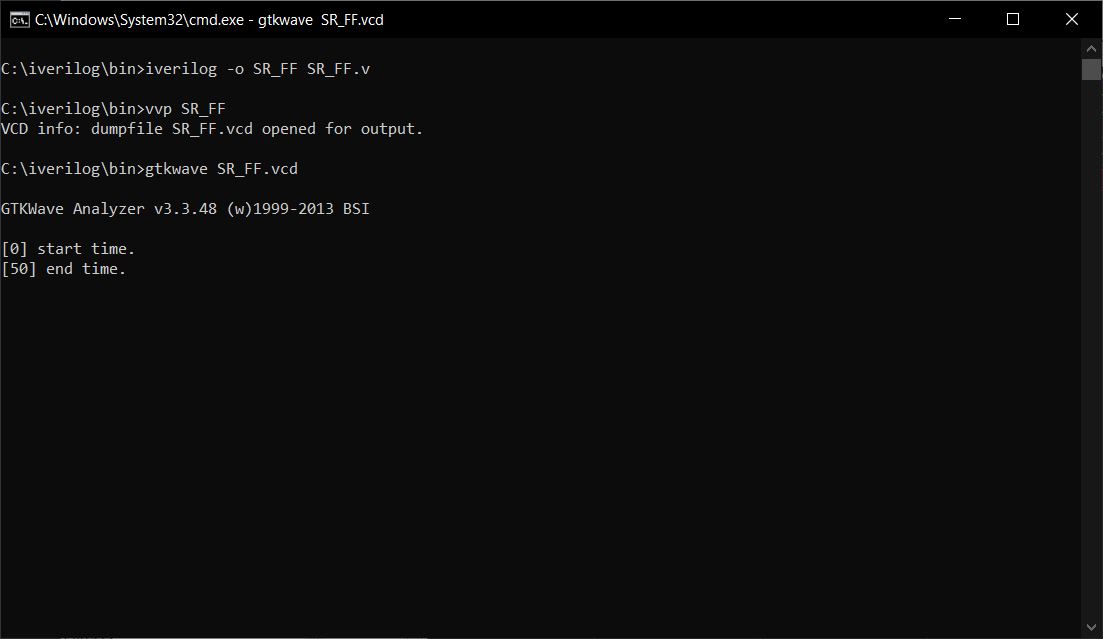
    A=2'b10;#10

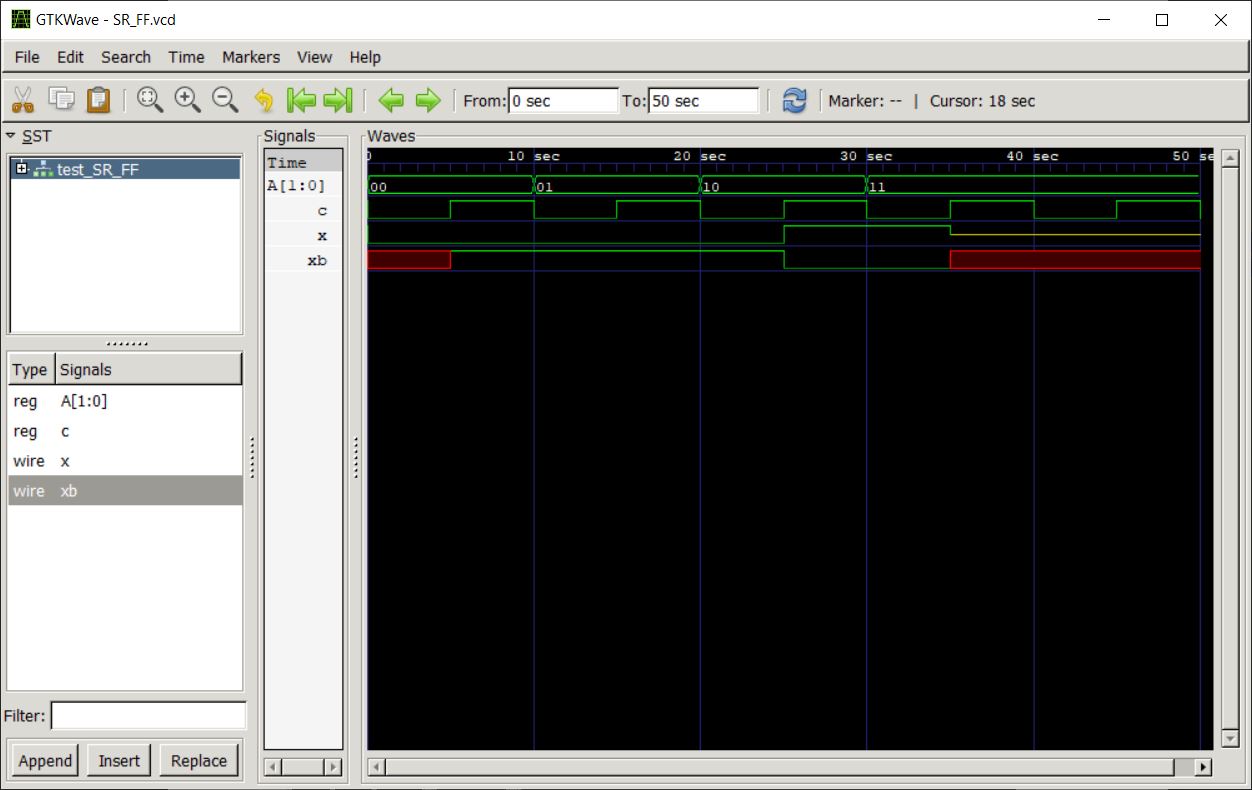
    A=2'b11;

    #20 $finish;

    end

endmodule





**Experiment 8:**

**Write HDL implementation for a JK flip-flop using behavioral model. Simulate the same using behavioral model and depict the timing diagram for valid inputs.**

**Main Module:**

module jk\_flipflop1(j,k,clk,reset,q,q\_bar);

input j,k,clk,reset;

output q,q\_bar;

wire j,k,clk,reset;

reg q,q\_bar;

always@(posedge clk)begin

if(reset)begin

q=1'b0;

q\_bar=1'b1;

end else begin

case({j,k})

{1'b0,1'b0}:begin q=q;q\_bar=q\_bar;end

{1'b0,1'b1}:begin q=1'b0;q\_bar=1'b1;end

{1'b1,1'b0}:begin q=1'b1;q\_bar=1'b0;end

{1'b1,1'b1}:begin q=~q;q\_bar=~q\_bar;end

endcase

end

end

endmodule

**Test Module:**

`include "jk\_flipflop1.v"

 module JK\_ff\_tb;

 reg clk;

 reg reset;

 reg j,k;

 wire q;

 wire qb;

jk\_flipflop1 flipflop(.clk(clk),.reset(reset),.j(j),.k(k),.q(q),.q\_bar(qb));

 initial begin

 $dumpfile("jk.vcd");

$dumpvars(0,JK\_ff\_tb);

$monitor(clk,j,k,q,qb,reset);

j=1'b0;

k=1'b0;

reset=1;

clk=1;

#10

reset=0;

j=1'b1;

k=1'b0;

#100

reset=0;

j=1'b0;

k=1'b1;

#100

reset=0;

j=1'b1;

k=1'b1;

#100

reset=0;

j=1'b0;

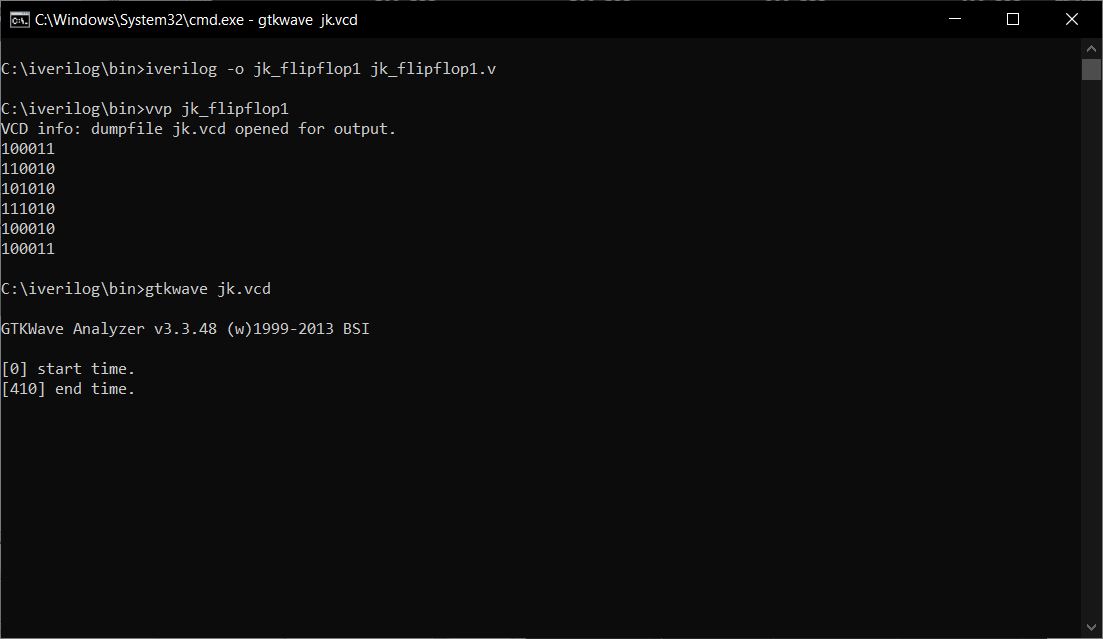
k=1'b0;

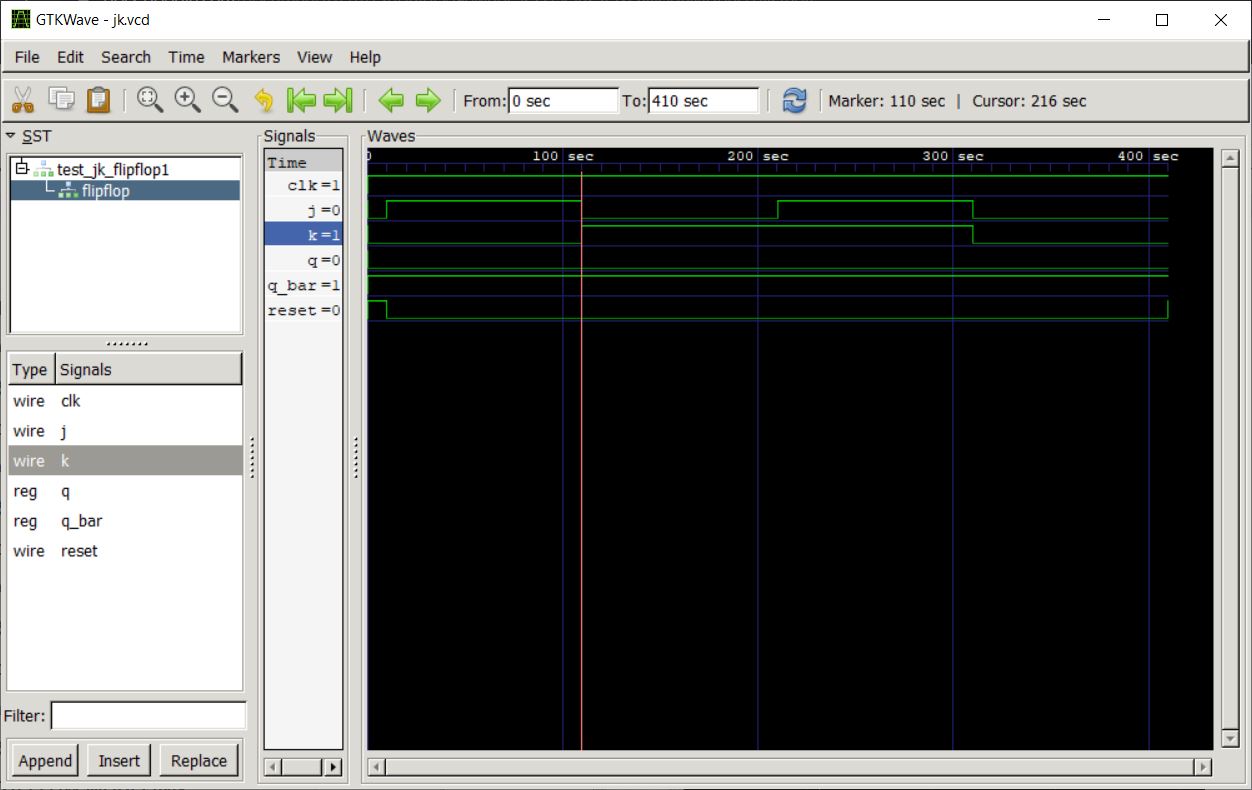
#100

reset=1;$finish;

 end

 endmodule





**Experiment 9:**

**Write HDL implementation for a 4-bit right shift register using behavioral model. Simulate the same using behavioral model and depict the timing diagram for valid inputs.**

**Main Module:**

module rshift(input clk,input clrb,input sdr,output reg [3:0]q);

always @(posedge(clk),negedge(clrb))

if(~clrb)

q<=4'b0000;

else

q<={sdr,q[3:1]};

endmodule

**Test Module:**

`include "rshift.v"

module test\_rshift;

reg clk,clrb,sdr;

wire [3:0]q;

rshift rs(clk,clrb,sdr,q);

initial

begin

$dumpfile("shift.vcd");

$dumpvars(0,test\_rshift);

clk=1;

clrb=0;

sdr=1;

#100

clrb=1;

sdr=1;

#150

sdr=0;

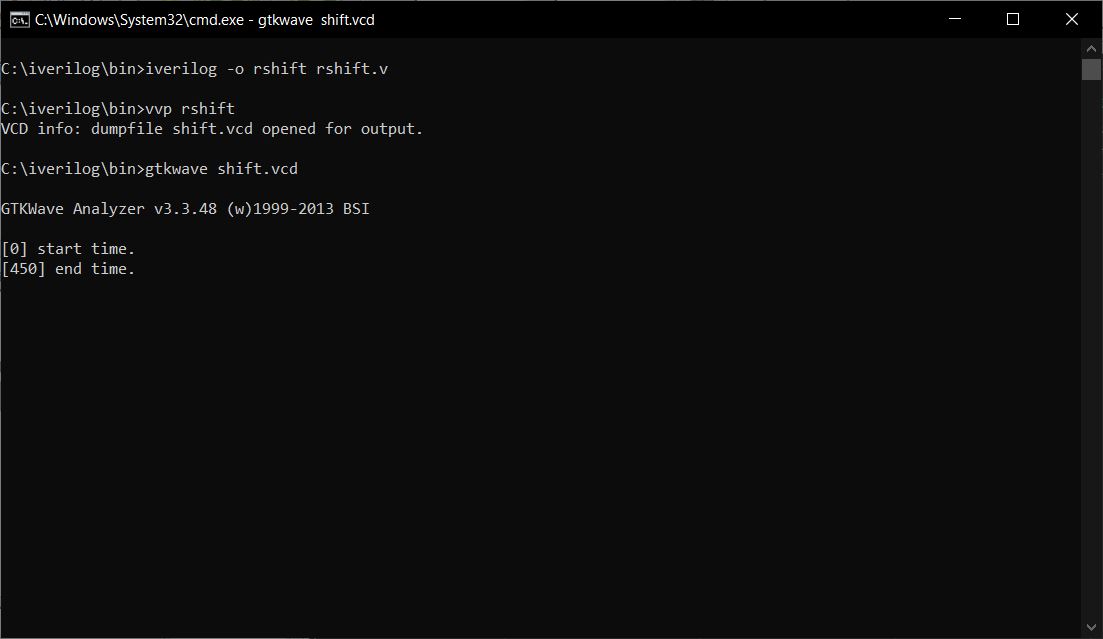
#200

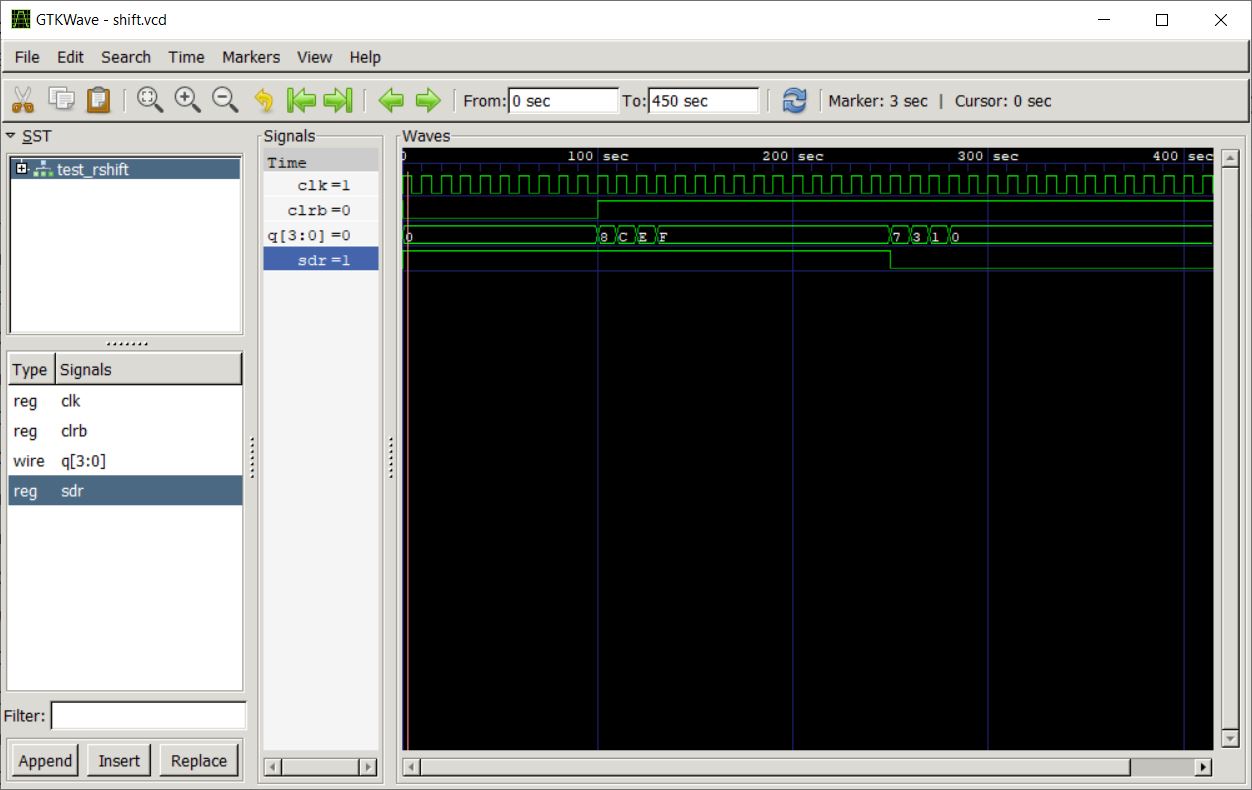
$finish;

end

always #5 clk=~clk;

endmodule





**Experiment 10:**

**Write HDL implementation for a 3-bit up-counter using behavioral model. Simulate the same using behavioral model and depict the timing diagram for valid inputs**

**Main Module:**

module counter2 ( count,rst,clk);

input rst, clk;

output reg [2:0] count;

always @(posedge (clk))

if (rst)

count<= 3'b000;

else

count<= count + 1;

endmodule

**Test Module:**

`include "counter2.v"

module testmod;

reg r,c;

wire  [2:0] ct;

counter2 countbeh (ct,r,c);

initial

begin

   $dumpfile("count.vcd");

$dumpvars(0,testmod);

   r =1;

 c=0;

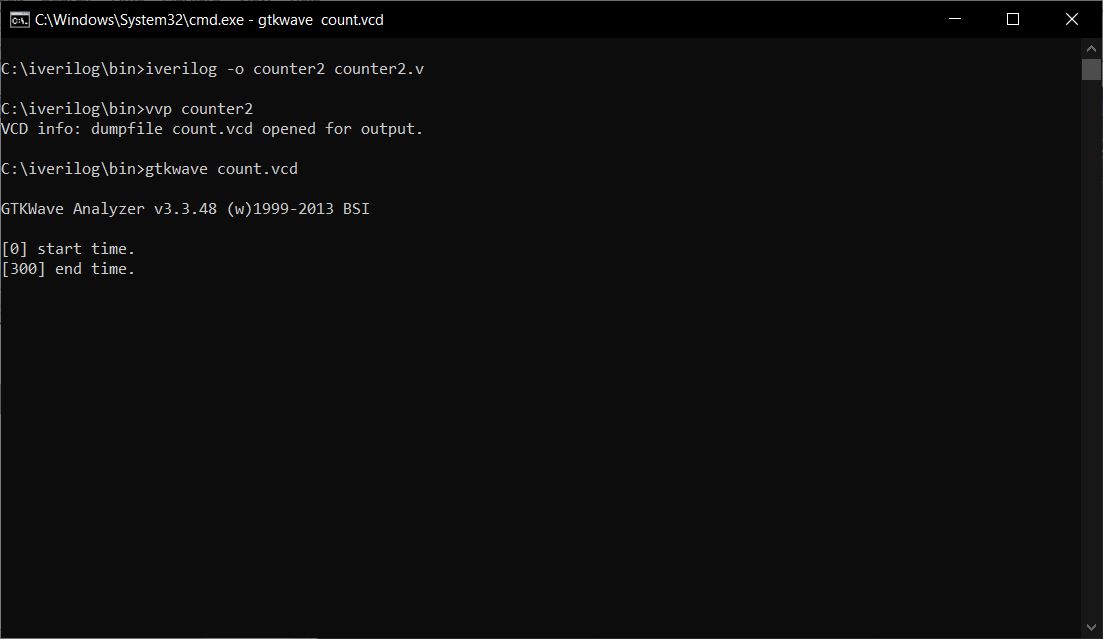
    #100 r=0;

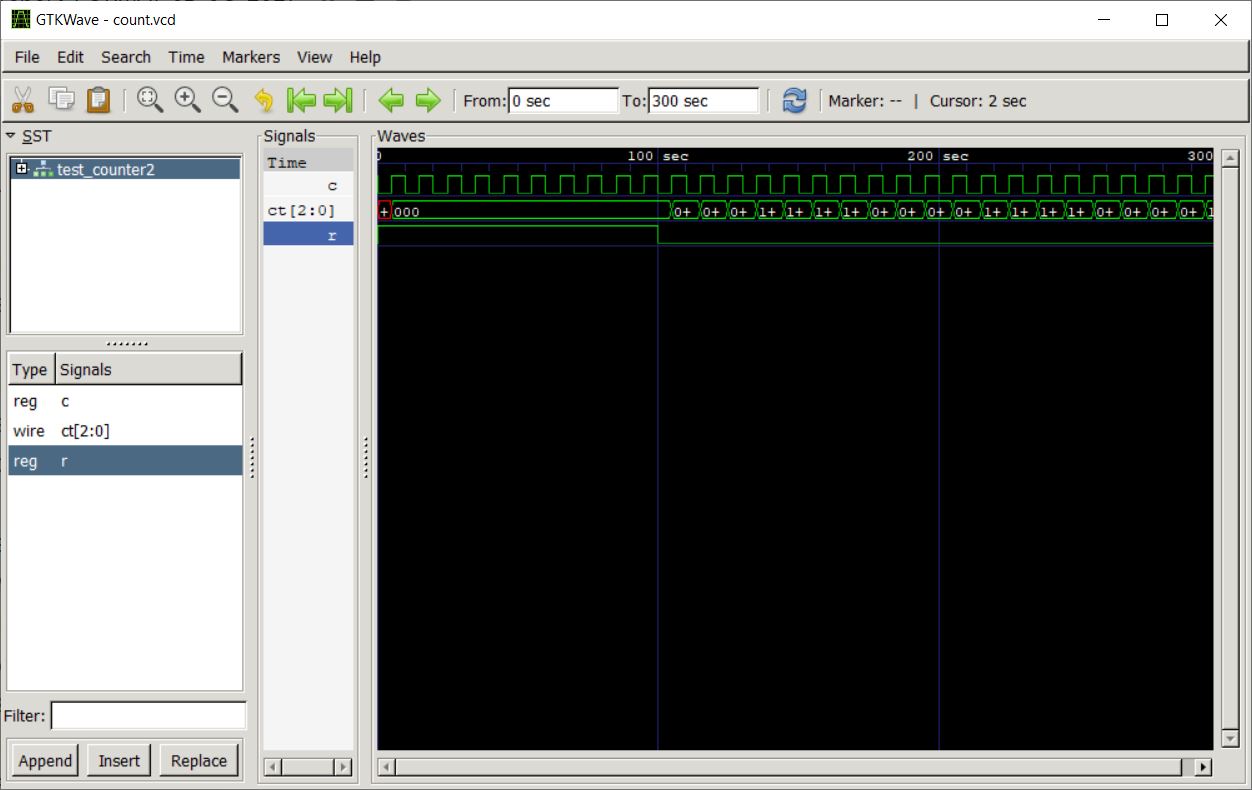
    #200 $finish;

end

always #5 c=~c;

endmodule





**DATA FLOW MODELING**

**Experiment 11:**

**Write HDL implementation for AND/OR/NOT gates using data flow model. Simulate the same using data flow model and depict the timing diagram for valid inputs.**

**Main Module:**

module gates(input a,b,output [2:0]y);

assign y[2]=a&b;

assign y[1]=a|b;

    assign y[0]=~a;

    endmodule

**Test Module:**

`include "gates.v"

module test\_gates;

wire [2:0]y;

reg a,b;

gates dut(.y(y),.a(a),.b(b));

initial

begin

$dumpfile("gates.vcd");

$dumpvars(0,test\_gates);

a=1'b0;

b=1'b0;

#50;

a=1'b0;

b=1'b1;

#50;

a=1'b1;

b=1'b0;

#50;

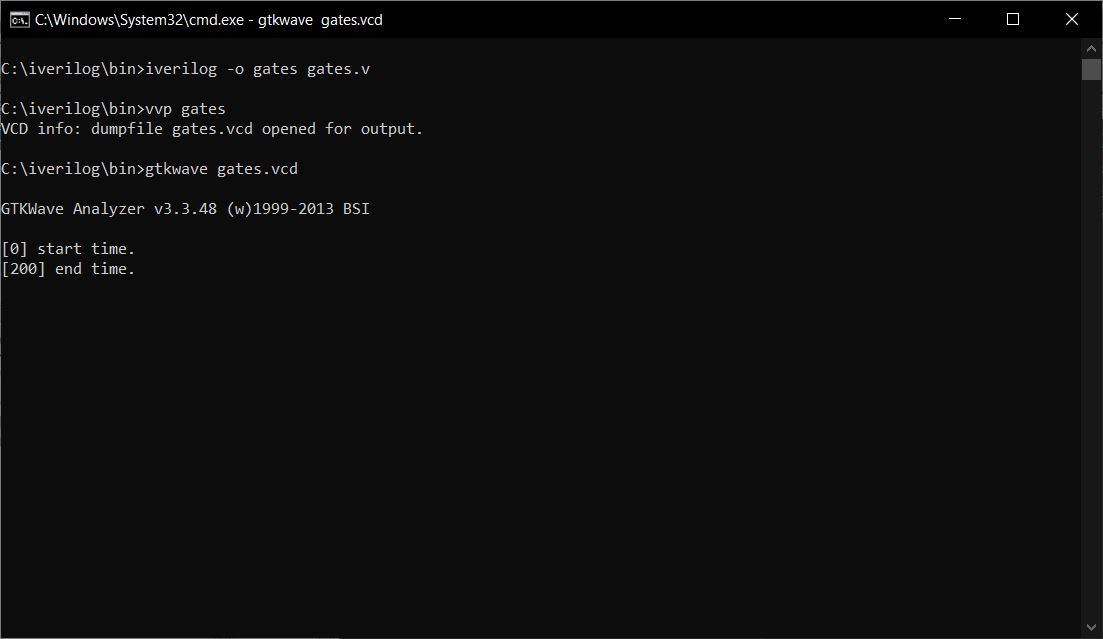
a=1'b1;

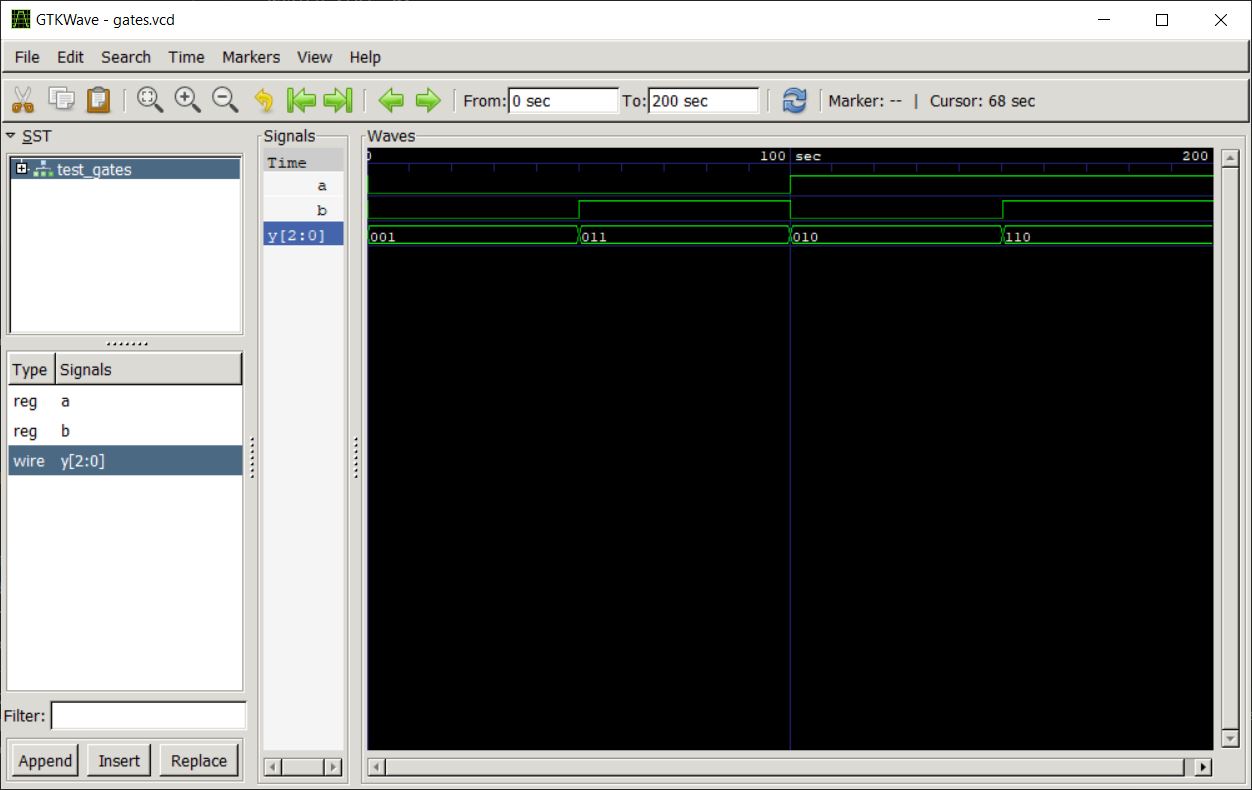
b=1'b1;

#50;

end

endmodule





**Experiment 12:**

**Write HDL implementation for a 3-bit full adder using data flow model. Simulate the same using data flow model and depict the timing diagram for valid inputs**.

**Main Module:**

module fa1(a,b,cin,s,cout);

input a,b,cin;

output s,cout;

assign s=a^b^cin;

assign cout=(a&b)|(b&cin)|(cin&a);

endmodule

**Test Module:**

`include "fulladd.v"

module test\_fa;

reg a,b,cin;

wire s,cout;

fulladd f1(a,b,cin,s,cout);

initial

begin

$dumpfile("fa.vcd");

$dumpvars(0,test\_fa);

    a=1;b=1;cin=0;

    #5

    a=1;b=1;cin=1;

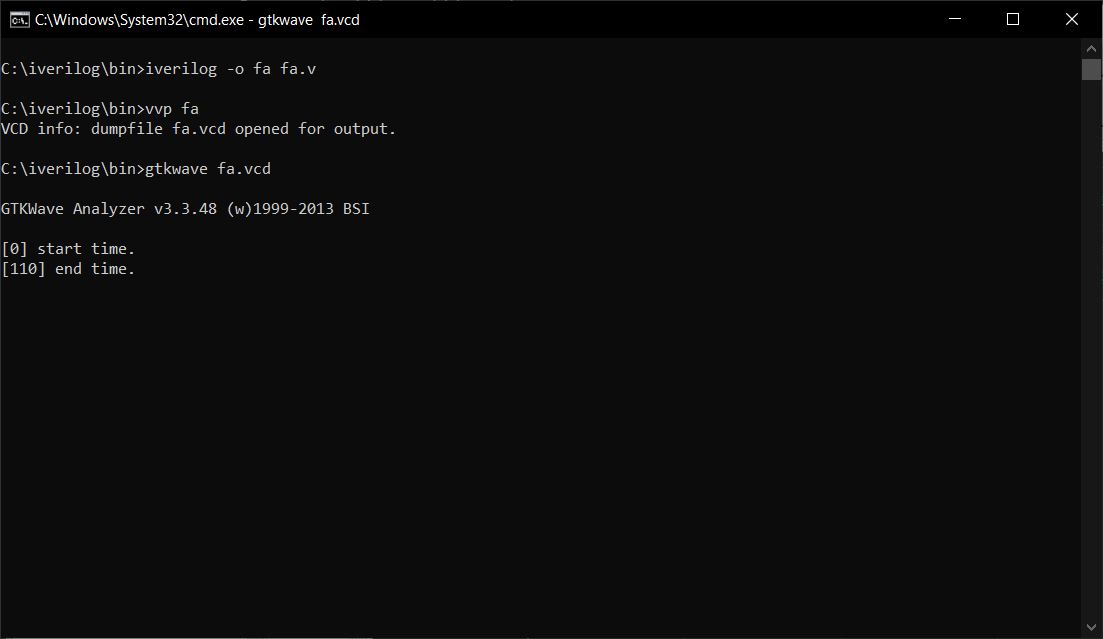
    #5

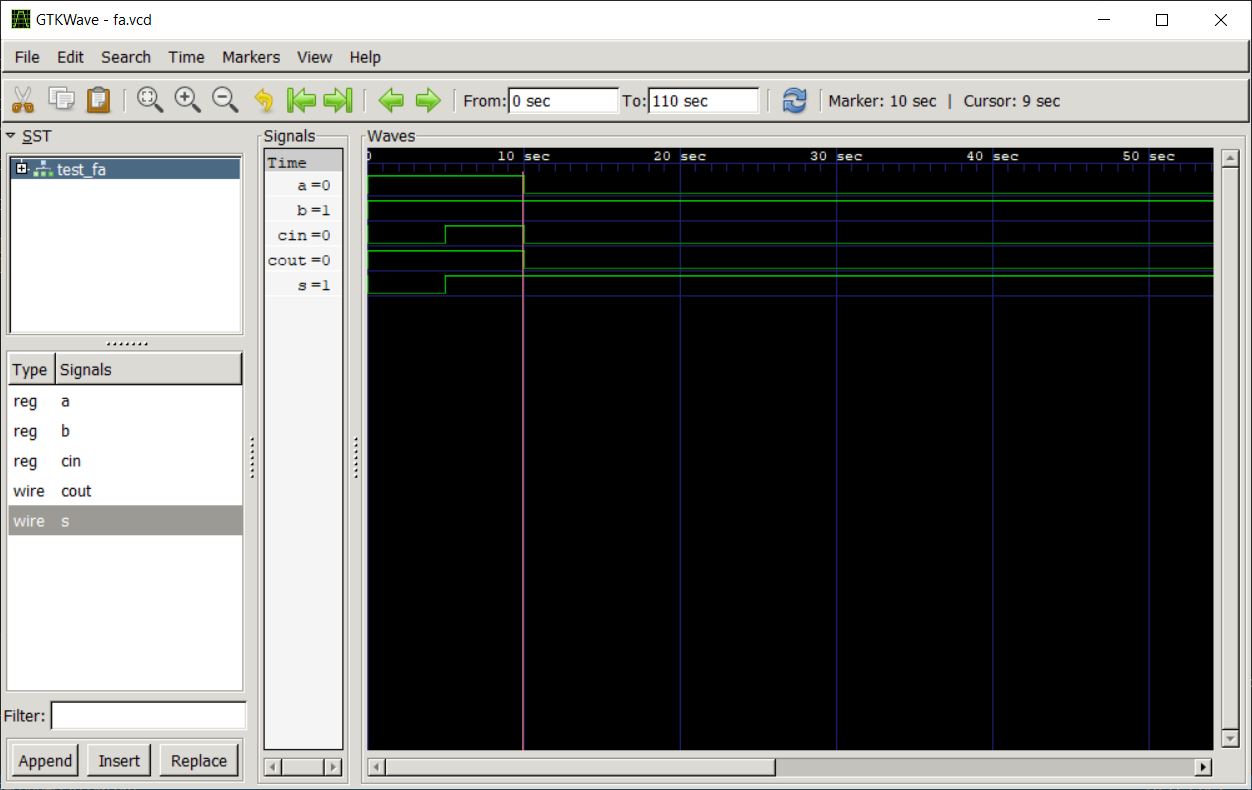
    a=0;b=1;cin=0;

    #100 $finish;

    end

endmodule





**Signature of the staff in-charge**