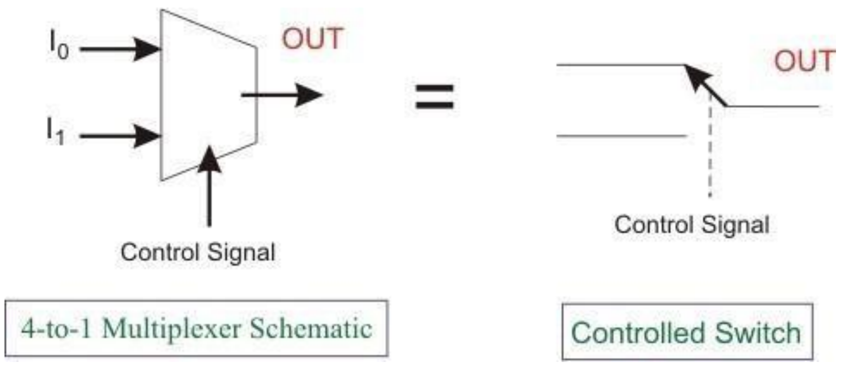
Experiment No.2

Student Name: Priyanshu Mathur **UID:** 20BEC1073
Branch: B.E. ECE **Section/Group:** A
Semester: 6th **Date of Performance:** 2/3/2023
Subject Name: Digital System Design Using Verilog **Subject Code:** 20ECP-373

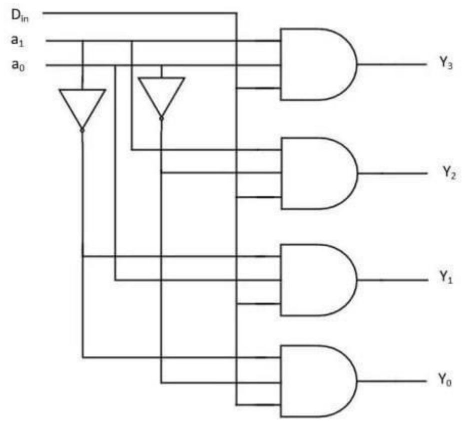
1. **Aim:** Write an HDL code for MUX / DMUX **2.**
Software used: Xilinx 12.4 ISE Design suite 14.7
3. **Modeling style:** Data flow model.
4. **Theory:**

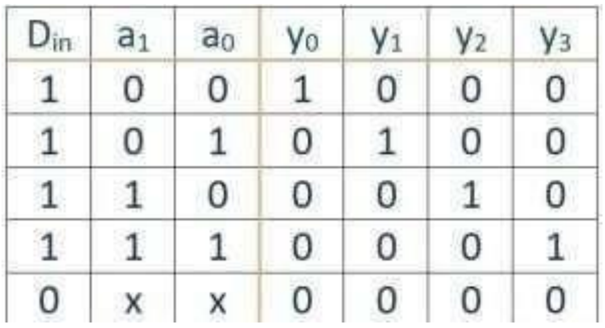
Multiplexer: A multiplexer (sometimes spelled multiplexor and also known as a MUX) is defined as a combinational circuit that selects one of several data inputs and forwards it to the output. The inputs to a multiplexercanbe analog or digital. Multiplexers are also known as data selectors. A multiplexer is useful for transmitting a large amount of data over the network within a certain amount of time and bandwidth.
In digital systems, many times it is necessary to select a single data line from several data-input lines and the datafrom the selected data input line should be available on the output line. The digital circuit which does this task is a multiplexer.
A multiplexer is a digital circuit that selects one of the n data inputs and forwards it to the output. The selection of one of the n inputs is done by the select inputs. To select one of several inputs, we need m select lines such that 2^m=n.



Demultiplexer: A demultiplexer is a circuit that places the value of a single data input onto multiple data outputs. The demultiplexer circuit can also be implemented using a decoder circuit.
Here we are going to work with 1-to-4 demultiplexer.

The below diagram shows the circuit of the 1-to-4 demultiplexer. Here a_1 and a_0 are control or select lines y_0, y_1, y_2, y_3 are outputs, and D_{in} is the data line.





D_{in}	a_1	a_0	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0

The values of a_1a_0 determine which of the outputs are set to the value of D_{in}. When D_{in}=0, all the outputs are set to 0, including the one selected by the valuation of a_1a_0. When D_{in}=1, the valuation of a_1a_0 sets the appropriate output (anyone from y0, y1, y2, y3) to 1.

Now that we have thoroughly understood the concepts of the demultiplexer, let's dive directly into the Verilog code for the demultiplexer.

4 Verilog CODE:

- **For Multiplexer**

```
module MUX(input a, input
b, input c, input d, input s1,
input s2, output Out   );
assign Out = s1 ? (s2 ? a : b) : (s2 ? c :d);
endmodule
```

- **For Demultiplexer**

```
 module
Demux(      input
  [1:0] sel, input
  i,
  output reg y0,y1,y2,y3);

  always @(*) begin
    case(sel)
      2'h0: {y0,y1,y2,y3} = {i,3'b0};
      2'h1: {y0,y1,y2,y3} = {1'b0,i,2'b0};
      2'h2: {y0,y1,y2,y3} = {2'b0,i,1'b0};
      2'h3: {y0,y1,y2,y3} = {3'b0,i};
      default: $display("Invalid sel input");
      endcase
 end
 endmodule
```

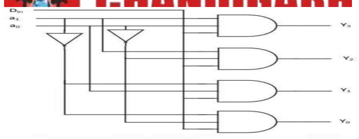
6. Test Bench Code:

 For Multiplexer

```
module logicgates;
        // Inputs
        reg A; reg
        B; //
        Outputs
        wire S; wire
        H; wire F;
        wire E; wire
        I; wire R;
        wire M;
        // Instantiate the Unit Under Test (UUT)
        Logic_gates uut (
                .A(A),
                .B(B),
                .S(S),
                .H(H),
                .F(F),
                .E(E),
                .I(I),
                .R(R),
                .M(M) );
        initial begin
                // Initialize Inputs
                A = 0;B = 0;
                // Wait 100 ns for global reset to finish
                #100
                // Add stimulus here
A = 0; B = 0; #100;
A = 0; B = 1; #100;
A = 1; B = 0; #100;
A = 1; B = 1;
#100; end
endmodule
```

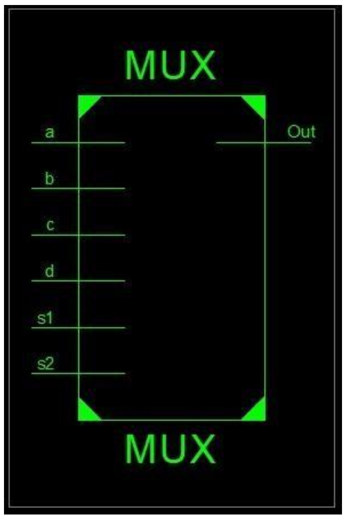
 For Demultiplexer

```
        tbmodule;
 reg [1:0] sel;
 reg i;
 wire y0,y1,y2,y3;
```

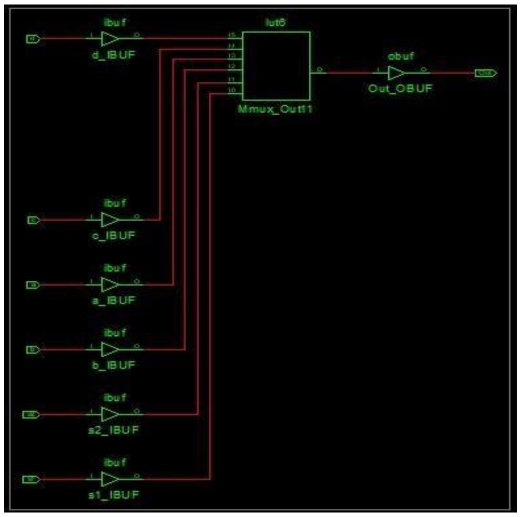


```
  demux_1_4 demux(sel, i, y0, y1, y2, y3); initial begin $monitor("sel = %b, i = %b -> y0 = %0b, y1 = %0b
  ,y2 = %0b, y3 = %0b", sel,i, y0,y1,y2,y3);
    sel=2'b00; i=0; #1;
    sel=2'b00; i=1; #1;
    sel=2'b01; i=0; #1;
    sel=2'b01; i=1; #1;
    sel=2'b10; i=0; #1;
    sel=2'b10; i=1; #1;
    sel=2'b11; i=0; #1;
    sel=2'b11; i=1; #1;
  end
endmodule
```

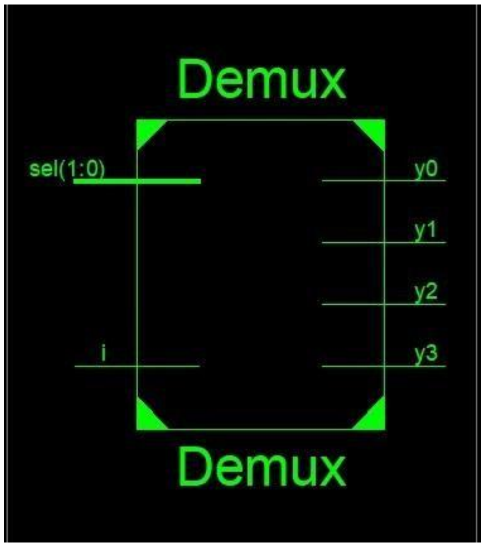
MUX RTL Schematic Screenshot:



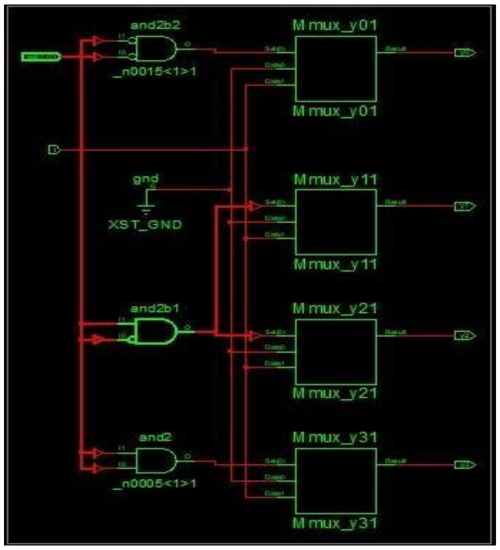
Mux Technology Schematic:



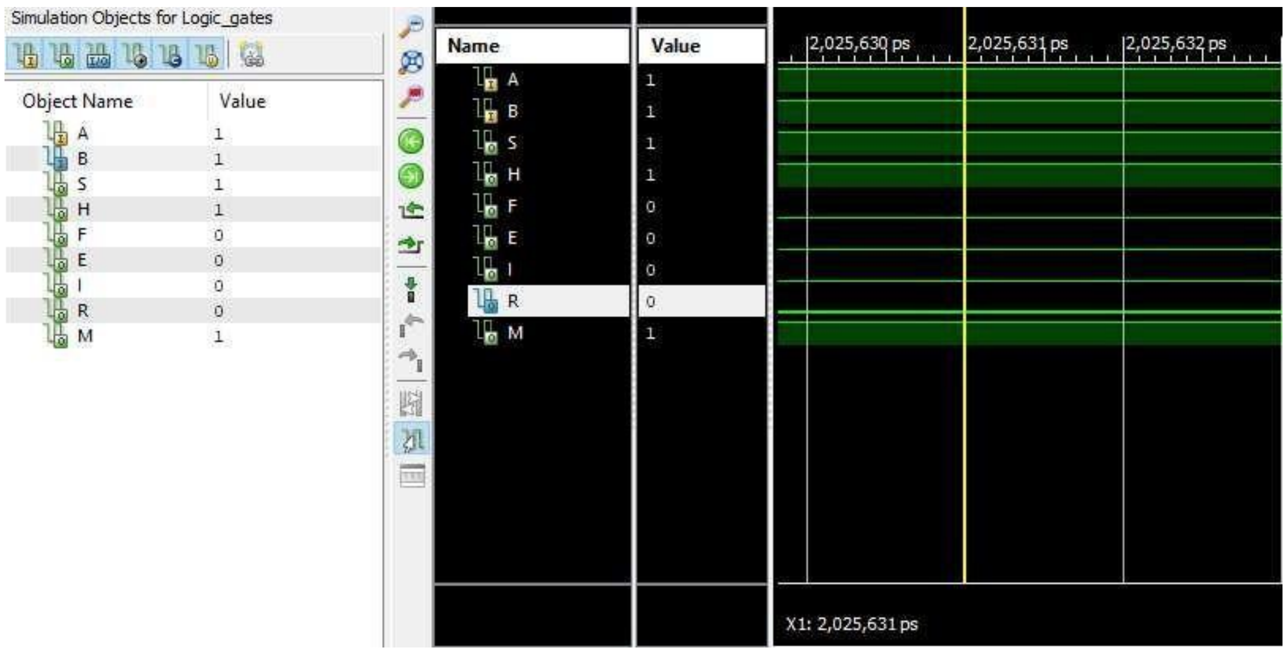
DEMUX RTL Schematic Screenshot:



DEMux Technology Schematic:



5 Waveform:


Simulation Objects for Logic_gates

Object Name	Value
A	1
B	1
S	1
H	1
F	0
E	0
I	0
R	0
M	1

Name	Value
A	1
B	1
S	1
H	1
F	0
E	0
I	0
R	0
M	1

6 Result:

Thus, an AND, OR, NOR, NAND, XOR and XNOR, gate is designed with inputs using Behavioral modeling and the output is compared with the theoretical values using the truth table.

7 Learning Outcomes:

1. We have learnt how to use Verilog.
2. We have learnt how to write the code for a 4:1 MUX/ DEMUX.
3. We have also learnt how to check its RTL, Technology schematic, how to force constant while simulation**.**

Experiment No.3

Student Name: Priyanshu Mathur **UID:** 20BEC1073
Branch: Electronics and Communication **Section/Group:** A
Semester: 6th **Date of Performance:** 10/3/2023
Subject Name: Digital System Design Using Verilog **Subject Code:** 20ECP-373

1. Aim: Write an HDL code for Encoder and Decoder **2.**
Software used: Xilinx 12.4 ISE Design suite 14.7

3. Modeling style: Data flow model.

4. Theory:

Encoder: An encoder is a combinational circuit. It has 2^n input lines and n output lines. It takes up these 2^n input data and encodes them into n-bit data. And it produces the binary code equivalent of the input line, which is active high.

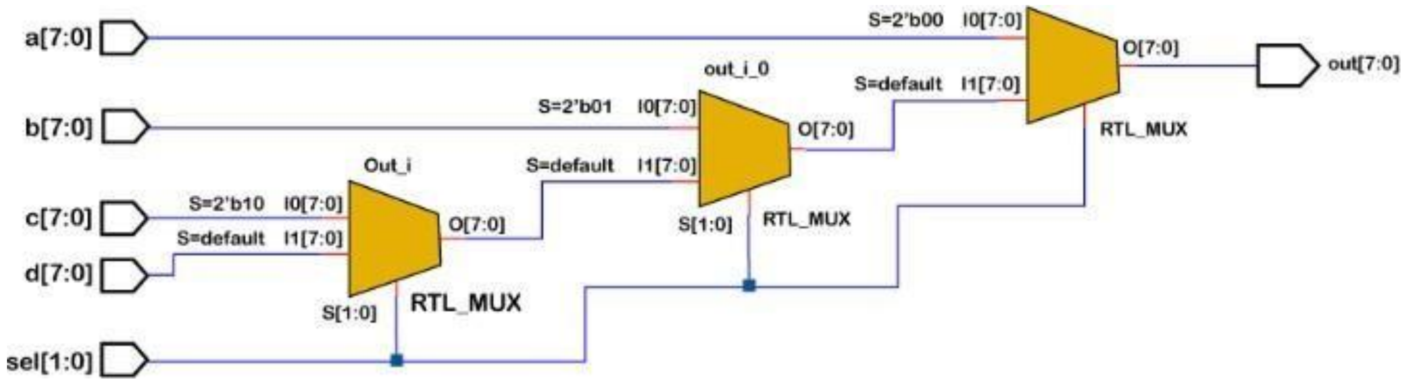


Figure: Hardware Schematic

Decoder: The combinational circuit that change the binary information into 2^N output lines is known as Decoders. The binary information is passed in the form of N input lines. The output lines define the 2^Nbit code for the binary information. In simple words, the Decoder performs the reverse operation of the Encoder**.** At a time, only one input line is activated for simplicity. The produced 2^N-bit output code is equivalent to the binary information.

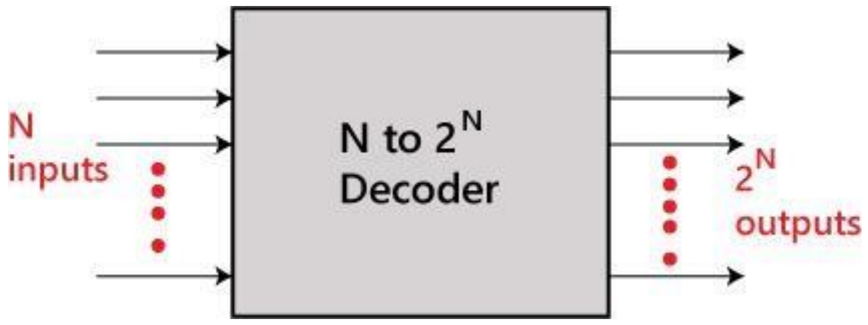


Figure: Decoder block diagram

4 Verilog CODE:

- **For Encoder**

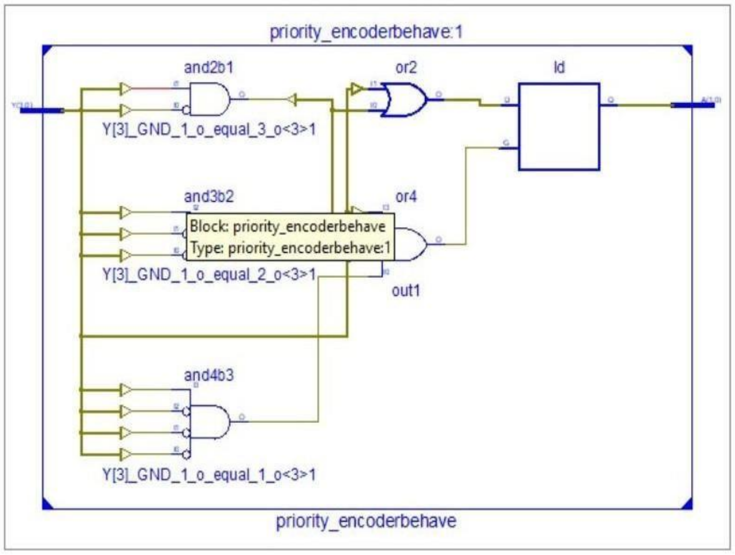
```
module priority_encoderbehave(A,
Y); input [3: 0] Y; output reg [1: 0] A;
always@(Y) begin casex(Y) 4'b0001:
A = 2'b00;
4'b001x: A = 2'b01;
4'b01xx: A = 2'b10;
4'b1xxx: A = 2'b11; default:
$display("Error!");endcase
end endmodule
```

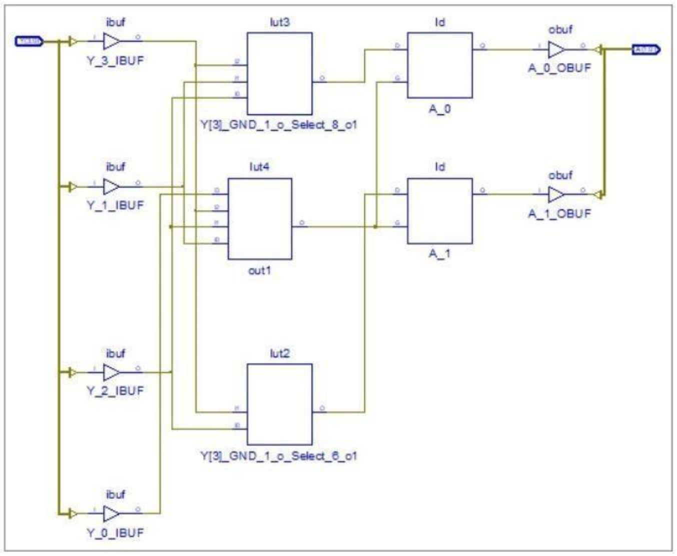
- **For Decoder**

```
module decoder24_behaviour(en,a,b,y);
  // input port
  input en,a,b;

  // use reg to store the output
  value output reg [3:0]y; // always
  is used in design block // only in
  Behavioural modeling.

  always @(en,a,b)
    begin
      // using condition if statement
      // implement the 2:4 truth table
      if(en==0)
        begin
          if(a==1'b0 & b==1'b0) y=4'b1110;
          else if(a==1'b0 & b==1'b1)
          y=4'b1101; else if(a==1'b1 &
          b==1'b0) y=4'b1011; else if(a==1 &
          b==1) y=4'b0111; else y=4'bxxxx;
        end else
    y=4'b1111;
    end
    endmodule
```

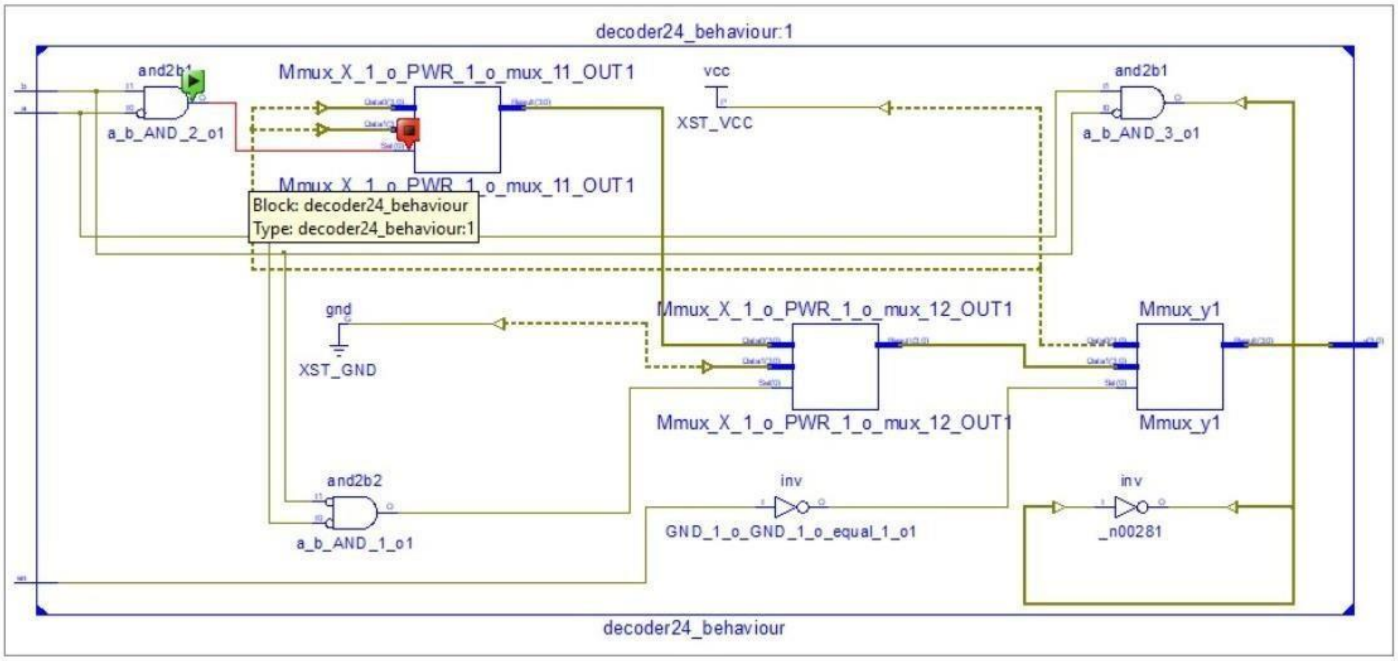
**5. Schematic Diagrams **

Encoder:


priority_encoderbehave:1
and2b1
or2
ld
Y[3]_GND_1_o_equal_3_o<3>1
and3b2
or4
Block: priority_encoderbehave
Type: priority_encoderbehave:1
Y[3]_GND_1_o_equal_2_o<3>1
out1
and4b3
Y[3]_GND_1_o_equal_1_o<3>1
priority_encoderbehave

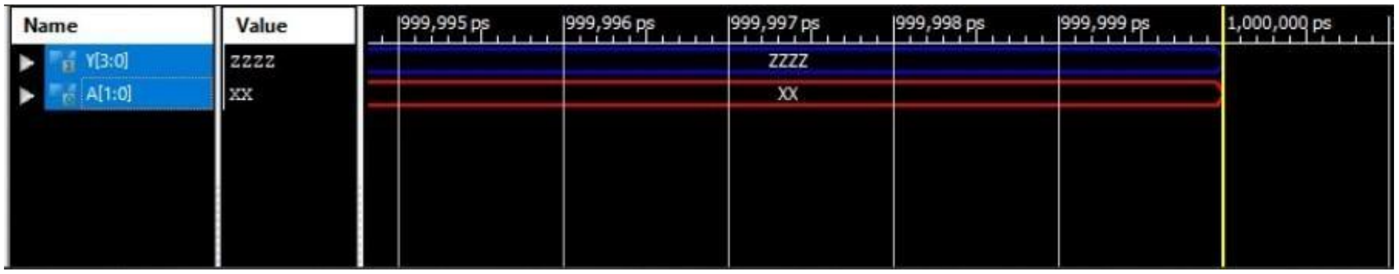


 Decoder

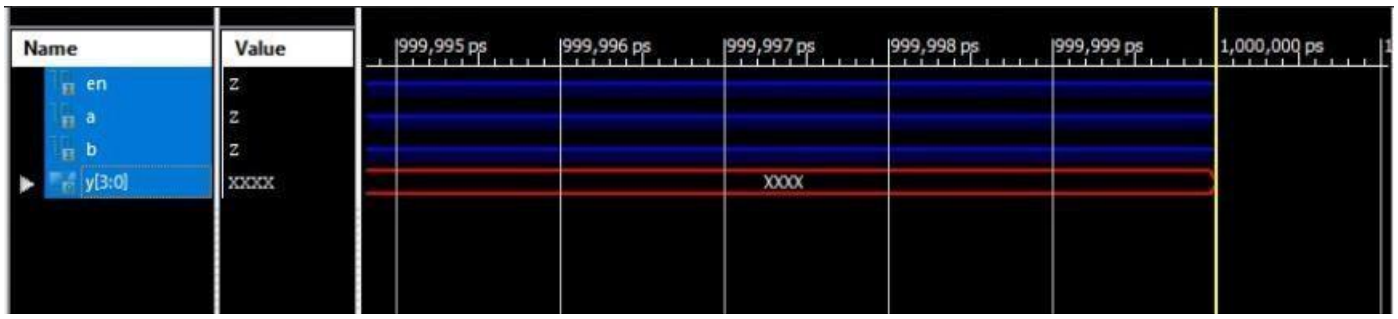

decoder24_behaviour:1
and2b1
a_b_AND_2_o1
Mmux_X_1_o_PWR_1_o_mux_11_OUT1
Mmux_X_1_o_PWR_1_o_mux_11_OUT1
Block: decoder24_behaviour
Type: decoder24_behaviour:1
vcc
XST_VCC
and2b1
a_b_AND_3_o1
gnd
XST_GND
Mmux_X_1_o_PWR_1_o_mux_12_OUT1
Mmux_X_1_o_PWR_1_o_mux_12_OUT1
Mmux_y1
Mmux_y1
and2b2
a_b_AND_1_o1
inv
GND_1_o_GND_1_o_equal_1_o1
inv
_n00281
decoder24_behaviour

6. Waveform:

 Encoder



- **Decoder**



5 Result:

Thus, an encoder and decoder are designed with inputs using Behavioral modeling and output is compared with the theoretical values using the truth table.

6 Learning Outcomes:

1. We have learnt how to use Verilog and its basic to intermediate functionalities.
2. We have learnt how to write the code for an encoder/ decoder
3. We have also learnt how to check its RTL, Technology schematic, how to force constant while simulation**.**

Experiment No. 4

Student Name: Priyanshu Mathur **UID:** 20BEC1073
Branch: Electronics and Communication **Section/Group:** A
Semester: 6th **Date of Performance:** 13/3/2023
Subject Name: Digital System Design Using Verilog **Subject Code:** 20ECP-373

1. **Aim:** Write an HDL code for Half and Full Adder **2.** **Software used:** Xilinx 12.4 ISE Design suite 14.7

3. **Modeling style:** Data flow model.

4. **Theory:**

Half Adder: A half adder is a digital logic circuit that performs binary addition of two single-bit binary numbers. It has two inputs, A and B, and two outputs, SUM and CARRY. The SUM output is the least significant bit (LSB) of the result, while the CARRY output is the most significant bit (MSB) of the result, indicating whether there was a carry-over from the addition of the two inputs. The half adder can be implemented using basic gates such as XOR and AND gates.

Full Adder: Full Adder is the adder that adds three inputs and produces two outputs. The first two inputs are A and B and the third input is an input carry as C-IN. The output carry is designated as C-OUT and the normal output is designated as S which is SUM. A full adder logic is designed in such a manner that can take eight inputs together to create a byte-wide adder and cascade the carry bit from one adder to another. we use a full adder because when a carry-in bit is available, another 1-bit adder must be used since a 1-bit half-adder does not take a carry-in bit. A 1-bit full adder adds three operands and generates 2-bit results.

4 Verilog CODE:

- **For Encoder**

```
module HalfAdder(a,b,sum,carry);
input a,b; output
sum,carry;
xor(sum,a,b);
and(carry,a,b);
endmodule
```

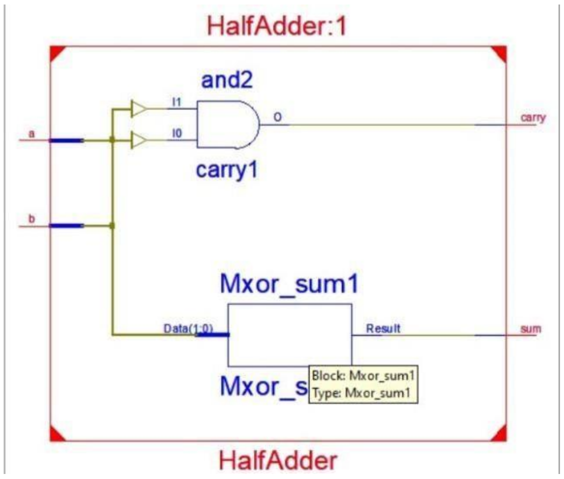
- **For Decoder**

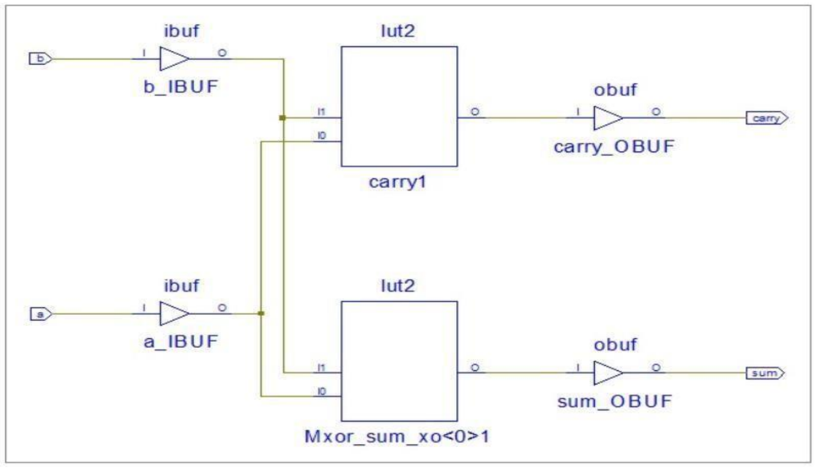
```
module full_add(a,b,cin,sum,cout);
  input a,b,cin;
  output sum,cout;
  wire x,y,z;
// instantiate building blocks of full adder
```

```
  half_add h1(.a(a),.b(b),.s(x),.c(y));
  half_add h2(.a(x),.b(cin),.s(sum),.c(z));
  or o1(cout,y,z);
endmodule : full_add // code
your half adder design
module half_add(a,b,s,c);
input a,b;
  output s,c;
// gate level design of half adder
  xor x1(s,a,b); and a1(c,a,b);
endmodule :half_add
```

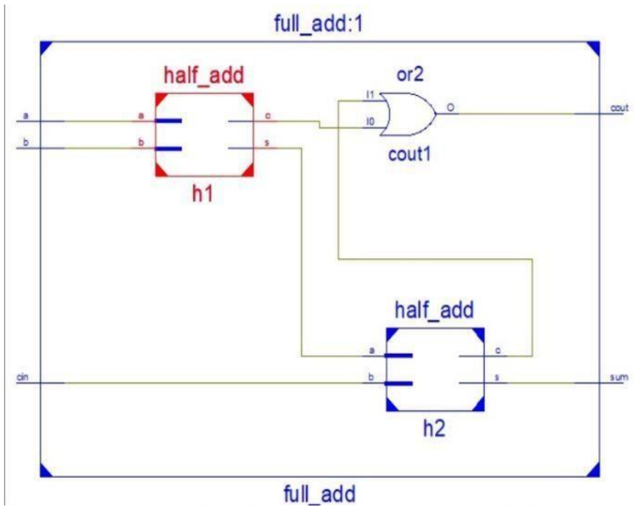
**5. Schematic Diagrams **

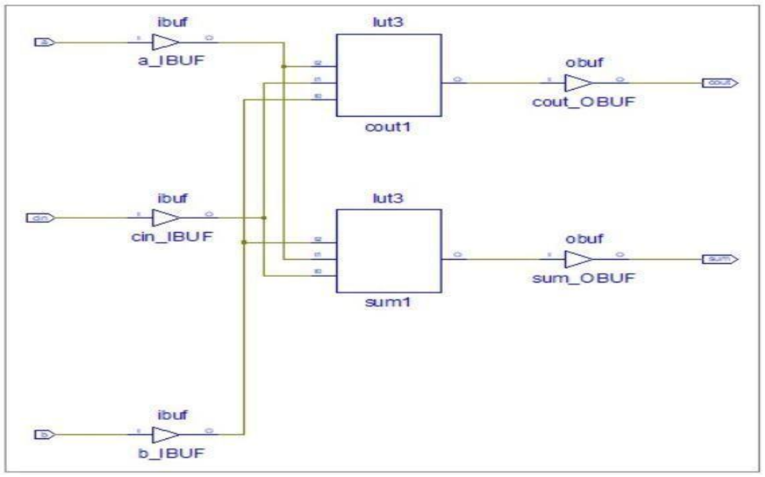
Half Adder:





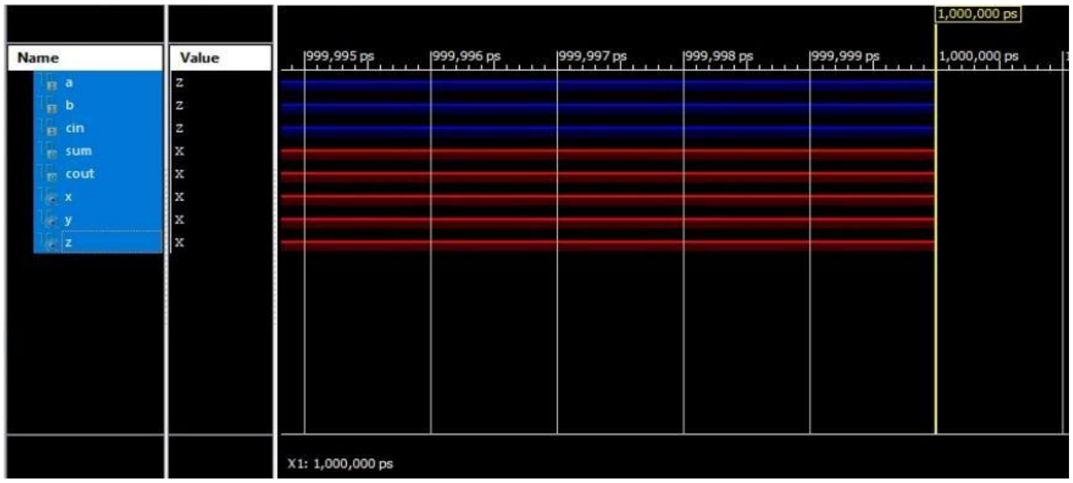
** Full Adder:**



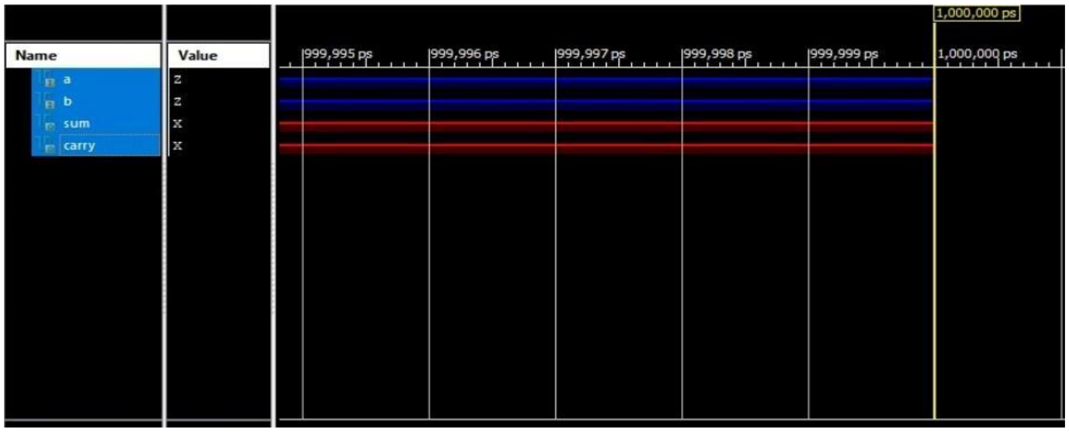


5 Waveform:

- **Half Adder**



- **Full Adder**



6 Result

Thus, ahalfandfulladderare designed with inputs using Behavioral modeling and output is compared with the theoretical values using the truth table.

7 Learning Outcomes:

1. We have learnt how to use Verilog and its basic to intermediate functionalities.
2. We have learnt how to write the code for a half and full adder
3. We have also learnt how to check its RTL, Technology schematic, how to force constant while simulation**.**

Experiment No. 5

Student Name: Priyanshu Mathur **UID:** 20BEC1073
Branch: Electronics and Communication **Section/Group:** A
Semester: 6th **Date of Performance:** 20/3/2023
Subject Name: Digital System Design Using Verilog **Subject Code:** 20ECP-373

1. **Aim:** Write an HDL code for Half and Full Subtractor

2. **Software used:** Xilinx 12.4 ISE Design suite 14.7

3. **Modeling style:** Data flow model.

4. **Theory:**

Half Subtractor: A half-subtractor is a combinational logic circuit that have two inputs and two outputs (i.e. difference and borrow). The half subtractor produces the difference between the two binary bits at the input and also produces a borrow output (if any).

Full Subtractor: A full-subtractor is a combinational circuit that has three inputs A, B, bin and two outputs d and b. Where, A is the minuend, B is subtrahend, b_{in} is borrow produced by the previous stage, d is the difference output and b is the borrow output.

4 Verilog CODE:

- **For Half Subtractor** module

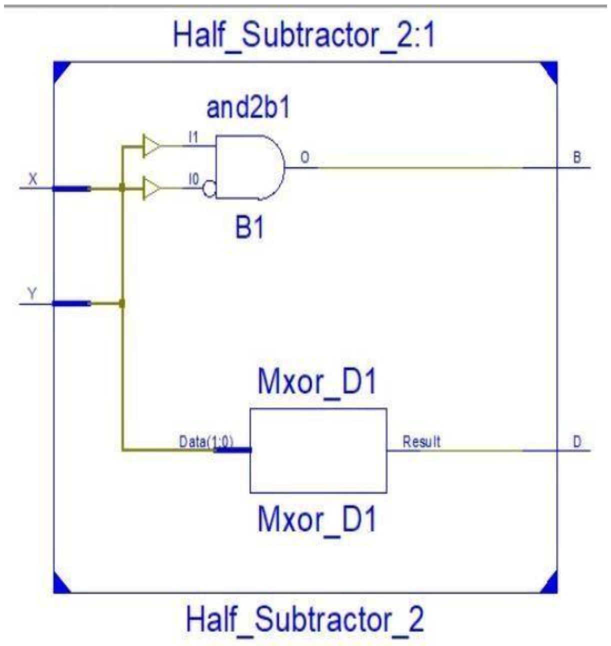
```
Half_Subtractor_2(output D, B, input X, Y);
assign D = X ^ Y;
assign B = ~X & Y;
endmodule
```

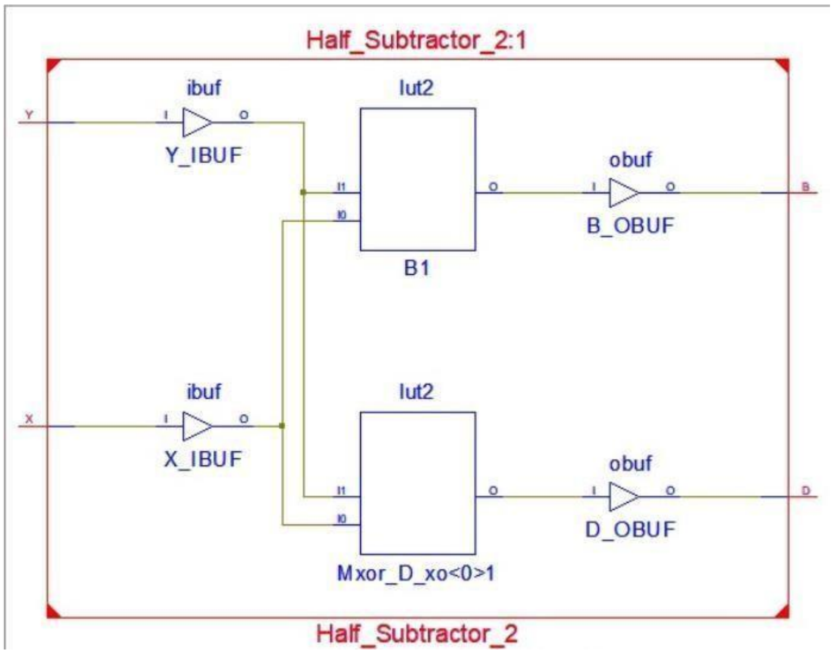
- **For Full subtractor** module

```
full_subtractor(input a, b, Bin, output D, Bout);
  assign D = a ^ b ^ Bin; assign Bout = (~a
& b) | (~(a ^ b) & Bin); endmodule
```

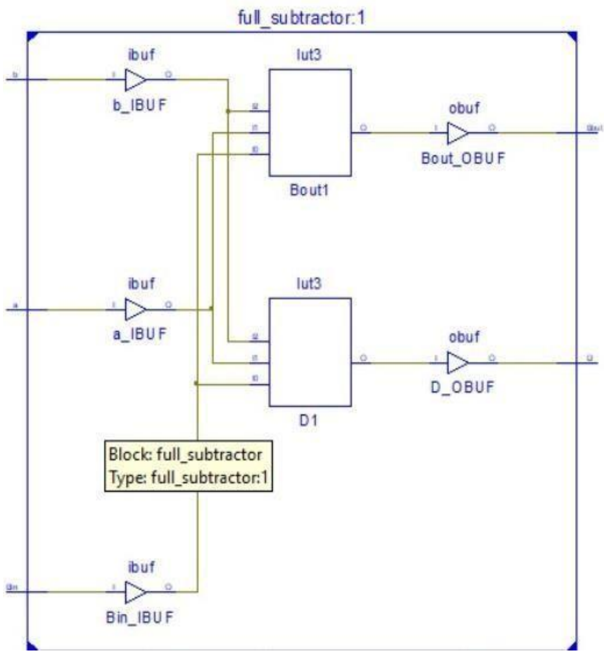
5. Schematic Diagrams

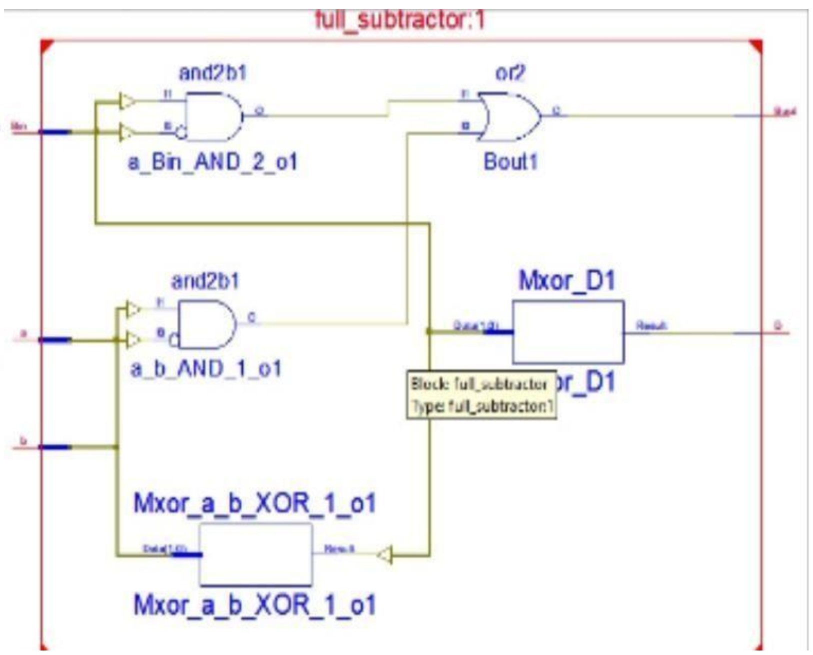
- **Half Subtractor:**





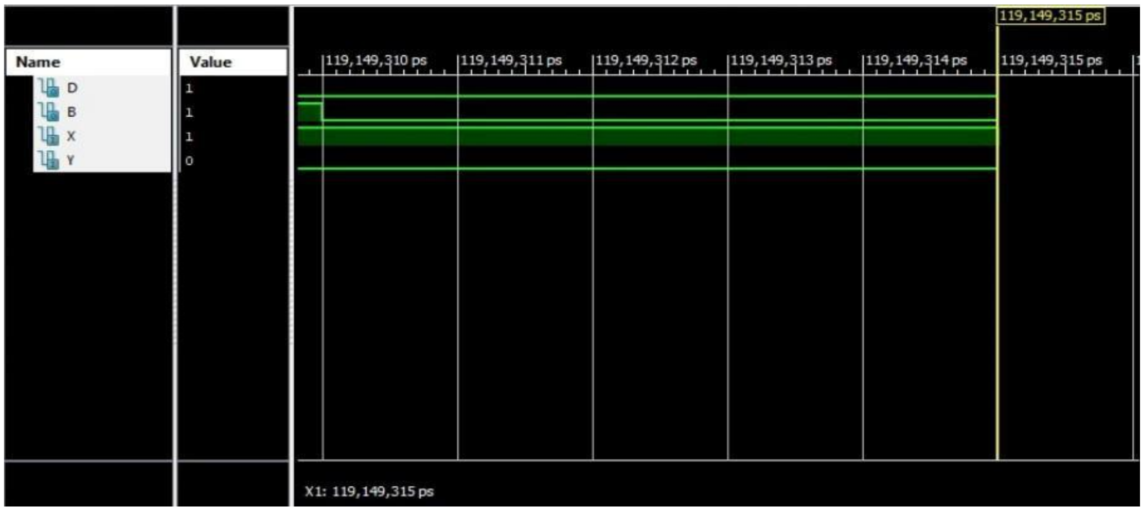
- **Full Subtractor:**



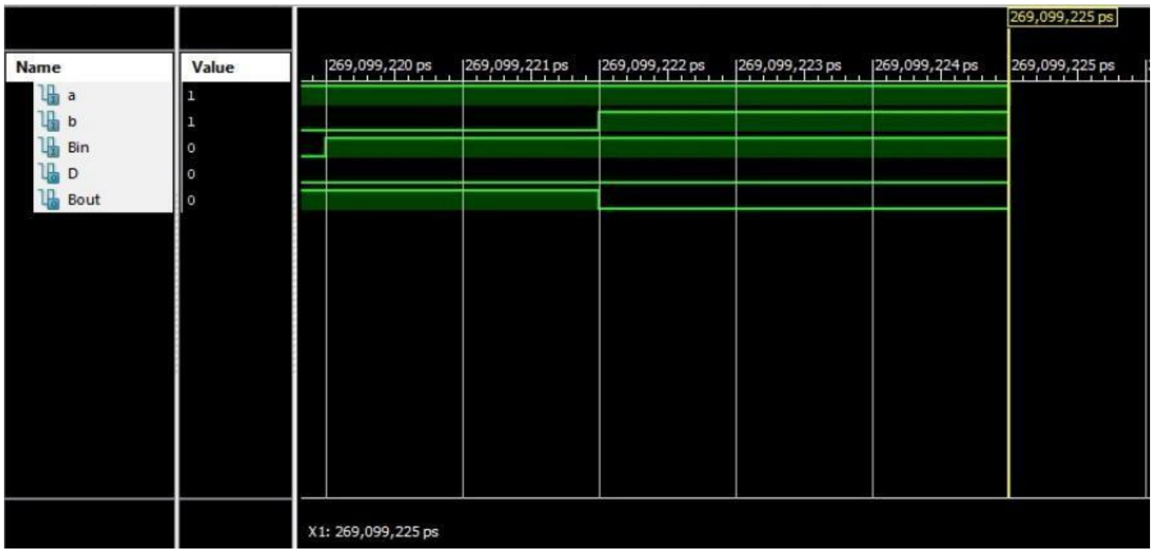


5 Waveform:

- **Half Subtractor**



- **Full Subtractor**



6 Result

Thus, a halfand fullsubtractorare designed with inputs using Behavioral modeling and output is compared with the theoretical values using the truth table.

7 Learning Outcomes:

1. We have learnt how to use Verilog and its basic to intermediate functionalities.
2. We have learnt how to write the code for a half and full subtractor
3. We have also learnt how to check its RTL, Technology schematic, how to force constant while simulation.

Experiment No. 6

Student Name: Priyanshu Mathur **UID:** 20BEC1073
Branch: Electronics and Communication **Section/Group:** A
Semester: 6th **Date of Performance:** 20/3/2023
Subject Name: Digital System Design Using Verilog **Subject Code:** 20ECP-373

1. **AIM:** Write an HDL code for D-FF, T-FF, JK-FF **2.**

SOFTWARE USED: Xilinx 12.4 ISE Design suite 14.7

3. **MODELING STYLE:** Data flow model.

4. **THEORY:**

D-flip flop: A D (or Delay) Flip Flop (Figure 1) is a digital electronic circuit used to delay the change of state of its output signal (Q) until the next rising edge of a clock timing input signal occurs.

T-flip flop: T Flip-Flop is single input logic circuit that holds or toggles its output according to the input state. Toggling means changing the next state output to complement the current state. T is an abbreviation for Toggle.

JK-flip flop: The JK flip flop is one of the most used flip flops in digital circuits. The JK flip flop is a universal flip flop having two inputs 'J' and 'K'. In SR flip flop, the 'S' and 'R' are the shortened abbreviated letters for Set and Reset, but J and K are not. The J and K are themselves autonomous letters which are chosen to distinguish the flip flop design from other types.

4 VERILOG CODE:

- **D flip flop with ASYNC active low range**

```
module dff (input d,
        input     rstn,
        input     clk,
        output    reg
        q);
 always @ (posedge clk or negedge rstn)
    if (!rstn) q <= 0;
    else       q    <=    d;
endmodule
```

- **D flip flop with SYNC active low reset** module

```
dff (input d,
        input     rstn,
        input     clk,
        output    reg
        q);
 always @ (posedge clk)
```

```
        if (!rstn)    q <= 0;
        else q <= d;
endmodule
```

- **T flip flop**

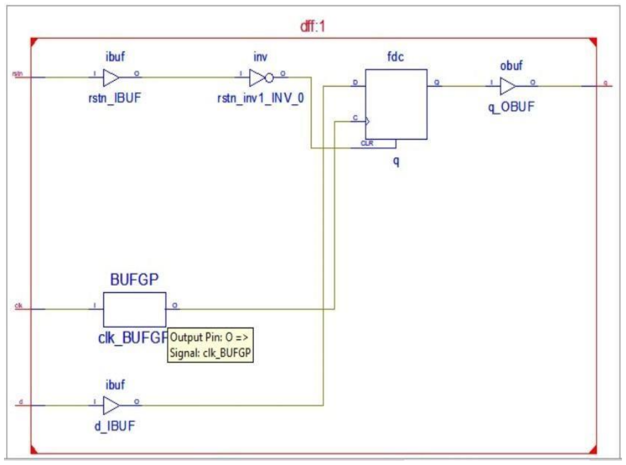
```
module tff ( input clk, input rstn, input t, output reg q);
  always @ (posedge clk) begin
    if (!rstn) q <= 0; else
        if (t) q <= ~q; else
        q <= q;
  end endmodule
```

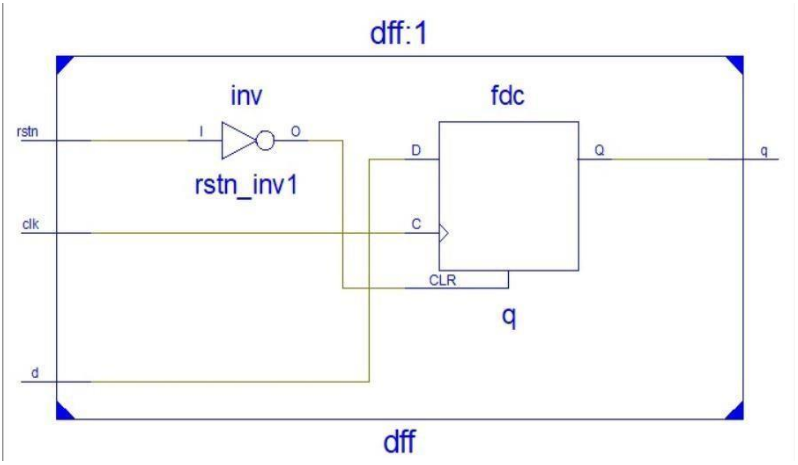
- **JK flip flop**

```
module jk_ff ( input j, input k, input clk, output q);
   reg     q;     always     @
   (posedge clk)
      case ({j,k})
         2'b00 : q <= q;
         2'b01 : q <= 0;
         2'b10 : q <= 1;
         2'b11 : q <= ~q; endcase
endmodule
```

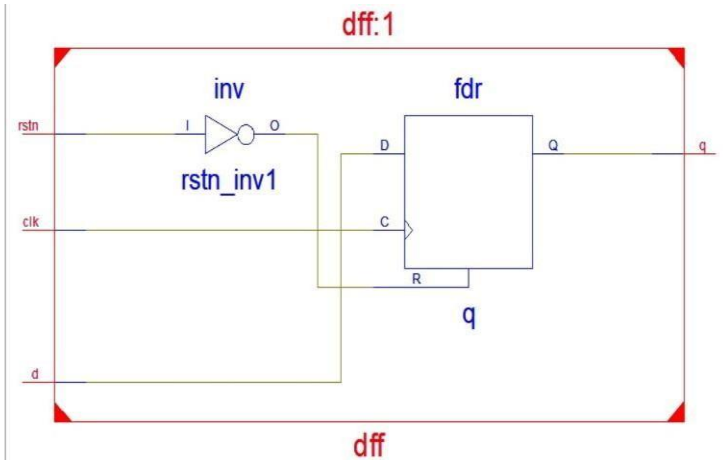
5. SCHEMATIC DIAGRAMS

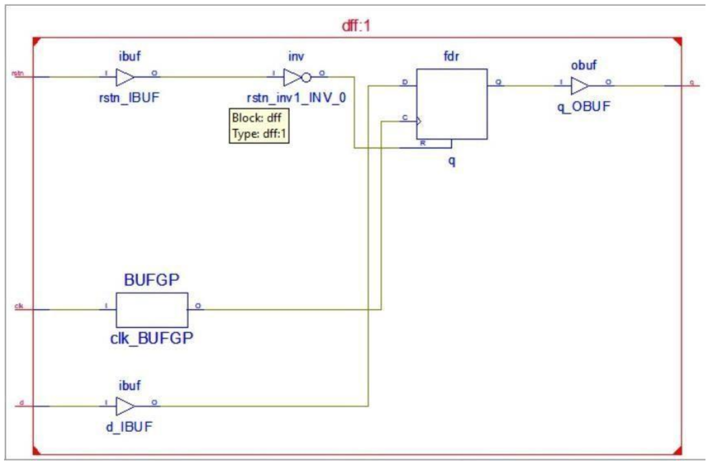
- **D flip flop with ASYNC active low range:**



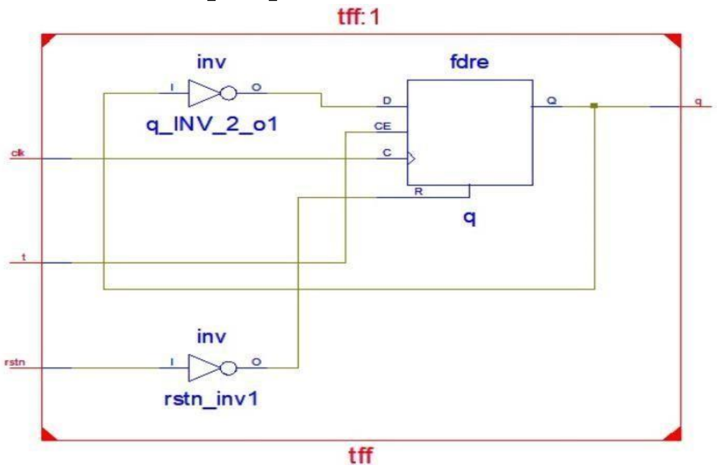


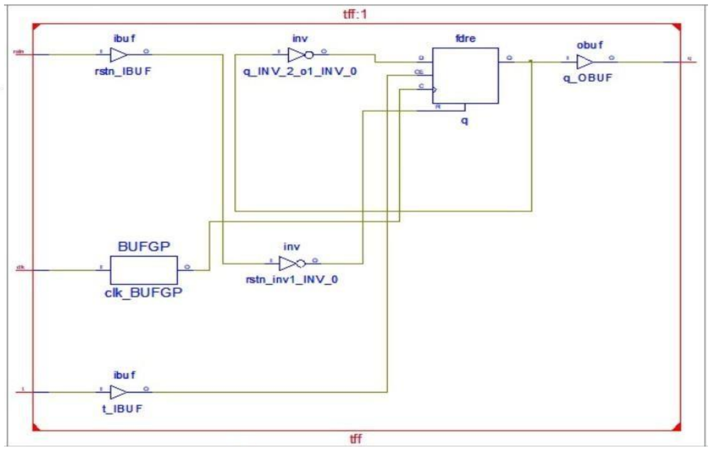
- **D flip flop SYNC with active low reset:**



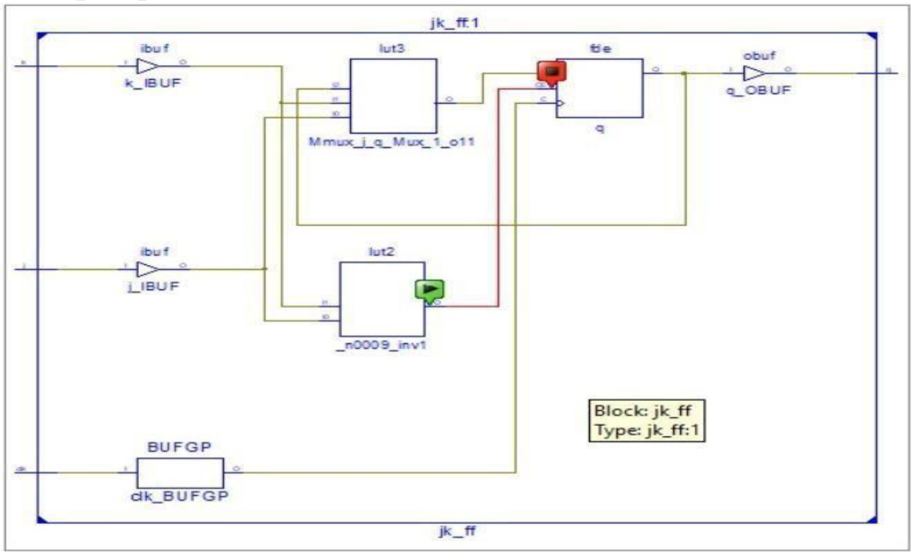


- **T flip flop**



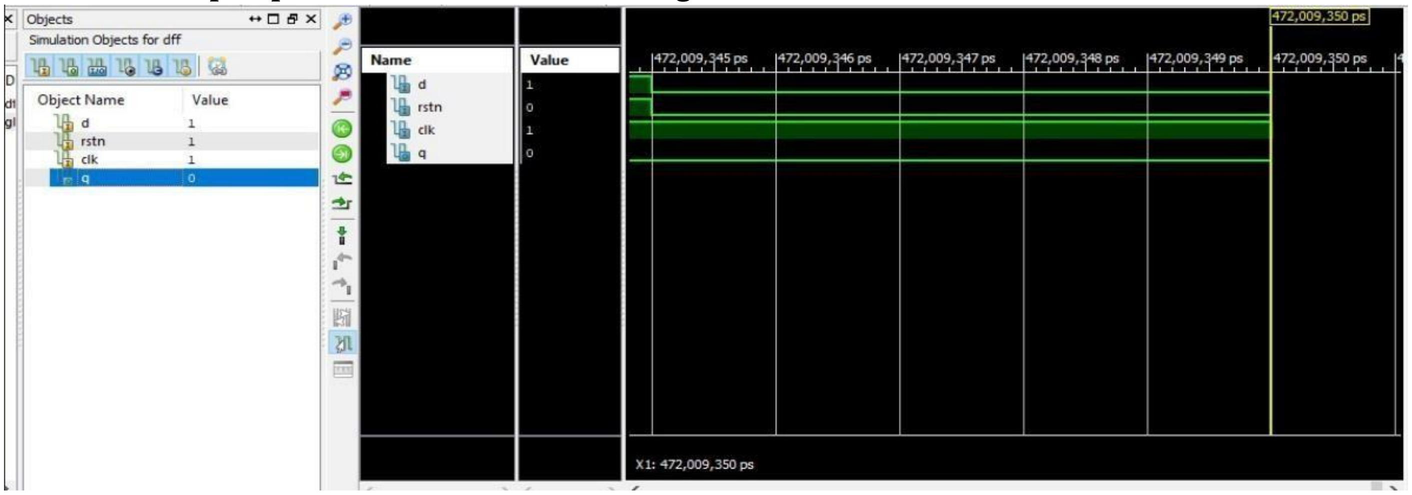


- **JK flip flop**



5 Waveform:

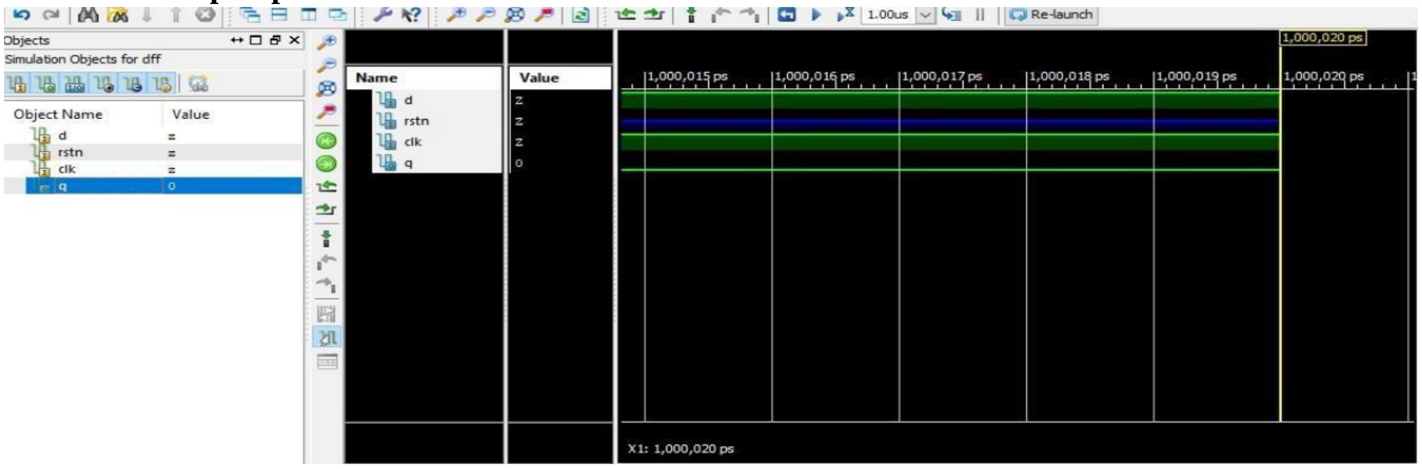
- **D flip flop with ASYNC active low range:**


Objects
Simulation Objects for dff

Object Name	Value
d	1
rstn	1
clk	1
q	0

Name	Value
d	1
rstn	0
clk	1
q	0

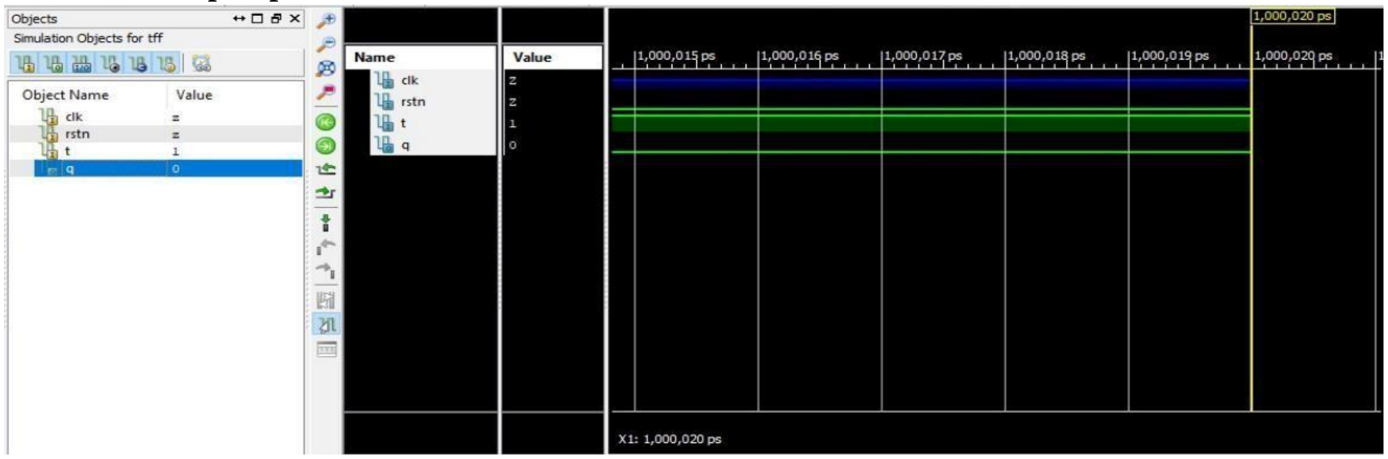
- **D flip flop with SYNC active low reset**


Objects
Simulation Objects for dff

Object Name	Value
d	z
rstn	z
clk	z
q	0

Name	Value
d	z
rstn	z
clk	z
q	0

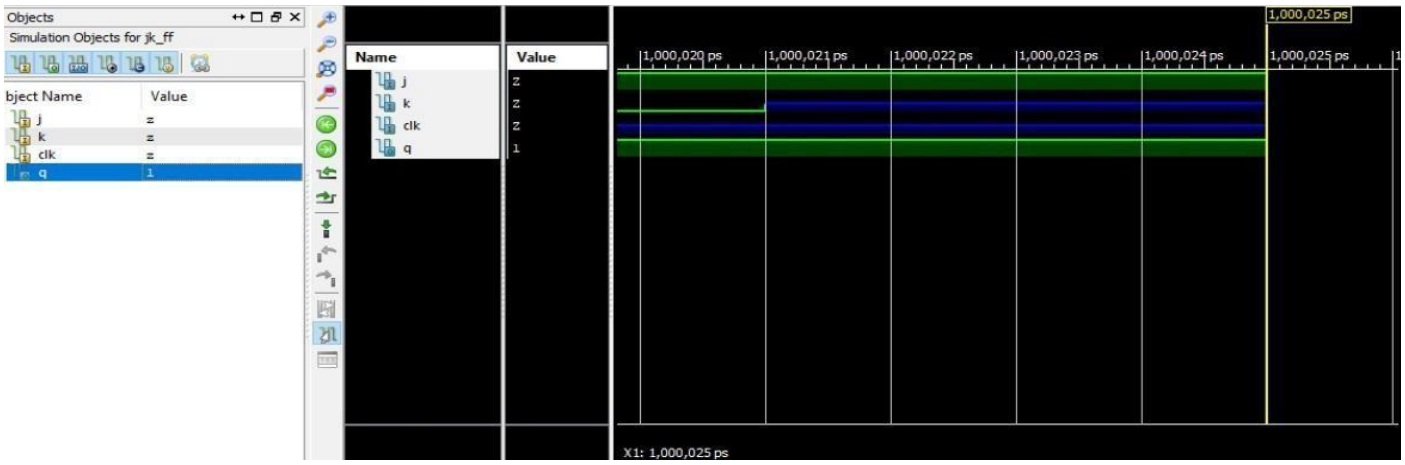
- **T flip flop**


Objects
Simulation Objects for tff

Object Name	Value
clk	z
rstn	z
t	1
q	0

Name	Value
clk	z
rstn	z
t	1
q	0

- **JK flip flop**



bject Name	Value
j	z
k	z
clk	z
q	1

Name	Value
j	z
k	z
clk	z
q	1

6 Result

Thus, thevariousflipflopssuchasd flipflop,t flipflopandjk flipflopare designed with inputs using Behavioral modeling and output is compared with the theoretical values using the truth table.

7 Learning Outcomes:

1. We have learnt how to use Verilog and its basic to intermediate functionalities.
2. We have learnt how to write the code for a d,t, jk flip flops
3. We have also learnt how to check its RTL, Technology schematic, how to force constant while simulation**.**

Experiment No.1

Student Name: Priyanshu Mathur UID: 20BEC1073

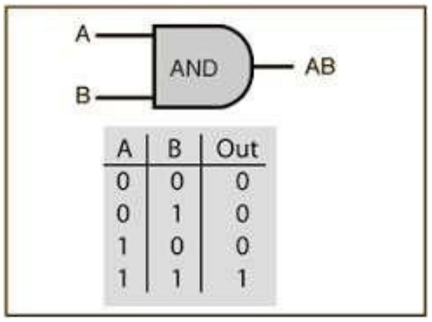
Branch: Electronics and Communication Section/Group: A

Semester: 6th Date of Performance: 16/2/2023

Subject Name: Digital System Design Using VERILOG Subject Code: 20ECP-373

1. Aim : Write an HDL code for Basic Logic Gates.
2. Software used :Xilinx 12.4 ISE Design suite 14.7
3. Modeling style : Data flow model.
4. Theory :

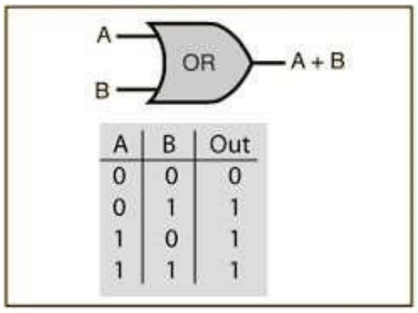
4.1AND GATE: An AND gate is an electrical circuit that combines two signals so that the output is on if both signals are present. The output of the AND gate is connected to a base driver which is coupled to the bases of transistors, and alternately switches the transistors at opposite corners of the inverter.



A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

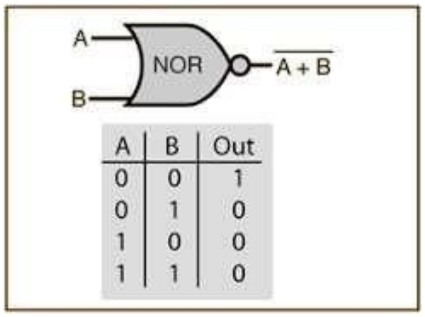
4.2OR GATE: The output of an OR gate is true when one or more of its inputs are true. If all of an OR gate's inputs are false, then the output of the OR gate is false.

. The truth table for an OR gate with two inputs is given below.



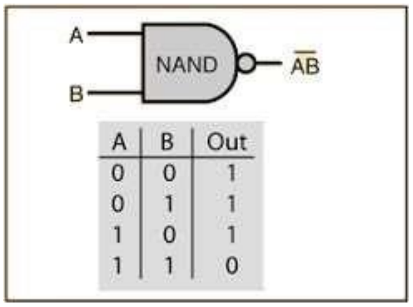
A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

4.3NOR GATE: The NOR gate is a logic gate that produces a high output (1) only if all its inputs are false, and low output (0) otherwise. Hence the NOR gate is the inverse of and or gate, and its circuit is produced by connecting an OR gate to a NOT gate.



A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

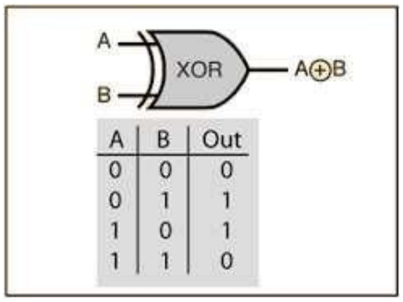
4.4NAND GATE: The NAND gate is a basic digital logic gate that implements logical conjunction it behaves according to the truth table. A LOW output (0) results only if both the inputs to the NAND gate are HIGH (1). If neither or only one input to the AND gate is HIGH, a HIGH output results. The NAND gate is significant because any Boolean function can be implemented by using a combination of NAND gates. This property is called functional completeness.



A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

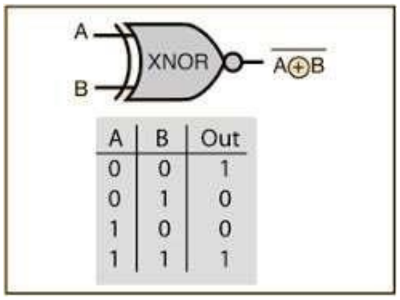
4.5XOR GATE: The XOR gate is a digital logic gate that implements an exclusive or; that is, a true output (1/ HIGH) results if one, and only one, of the inputs to the gate is true. If both inputs are false (0/LOW) and both are true, a false output results. XOR represents the inequality function, i.e., the output is true if the inputs are not alike otherwise the output is false.

A way to remember XOR is "one or the other but not both”. XOR can also be viewed as addition modulo 2. As a result, XOR gates are used to implement binary addition in computers.



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

4.6 XNOR GATE: The XNOR gate (sometimes spelled "Exnor" or "enor" and rarely written NXOR) is a digital logic gate whose function is the logical complement of the exclusive OR (XOR) gate. The twoinput version implements logical equality, behaving according to the truth table to the right. A HIGH output (1) results if both of the inputs to the gate are the same. If one but not both inputs are HIGH (1), a LOW output (0) results.



A	B	Out
0	0	1
0	1	0
1	0	0
1	1	1

5. Verilog CODE:

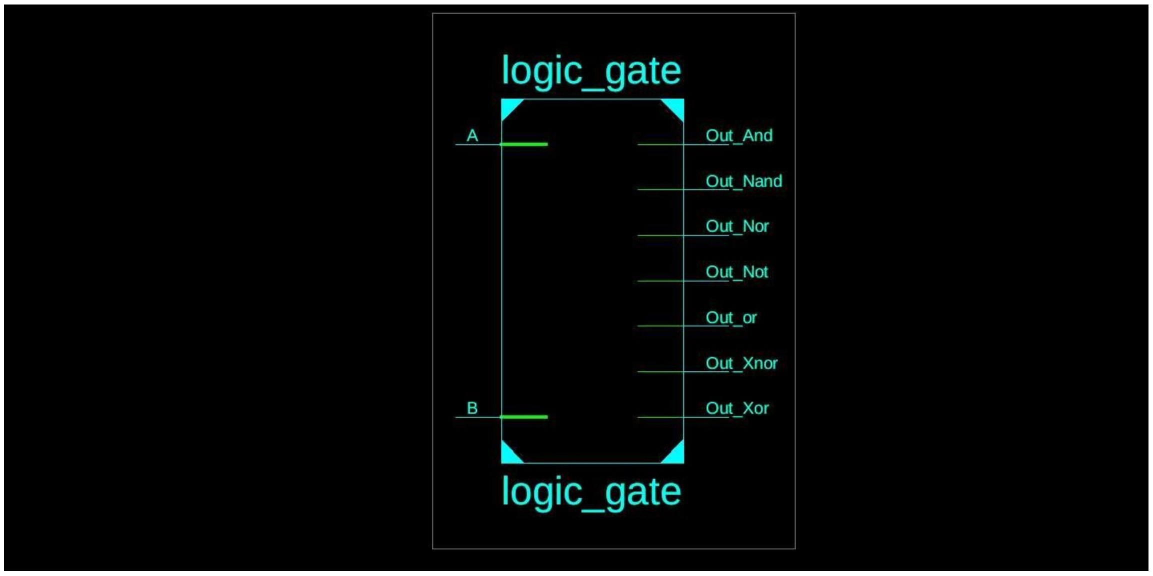
```
module logic_gate(output Out_And,output Out_or,output Out_Not,output Out_Nand,  output
                     Out_Nor,output Out_Xor,output Out_Xnor,input A, input B );

                and(Out_And,A,B);  //And gate
                or(Out_or,A,B);              //or     gate
               not(Out_Not,A);               //Not    gate
               nand(Out_Nand,A,B);      //Nand gate
               nor(Out_Nor,A,B);             //Nor    gate
               xor(Out_Xor,A,B);             //Xor    gate
               xnor(Out_Xnor,A,B);      //Xnor gate

endmodule
```

6. Test Bench Code:

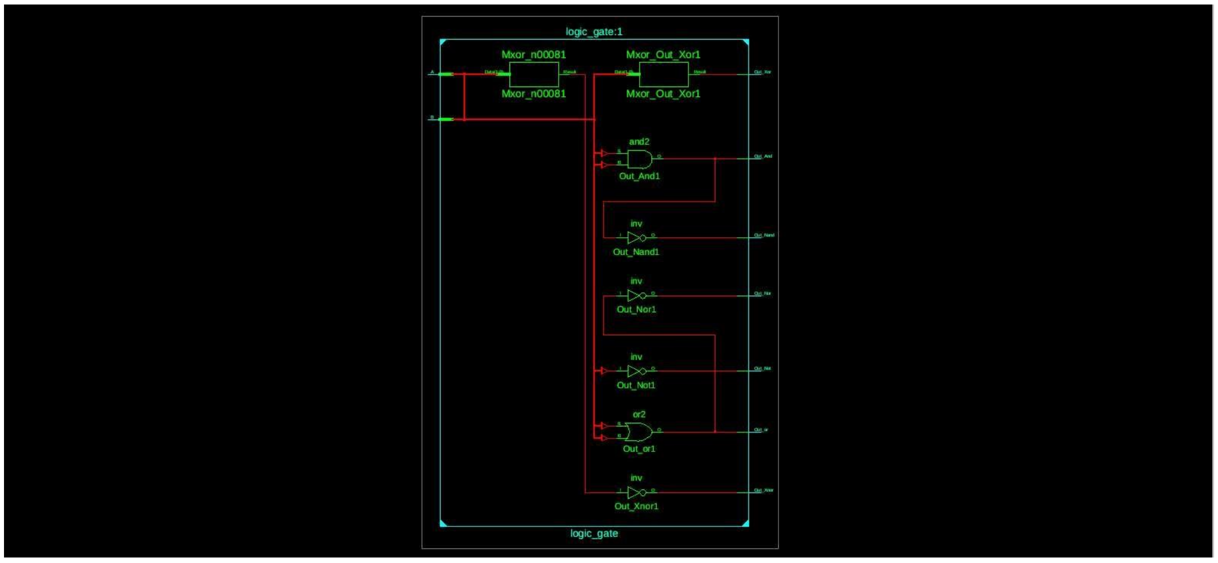
```
        module test_logic_gate;

                // Inputs
                reg A;
                reg B;

                // Outputs
                wire Out_And;          //AND OUTPUT
                wire Out_or;        //OR OUTPUT wire
                Out_Not;        //NOT OUTPUT wire
                Out_Nand;       //NAND OUTPUT wire
                Out_Nor;        //NOR OUTPUT wire
```

```
            Out_Xor;        //XOR OUTPUT wire
            Out_Xnor;       //XNOR OUTPUT //
            Instantiate the Unit Under Test (UUT)
            logic_gate uut (.Out_And(Out_And),
            .Out_or(Out_or), .Out_Not(Out_Not),
                .Out_Nand(Out_Nand), .Out_Nor(Out_Nor), .Out_Xor(Out_Xor),
                .Out_Xnor(Out_Xnor),
                .A(A),
                .B(B)
            );
        initial begin
                // Initialize Inputs
                A = 0;
                B = 0;

                // Wait 100 ns for global reset to finish
                #100;

                // Add stimulus here
         A = 0; B = 0; #100;
             A = 0; B = 1; #200;
             A = 1; B = 0; #100;
             A = 1; B = 1; #200;
            end

    endodule
```

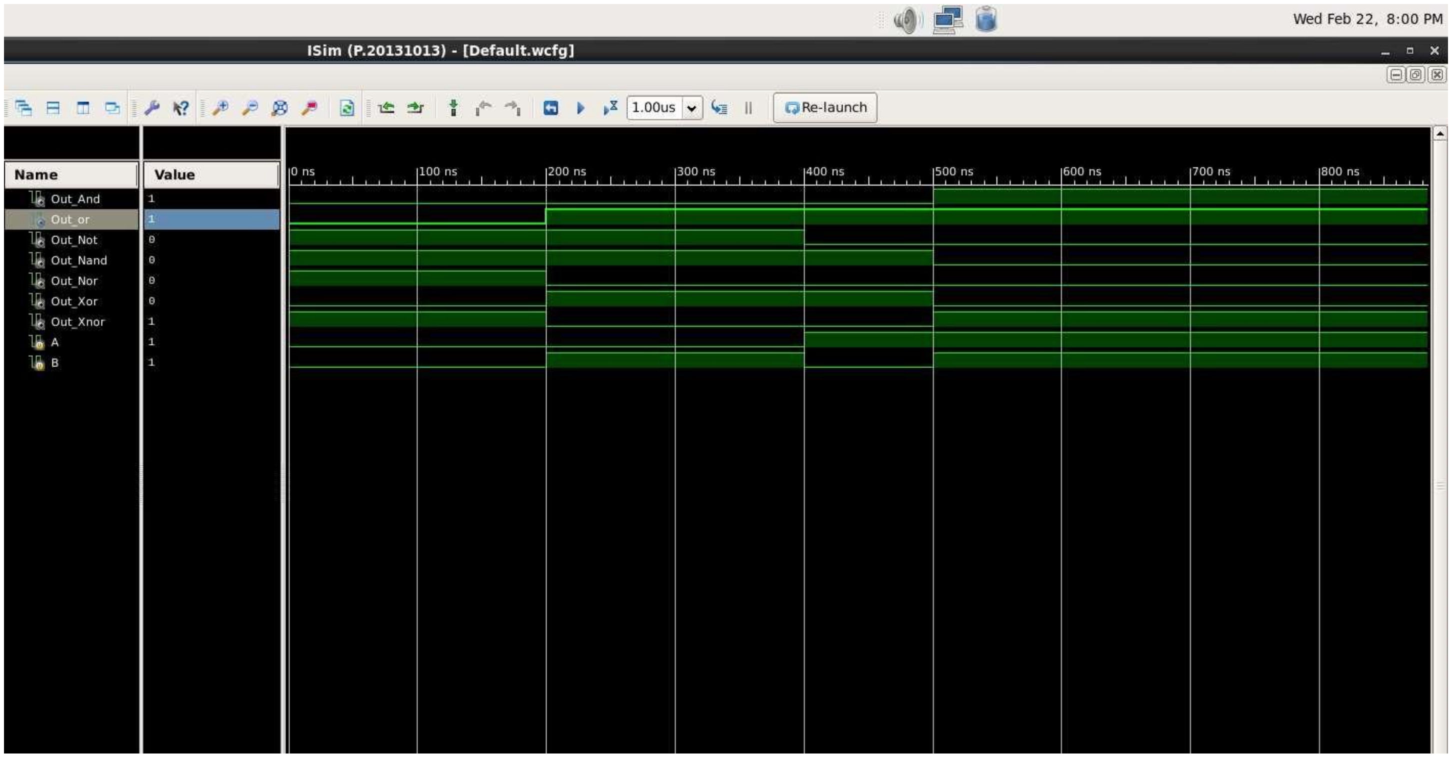
7. RTL Schematic Screenshot:



7. Technology Schematic:



9. Waveform:



Name	Value
Out_And	1
Out_or	1
Out_Not	0
Out_Nand	0
Out_Nor	0
Out_Xor	0
Out_Xnor	1
A	1
B	1

10. Result:

Thus, an AND, OR, NOR, NAND, XOR and XNOR, gate is designed with inputs using Behavioral modeling and the output is compared with the theoretical values using the truth table.

11.Learning Outcomes

1. We have learnt how to use Verilog.
2. We have learnt how to write the code for different logic gates.
3. We have also learnt how to check its RTL , Technology schematic, how to force constant while simulation.