LAB ASSIGNMENT

Submitted for

EMBEDDED SYSTEM DESIGN (UCS704)

Submitted by:

AAYUSH BANSAL 102103212

4 CO8

Submitted to:

DR. DEEPSHIKHA TIWARI

Computer Science and Engineering Department Thapar Institute of Engineering and Technology, Patiala

EXPERIMENT – 1 (Truth Table and Logic Gates)

AIM: - To study and verify the truth table of various logic gates (AND, OR, NAND, NOR, XOR, XNOR, NOT).

AND GATE

CODE: -

```
module andGate; reg x;
reg y; wire z;
andComp uut (
.x(x),
.y(y),
.z(z)
);
initial begin x = 0;
y = 0;
#20 x = 1;
#20 y = 1;
#20 x = 0;
end
initial begin
$display("INPUT\tOUTPUT");
$monitor("x=%d,y=%d,z=%d \n",x,y,z);
end endmodule
module andComp( input x,
input y, output z
);
assign z = x&y; endmodule
```

OUTPUT: -

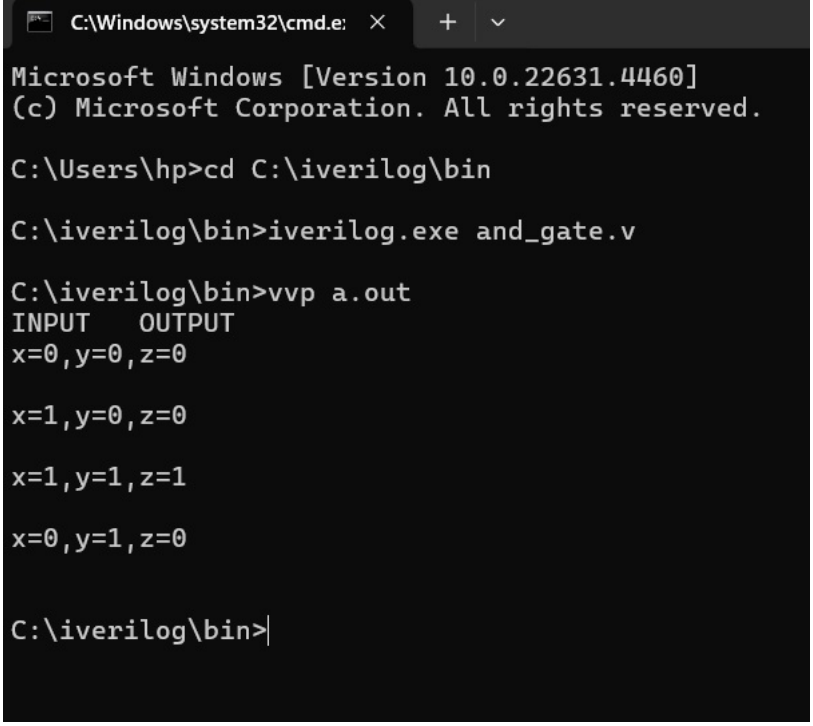


Fig 1.1 Output for AND gate

OR GATE

CODE: -

```
module orGate; reg x;
reg y; wire z;
orComp uut (
.x(x),
.y(y),
.z(z)
);
initial begin x = 0;
y = 0;
#20 x = 1;
#20 y = 1;
#20 x = 0;
end
initial begin
$display("INPUT\tOUTPUT");
$monitor("x=%d,y=%d,z=%d \n",x,y,z); end
endmodule module orComp(
input x,
input y, output z
);
assign z = x|y; endmodule
```

OUTPUT: -

```
C:\iverilog\bin>iverilog.exe or_gate.v

C:\iverilog\bin>vvp a.out
INPUT   OUTPUT
x=0,y=0,z=0

x=1,y=0,z=1

x=1,y=1,z=1

x=0,y=1,z=1


C:\iverilog\bin>
```

Fig 1.2 Output for OR gate

NAND GATE

CODE: -

```
module nandGate; reg x;

reg y; wire z;

nandComp uut (

.x(x),

.y(y),

.z(z)

);

initial begin x = 0;

y = 0;

#20 x = 1;

#20 y = 1;

#20 x = 0;

end

initial begin

$display("INPUT\tOUTPUT");

$monitor("x=%d,y=%d,z=%d \n",x,y,z); end

endmodule

module nandComp( input x,

input y, output z

);

assign z = x~&y; endmodule
```

OUTPUT: -

```
C:\iverilog\bin>iverilog.exe nand_gate.v

C:\iverilog\bin>vvp a.out
INPUT   OUTPUT
x=0,y=0,z=1

x=1,y=0,z=1

x=1,y=1,z=0

x=0,y=1,z=1


C:\iverilog\bin>
```

Fig 1.3 Output for NAND gate

NOR GATE

CODE: -

```
module norGate; reg x;

reg y; wire z;

norComp uut (

.x(x),

.y(y),

.z(z)

);

initial begin x = 0;

y = 0;

#20 x = 1;

#20 y = 1;

#20 x = 0;

end

initial begin

$display("INPUT\tOUTPUT");

$monitor("x=%d,y=%d,z=%d \n",x,y,z); end

endmodule module norComp(

input x,

input y, output z

);

assign z = x~|y; endmodule
```

OUTPUT: -

```
C:\iverilog\bin>iverilog.exe nor_gate.v

C:\iverilog\bin>vvp a.out
INPUT   OUTPUT
x=0,y=0,z=1

x=1,y=0,z=0

x=1,y=1,z=0

x=0,y=1,z=0


C:\iverilog\bin>
```

Fig 1.4 Output for NOR gate

XOR GATE

CODE: -

```
module xorGate; reg x;

reg y; wire z;

xorComp uut (

.x(x),

.y(y),

.z(z)

);
```

initial begin x = 0;

y = 0;

#20 x = 1;

#20 y = 1;

#20 x = 0;

end

initial begin

$display("INPUT\tOUTPUT");

$monitor("x=%d,y=%d,z=%d \n",x,y,z); end

endmodule module xorComp(

input x,

input y, output z

);

assign z = x^y; endmodule

OUTPUT:-

```
C:\iverilog\bin>iverilog.exe xor_gate.v

C:\iverilog\bin>vvp a.out
INPUT   OUTPUT
x=0,y=0,z=0

x=1,y=0,z=1

x=1,y=1,z=0

x=0,y=1,z=1


C:\iverilog\bin>
```

Fig 1.5 Output for XOR gate

XNOR GATE

CODE: -

module xnorGate; reg x;

reg y; wire z;

xnorComp uut (

.x(x),

.y(y),

.z(z)

);

initial begin x = 0;

y = 0;

#20 x = 1;

#20 y = 1;

#20 x = 0;

end

initial begin

$display("INPUT\tOUTPUT");

$monitor("x=%d,y=%d,z=%d \n",x,y,z); end

endmodule module xnorComp(

input x,

input y, output z

);

assign z = x~^y; endmodule

OUTPUT:-

```
C:\iverilog\bin>iverilog.exe xnor_gate.v

C:\iverilog\bin>vvp a.out
INPUT   OUTPUT
x=0,y=0,z=1

x=1,y=0,z=0

x=1,y=1,z=1

x=0,y=1,z=0


C:\iverilog\bin>
```

Fig 1.6 Output for XNOR gate

NOT GATE

CODE: -

module notGate; reg x;

wire z; notComp uut (

.x(x),

.z(z)

);

initial begin x = 0;

#20 x = 1;

end

initial begin

$monitor("x=%d,z=%d \n",x,z); end

endmodule module notComp(

input x, output z

);

assign z = ~x; endmodule

OUTPUT: -

```
C:\iverilog\bin>iverilog.exe not_gate.v

C:\iverilog\bin>vvp a.out
x=0,z=1

x=1,z=0


C:\iverilog\bin>
```

Fig 1.7 Output for NOT gate

EXPERIMENT – 2 (HALF ADDER)

AIM: - To design and verify a half adder using S= (x+y)(x'+y') C= xy

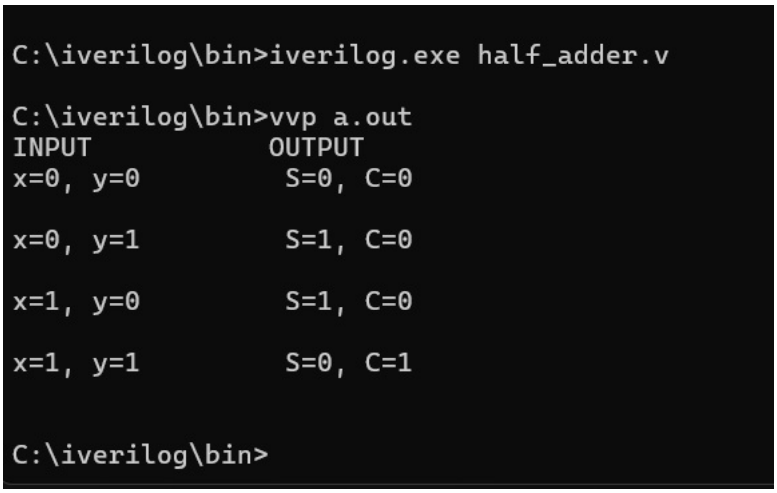
CODE: -

module halfAdder; reg x;

reg y; wire S; wire C;

```
halfAdderComp uut (
.x(x),
.y(y),
.S(S),
.C(C)
);
initial begin x = 0;
y = 0;
#20 y = 1;
#20 y = 0; x = 1;
#20 x = 1; y = 1;
end
initial begin
$display("INPUT\t\tOUTPUT");
$monitor("x=%d, y=%d \t S=%d, C=%d \n", x, y, S, C);
end endmodule
module halfAdderComp( input x,
input y, output S, output C
);
assign S = x ^ y; assign C = x & y;
endmodule
```

OUTPUT: -



```
C:\iverilog\bin>iverilog.exe half_adder.v

C:\iverilog\bin>vvp a.out
INPUT           OUTPUT
x=0, y=0         S=0, C=0

x=0, y=1         S=1, C=0

x=1, y=0         S=1, C=0

x=1, y=1         S=0, C=1


C:\iverilog\bin>
```

Fig 2 Output for HALF ADDER

EXPERIMENT – 3 (FULL ADDER)

AIM: - To design and verify a full adder using S = x'y'z+x'yz'+xy'z'+xyz C=xy+xz+yz

CODE: -

```
module fullAdder; reg x;
reg y; reg Ci; wire S;
wire Cout; fullAdderComp uut (
.x(x),
.y(y),
.Ci(Ci),
.S(S),
.Cout(Cout)
);
```

```
initial begin x = 0;

y = 0; Ci=0;

#20 Ci=1;

#20 y = 1; Ci=0;

#20 Ci=1;

#20 y=0; x=1; Ci=0;

#20 Ci=1;

#20 x = 1; y=1; Ci=0;

#20 Ci=1;

end

initial begin

$display("Full Adder\n");

$display("X Y Ci\tS Cout");

$monitor("%d %d %d \t%d %d \n",x,y,Ci,S,Cout); end

endmodule

module fullAdderComp( input x,

input y, input Ci, output S, output Cout

);

assign S = x^y^Ci;

assign Cout = Ci&(x^y) | x&y;

endmodule
```

OUTPUT: -

```
C:\iverilog\bin>iverilog.exe full_adder.v

C:\iverilog\bin>vvp a.out
Full Adder

X  Y  Ci        S  Cout
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


C:\iverilog\bin>
```

Fig 3 Output for FULL ADDER

EXPERIMENT – 4 (Half Subtractor)

AIM: - To design and verify a half subtractor using D = x'y +xy' B=x'y

CODE: -

```
module halfSub; reg x;

reg y; wire D; wire B;

halfSubComp uut (
```

```
.x(x),

.y(y),

.D(D),

.B(B)

);

initial begin x = 0;

y = 0;

#20 y = 1;

#20 y=0; x=1;

#20 x = 1; y=1;

end

initial begin

$display("Half Subtractor\n");

$display("X Y\tD B");

$monitor("%d %d\t%d %d \n",x,y,D,B); end

endmodule

module halfSubComp( input x,

input y, output D, output B

);

assign D = (~x&y)|(x&~y); assign B = ~x&y;

endmodule
```

OUTPUT: -

```
C:\iverilog\bin>iverilog.exe half_subtractor.v

C:\iverilog\bin>vvp a.out
Half Subtractor

X  Y    D  B
0  0    0  0

0  1    1  1

1  0    1  0

1  1    0  0


C:\iverilog\bin>
```

Fig 4 Output for HALF SUBTRACTOR

EXPERIMENT – 5 (Number Converter)

AIM: - Design a BCD to Excess 3 code converter using combinational circuits.

CODE: -

```
module BCD2Ex3(A, B, C, D, W, X, Y, Z);

input A, B, C, D; output W, X, Y, Z;

assign W = A | (B&C) | (B&D);

assign X = (~B&C) | (~B & D) | (B & ~C & ~D); assign Y = ~( C ^ D);

assign Z = ~D; endmodule

module test; wire W, X,Y,Z;

reg A,B,C,D;

BCD2Ex3 object(A,B,C,D,W,X,Y,Z);

initial begin
```

```
$dumpfile("bcd.vcd");

$dumpvars(0,test);

$display (" A B C D | W X Y Z");

$monitor(" ",A, " ",B, " ",C, " ",D, " | ", W, " ",X, " ",Y, " ",Z); A = 0; B = 0; C = 0; D = 0;

#5 A = 0; B = 0; C = 0; D = 0;

#5 A = 0; B = 0; C = 0; D = 1;

#5 A = 0; B = 0; C = 1; D = 0;

#5 A = 0; B = 0; C = 1; D = 1;

#5 A = 0; B = 1; C = 0; D = 0;

#5 A = 0; B = 1; C = 0; D = 1;

#5 A = 0; B = 1; C = 1; D = 0;

#5 A = 0; B = 1; C = 1; D = 1;

#5 A = 1; B = 0; C = 0; D = 0;

#5 A = 1; B = 0; C = 0; D = 1;

end endmodule
```

OUTPUT: -

```
C:\iverilog\bin>iverilog.exe number_convertor.v

C:\iverilog\bin>vvp a.out
VCD info: dumpfile bcd.vcd opened for output.
 A  B  C  D |  W  X  Y  Z
 0  0  0  0 | 0  0  1  1
 0  0  0  1 | 0  1  0  0
 0  0  1  0 | 0  1  0  1
 0  0  1  1 | 0  1  1  0
 0  1  0  0 | 0  1  1  1
 0  1  0  1 | 1  0  0  0
 0  1  1  0 | 1  0  0  1
 0  1  1  1 | 1  0  1  0
 1  0  0  0 | 1  0  1  1
 1  0  0  1 | 1  1  0  0

C:\iverilog\bin>
```

Fig 5 Output for NUMBER CONVERTOR

EXPERIMENT – 6 (Multiplexer)

AIM: - To design and implement a 4:1 multiplexer

CODE: -

```
module mux(s1,s2,a,b,c,d,y); input s1,s2,a,b,c,d; output y;

assign y = ~s1&~s2&a | ~s1&s2&b | s1&~s2&c | s1&s2&d ; endmodule

module test;

reg a, b, c, d, s1, s2; wire y;

mux obj(s1,s2,a,b,c,d,y); initial begin

//$dumpfile("mux.vcd");

//$dumpvars( 0, test);

$display("S1\t S2\t A \t B \t C \t D | Y");

$monitor("%b \t %b \t %b \t %b \t %b \t %b | %b",s1,s2,a,b,c,d,y);

a=0; b=0; c=0; d=0; s1=0; s2=0;

#5 a=0; b=0; c=0; d=0; s1=0; s2=0; #5 a=0; b=0; c=0; d=1; s1=0; s2=1; #5 a=0; b=0; c=1; d=0; s1=1; s2=0; #5 a=0; b=0; c=1; d=1; s1=1; s2=1;

#5 a=0; b=1; c=0; d=0; s1=0; s2=0; #5 a=0; b=1; c=0; d=1; s1=0; s2=1; #5 a=0; b=1; c=1; d=0; s1=1; s2=0; #5 a=0; b=1; c=1; d=1; s1=1; s2=0; #5 a=1; b=0; c=0; d=0; s1=0; s2=1; #5
a=1; b=0; c=0; d=1; s1=0; s2=0;

#5 $finish;
```

end endmodule

OUTPUT: -

```
C:\iverilog\bin>iverilog.exe 4_1_Mux.v

C:\iverilog\bin>vvp a.out
S1        S2       A        B        C        D |  Y
0         0        0        0        0        0 |  0
0         1        0        0        0        1 |  0
1         0        0        0        1        0 |  1
1         1        0        0        1        1 |  1
0         0        0        1        0        0 |  0
0         1        0        1        0        1 |  1
1         0        0        1        1        0 |  1
1         0        0        1        1        1 |  1
0         1        1        0        0        0 |  0
0         0        1        0        0        1 |  1
4_1_Mux.v:28: $finish called at 55 (1s)

C:\iverilog\bin>
```

Fig 6 Output for 4:1 Multiplexer

EXPERIMENT – 7 (Demultiplexer)

AIM: - To design and implement a 1:4 demultiplexer.

CODE: -

module demux(s1,s0,a,b,c,d,e,i); input s1,s0,e,i;

output a,b,c,d;

assign a =i&e&~s1&~s0; assign b =i&e&~s1&s0; assign c =i&e&s1&~s0; assign d =i&e&s1&s0;

endmodule module test;

reg s1, s0, e, i;

wire a, b, c, d;

demux obj(s1,s0,a,b,c,d,e,i); initial

begin

$display("e\ts1\ts0\td\tc\tb\ta");

$monitor("%b\t%b\t%b\t%b\t%b\t%b\t%b" ,e,s1,s0,d,c,b,a); i=1; e=0; s1=0; s0=0;

#10 i=1; e=1; s1=0; s0=0; #10 i=1; e=1; s1=0; s0=1; #10 i=1; e=1; s1=1; s0=0; #10 i=1; e=1; s1=1; s0=1;

end endmodule

OUTPUT: -

```
C:\iverilog\bin>iverilog.exe 1_4_Demux.v

C:\iverilog\bin>vvp a.out
e        s1       s0       d        c        b        a
0        0        0        0        0        0        0
1        0        0        0        0        0        1
1        0        1        0        0        1        0
1        1        0        0        1        0        0
1        1        1        1        0        0        0

C:\iverilog\bin>
```

Fig 7 Output for 1:4 Demultiplexer

EXPERIMENT – 8 (Decoder)

AIM: - To design and verify a 2:4 decoder.

CODE: -

module decoder(a,b,c,d,e,f,E); input a,b,E;

output c,d,e,f; assign c = E&a&b;

```
assign d = E&a&(~b); assign e = E&(~a)&b; assign f = E&(~a)&(~b);

endmodule module testbench;

reg a, b, E;

wire c,d,e,f;

decoder obj(a,b,c,d,e,f,E); initial begin

$display("Inputs | Outputs");

$display("E a b | c d e f");

$monitor("%b %b %b | %b %b %b %b",E,a,b,c,d,e,f); E=0 ; a=0; b=0;

#5 E=1; a=0; b=0; #5 E=1; a=0; b=1; #5 E=1; a=1; b=0; #5 E=1; a=1; b=1;

#5 $finish;

end endmodule
```

OUTPUT: -

```
C:\iverilog\bin>vvp a.out
Inputs      |  Outputs
E  a  b  |  c  d  e  f
0  0  0  |  0  0  0  0
1  0  0  |  0  0  0  1
1  0  1  |  0  0  1  0
1  1  0  |  0  1  0  0
1  1  1  |  1  0  0  0
file.v:22: $finish called at 25 (1s)
```

Fig 8 Output for 2:4 decoder.

EXPERIMENT – 9 (Encoder)

AIM: - To design and implement a 4:2 encoder.

CODE: -

```
module Encoder42; reg i1;

reg i2;

reg i3; reg i4; wire o1; wire o2;

Encoder42Comp enc (

.i1(i1),

.i2(i2),

.i3(i3),

.i4(i4),

.o1(o1),

.o2(o2)

);

initial begin i1 = 0;

i2 = 0;

i3 = 0;

i4 = 0;

#20 i1 = 1;

#20 i1 = 0; i2 = 1;

#20 i2 = 0; i3 = 1;

#20 i3 = 0; i4 = 1;

end

initial begin

$display("4*2 Encoder\n");

$display("I4 I3 I2 I1\t\tO2 O1");

$monitor("%d %d %d %d\t\t%d %d", i4, i3, i2, i1, o2, o1); end
```

endmodule

module Encoder42Comp(input i1,

input i2, input i3, input i4, output o1, output o2

);

assign o1 = i3 | i4; assign o2 = i2 | i4;

endmodule

OUTPUT: -

```
C:\iverilog\bin>vvp a.out
4*2  Encoder

I4  I3  I2  I1          O2  O1
0   0   0   0           0   0
0   0   0   1           0   0
0   0   1   0           1   0
0   1   0   0           0   1
1   0   0   0           1   1

C:\iverilog\bin>
```

Fig 9 Output for 4:2 encoder.

EXPERIMENT – 10 (Flip-Flops)

AIM: - To design and verify the operation of D flip-flops using logic gates.

module dff(d,clk,q,qn); input d,clk;

output q,qn; reg q,qn;

dffComp ff(q,qn,clk,d);

initial begin q=0; qn=1; end always @(posedge clk) begin

end endmodule

q = d; qn = !d;

module dffComp(q,qn,clk,d); input q,qn;

output clk,d; reg clk,d; initial begin

clk=0;

d=0; #9 d=1; #1 d=0; #1 d=1; #2 d=0; #1 d=1; #12 d=0;

#1 d=1; #2 d=0; #1 d=1; #1 d=0; #1 d=1; #1 d=0; # 7 d=1;

#8 $finish;

end

always begin

#4 clk=!clk;

end initial begin

$monitor("q=%d\tqn=%d\tclk=%d\td=%d\t",q,qn,clk,d); end

endmodule

OUTPUT: -

```
C:\iverilog\bin>vvp a.out
q=0     qn=1    clk=0   d=0
q=0     qn=1    clk=1   d=0
q=0     qn=1    clk=0   d=0
q=0     qn=1    clk=0   d=1
q=0     qn=1    clk=0   d=0
q=0     qn=1    clk=0   d=1
q=1     qn=0    clk=1   d=1
q=1     qn=0    clk=1   d=0
q=1     qn=0    clk=1   d=1
q=1     qn=0    clk=0   d=1
q=1     qn=0    clk=1   d=1
q=1     qn=0    clk=0   d=1
q=1     qn=0    clk=0   d=0
q=1     qn=0    clk=0   d=1
q=1     qn=0    clk=1   d=1
q=1     qn=0    clk=1   d=0
q=1     qn=0    clk=1   d=1
q=1     qn=0    clk=1   d=0
q=1     qn=0    clk=0   d=1
q=1     qn=0    clk=0   d=0
q=0     qn=1    clk=1   d=0
q=0     qn=1    clk=0   d=1
q=1     qn=0    clk=1   d=1
file.v:25: $finish called at 48 (1s)
q=1     qn=0    clk=0   d=1

C:\iverilog\bin>
```

Fig 10 Output for D flip-flops

EXPERIMENT – 11 (Flip-Flops)

AIM: - To design and verify the operation of JK flip-flops using logic gates.

```
module jkff(J, K, clk, q, qn); input J, K, clk;

output q, qn; reg q, qn; initial begin

q = 0;

qn = 1; end

always @(posedge clk) begin case ({J, K})

2'b00: q = q;

2'b01: q = 0;

2'b10: q = 1;

2'b11: q = ~q; endcase

qn = ~q; end

endmodule

module jkff_tb; reg J, K, clk; wire q, qn;

jkff uut (.J(J), .K(K), .clk(clk), .q(q), .qn(qn)); initial begin

clk = 0;

J = 0; K = 0; #10;

J = 0; K = 1; #10;

J = 1; K = 0; #10;

J = 1; K = 1; #10;

J = 1; K = 1; #10;

J = 0; K = 0; #10;

J = 1; K = 1; #10;

$finish; end

always #5 clk = ~clk; initial begin

$monitor("Time=%0d J=%b K=%b clk=%b q=%b qn=%b", $time, J, K, clk, q, qn);

end endmodule
```

OUTPUT: -

```
C:\iverilog\bin>vvp a.out
Time=0 J=0 K=0 clk=0 q=0 qn=1
Time=5 J=0 K=0 clk=1 q=0 qn=1
Time=10 J=0 K=1 clk=0 q=0 qn=1
Time=15 J=0 K=1 clk=1 q=0 qn=1
Time=20 J=1 K=0 clk=0 q=0 qn=1
Time=25 J=1 K=0 clk=1 q=1 qn=0
Time=30 J=1 K=1 clk=0 q=1 qn=0
Time=35 J=1 K=1 clk=1 q=0 qn=1
Time=40 J=1 K=1 clk=0 q=0 qn=1
Time=45 J=1 K=1 clk=1 q=1 qn=0
Time=50 J=0 K=0 clk=0 q=1 qn=0
Time=55 J=0 K=0 clk=1 q=1 qn=0
Time=60 J=1 K=1 clk=0 q=1 qn=0
Time=65 J=1 K=1 clk=1 q=0 qn=1
file.v:38: $finish called at 70 (1s)
Time=70 J=1 K=1 clk=0 q=0 qn=1

C:\iverilog\bin>
```

Fig 11 Output for JK flip-flops

EXPERIMENT – 12 (Counter)

AIM: - To verify the operation of asynchronous counter.

CODE: -

```
module cntr4bit();

reg clock, reset, enable; wire [3:0] counter_out;

initial begin

$display ("time\t clk reset enable counter");

$monitor ("%g\t %b %b %b %b",

$time, clock, reset, enable, counter_out); clock = 1;

reset = 0;

enable = 0;

#5 reset = 1;

#10 reset = 0;

#10 enable = 1;

#100 enable = 0;

#30 enable=1;

#50 $finish; end

always begin

#5 clock = ~clock; end

cntr4bits cntr ( clock,

reset, enable, counter_out

);

endmodule

module cntr4bits (clock,reset,enable,counter_out); input clock ;

input reset ; input enable ;

output [3:0] counter_out ; wire clock ;

wire reset ; wire enable ;

reg [3:0] counter_out ; always @ (posedge clock) begin :Counter

if (reset == 1) begin counter_out = #1 4'b0000; end

else if (enable == 1) begin counter_out = #1 counter_out + 1; end

end endmodule
```

OUTPUT: -

```
C:\iverilog\bin>vvp a.out
time     clk reset enable counter
0        1   0     0      xxxx
5        0   1     0      xxxx
10       1   1     0      0000
15       0   0     0      0000
20       1   0     0      0000
25       0   0     1      0000
30       1   0     1      0001
35       0   0     1      0001
40       1   0     1      0010
45       0   0     1      0010
50       1   0     1      0011
55       0   0     1      0011
60       1   0     1      0100
65       0   0     1      0100
70       1   0     1      0101
75       0   0     1      0101
80       1   0     1      0110
85       0   0     1      0110
90       1   0     1      0111
95       0   0     1      0111
100      1   0     1      1000
105      0   0     1      1000
110      1   0     1      1001
115      0   0     1      1001
120      1   0     1      1010
125      0   0     0      1010
130      1   0     0      1010
135      0   0     0      1010
140      1   0     0      1010
145      0   0     0      1010
150      1   0     0      1010
155      0   0     1      1010
160      1   0     1      1011
165      0   0     1      1011
170      1   0     1      1100
175      0   0     1      1100
180      1   0     1      1101
185      0   0     1      1101
190      1   0     1      1110
195      0   0     1      1110
200      1   0     1      1111
file.v:17: $finish called at 205 (1s)
205      0   0     1      1111

C:\iverilog\bin>
```

Fig 12 Output for asynchronous counter