EXP NO: 01

VERIFICATION OF LOGIC GATES

AIM:

To develop the source code for logic gates by using VHDL/VERILOG and obtain the simulation.

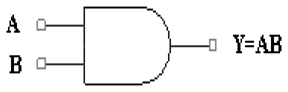
ALGORITM:

Step1: Define the specifications and initialize the design.
Step2: Declare the name of the entity and architecture by using VHDL source code.
Step3: Write the source code in VERILOG.
Step4: Check the syntax and debug the errors if found, obtain the synthesis report.
Step5: Verify the output by simulating the source code.
Step6: Write all possible combinations of input using the test bench.
Step7: Obtain the place and route report.

LOGIC DIAGRAM:

AND GATE:

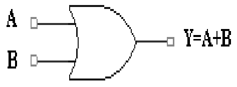
LOGIC DIAGRAM: TRUTH TABLE:



A	B	Y=AB
0	0	0
0	1	0
1	0	0
1	1	1

OR GATE:

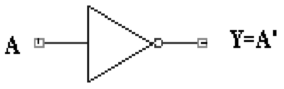
LOGICDIAGRAM TRUTH TABLE:



A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

NOT GATE:

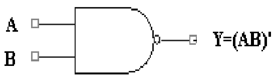
LOGIC DIAGRAM: TRUTH TABLE:



A	Y=A’
0	1
1	0

NAND GATE:

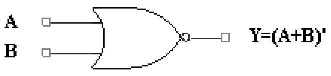
LOGICDIAGRAM TRUTH TABLE



A	B	Y=(AB)’
0	0	1
0	1	1
1	0	1
1	1	0

NOR GATE:

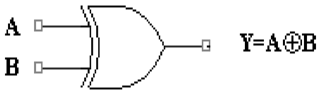
LOGIC DIAGRAM: TRUTH TABLE:



A	B	Y=(A+B)’
0	0	1
0	1	0
1	0	0
1	1	0

XOR GATE:

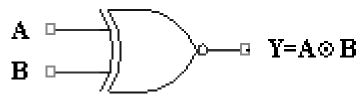
LOGICDIAGRAM TRUTH TABLE



A	B	Y=A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

XNOR GATE:

LOGIC DIAGRAM:



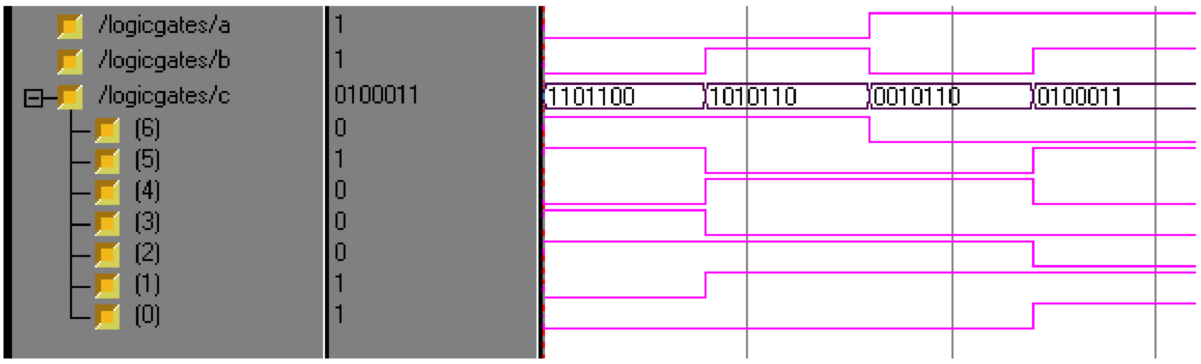
TRUTH TABLE:

A	B	Y=A⊙B
0	0	1
0	1	0
1	0	0
1	1	1

VERILOG SOURCE CODE:

```
module logicgates1(a, b, c);
   input a;
   input b;
   OUTPUT: [6:0] c;
          assign c[0]= a & b;
          assign c[1]= a | b;
          assign c[2]= ~(a & b);
          assign c[3]= ~(a | b);
          assign c[4]= a ^ b;
          assign c[5]= ~(a ^ b);
          assign c[6]= ~ a;
endmodule
```

Simulation output:



```
/logicgates/a   1
/logicgates/b   1
/logicgates/c   0100011     1101100  1010110  0010110  0100011
   (6)          0
   (5)          1
   (4)          0
   (3)          0
   (2)          0
   (1)          1
   (0)          1
```

==

RESULT:

Thus the outputs of Basic Logic Gates are verified by simulating and synthesizing the VERILOG code.

EXP NO: 02

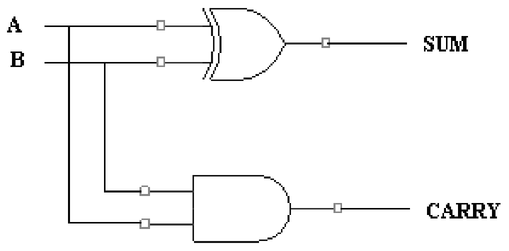
ADDERS AND SUBTRACTORS

AIM:
To develop the source code for adders and subtractors by using VERILOG and obtain the simulation.

ALGORITM:
Step1: Define the specifications and initialize the design.
Step2: Declare the name of the entity and architecture by using VHDL source code.
Step3: Write the source code in VERILOG.
Step4: Check the syntax and debug the errors if found, obtain the synthesis report.
Step5: Verify the output by simulating the source code.
Step6: Write all possible combinations of input using the test bench.
Step7: Obtain the place and route report.

BASIC ADDERS & SUBTRACTORS:

HALF ADDER:
LOGIC DIAGRAM:



TRUTH TABLE:

A	B	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

VERILOG SOURCE CODE:

Dataflow Modeling:
```
module ha_dataflow(a, b, s, ca);
   input a;
   input b;
   output s;
   output ca;
              assign#2 s=a^b;
              assign#2 ca=a&b;
endmodule
```

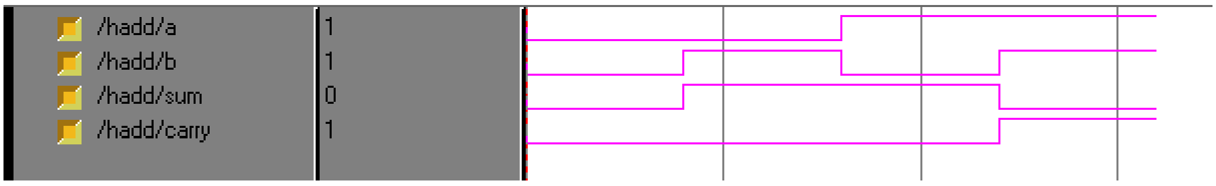
Behavioral Modeling:
```
module ha_behv(a, b, s, ca);
   input a;
   input b;
   output s;
   output ca;
              reg s,ca;
              always @ (a or b) begin
              s=a^b;
              ca=a&b;
              end
```

```
endmodule
```

Structural Modeling:

```
module ha_struct(a, b, s, ca);
   input a;
   input b;
   output s;
   output ca;
            xor
            x1(s,a,b);
            and
            a1(ca,a,b);
endmodule
```

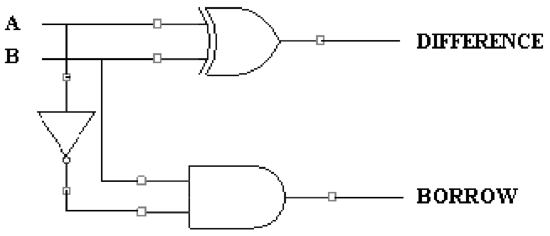
Simulation output:



```
/hadd/a      1
/hadd/b      1
/hadd/sum    0
/hadd/carry  1
```

HALF SUBSTRACTOR:

LOGIC DIAGRAM:



TRUTH TABLE

A	B	DIFFERENCE	BORROW
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

VERILOG SOURCE CODE:
Dataflow Modeling:

```
module hs_dataflow(a, b, dif, bor);
   input a;
   input b;
   output dif;
   output bor;
            wire abar;
            assign#3 abar=~a;
            assign#3 dif=a^b;
            assign#3 bor=b&abar;
endmodule
```

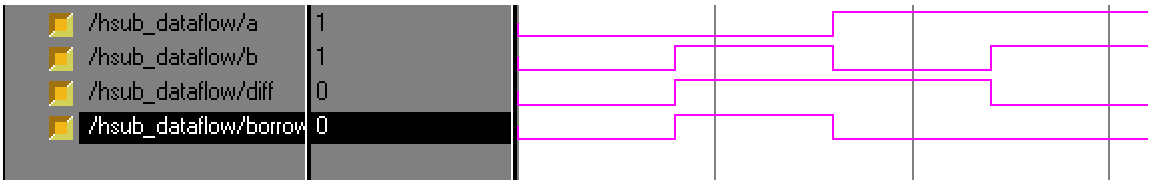
Behavioral Modeling:

```
module hs_behv(a, b, dif, bor);
```

```
   input a;
   input b;
   output dif;
   output bor;
              reg dif,bor;
              reg abar;
              always@(a or b) begin
              abar=~a;
              dif=a^b;
              bor=b&abar;
              end
endmodule
```

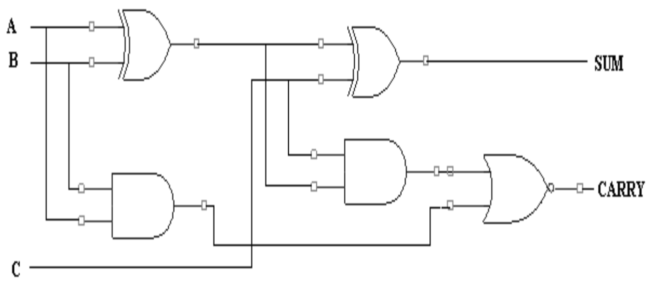
Structural Modeling:

```
module hs_struct(a, b, dif, bor);
   input a;
   input b;
   output dif;
   output bor;
              wire abar;
              xor
              x1(dif,a,b);
              not
              n1(abar,a);
              and
              a1(bor,abar,b);
endmodule
```



FULL ADDER:

LOGIC DIAGRAM:



TRUTH TABLE:

A	B	C	SUM	CARRY
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

VERILOG SOURCE CODE:

Dataflow Modeling:

```
module fulladddataflow(a, b, cin, sum, carry);
```

```
   input a;
   input b;
   input cin;
   output sum;
   output carry;
 assign sum=a^b^cin;
 assign carry=(a & b) | (b & cin) | (cin & a);
 endmodule
```

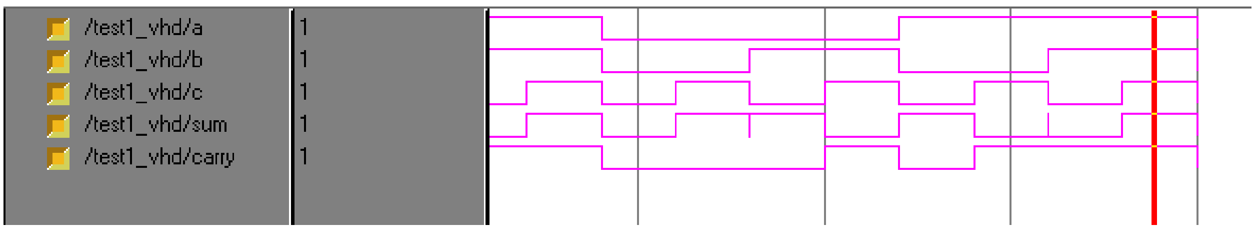
Behavioral Modeling:

```
module fuladbehavioral(a, b, c, sum, carry);
   input a;
   input b;
   input c;
   output sum;
   output carry;
            reg sum,carry;
            reg t1,t2,t3;
            always @ (a or b or c) begin
            sum = (a^b)^c;
            t1=a & b;
            t2=b & c;
            t3=a & c;
            carry=(t1 | t2) | t3;
            end
endmodule
```

Structural Modeling:

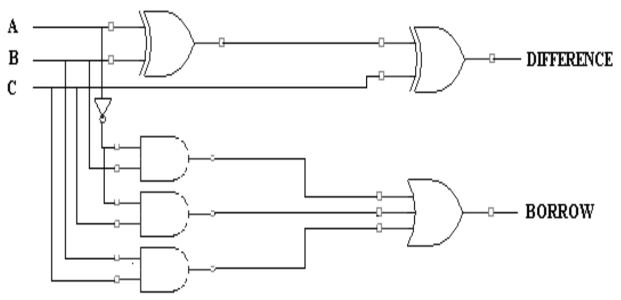
```
module fa_struct(a, b, c, sum, carry);
   input a;
   input b;
   input c;
   output sum;
   output carry;
            wire p,q,r,s;
            xor
            x1(p,a,b),
            x2(sum,p,c);
            and
            a1(q,a,b),
            a2(r,b,c),
            a3(s,a,c);
            or
            o1(carry,q,r,s);
endmodule
```

Simulation output:



FULL SUBSTRACTOR:

LOGIC DIAGRAM:



TRUTH TABLE:

A	B	C	DIFFERENCE	BORROW
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

VERILOG SOURCE CODE:

Dataflow Modeling:

```
module fulsubdataflow(a, b, cin, diff, borrow);
   input a;
   input b;
   input cin;
   output diff;
   output borrow;
              wire abar;
              assign abar= ~ a;
              assign diff=a^b^cin;
              assign borrow=(abar & b) | (b & cin) |(cin & abar);

endmodule
```

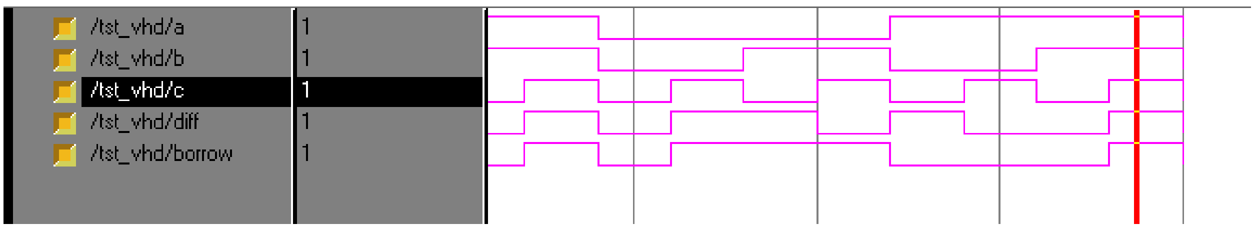
Behavioral Modeling:

```
module fulsubbehavioral(a, b, cin, diff, borrow);
   input a;
   input b;
   input cin;
   output diff;
   output borrow;
              reg t1,t2,t3;
              reg diff,borrow;
              reg abar;
              always @ (a or b or cin) begin
              abar= ~ a;
              diff = (a^b)^cin;
              t1=abar & b;
              t2=b & cin;
              t3=cin & abar;
              borrow=(t1 | t2) | t3;
              end
             endmodule
```

Structural Modeling:

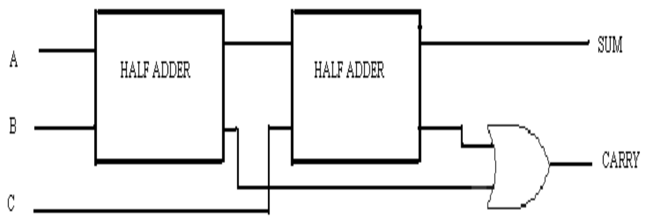
```
module fs_struct(a, b, c, diff, borrow);
   input a;
   input b;
   input c;
   output diff;
   output borrow;
           wire abar,p,q,r,s;
           not
           n1(abar,a);
           xor
           x1(p,a,b),
           x2(diff,p,c);
           and
           a1(q,abar,b),
           a2(r,abar,c),
           a3(s,a,c);
           or
           o1(borrow,q,r,s);
endmodule
```

Simulation output:



FULL ADDER USING TWO HALF ADDERS:

LOGIC DIAGRAM:



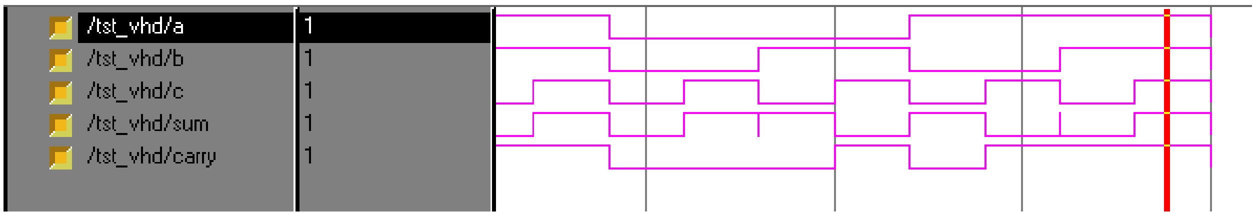
TRUTH TABLE:

A	B	C	SUM	CARRY
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

VERILOG SOURCE CODE:
Structural Modeling:

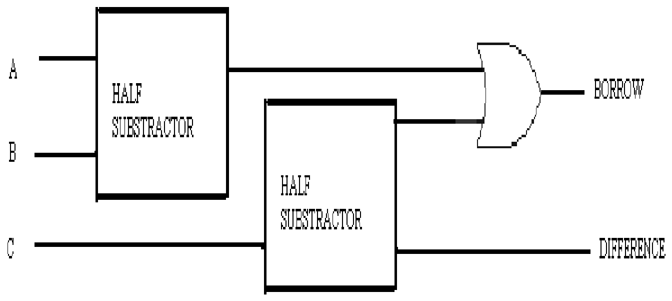
```
module fa_2ha(a, b, c, sum, carry);
   input a;
   input b;
   input c;
   output sum;
   output carry;
            wire p,q,r;
            ha_dataflow
            h1(a,b,p,q),
            h2(p,c,sum,r);
            or
            o1(carry,q,r);
            endmodule
module ha_dataflow(a, b, s, ca);
   input a;
   input b;
   output s;
   output ca;
            assign#2 s=a^b;
            assign#2 ca=a&b;
endmodule
```

Simulation output:



FULL SUBTRACTOR USING TWO HALF SUBTRACTORS:

LOGIC DIAGRAM:



TRUTH TABLE:

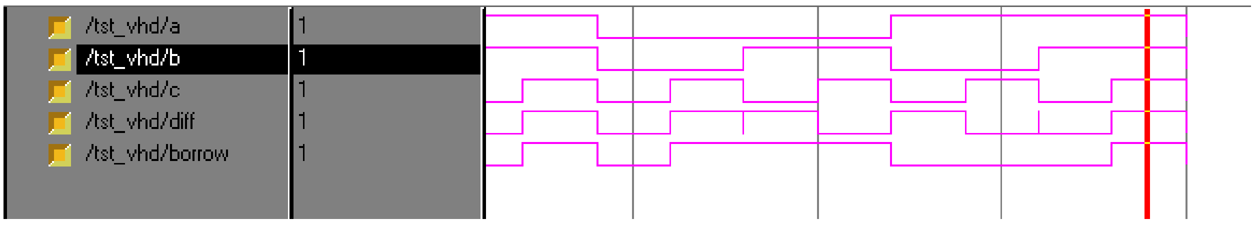
A	B	C	DIFFERENCE	BORROW
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

VERILOG SOURCE CODE:
Structural Modeling:

```
module fs_2hs(a, b, c, diff, borrow);
   input a;
   input b;
   input c;
   output diff;


   output borrow;
   wire p,q,r;
```

```
            hs_dataflow
            h1(a,b,p,q),
            h2(p,c,diff,r);
            or
            o1(borrow,q,r);
endmodule

module hs_dataflow(a, b, dif, bor);
   input a;
   input b;
   output dif;
   output bor;
            wire abar;
            assign#3 abar=~a;
            assign#3 dif=a^b;
            assign#3 bor=b&abar;
endmodule
```

Simulation output:



EXP NO: 03

ENCODERS AND DECODERS

AIM

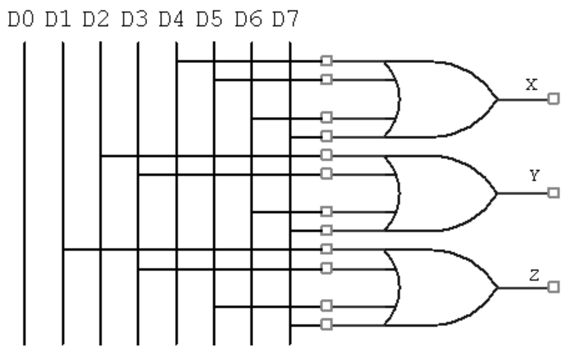
To develop the source code for encoders and decoders by using VERILOG and obtain the simulation.

ALGORITHM

Step1: Define the specifications and initialize the design.
Step2: Declare the name of the entity and architecture by using VHDL source code.
Step3: Write the source code in VERILOG.
Step4: Check the syntax and debug the errors if found, obtain the synthesis report.
Step5: Verify the output by simulating the source code.
Step6: Write all possible combinations of input using the test bench.
Step7: Obtain the place and route report.

ENCODER

LOGIC DIAGRAM



TRUTH TABLE:

D0	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**X**	**Y**	**Z**
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

VERILOG SOURCE CODE:

Dataflow Modeling:

```
module encod_data(d, x, y, z);
   input [7:0] d;
   output x;
   output y;
   output z;
             assign#3 x=d[4]|d[5]|d[6]|d[7];
             assign#3 y=d[2]|d[3]|d[6]|d[7];
             assign#3 z=d[1]|d[3]|d[5]|d[7];
endmodule
```

Behavioral Modeling:

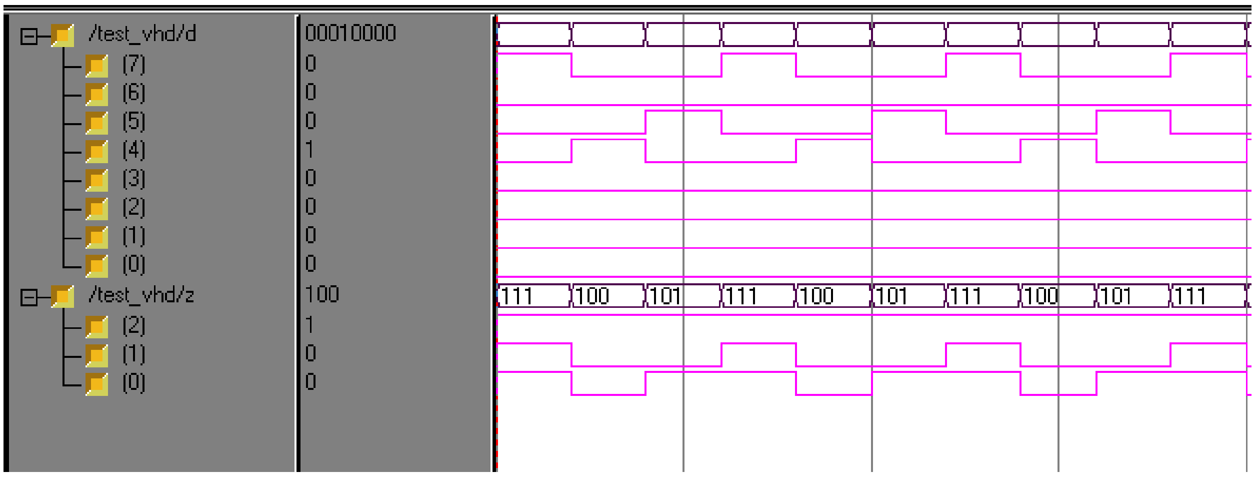
```
module encoderbehav(d, x, y, z);
   input [7:0] d;
   output x;
   output y;
   output z;
             reg x,y,z;
             always @ (d [7:0]) begin
             x= d[4] | d[5] | d[6] | d[7];
             y= d[2] | d[3] | d[6] | d[7];
```

```
            z= d[1] | d[3] | d[5] | d[7];
            end
    endmodule
```

Structural Modeling:

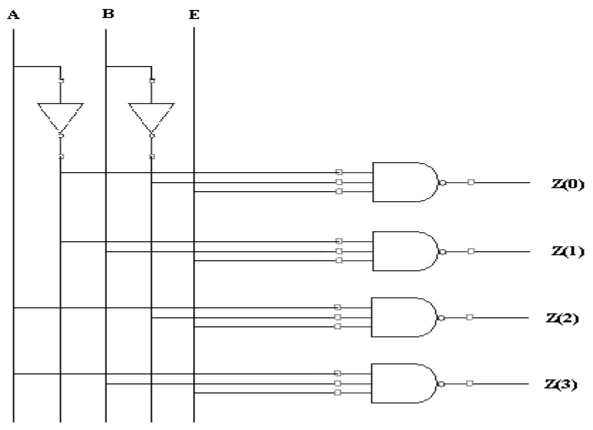
```
module encod_struct(d, x, y, z);
    input [7:0] d;
    output x;
    output y;
    output z;
            or
            o1(x,d[4],d[5],d[6],d[7]),
            o2(y,d[2],d[3],d[6],d[7]),
            o3(z,d[1],d[3],d[5],d[7]);
endmodule
```

Simulation output:



DECODERS:

LOGIC DIAGRAM:



TRUTH TABLE:

A	B	C	Z(0)	Z(1)	Z(2)	Z(3)
0	0	1	0	1	1	1
0	1	1	1	0	1	1
1	0	1	1	1	0	1
1	1	1	1	1	1	0

VERILOG SOURCE CODE:
Dataflow Modeling:

```
module decoderdataflow(a,b,en, z);
   input a,b,en;
   output [0:3] z;
            wire  abar,bbar;
            assign # 1 abar=~a;
            assign # 1 bbar =~b;
            assign # 2 z[0]=(abar & bbar & en);
            assign # 2 z[1]=(abar & b & en);
                        assign # 2 z[2]=(a & bbar & en);
                                    assign # 2 z[3]=(a & b & en);
endmodule
```

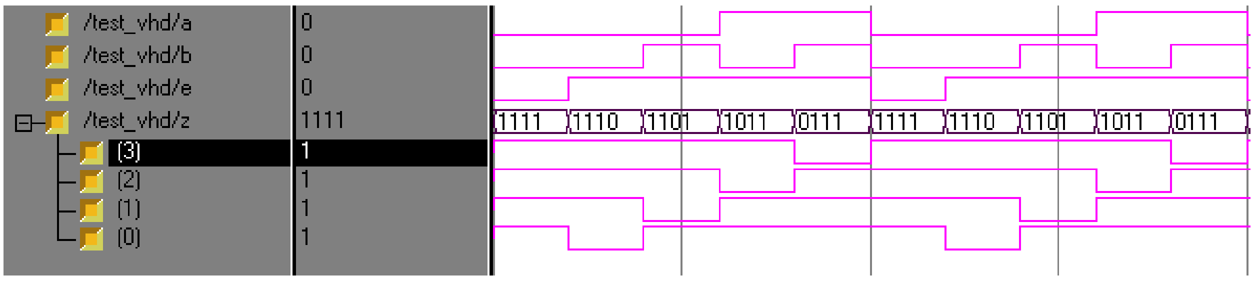
Behavioral Modeling:

```
module decoderbehv(a, b, en, z);
   input a;
   input b;
   input en;
   output [3:0] z;

             reg [3:0] z;

             always @ (a,b,en) begin

             z[0] = ~ ((~a) & (~b) & en);
             z[1] = ~ ((~a) & b & en);
             z[2] = ~ (a & (~b) & en);
             z[3] = ~ (a & b & en);

             end
endmodule
```

Structural Modeling:

```
module decod_struct(a, b, e, z);
   input a;
   input b;
   input e;
   output [3:0] z;
             wire abar,bbar;
             not
             n1(abar,a),
             n2(bbar,b);
             and
             a1(z[0],abar,bbar,e),
             a2(z[1],abar,b,e),
             a3(z[2],a,bbar,e),
             a4(z[3],a,b,e);
             endmodule
```

Simulation output:



RESULT:

Thus the OUTPUT's of Encoder and Decoder are verified by synthesizing and simulating the VERILOG code.

EXP NO: 04

MULTIPLEXER AND DEMULTIPLEXER

AIM

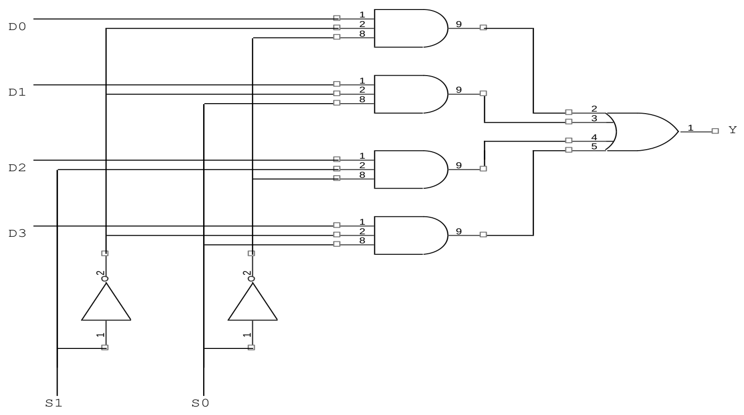
To develop the source code for multiplexer and demultiplexer by using VERILOG and obtain the simulation.

ALGORITHM

Step1: Define the specifications and initialize the design.
Step2: Declare the name of the entity and architecture by using VHDL source code.
Step3: Write the source code in VERILOG.
Step4: Check the syntax and debug the errors if found, obtain the synthesis report.
Step5: Verify the output by simulating the source code.
Step6: Write all possible combinations of input using the test bench.
Step7: Obtain the place and route report.

MULTIPLEXER:

LOGIC DIAGRAM:



TRUTH TABLE:

SELECT INPUT		OUTPUT
S1	S0	Y
0	0	D0
0	1	D1
1	0	D2
1	1	D3

VERILOG SOURCE CODE:

Dataflow Modeling:

```
module muxdataflow(s, i, y);
   input [1:0] s;
   input [3:0] i;
   output y;
          wire s2,s3,s4,s5,s6,s7;
          assign s2 = ~s[1];
          assign s3 = ~s[0];
          assign s4 = i[0]&s2&s3;
          assign s5 = i[1]&s2&s[0];
          assign s6 = i[2]&s[1]&s3;
          assign s7 = i[3]&s[1]&s[0];
          assign y = s4 | s5 | s6 | s7;
endmodule
```

Behavioral Modeling:

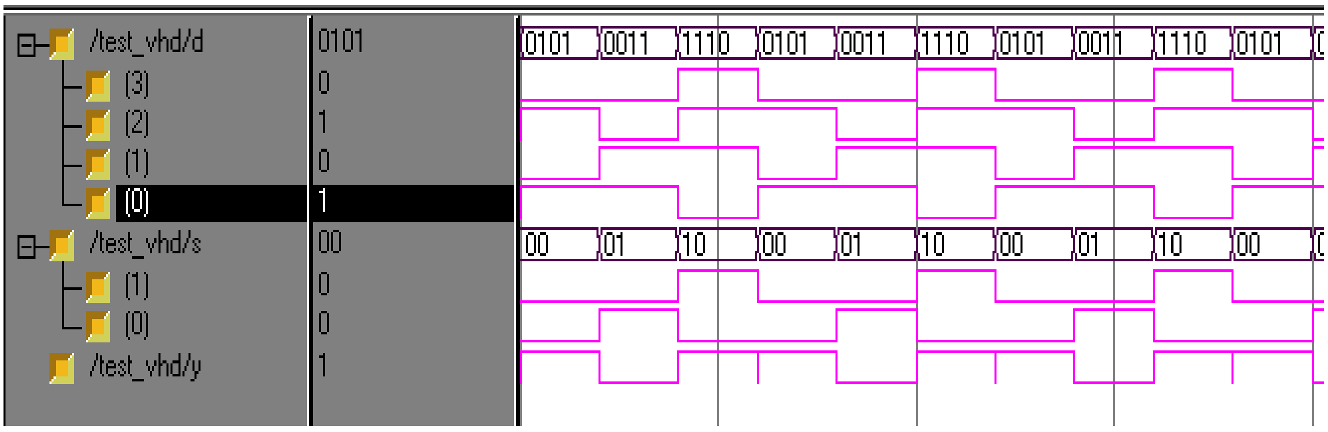
```
module mux_behv(i, s0, s1, y);
   input [3:0] i;
   input s0;
   input s1;
   output y;
```

```
            reg y;
            always@(i or s0 or s1)
            case({s1,s0})
            2'd0:y=i[0];
            2'd1:y=i[1];
            2'd2:y=i[2];
            2'd3:y=i[3];
            default:$display("invalid control signals");
            endcase
endmodule
```

Structural Modeling:

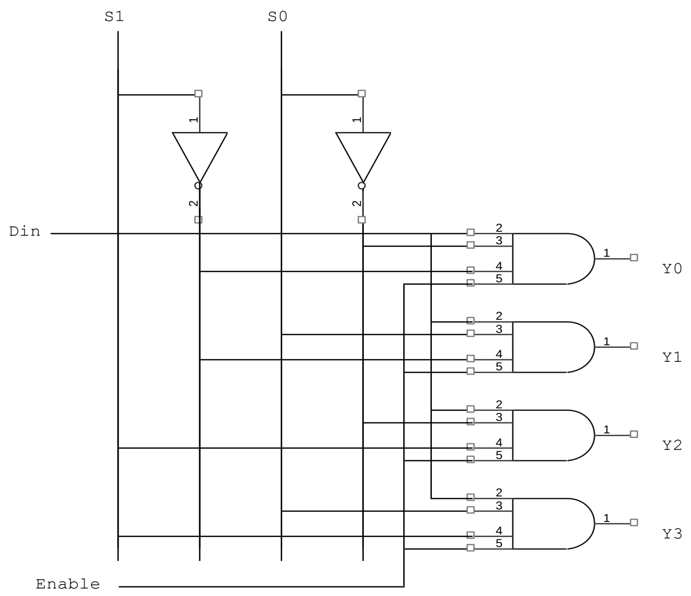
```
module mux_struct(i, s0, s1, y);
   input [3:0] i;
   input s0;
   input s1;
   output y;
            wire s2,s3,s4,s5,s6,s7;
            not
            n1(s2,s1),
            n2(s3,s0);
            and
            a1(s4,i[0],s2,s3),
            a2(s5,i[1],s2,s0),
            a3(s6,i[2],s1,s3),
            a4(s7,i[3],s1,s0);
            or
            o1(y,s4,s5,s6,s7);
endmodule
```

Simulation output:



DEMULTIPLEXER:

LOGIC DIAGRAM:



`

TRUTH TABLE:

INPUT			OUTPUT			
D	S0	S1	Y0	Y1	Y2	Y3
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

VERILOG SOURCE CODE:

Dataflow Modeling:

```
module demuxdataflow(s0,s1,i,y);
   input s0,s1,i;
   output [3:0] y;
            wire s2,s3;
            assign #2 s2=~s0;
            assign #2 s3=~s1;
            assign #3 y[0]=i&s2&s3;
            assign #3 y[1]=i&s2&s1;
            assign #3 y[2]=i&s0&s3;
            assign #3 y[3]=i&s0&s1;
endmodule
```

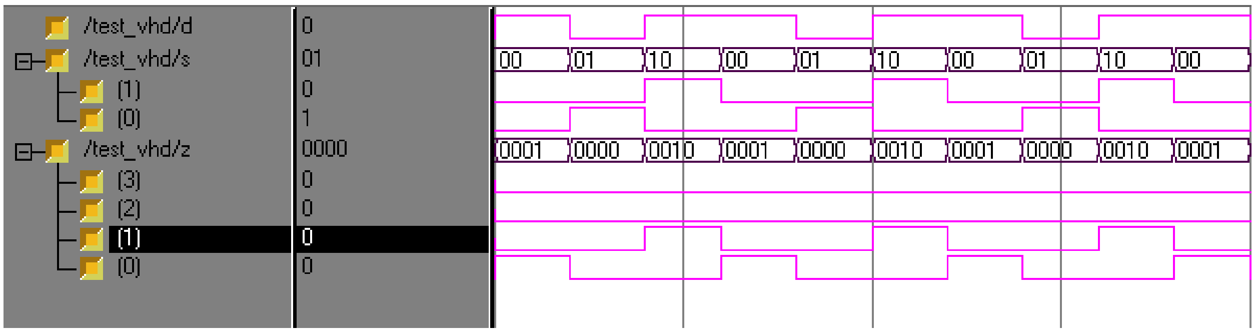
Behavioral Modeling:

```
module demux_behv(s0, s1, i, y);
   input s0;
   input s1;
   input i;
   output [3:0] y;
            reg [3:0] y;
            reg s2,s3;
            always@(i or s0 or s1)
            begin
            s2=~s0;
            s3=~s1;
            y[0]=i & s2 & s3;
            y[1]=i & s2 & s1;
            y[2]=i & s0 & s3;
            y[3]=i & s0 & s1;
            end
            endmodule
```

Structural Modeling:

```
module demux_struct(s0, s1, i, y);
   input s0;
   input s1;
   input i;
   output [3:0] y;
            wire s2,s3;
            not
            n1(s2,s0),
            n2(s3,s1);
            and
            a1(y[0],i,s2,s3),
            a2(y[1],i,s2,s1),
            a3(y[2],i,s0,s3),
            a4(y[3],i,s0,s1);
endmodule
```

Simulation output:



RESULT:

Thus the OUTPUT's of Multiplexers and Demultiplexers are verified by synthesizing and simulating the VERILOG code.

EXP NO: 05

CODE CONVERTERS AND COMPARATOR

AIM:

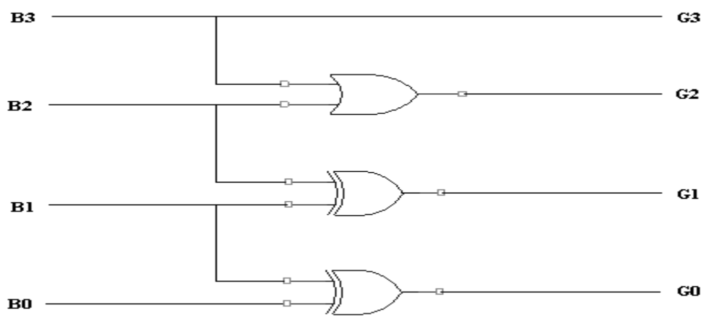
To develop the source code for code converters and comparator by usingVERILOG and obtained the simulation.

ALGORITHM:

Step1: Define the specifications and initialize the design.
Step2: Declare the name of the entity and architecture by using VHDL source code.
Step3: Write the source code in VERILOG.
Step4: Check the syntax and debug the errors if found, obtain the synthesis report.
Step5: Verify the output by simulating the source code.
Step6: Write all possible combinations of input using the test bench.
Step7: Obtain the place and route report.

CODE CONVERTER (BCD TO GRAY):

LOGIC DIAGRAM:



TRUTH TABLE:

BCD	GRAY
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101

VERILOG SOURCE CODE:

Dataflow Modeling:

```
module b2g_data(b, g);
   input [3:0] b;
   output [3:0] g;
            assign#2 g[3]=b[3];
            assign#2 g[2]=b[3]^b[2];
            assign#3 g[1]=b[2]^b[1];
            assign#3 g[0]=b[1]^b[0];
endmodule
```

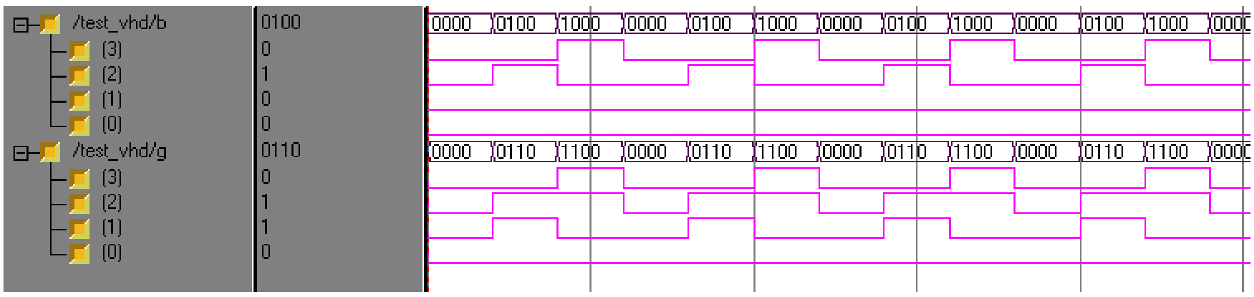
Behavioral Modeling:

```
module b2g_behv(b, g);
   input [3:0] b;
   output [3:0] g;
            reg [3:0] g;
            always@(b) begin
            g[3]=b[3];
            g[2]=b[3]^b[2];
            g[1]=b[2]^b[1];
            g[0]=b[1]^b[0];
            end
endmodule
```

Structural Modeling:

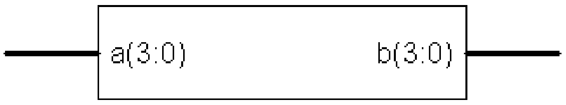
```
module b2gstructural(b, g);
   input [3:0] b;
   output [3:0] g;
            xor
            x1(g[0],b[0],b[1]),
            x2(g[1],b[1],b[2]),
            x3(g[2],b[2],b[3]);
            and
            a1(g[3],b[3]);
endmodule
```

Simulation output:



CODE CONVERTER (BCD TO EXCESS 3):

LOGIC DIAGRAM:



VERILOG SOURCE CODE:

Dataflow Modeling:

```
module bcd2ex3_data(b, e);
   input [3:0] b;
   output [3:0] e;
            wire s1,s2,s3,s4,s5,s6,s7;
            assign#2 s1=~b[2];
            assign#2 s2=~b[1];
            assign#2 s3=~b[0];
            assign#3 s4=b[1]|b[0];
            assign#4 s5=b[1]&s2&s3;
            assign#5 s6=s1&s4;
            assign#5 s7=b[2]&s4;
            assign#5 e[3]=b[3]|s7;
            assign#5 e[2]=s5|s6;
            assign#5 e[1]=b[0]^~b[1];
            assign#6 e[0]=s3;
endmodule
```

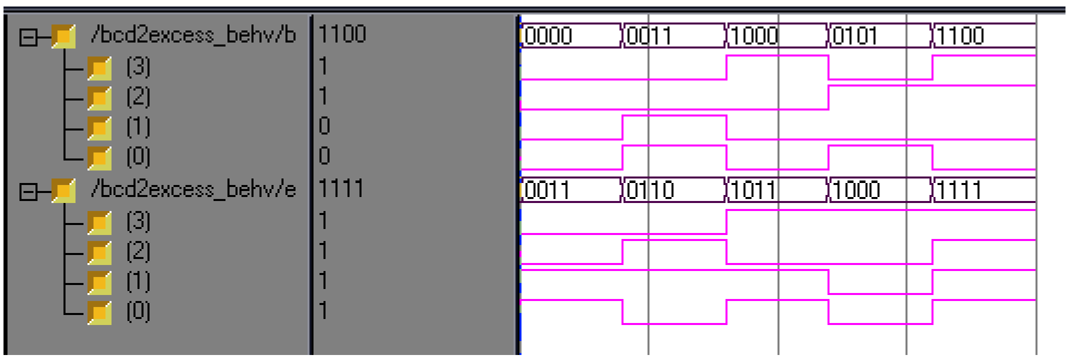
Behavioral Modeling:

```
module bcd2ex3_behv(b, e);
   input [3:0] b;
   output [3:0] e;
           reg [3:0] e;
           reg s1,s2,s3,s4,s5,s6,s7;
           always@(b) begin
           s1=~b[2];
           s2=~b[1];
           s3=~b[0];
           s4=b[1]|b[0];
           s5=b[1]&s2&s3;
           s6=s1&s4;
           s7=b[2]&s4;
           e[3]=b[3]|s7;
           e[2]=s5|s6;
           e[1]=b[0]^~b[1];
           e[0]=s3;
           end
endmodule
```

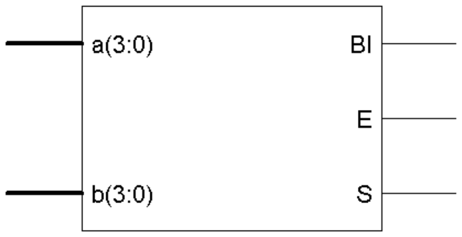
Structural Modeling:

```
module bcd2excessstructural(b, e);
   input [3:0] b;
   output [3:0] e;
           wire s1,s2,s3,s4,s5,s6,s7;
           not
           n1(s1,b[2]),
           n2(s2,b[1]),
           n3(s3,b[0]);
           or
           o1(s4,b[1],b[0]),
           o2(e[3],b[3],s7),
           o3(e[2],s5,s6);
           xor
           x1(e[1],b[0],b[1]);
           and
           a1(s5,b[1],s2,s3),
           a2(s6,s1,s4),
           a3(s7,b[2],s4),
           a4(e[0],s3);
endmodule
```

Simulation output:



4 BIT COMPARATOR:
LOGIC DIAGRAM:

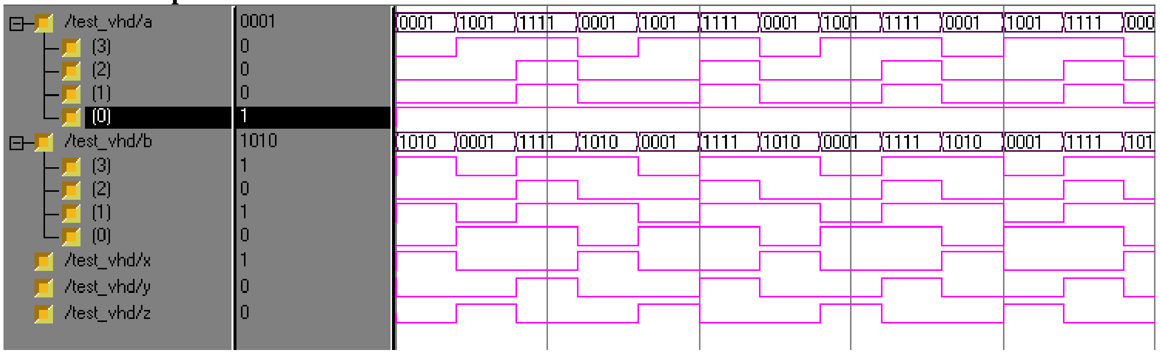


VERILOG SOURCE CODE:

Behavioral Modeling:

```
module comparatorbehvioral(a, b, x, y, z);
   input [3:0] a;
   input [3:0] b;
   output x;
   output y;
   output z;
           reg x,y,z;
           always @ (a,b) begin
           if(a==b)
           begin
           x=1'b1;
           y=1'b0;
           z=1'b0;
           end
           else if (a<b)  begin
           x=1'b0;
           y=1'b1;
           z=1'b0;
   end
           else
           begin
           x=1'b0;
           y=1'b0;
           z=1'b1;
           end
           end
           endmodule
```

Simulation output:



RESULT:

Thus the OUTPUT's of Code converters and comparator are verified by synthesizing and simulating the VERILOG code.

EXP NO: 06

FLIP FLOPS

AIM:

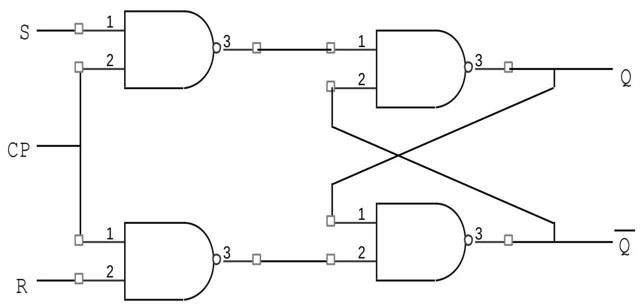
To develop the source code for flip flops by using VERILOG and Obtained the simulation.

ALGORITHM:

Step1: Define the specifications and initialize the design.
Step2: Declare the name of the entity and architecture by using VHDL source code.
Step3: Write the source code in VERILOG.
Step4: Check the syntax and debug the errors if found, obtain the synthesis report.
Step5: Verify the output by simulating the source code.
Step6: Write all possible combinations of input using the test bench.
Step7: Obtain the place and route report.

SR FLIPFLOP:

LOGIC DIAGRAM:



TRUTH TABLE:

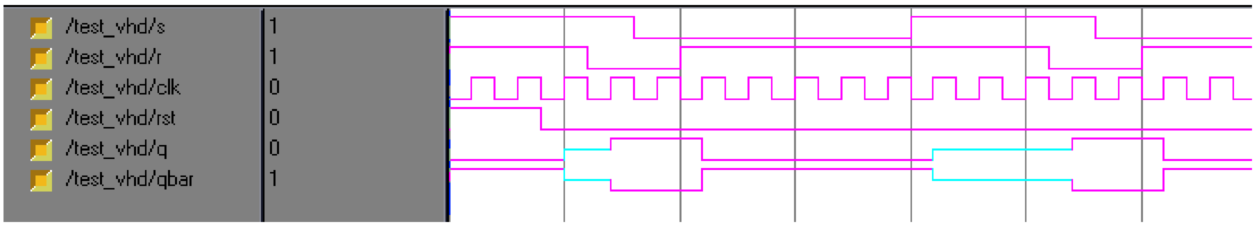
Q(t)	S	R	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	X
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	X

VERILOG SOURCE CODE:

Behavioral Modeling:

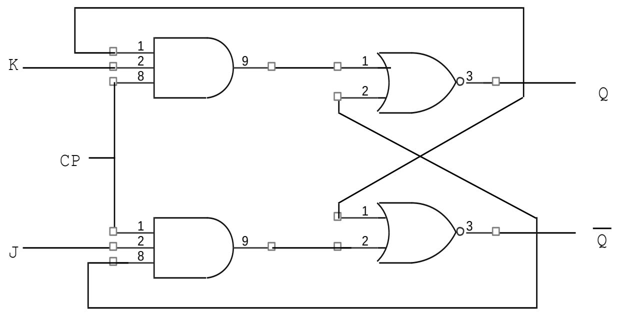
```
module srflipflop(s, r, clk, rst, q, qbar);
   input s;
   input r;
   input clk;
   input rst;
   output q;
   output qbar;
           reg q,qbar;
           always @ (posedge(clk) or posedge(rst)) begin
           if(rst==1'b1) begin
           q= 1'b0;qbar= 1'b1;
           end
           else if(s==1'b0 &&          r==1'b0)
            begin
           q=q; qbar=qbar;
           end
                     else if(s==1'b0 &&          r==1'b1)
                      begin
           q= 1'b0; qbar= 1'b1;
           end
                     else if(s==1'b1 &&          r==1'b0)
                      begin
           q= 1'b1; qbar= 1'b0;
           end
           else
           begin
           q=1'bx;qbar=1'bx;
           end
           end
endmodule
```

Simulation output:



JK FLIPFLOP:

LOGIC DIAGRAM:



TRUTH TABLE:

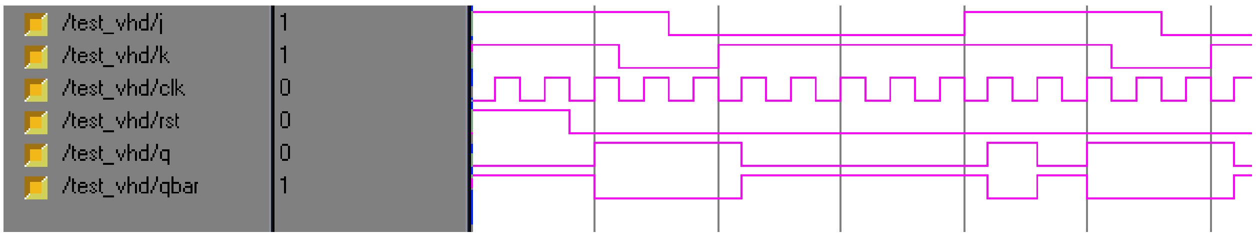
Q(t)	**J**	**K**	**Q(t+1)**
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

VERILOG SOURCE CODE:

Behavioral Modeling:

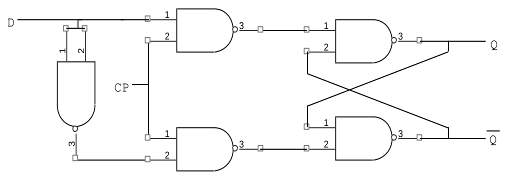
```
module jkff(j, k, clk, rst, q, qbar);
   input j;
   input k;
   input clk;
   input rst;
   output q;
   output qbar;
            reg q;
            reg qbar;
            always @ (posedge(clk) or posedge(rst))  begin
            if (rst==1'b1)
            begin
            q=1'b0;
            qbar=1'b1;
            end
            else if (j==1'b0 && k==1'b0)
            begin
            q=q;
            qbar=qbar;
            end
            else if (j==1'b0 && k==1'b1)
            begin
            q=1'b0;
            qbar=1'b1;
            end
            else if (j==1'b1 && k==1'b0)
            begin
            q=1'b1;
            qbar=1'b0;
            end
            else
            begin
            q=~q;
            qbar=~qbar;
            end
            end
            endmodule
```

Simulation output:



D FLIPFLOP:

LOGIC DIAGRAM:



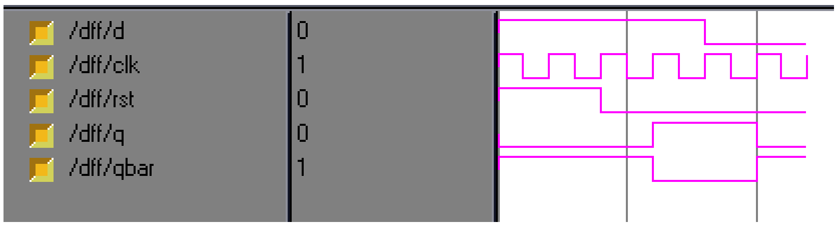
TRUTH TABLE:

Q(t)	D	Q(t+1)
0	0	0
0	1	1
1	0	0
1	1	1

VERILOG SOURCE CODE:
Behavioral Modeling:

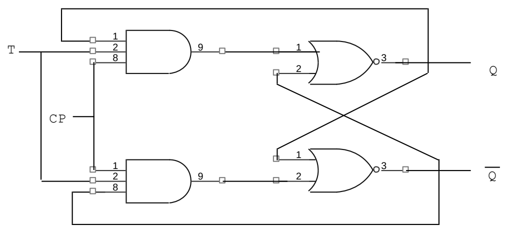
```
module dff(d, clk, rst, q, qbar);
   input d;
   input clk;
   input rst;
   output q;
   output qbar;
            reg q;
            reg qbar;
            always @ (posedge(clk) or posedge(rst)) begin
            if (rst==1'b1)
            begin
            q=1'b0;
            qbar=1'b1;
            end
            else if (d==1'b0)
            begin
            q=1'b0;
            qbar=1'b1;
            end
            else
            begin
            q=1'b1;
            qbar=1'b0;
            end
           end
           endmodule
```

Simulation output:



T FLIPFLOP:

LOGIC DIAGRAM:



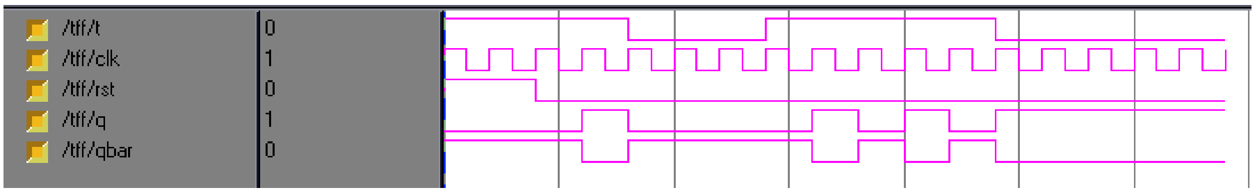
TRUTH TABLE:

Q(t)	T	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

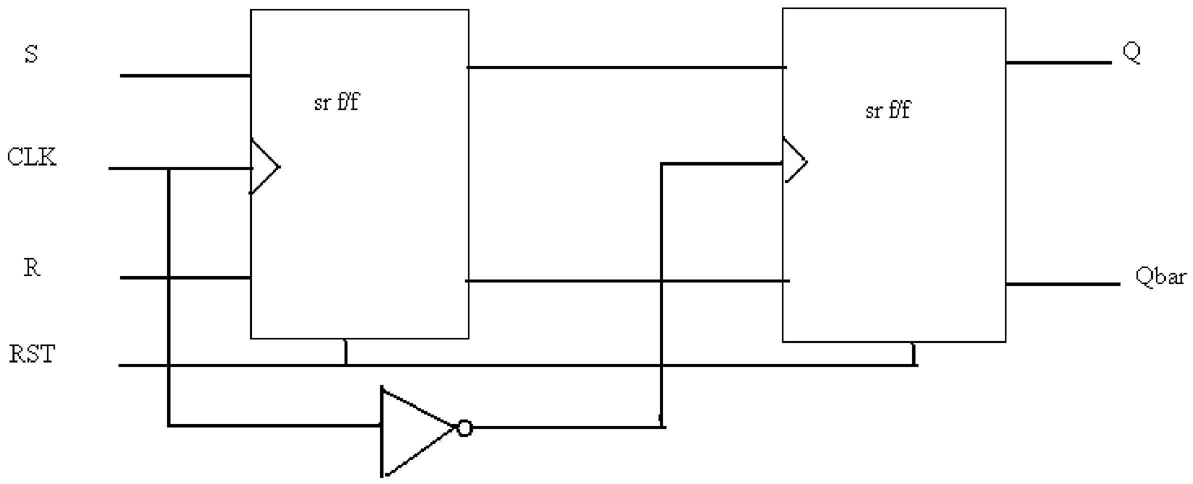
VERILOG SOURCE CODE:
Behavioral Modeling:

```
module tff(t, clk, rst, q, qbar);
   input t;
   input clk;
   input rst;
   output q;
   output qbar;
            reg q,qbar;
            always @ (posedge(clk) or posedge(rst)) begin
            if(rst==1'b1) begin
            q= 1'b0;qbar= 1'b1;
            end
            else if (t==1'b0)
            begin
            q=q; qbar=qbar;
            end
            else
            begin
            q=~q; qbar=~qbar;
            end
            end
            endmodule
```

Simulation output:



MASTER-SLAVE SR FLIP-FLOP:
LOGIC DIAGRAM:



VERILOG SOURCE CODE:

Structurl modeling:

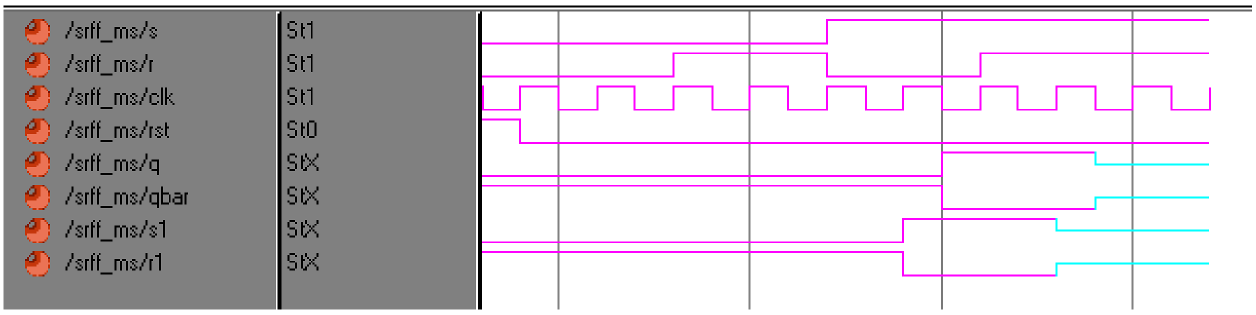
```
module srff_ms(s, r, clk, rst, q, qbar);
   input s;
   input r;
   input clk;
   input rst;
   output q;
   output qbar;
            wire s1,r1;
            srflipflop
            sr1(s,r,clk,rst,s1,r1),
            sr2(s1,r1,~clk,rst,q,qbar);
endmodule
```

srflipflop component source code:

```
module srflipflop(s, r, clk, rst, q, qbar);
   input s;
   input r;
   input clk;
   input rst;
   output q;
```

```
    output qbar;
            reg q,qbar;
            always @ (posedge(clk) or posedge(rst)) begin
            if(rst==1'b1) begin
            q= 1'b0;qbar= 1'b1;
            end
            else if(s==1'b0 &&          r==1'b0)
            begin
            q=q; qbar=qbar;
            end
            else if(s==1'b0 &&          r==1'b1)
             begin
            q= 1'b0; qbar= 1'b1;
            end
            else if(s==1'b1 &&          r==1'b0)
            begin
            q= 1'b1; qbar= 1'b0;
            end
            else
            begin
            q=1'bx;qbar=1'bx;
            end
            end
endmodule
```

Simulation output:



```
/srff_ms/s      St1
/srff_ms/r      St1
/srff_ms/clk    St1
/srff_ms/rst    St0
/srff_ms/q      StX
/srff_ms/qbar   StX
/srff_ms/s1     StX
/srff_ms/r1     StX
```

RESULT:

Thus the OUTPUT's of Flip Flops are verified by synthesizing and simulating the VERILOG code.