

STRUCTURAL MODELING

Experiment 1

1. Write HDL implementation for the following Logic
 - a. AND/OR/NOT

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

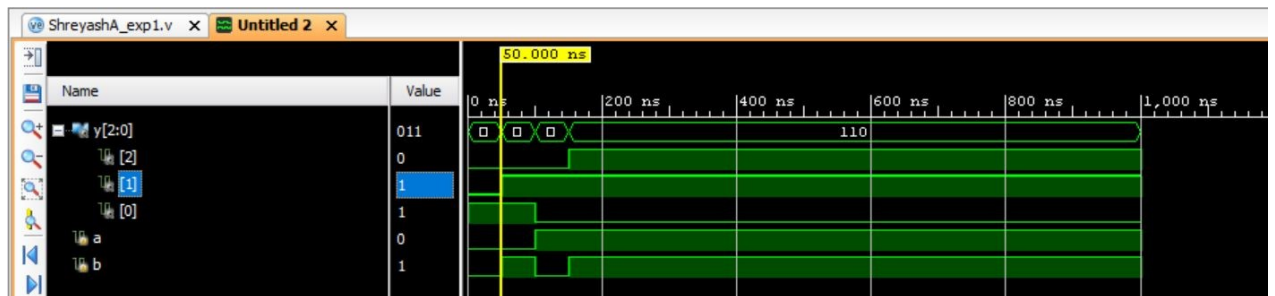
MAIN MODULE

```
module gates(input a, b, output [2:0]y);
    assign y[2]= a & b; // AND gate
    assign y[1]= a | b; // OR gate
    assign y[0]= ~a; // NOT gate
endmodule
```

TESTBENCH MODULE

```
module gates_tb;
    wire [2:0]y;
    reg a, b;
    gates dut(a,b,y);
    initial
    begin
        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
```

OUTPUT:



Experiment 2

Write HDL implementation for the following Logic

a. NAND/NOR

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

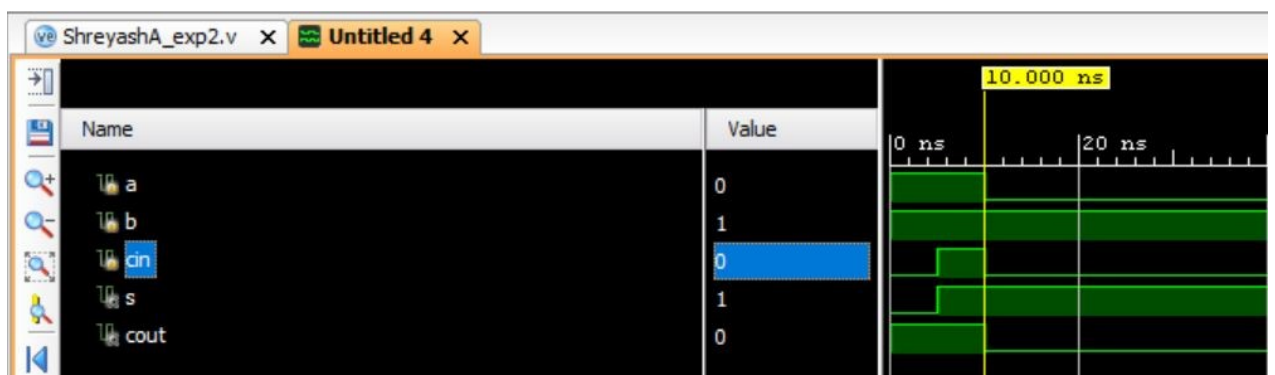
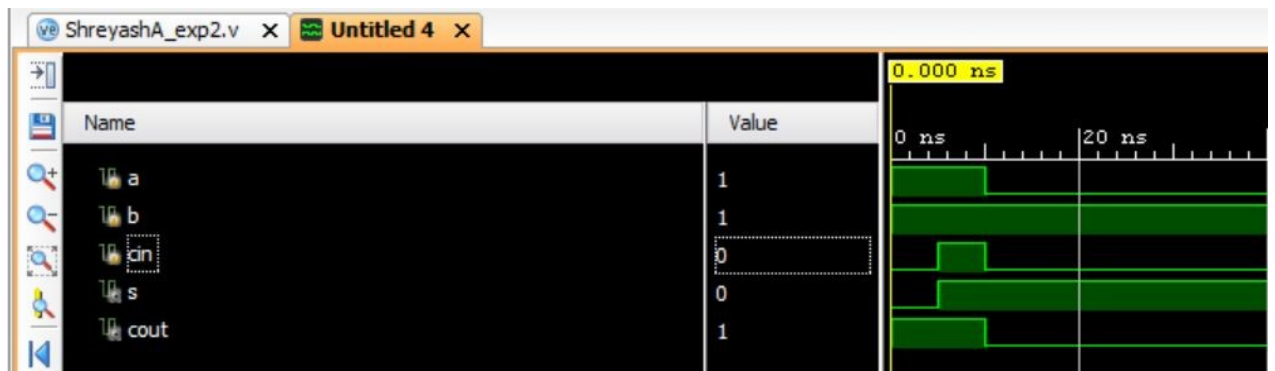
MAIN MODULE

```
module fa(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

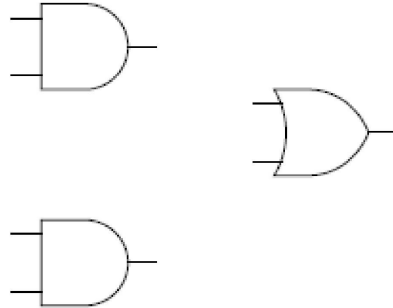
```
module fa_test;
    reg a,b,cin;
    wire s, cout;
    fa f1(a,b,cin,s,cout);
    initial
        begin
            a=1;  b=1; cin=0;
            #5
            a=1;  b=1; cin=1;
            #5
            a=0;  b=1; cin=0;
            #100 $finish;
        end
endmodule
```

OUTPUT:



Experiment 3

Write HDL implementation for the following Logic



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

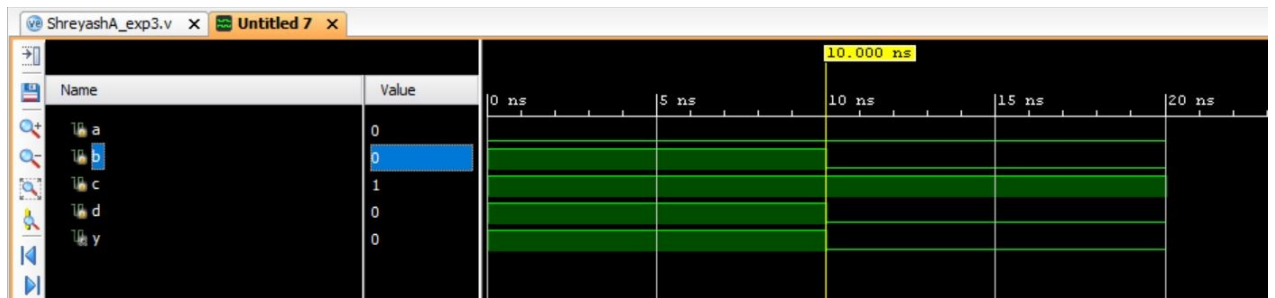
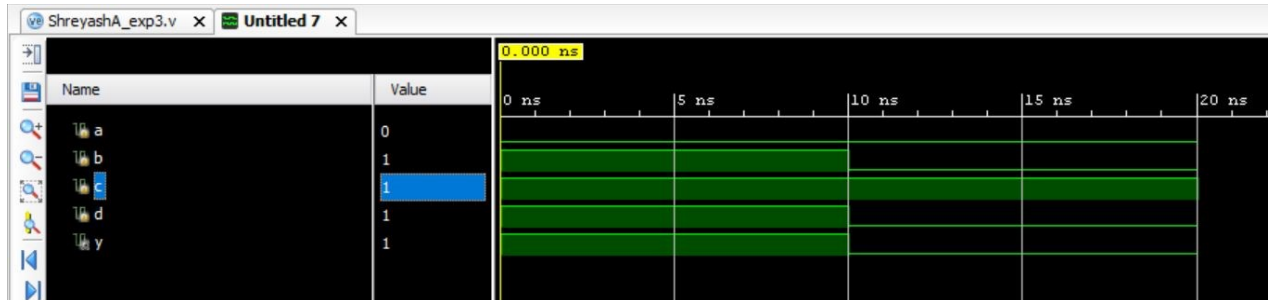
MAIN MODULE

```
module addor(A,B,C,D,Y);  
input A,B,C,D;  
output Y;  
wire and_op1, and_op2;  
and g1(and_op1,A,B);  
and g2(and_op2,C,D); // g2 represents lower AND  
or g3(Y,and_op1,and_op2); // g3 represents the OR gate  
endmodule
```

TEST MODULE

```
module test_andor;  
reg a,b,c,d;  
wire y;  
addor ao(a,b,c,d,y);  
initial  
begin  
a=0; b=1; c=1; d=1; #10  
a=0; b=0; c=1; d=0; #10  
$finish;  
end  
endmodule
```

OUTPUT:



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 Multiplexer4to1(Do, Din, En);
input En;
input [3:0] Din;
output Do;

reg [1:0]Do;

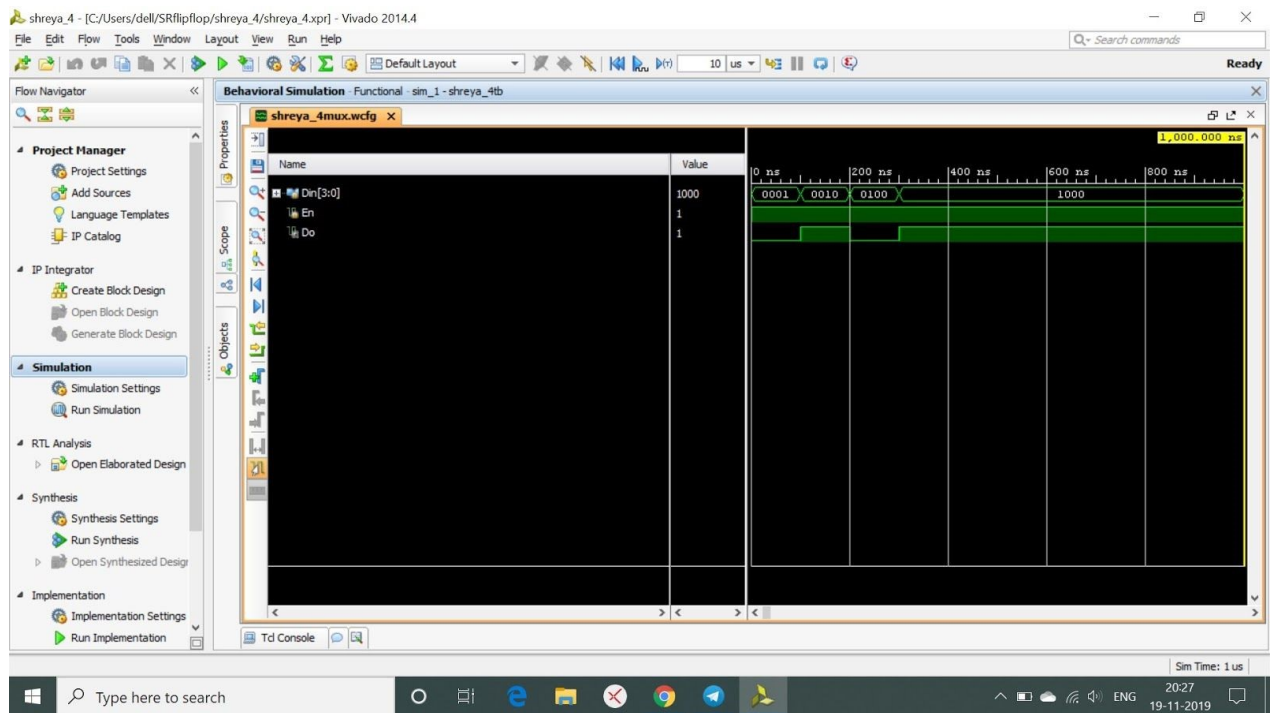
always @ (En or Din)
begin
if (En)
begin
case (Din)
4'b0001: Do = 2'b00;
4'b0010: Do = 2'b01;
4'b0100: Do = 2'b10;
4'b1000: Do = 2'b11;
default: Do=2'bzz;
endcase
end
end
endmodule
```

TESTBENCH MODULE

```
module multiplexer_tb;
reg [3:0] Din;
reg En;
wire Do;

Multiplexer4to1 mux(
.D0(Do),
.Din(Din),
.En(En)
);
initial begin
// Initialize Inputs
En = 1;
Din = 4'b0001; #100;
Din = 4'b0010; #100;
Din = 4'b0100; #100;
Din = 4'b1000; #100;
end
endmodule
```

OUTPUT:



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_case(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;

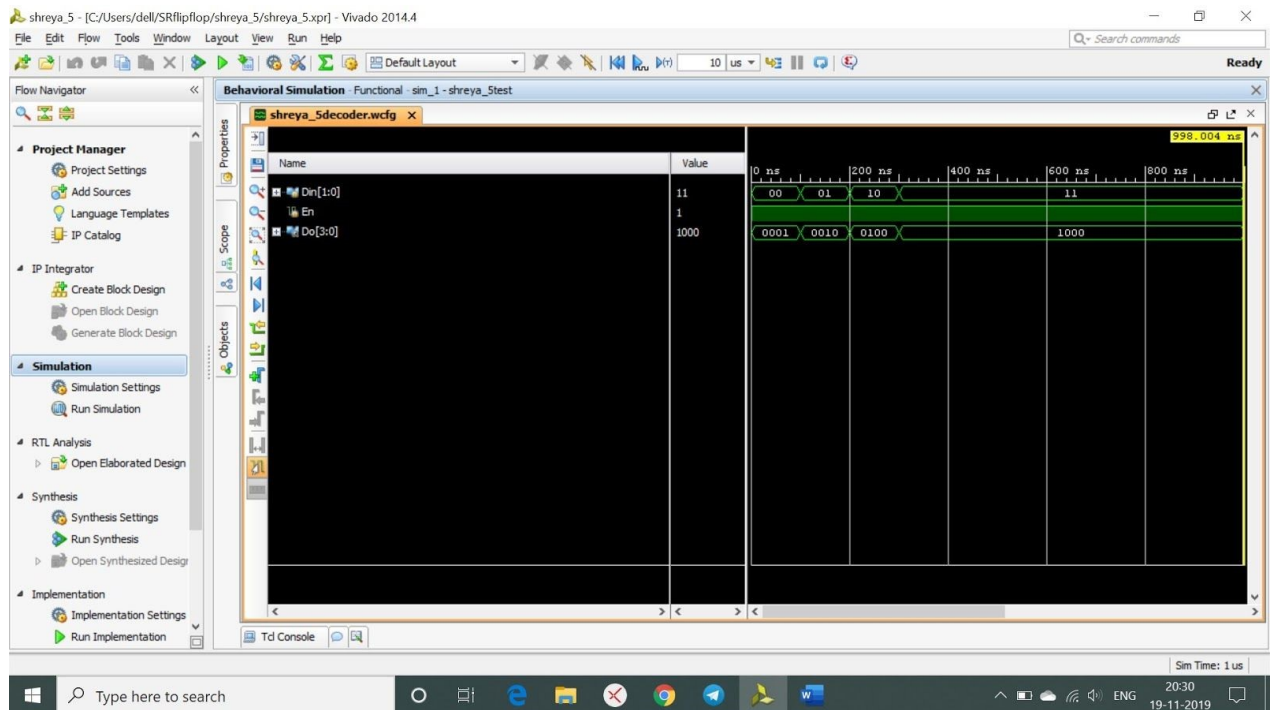
default: Do=4'bzzzz;
endcase
end
end
endmodule
```

TEST BENCH MODULE

```
module decoder_tb_v;
reg [1:0] Din;
reg En;
wire [3:0] Do;

decoder_case uut(
.D0(D0),
.Din(Din),
.En(En)
);
initial begin
// Initialize Inputs
En = 1;
Din =2'b00; #100;
Din = 2'b01; #100;
Din = 2'b10; #100;
Din = 2'b11; #100;
end
endmodule
```

OUTPUT:



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;

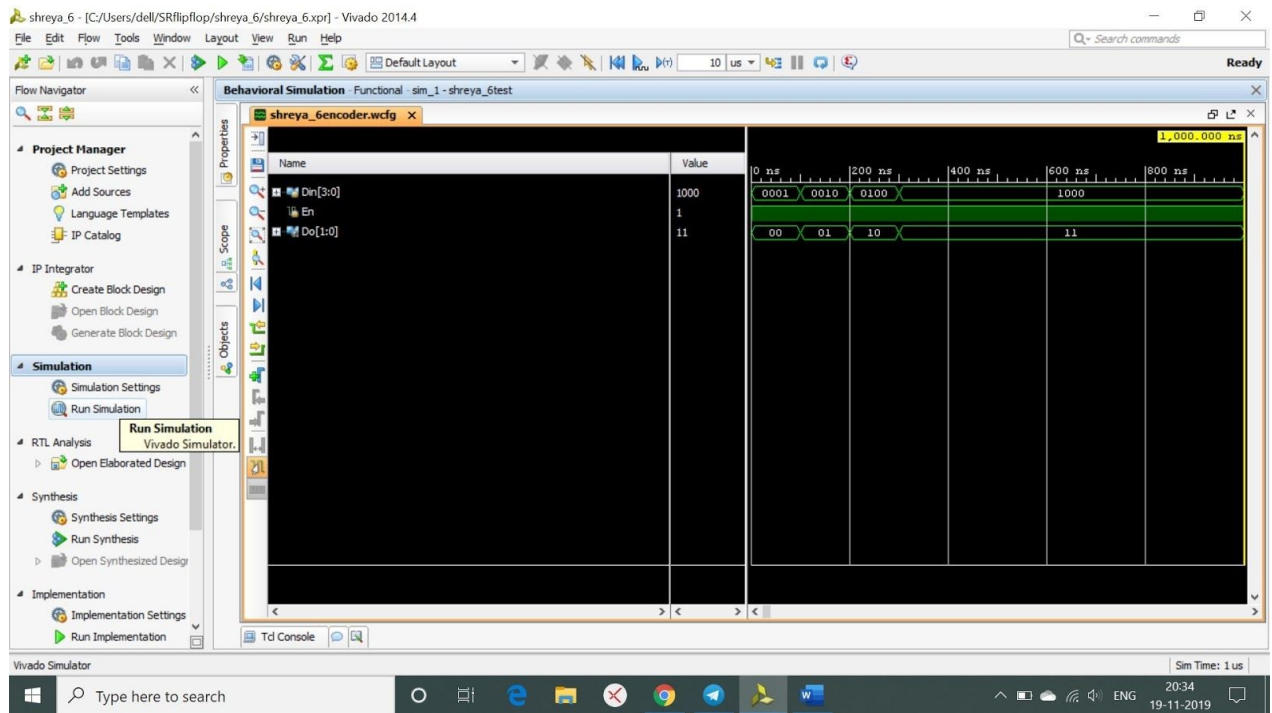
default: Do=2'bzz;
endcase
end
end
endmodule
```

TESTBENCH MODULE

```
module encoder_tb_v;
reg [3:0] Din;
reg En;
wire [1:0] Do;

Encoder uut(
.D0(D0),
.Din(Din),
.En(En)
);
initial begin
// Initialize Inputs
En = 1;
Din = 4'b0001; #100;
Din = 4'b0010; #100;
Din = 4'b0100; #100;
Din = 4'b1000; #100;
end
endmodule
```

OUTPUT:



BEHAVIOR MODELING

Experiment 7

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

MAIN MODULE

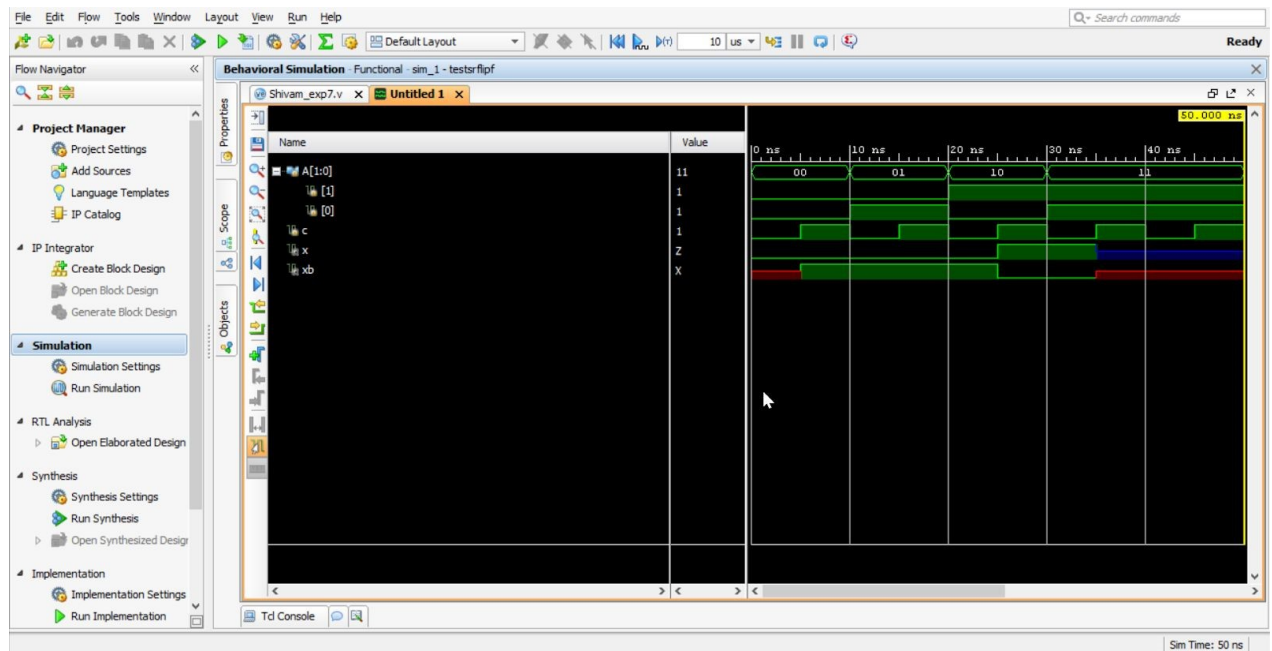
```
module SR_FF (sr, clk, q, qb);
input [1:0] sr;
input clk;
output reg q=1'b0;
output reg qb;
always @(posedge clk)
begin
    case (sr)
        2'b00 : q = q ;
                2'b01 : q = 1'b0 ;
                2'b10 : q = 1'b1 ;
                2'b11 : q = 1'bz ;
    endcase
    qb =~ q;
end
endmodule
```

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
```

OUTPUT:



Experiment 8

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

MAIN MODULE

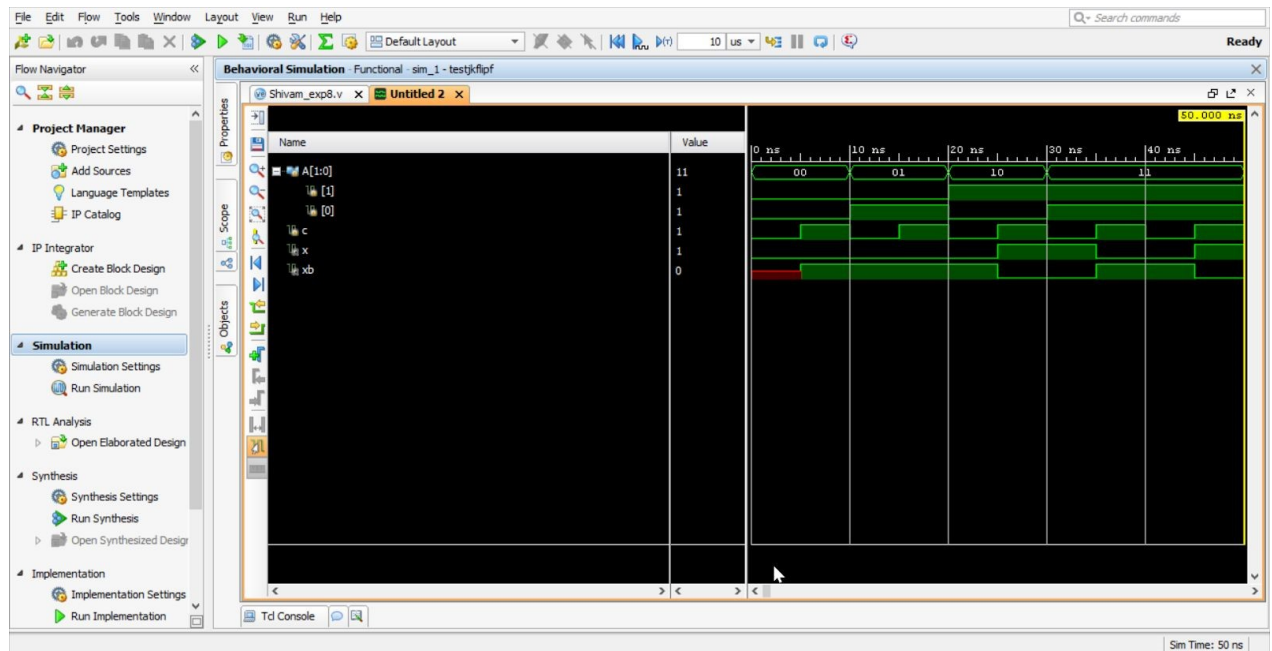
```
module JK_FF (jk, clk, q, qb);
input [1:0] jk;
input clk;
output reg q=1'b0;
output reg qb;
always @ (posedge clk)
begin
    case (jk)
        2'b00 : q = q ;
        2'b01 : q = 1'b0 ;
        2'b10 : q = 1'b1 ;
        2'b11 : q = ~q ;
    endcase
    qb = ~ q;
end
endmodule
```

TEST MODULE

```
module testjkflipf;
reg [1:0] A;
reg c;
wire x, xb;
JK_FF jkff(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
```

OUTPUT:



Experiment 9

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

MAIN MODULE

```
module Rshiftregister( input clk, input clrb, input SDR, output reg [3:0] Q );
//serial in, parallel out
    always @ ( posedge(clk), negedge(clrb))

        if (~clrb) Q<=4'b0000;
        else

            Q<={SDR,Q[3:1]};
endmodule
```

TEST MODULE

```
module testRshiftregister;
    reg clk,clrb,SDR;
    wire [3:0]Q;

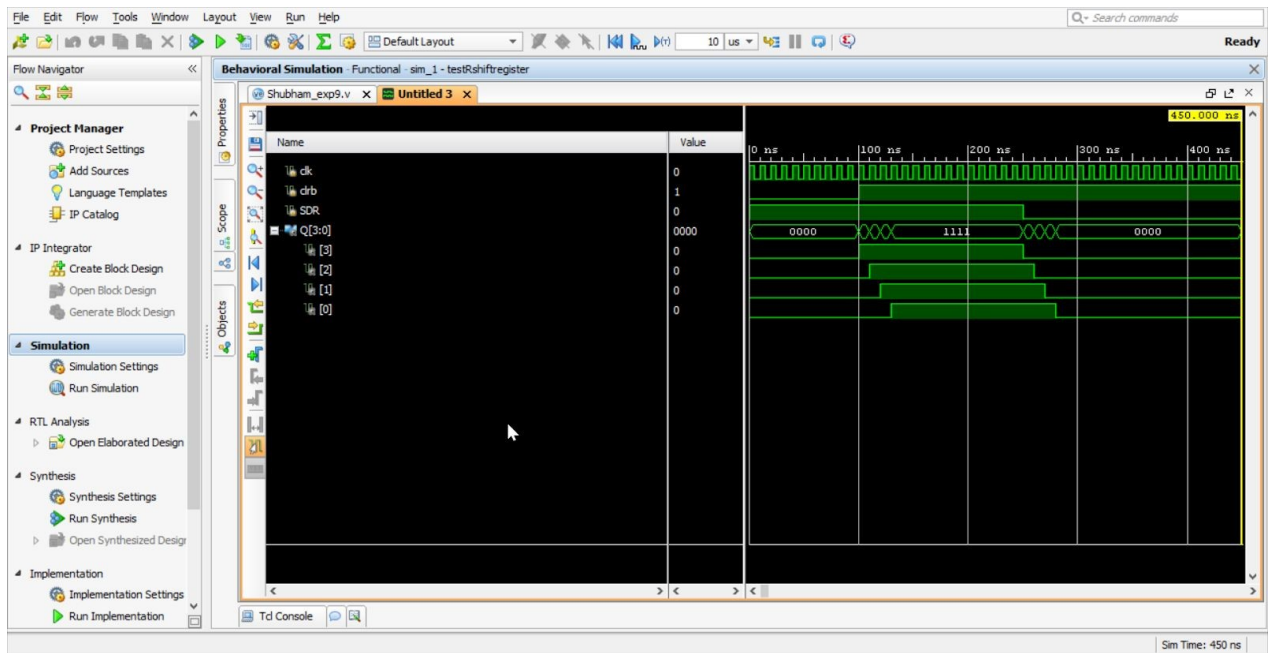
    Rshiftregister RS(clk, clrb, SDR, Q );
    initial
    begin

        clk =1;
        clrb=0;

        SDR=1;

        #100
        clrb=1;
        SDR=1;
        #150
        SDR=0;
        #200 $finish;
    end
    //initial and always run in parallel and starts its execution at 0ns
    always #5 clk=~clk;
endmodule
```

OUTPUT:



Experiment 10

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

Main Module

```
module counter_behav ( 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

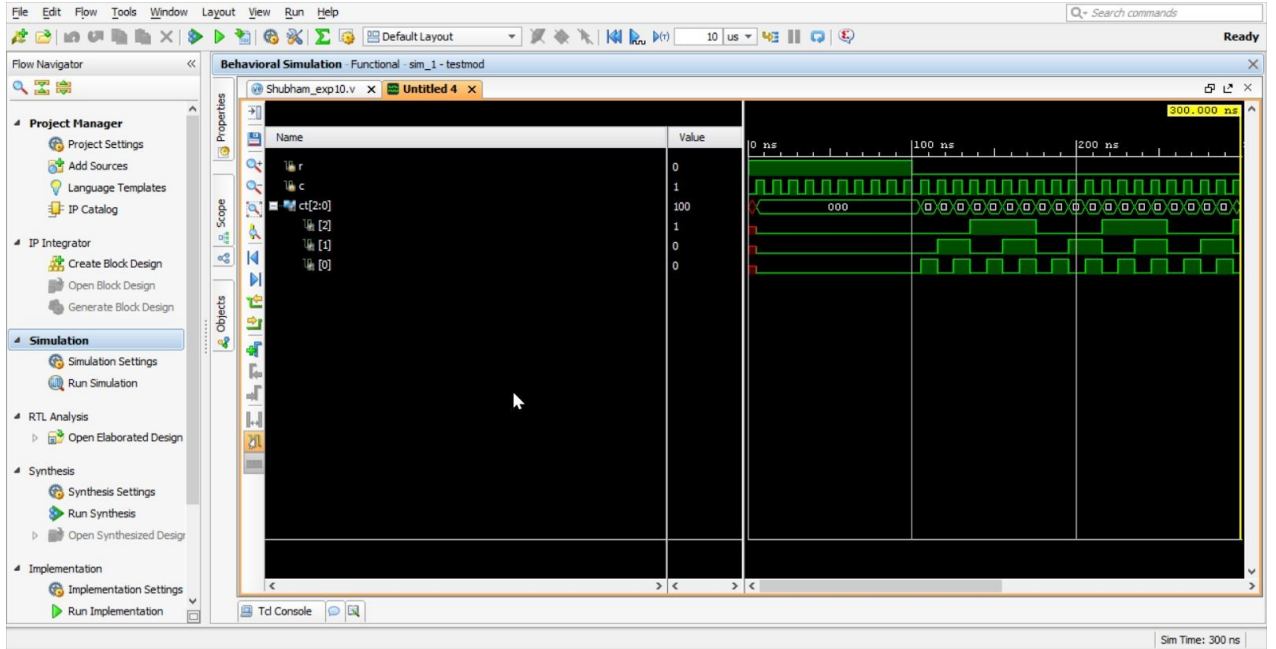
```
module testmod;
reg r,c;
wire [2:0] ct;

counter_behav countbeh (ct,r,c);
initial
begin

    r=1;

    c=0;
    #100 r=0;
    #200 $finish;
end
//initial and always run in parallel and starts its execution at 0ns
always #5 c=~c;
endmodule
```

OUTPUT:



DATA FLOW MODELING

Experiment 11

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

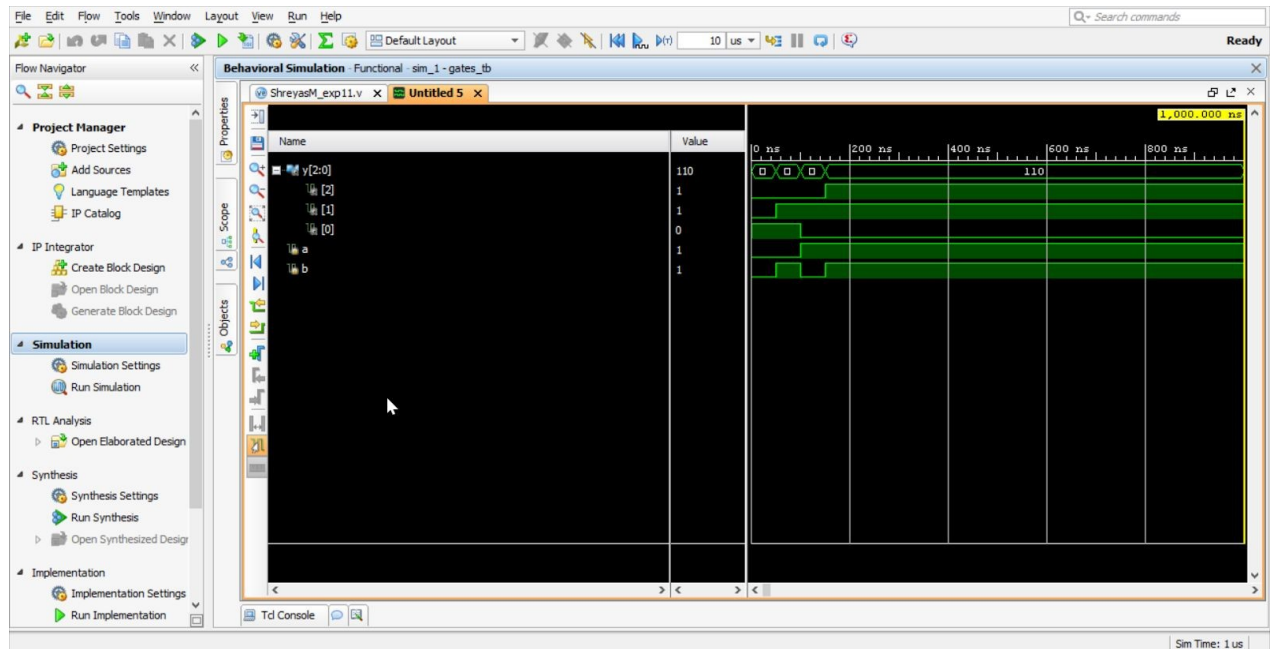
MAIN MODULE

```
module gates(input a, b, output [2:0]y);
assign y[2]= a & b; // AND gate
assign y[1]= a | b; // OR gate
assign y[0]= ~a; // NOT gate
endmodule
```

TESTBENCH MODULE

```
module gates_tb;
wire [2:0]y;
reg a, b;
gates dut(.y(y), .a(a), .b(b));
initial
begin
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
```

OUTPUT:



Experiment 12

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

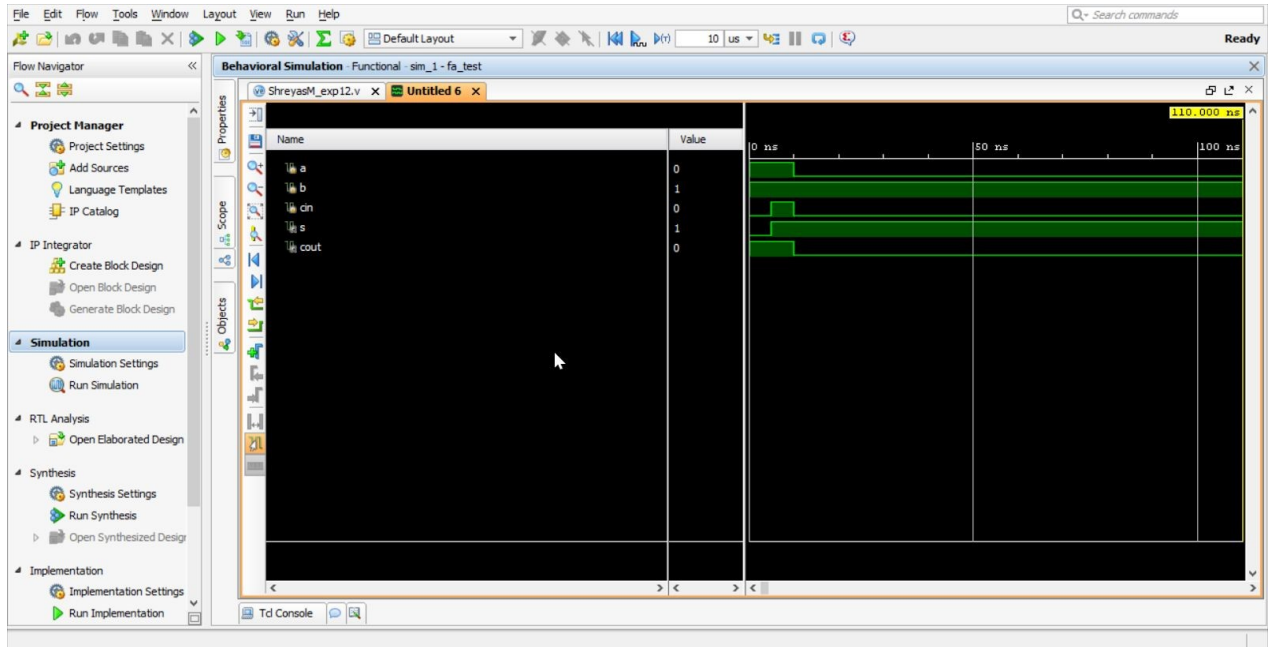
MAIN MODULE

```
module fa(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

```
module fa_test;
    reg a,b,cin;
    wire s, cout;
    fa f1(a,b,cin,s,cout);
    initial
        begin
            a=1;   b=1; cin=0;
            #5
            a=1;   b=1; cin=1;
            #5
            a=0;   b=1; cin=0;
            #100 $finish;
        end
endmodule
```

OUTPUT:



Signature of the staff in-charge