QMP 7.1 D/F

Channabasaveshwara Institute of Technology
(Affiliated to VTU, Belagavi & Approved by AICTE, New Delhi)
(ISO 9001:2015 Certified Institution)
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.

Department of Electronics & Communication Engineering

Digital System Design Using Verilog

21EC32

(CBCS SCHEME)

B.E - III Semester

Lab Manual 2022-23

Name: _____

USN: _____

Batch:_____Section: _____

.

QMP 7.1 D/F

Channabasaveshwara Institute of Technology
(Affiliated to VTU, Belagavi & Approved by AICTE, New Delhi)
(**ISO 9001:2015 Certified Institution)**
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.

Department of Electronics & Communication Engineering

Digital System Design Using Verilog

PRACTICAL COMPONENT OF IPCC

Prepared by:

Mrs. Tejaswini S

Reviewed by:

Mr. Sanjeev Kumar Harihar

Approved by:

Dr. Sekar R

HOD, Dept. of ECE

.

Channabasaveshwara Institute of Technology

(Affiliated to VTU, Belagavi & Approved by AICTE, New Delhi)
(ISO 9001:2015 Certified Institution)
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.

Department of Electronics & Communication Engineering

Department Vision

To create globally competent Electronics and Communication Engineering professionals with ethical and moral values for the betterment of the society.

Department Mission

- To nurture the technical/professional/engineering and entrepreneurial skills for overall self and societal upliftment through co-curricular and extra-curricular events.
- To orient the Faculty/Student community towards the higher education, research and development activities.
- To create the Centres of Excellence in the field of electronics and communication in collaboration with industries/Universities by training the faculty through latest technologies.
- To impart quality technical education in the field of electronics and communication engineering to meet over the current/future global industry requirements.

PROGRAM EDUCATIONAL OBJECTIVES (PEO's)

After four Years of Graduation, our graduates are able to:

- Provide technical solutions to real world problems in the areas of electronics and communication by developing suitable systems.
- Pursue engineering career in Industry and/or pursue higher education and research.
- Acquire and follow best professional and ethical practices in Industry and Society.
- Communicate effectively and have the ability to work in team and to lead the team.

PROGRAM SPECIFIC OUTCOMES (PSO'S)

At the end of the B.E Electronics & Communication Engineering program, students are expected to have developed the following program specific outcomes.

PSO1: Build Analog and Digital Electronic systems for Multimedia Applications, VLSI and Embedded Systems in Interdisciplinary Research / Development.

PSO2: Design and Develop Communication Systems as per Real Time Applications and Current Trends.

Department of Electronics & Communication Engineering

SYLLABUS

Digital System Design Using Verilog
PRACTICAL COMPONENT OF IPCC
[As per Choice Based Credit System (CBCS) scheme]
SEMESTER – III (EC)

Subject Code: 21EC32 **CIE Marks: 20**
Hours/ Week : 02 (02 Hours Laboratory) **Test Hours: 03**

Using suitable simulation software, demonstrate the operation of the following circuits: Using Xilinx Tool)

Sl.No	Experiments
1.	To simplify the given Boolean expressions and realize using Verilog program
2.	To realize Adder/Subtractor (Full/half) circuits using Verilog data flow description.
3.	To realize 4-bit ALU using Verilog program.
4.	To realize the following Code converters using Verilog Behavioral description a) Gray to binary and vice versa b) Binary to excess3 and vice versa
5.	To realize using Verilog Behavioral description: 8:1 mux, 8:3 encoder, Priority encoder
6.	To realize using Verilog Behavioral description: 1:8 Demux, 3:8 decoder, 2-bit Comparator
7.	To realize using Verilog Behavioral description: Flip-flops: a) JK type b) SR type c) T type and d) D type
8.	To realize Counters - up/down (BCD and binary) using Verilog Behavioral description.
Demonstration Experiments (For CIE only – not to be included for SEE) Use FPGA/CPLD kits for downloading Verilog codes and check the output for interfacing experiments.	
9.	Verilog Program to interface a Stepper motor to the FPGA/CPLD and rotate the motor in the specified direction (by N steps).
10.	Verilog programs to interface a Relay or ADC to the FPGA/CPLD and demonstrate its working.
11.	Verilog programs to interface DAC to the FPGA/CPLD for Waveform generation.
12.	Verilog programs to interface Switches and LEDs to the FPGA/CPLD and demonstrate its working.

Channabasaveshwara Institute of Technology
(Affiliated to VTU, Belagavi & Approved by AICTE, New Delhi)
(ISO 9001:2015 Certified Institution)
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.

DEPARTMENT OF ELECTRONICS AND COMMUNICATION

TABLE OF CONTENTS

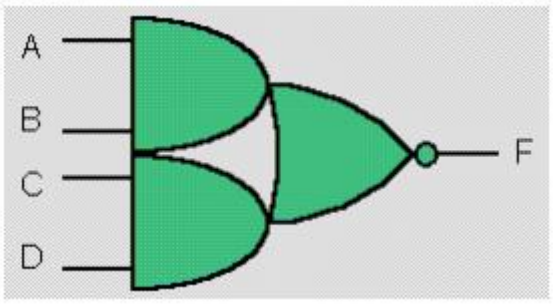
INDEX PAGE

Sl. No	Name of the Experiment	Date			Manual Marks (Max . 15)	Record Marks (Max. 10)	Signature (Student)	Signature (Faculty)
		Conduction	Repetition	Submission of Record				
1.								
2.								
3.								
4.								
5.								
6.								
7.								
8.								
9.								
10.								
11.								
12.								
Average								

General Instructions to Students

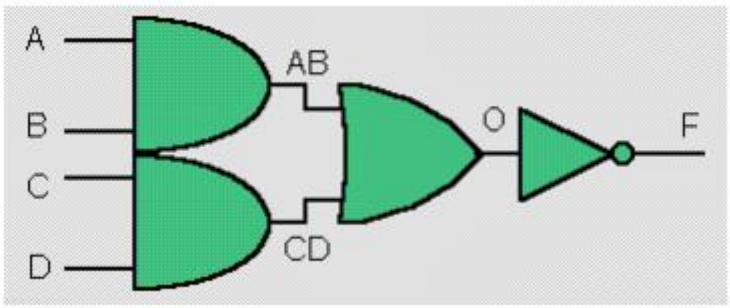
1. Students should come with thorough preparation for the experiment to be conducted.
2. Students should take prior permission from the concerned faculty before availing the leave.
3. Students should come with formals and to be present on time in the laboratory.
4. Students will not be permitted to attend the laboratory unless they bring the practical record fully completed in all respects pertaining to the experiments conducted in the previous session.
5. Students will be permitted to attend the laboratory unless they bring the observation book fully completed in all respects pertaining to the experiments conducted in the present session.
6. They should obtain the signature of the staff-in –charge in the observation book after completing each experiment.
7. Practical record should be neatly maintained.
8. Ask lab Instructor for assistance for any problem.
9. Do not download or install software without the assistance of laboratory Instructor.
10. Do not alter the configuration of system.
11. Turn off the systems after use.

a.



F = ~ (AB + CD)

b.



F = ~ (AB + CD)

Result:

Experiment No 1: **Date: __/__/_____**

To simplify the given Boolean expressions and realize using Verilog program.

a. //Using Data-flow Description

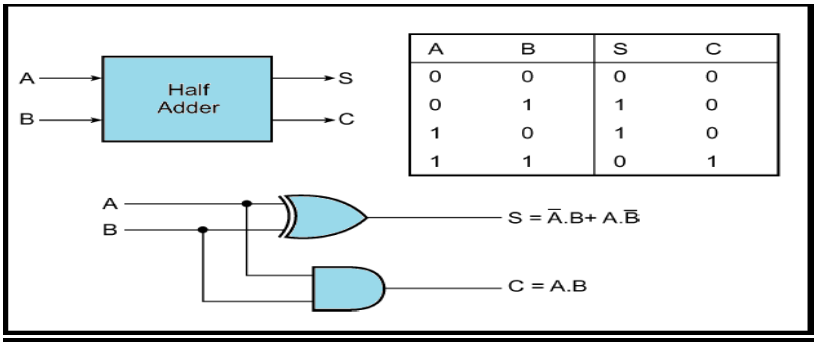
```
module bool_fun(F, A, B, C, D);
        output F;
         input A, B, C, D;
        assign F = ~ ((A & B) | (C & D));
endmodule
// end of Verilog code
```

OR

b. // Using Structural Description

```
module bool_fun(F, A, B, C, D);
        output F;
        input A, B, C, D;
        wire AB, CD, O; // necessary
        and a1(AB, A, B);
        and a2(CD, C, D);
        or o1(O, AB, CD);
        not n1(F, O);
endmodule
```

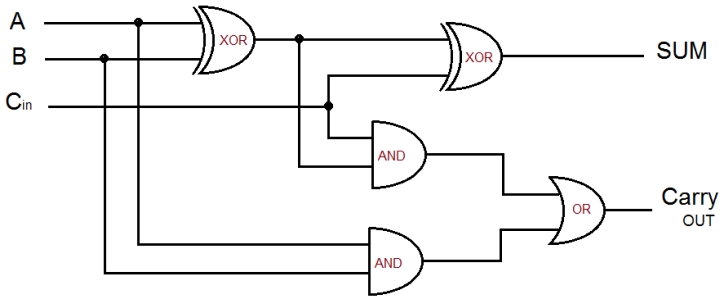
Half Adder (Dataflow Modeling):



A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Full Adder (Dataflow Modeling):

Inputs			Outputs	
A	B	C_{in}	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



SUM = A ^ B ^ Cin

Carry_OUT = AB + C(A^B)

Result:

Experiment No 2: **Date: __/_/_____**

To realize Adder/Subtractor (Full/half) circuits using Verilog data flow description.

a. Half adder

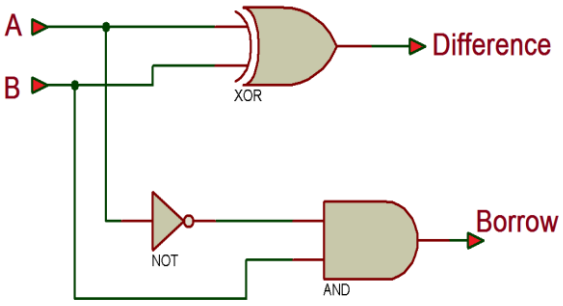
```
module halfadder(
    input A,
    input B,
    output S,
    output C );
assign S = A ^ B ;
assign C = A & B;
endmodule
```

b. Full adder

```
module fulladder(A, B, Cin, Sum, Carry_out);
input A, B, Cin;
output Sum, Carry_out;
assign Sum = A ^ B ^ Cin;
assign Carry_out = (A & B)|(B & Cin)|(A & Cin);
endmodule
```

Half Subtractor:

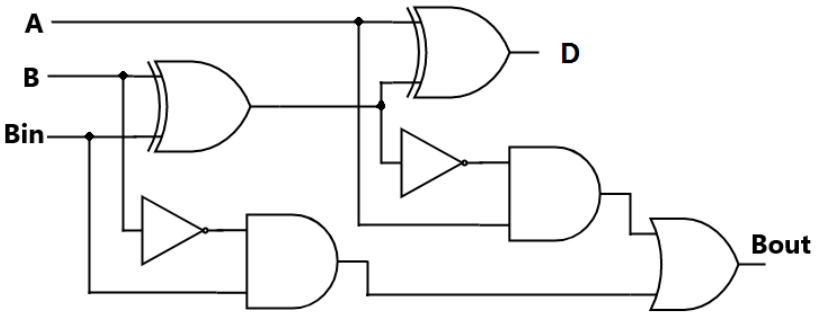
Inputs		Outputs	
A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0



Difference = A ^ B

Borrow = ~A & B

Full Subtractor:



A	B	B_{in}	D	B_{out}
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

D = A ^ B ^ Cin

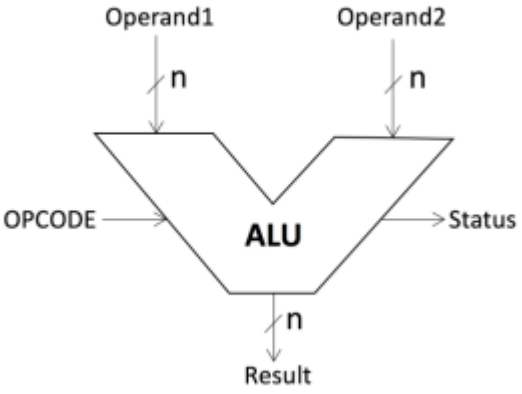
Bout = A′B + C(A^B)′

c. Half Subtractor

```
module halfsubtractor(
    input a,
    input b,
    output difference,
    output borrow );
assign difference = a ^ b ;
assign borrow = ~a & b;
endmodule
```

d. Full Subtractor

```
module fullsubtractor(a, b, bin, D, bout);
input a, b,  bin;
output D, bout;
assign D = a ^ b ^ bin;
assign bout = (~a & b)|(~a & bin)|(b & bin);
endmodule
```



Opcode	ALU Operation	
000	a + b	Addition of two numbers
001	a - b	Subtraction of two numbers
010	a << 1	Left shift a by 1position
011	a * b	Multiplication of two numbers
100	a && b	Logical AND
101	a || b	Logical OR
110	! a	Negation a
111	~ a	Complment a

The 4 - bit ALU has the following inputs:

- A: 4-bit input
- B: 4-bit input
- Output: 8-bit output
- Control: 3-bit control input

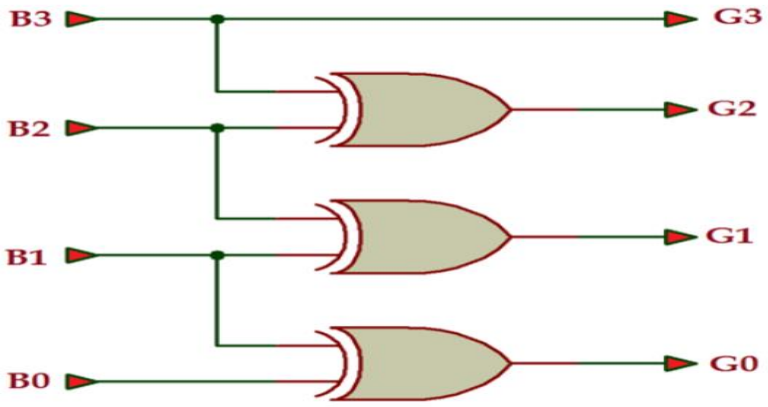
Result:

Experiment No 3: **Date: __/__/_____**

To realize 4-bit ALU using Verilog program.

```
//4 bit ALU
module p13(z,a,b,sel);
input [3:0]a,b;
input [2:0]sel;
output [7:0]z;
 reg [7:0]z;
always@(sel,a,b)
begin
case(sel)
3'b000: z=a+b;
3'b001: z=a-b;
3'b010: z=a<<1;
3'b011: z=a*b;
3'b100: z=a&&b;
3'b101: z=a||b;
3'b110: z=!a;
3'b111: z=~a;
endcase
end
endmodule
```

Binary to Gray:



B3	B2	B1	B0	Binary code	G3	G2	G1	G0	Gray code
0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	1	1
0	0	1	0	2	0	0	1	1	3
0	0	1	1	3	0	0	1	0	2
0	1	0	0	4	0	1	1	0	6
0	1	0	1	5	0	1	1	1	7
0	1	1	0	6	0	1	0	1	5
0	1	1	1	7	0	1	0	0	4
1	0	0	0	8	1	1	0	0	12
1	0	0	1	9	1	1	0	1	13
1	0	1	0	10	1	1	1	1	15
1	0	1	1	11	1	1	1	0	14
1	1	0	0	12	1	0	1	0	10
1	1	0	1	13	1	0	1	1	11
1	1	1	0	14	1	0	0	1	9
1	1	1	1	15	1	0	0	0	8

$G_3=\sum m\ (8, 9, 10, 11, 12, 13, 14, 15)$

$G_2=\sum m\ (4, 5, 6, 7, 8, 9, 10, 11)$

$G_1=\sum m\ (2, 3, 4, 5, 10, 11, 12, 13)$

$G_0=\sum m\ (1, 2, 5, 6, 9, 10, 13, 14)$

$G_3 = B_3$

$G_2 = B_3 \wedge B_2$

$G_1 = B_2 \wedge B_1$

$G0 = B_1 \wedge B_o$

Experiment No 4: **Date: __/_/_____**

To realize the following Code converters using Verilog Behavioral description.

a) Gray to binary and vice versa **b) Binary to excess3 and vice versa**

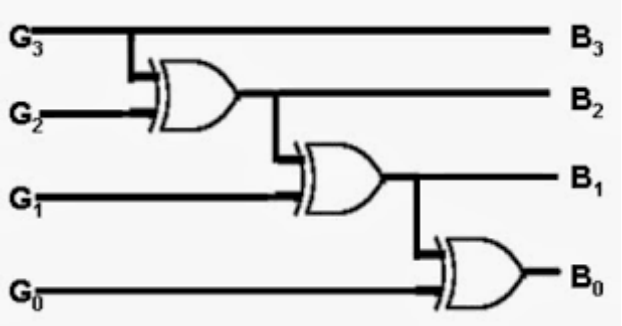
a. Binary to Gray

```
//Dataflow model
module binary_to_gray(b_in, g_op);
input [3:0] B;
output  [3:0]  G;
assign G[3] = B[3];
assign G[2] = B[3] ^ B[2];
assign G[1] = B[2] ^ B[1];
assign G[0] = B[1] ^ B[0];
endmodule
```

OR

```
// Behavioral model
module bin_gray(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
```

Gray to Binary:



Truth Table:

Gray Code Input				Binary Code Output			
G3	G2	G1	G0	B3	B2	B1	B0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0
1	1	1	0	1	0	1	1
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	0	0	1	1	1	1	0
1	0	0	0	1	1	1	1

```
b[3]=g[3];
b[2]=b[3]^g[2];
b[1]=b[2]^g[1];
b[0]=b[1]^g[0];
```

Result:

b. Gray to Binary

```
module graytobinary(g,b); //dataflow model
input [3:0] g;
output [3:0] b;
assign b[3]=g[3];
assign b[2]=b[3]^g[2];
assign b[1]=b[2]^g[1];
assign b[0]=b[1]^g[0];
endmodule
```

OR

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

Binary to Excess-3:

B3	B2	B1	B0	E3	E2	E1	E0
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
1	0	1	0	1	1	0	1
1	0	1	1	1	1	1	0
1	1	0	0	1	1	1	1
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x

E3 = B2(B1 + B0) + B3
E2 = B2′(B1 + B0) + B2B1′B0′
E1 = B1′B0′+B1B0
E0 = B0′

Excess-3 to Binary:

E3	E2	E1	E0	B3	B2	B1	B0
0	0	0	0	x	x	x	x
0	0	0	1	x	x	x	x
0	0	1	0	x	x	x	x
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	1
0	1	0	1	0	0	1	0
0	1	1	0	0	0	1	1
0	1	1	1	0	1	0	0
1	0	0	0	0	1	0	1
1	0	0	1	0	1	1	0
1	0	1	0	0	1	1	1
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	1
1	1	0	1	1	0	1	0
1	1	1	0	1	0	1	1
1	1	1	1	1	1	0	0

B3 = E3E2 + E3E1E0
B2 = E2E1E0 + E2′(E1′+E0′)
B1 = E1′E0 + E1E0′
B0 = E0′

Result:

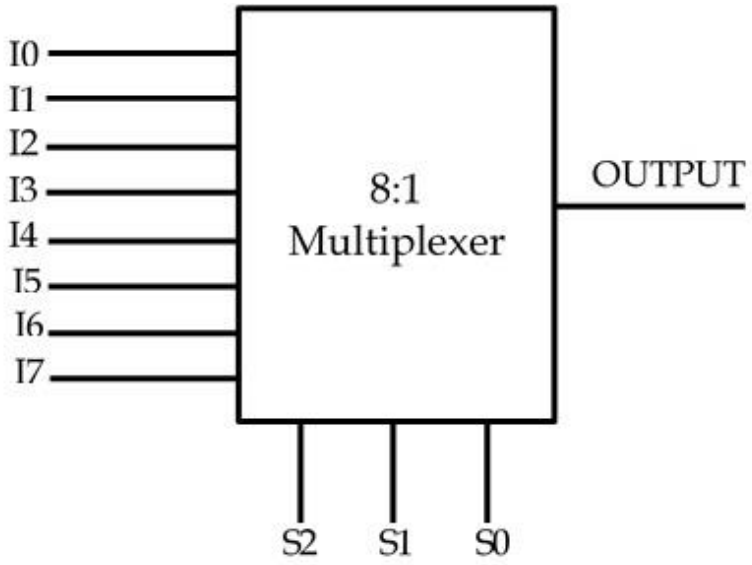
e. Binary to Excess-3

```
module binary2ex3(b,e);
input [3:0]b;
output [3:0]e;
reg [3:0]e;
always@(b)
begin
    e[3] = (b[2]&(b[0] | b[1])) | b[3];
    e2 = (~b[2])&(b[1] | b[0]) | (b2)&(~b1)&(~b0);
    e1 = (~b1)&(~b0) | b1&b0;
    e0 = ~b[0]
 end
endmodule
```

f. Excess-3 to Binary

```
module binary2ex3(b,e);
input [3:0]e;
output [3:0]b;
reg [3:0]b;
always@(e)
begin
    b[3] = (e[3] & e[2]) | (e[3] & e[1] & e[0])
    b2 = (e[2] & e[1] & e[0]) | (~e[2]&(~e[1] | ~e[0])
    b1 = (~e[1] & e[0]) | (e[1] & ~e[0])
    b0 = ~e[0]
 end
endmodule
```

a. 8:1 Mux



Truth Table:

S1	S2	S3	Y
0	0	0	I0
0	0	1	I1
0	1	0	I2
0	1	1	I3
1	0	0	I4
1	0	1	I5
1	1	0	I6
1	1	1	I7

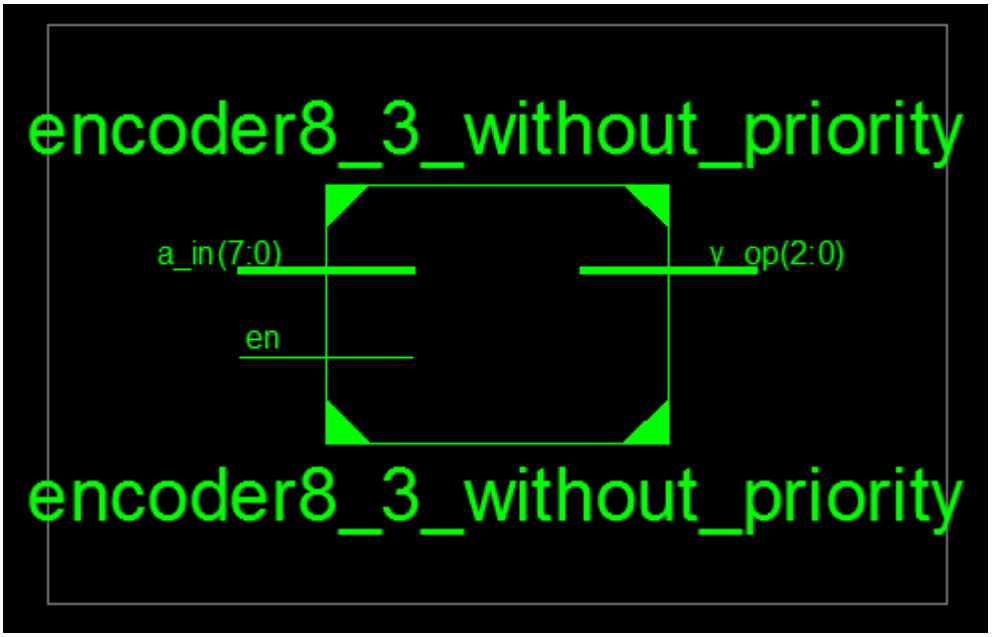
Experiment No 5: **Date: __/__/_____**

To realize using Verilog Behavioral description: 8:1 mux, 8:3 encoder, Priority encoder

a. 8:1 mux

```
module mux8_to_1(i,sel,y);
input [7:0] i;
input [2:0] sel;
output y;
reg y;
        always@(i,sel)
        begin
                case(sel)
                3'b000: y=i[0];
                3'b001: y=i[1];
                3'b010: y=i[2];
                3'b011: y=i[3];
                3'b100: y=i[4];
                3'b101: y=i[5];
                3'b110: y=i[6];
                3'b111: y=i[7];
                default:y=3'b000;
                endcase
                end
                endmodule
```

8:3 encoder



Truth Table:

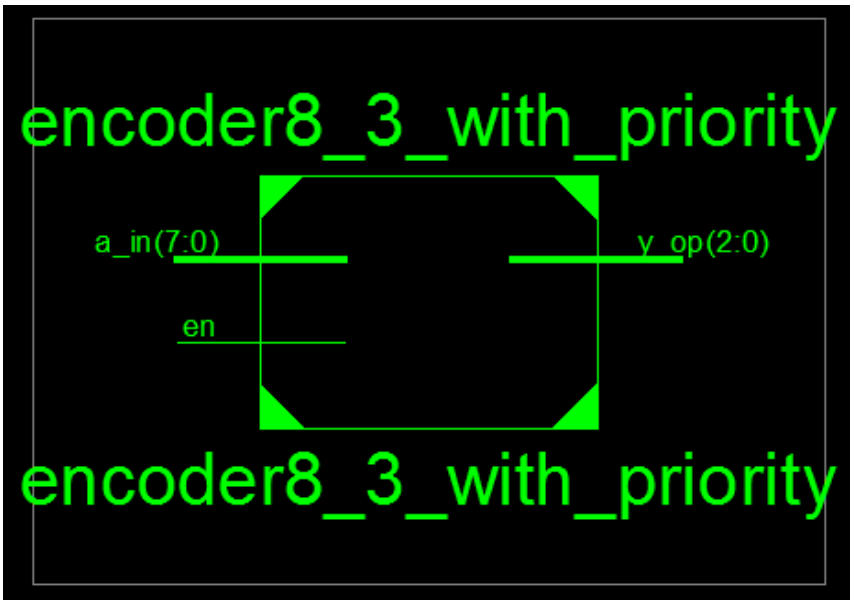
Inputs									Outputs		
en	a_in(7)	a_in(6)	a_in(5)	a_in(4)	a_in(3)	a_in(2)	a_in(1)	a_in(0)	y_op (2)	y_op (1)	y_op (0)
1	X	X	X	X	X	X	X	X	Z	Z	Z
0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	0	1	0	0	0	0	1	1
0	0	0	0	1	0	0	0	0	1	0	0
0	0	0	1	0	0	0	0	0	1	0	1
0	0	1	0	0	0	0	0	0	1	1	0
0	1	0	0	0	0	0	0	0	1	1	1

Result:

b. 8:3 Encoder without priority (Behavioral Modeling)

```
module encoder8_3_without_priority(en, a_in, y_op);
    input en;
    input [7:0] a_in;
    output [2:0] y_op;
    reg [2:0] y_op;
    always @ (a_in,en)
    begin
        if(en==1)
            y_op=3'bzzz;
         else
            case (a_in)
                8'b00000001: y_op = 3'b000;
                8'b00000010: y_op = 3'b001;
                8'b00000100: y_op = 3'b010;
                8'b00001000: y_op = 3'b011;
                8'b00010000: y_op = 3'b100;
                8'b00100000: y_op = 3'b101;
                8'b01000000: y_op = 3'b110;
                8'b10000000: y_op = 3'b111;
                default: y_op = 3'bxxx;
             endcase
    end
    endmodule
```

c. Priority encoder



Truth Table:

Inputs									Outputs		
en	a_in(7)	a_in(6)	a_in(5)	a_in(4)	a_in(3)	a_in(2)	a_in(1)	a_in (0)	y_op (2)	y_op (1)	y_op (0)
1	X	X	X	X	X	X	X	X	Z	Z	Z
0	1	X	X	X	X	X	X	X	1	1	1
0	0	1	X	X	X	X	X	X	1	1	0
0	0	0	1	X	X	X	X	X	1	0	1
0	0	0	0	1	X	X	X	X	1	0	0
0	0	0	0	0	1	X	X	X	0	1	1
0	0	0	0	0	0	1	X	X	0	1	0
0	0	0	0	0	0	0	1	X	0	0	1
0	0	0	0	0	0	0	0	1	0	0	0

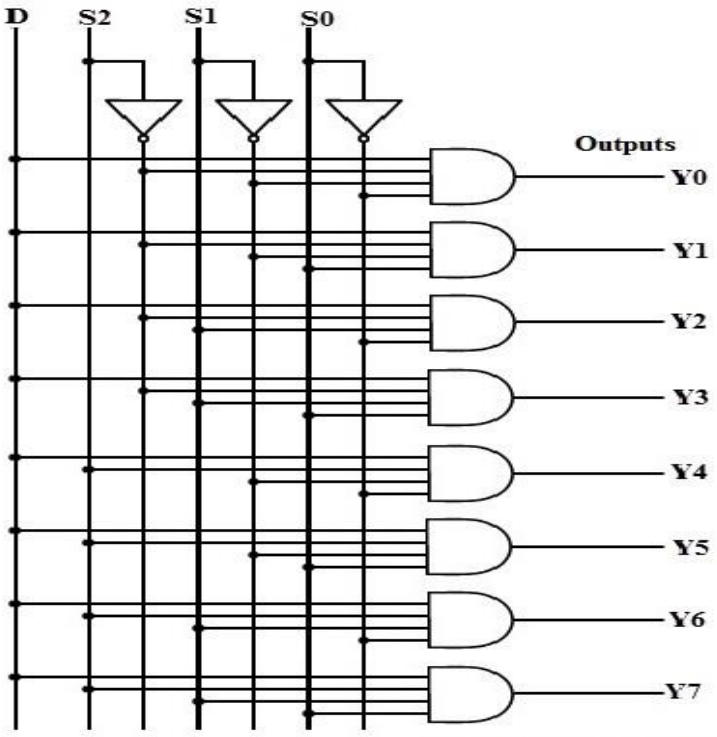
Result:

C. Realization of 8:3 Encoder with priority(Behavioral Modeling)

```
module encoder8_3_with_priority(en, a_in, y_op);
input en;
input [7:0] a_in;
output [2:0] y_op;
reg [2:0] y_op;
always @ (a_in,en)
begin
if(en==1)
y_op=3'bzzz;
else
casex(a_in)
      8'b00000001: y_op= 3'b000;
      8'b0000001x: y_op= 3'b001;
      8'b000001xx:  y_op=  3'b010;
      8'b00001xxx:  y_op=  3'b011;
      8'b0001xxxx:  y_op=  3'b100;
      8'b001xxxxx:  y_op=  3'b101;
      8'b01xxxxxx:  y_op=  3'b110;
      8'b1xxxxxxx: y_op= 3'b111;
            default: y_op=3'bxxx;
endcase
end
endmodule
```

1:8 Demux:

Data Input	Select Inputs			Outputs							
D	S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
D	0	0	0	0	0	0	0	0	0	0	D
D	0	0	1	0	0	0	0	0	0	D	0
D	0	1	0	0	0	0	0	0	D	0	0
D	0	1	1	0	0	0	0	D	0	0	0
D	1	0	0	0	0	0	D	0	0	0	0
D	1	0	1	0	0	D	0	0	0	0	0
D	1	1	0	0	D	0	0	0	0	0	0
D	1	1	1	D	0	0	0	0	0	0	0



Result:

Experiment No: 6 **Date: __/_/_____**

To realize using Verilog Behavioral description:

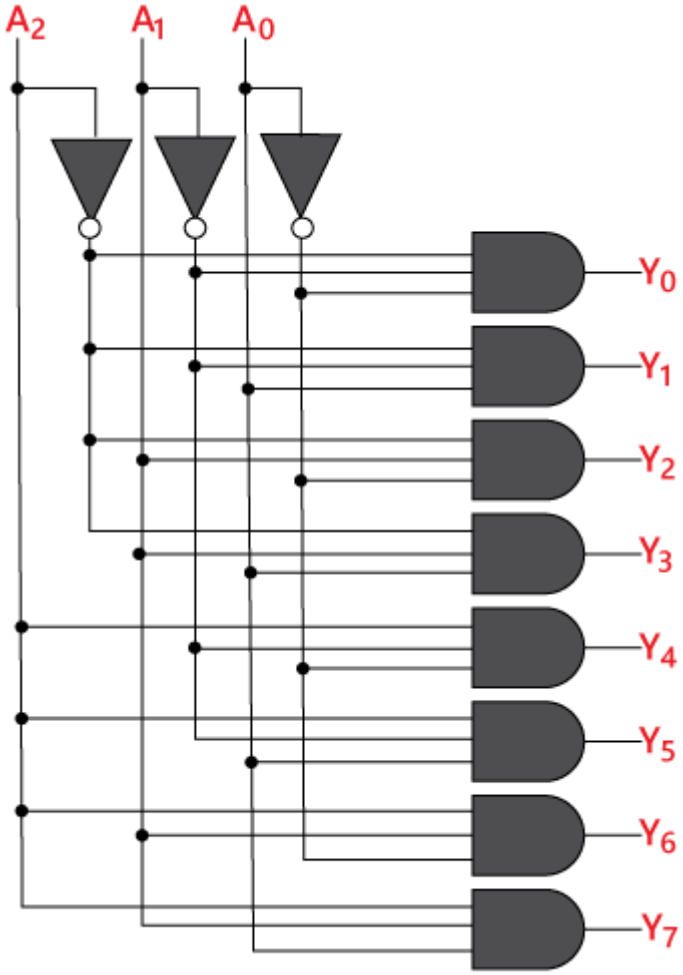
1:8 Demux, 3:8 decoder, 2-bit Comparator

a. 1:8 Demux

```
module 1_8_demux(
input D,
input s2 ,s1, s0,
output [7:0] y);

reg [7:0]y;
always @ (i or s0 or s1 or s2)
case ({s2,s1,s0})

0 : y0 = D;
1 : y1 = D;
2 : y2 = D;
3 : y3 = D;
4 : y4 = D;
5 : y5 = D;
6 : y6 = D;
7 : y7 = D;
default : y = 8'bxxxxxxx;
endcase
endmodule
```

3:8 Decoder:

Enable	INPUTS			Outputs							
E	A_2	A_1	A_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0



b. 3:8 Decoder:

```
module 3_8_DEC(
   input [3:0]A,
   output [7:0]Y
   );
reg [7:0]Y;
always @ (A)

case (A)
   0 : Y[0] = 1;
   1 : Y[1] = 1;
   2 : Y[2] = 1;
   3 : Y[3] = 1;
   4 : Y[4] = 1;
   5 : Y[5] = 1;
   6 : Y[6] = 1;
   7 : Y[7] = 1;
   default : Y = 8'bxxxxxxxx;
endcase
endmodule
```

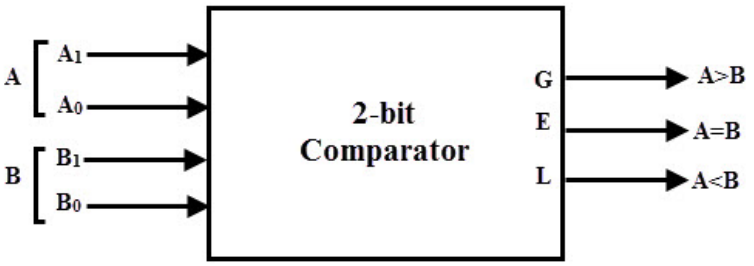
2-bit Comparator:

- ✓ Input: 2-bit A and B for comparison
- ✓ Output:

  A_greater_B: high if A > B else low

  A_equal_B: high if A = B else low

  A_less_B: high if A<B else low



Inputs				Outputs		
A_1	A_0	B_1	B_0	A>B	A=B	A<B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	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	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

(A > B) = A1 B1' + A0 B1' B0' + A0 A1 B0'

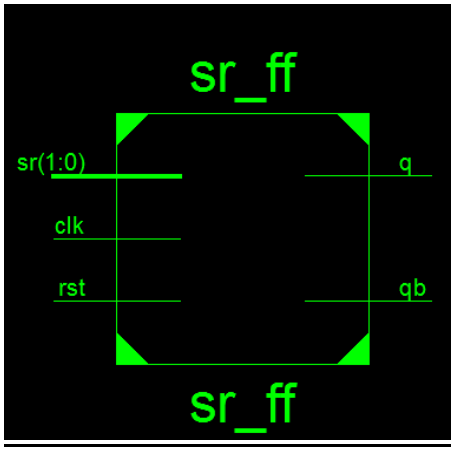
(A < B) = A1' B1 + A0' B1 B0 + A0' B1 B0

(A = B) = (A1 ʘ B1)(A0 ʘ B0)

c. 2-bit Comparator:

```
module compr_2(a,b,altb,aeqb,agtb);
input [1:0] a,b;
output reg altb,aeqb,agtb;
always @(a or b)
begin
 agtb = (A[1] & ~B[1]) | (A[0] & ~B[1] & ~B[0] ) | (A[0] & A[1] & ~B[0])
 altb = (~A[1] & B[1])  | (~A[0] & B[1] & B[0]) + (~A[0] & B[1] & B[0])
 aeqb = ~ (A[ 1] ^ B[1]) & ~(A[0] ^ B[0])
end
endmodule
```

RTL Schematic:



Truth Table:

Inputs				Outputs		
rst	clk	s	r	q	qb	Action
1	↑	X	X	q	qb	No Change
0	↑	0	0	q	qb	No Change
0	↑	0	1	0	1	Reset
0	↑	1	0	1	0	Set
0	↑	1	1	-	-	Illegal

Result:

Experiment No 7: **Date: __/__/_____**

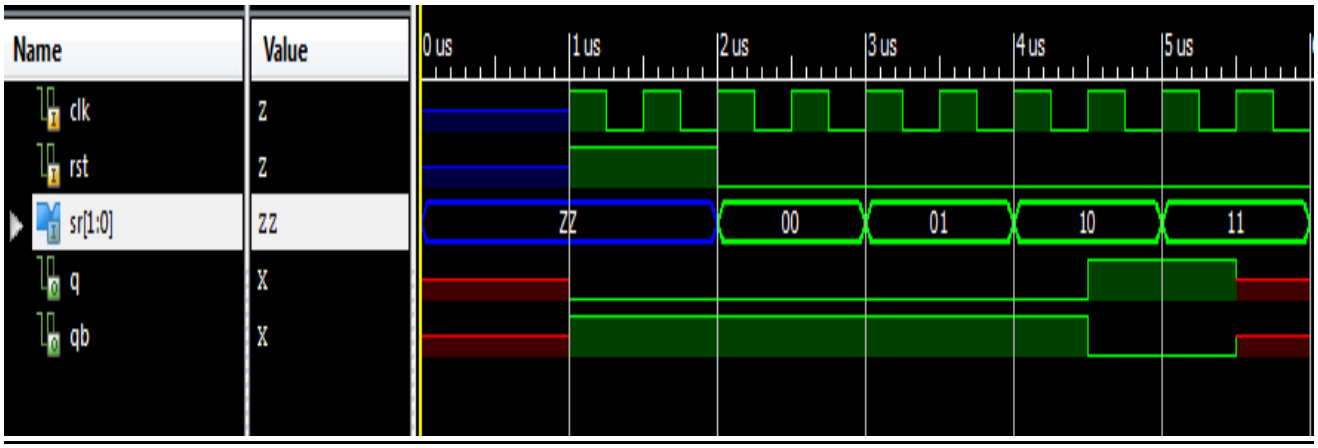
To realize using Verilog Behavioral description:

Flip-flops: a) JK type b) SR type c) T type and d) D type

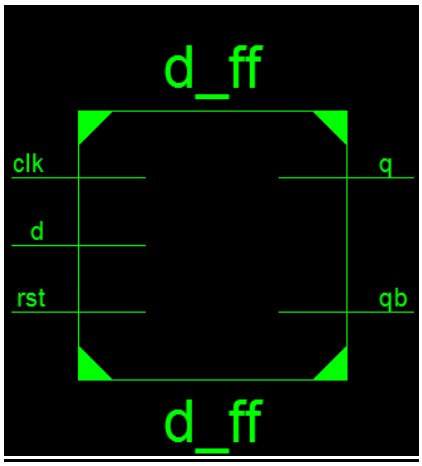
Verilog Code to Describe SR Flipflop:

```
module sr_ff(sr, clk, rst, q, qb);
input [1:0]sr;
input rst, clk;
output q,qb;
reg q,qb;
always @ (posedge clk)
begin
        if (rst==1)
        begin
                q=0;
                qb=1;
         end
         else   case (sr)
                        2'b00: begin q=q; qb=qb; end
                        2'b01:  begin q=0;  qb=1; end
                        2'b10:  begin q=1;  qb=0; end
                        2'b11: begin q=1'bx; qb=1'bx; end
                  endcase
               endmodule
```

Simulation Result:



RTL Schematic



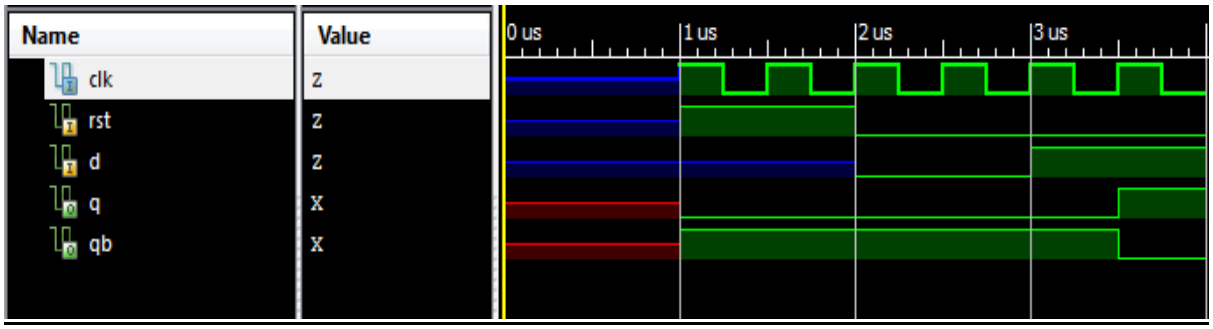
Truth Table:

Inputs			Outputs		
rst	clk	d	q	qb	Action
1	↑	X	q	qb	No Change
0	↑	0	0	1	Reset
0	↑	1	1	0	Set

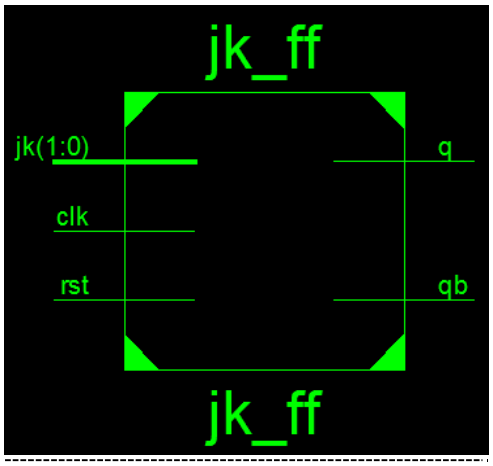
Verilog Code to Describe D Flipflop

```
module d_ff(d, rst, clk, q, qb);
input d;
input rst;
input clk;
output q;
output qb;
reg q,qb;
always@(posedge clk)
begin
        if (rst == 1)
        begin
                q = 0;

                qb = 1;
        end
        else
        begin
                q = d;
                qb = ~d;
        end
end
endmodule
```

Simulation Result:



RTL Schematic:



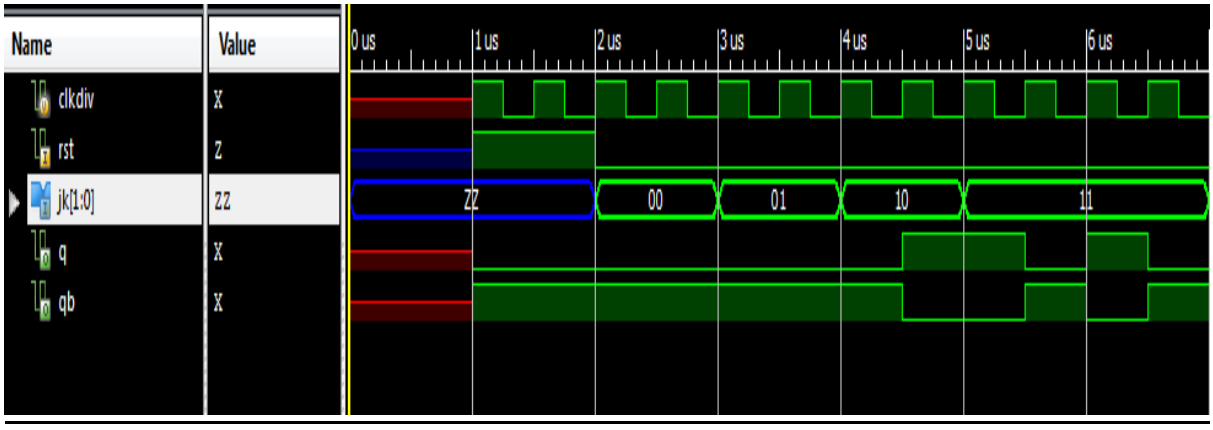
Truth Table:

Inputs				Outputs		
rst	clk	j	k	q	qb	Action
1	↑	X	X	q	qb	No Change
0	↑	0	0	q	qb	No Change
0	↑	0	1	0	1	Reset
0	↑	1	0	1	0	Set
0	↑	1	1	q'	q'	Toggle

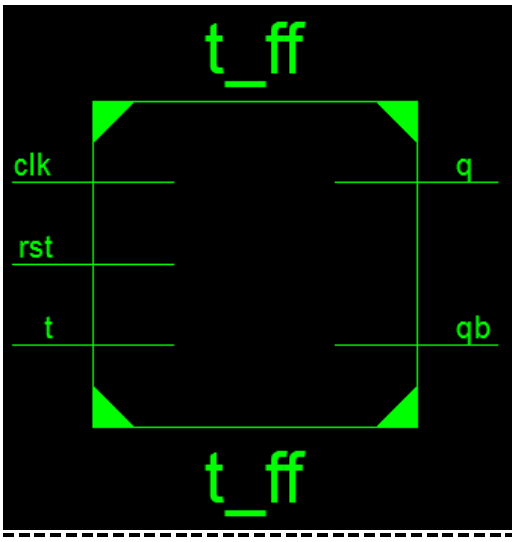
Verilog Code to Describe JK Flipflop

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

Simulation Result:



RTL Schematic:



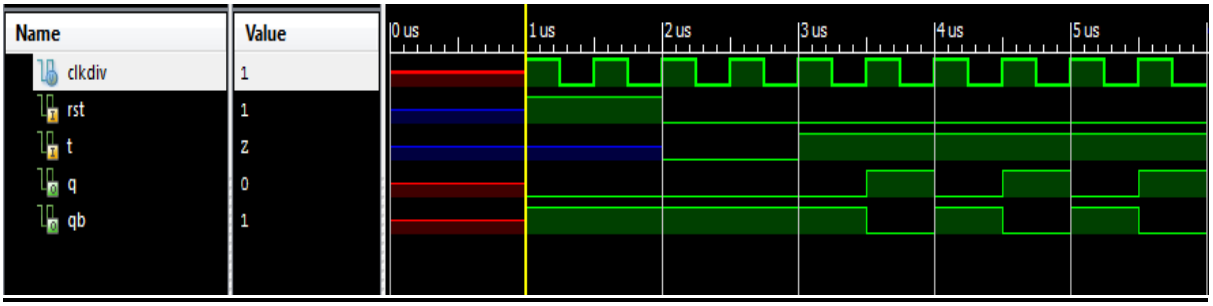
Truth Table:

Inputs			Outputs		
rst	clk	t	q	qb	Action
1	↑	X	q	qb	No Change
0	↑	0	q	qb	No Change
0	↑	1	q'	q'	Toggle

Verilog Code to Describe T Flipflop

```
module t_ff(t, clk, rst, q, qb);
input t, clk, rst;
output q, qb;
reg q,qb;

always @ (posedge clk)
begin
            if (rst==1)
             begin
                q=1'b0;
                qb=1'b1;
              end
              else
                    case (t)
                  1'b0:begin q=q; qb=qb; end
                  1'b1:begin q=~(q); qb=~(qb); end
                  endcase
end
endmodule
```

Simulation Result:

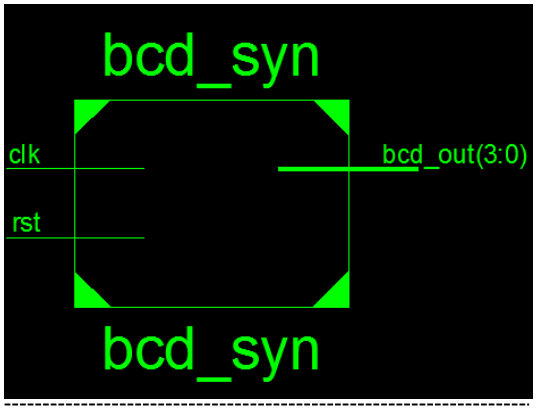


Experiment No 8: **Date: __/__/_____**

To realize Counters - up/down (BCD and binary) using Verilog Behavioral description.

```
//Verilog module for UpDown counter
//When Up mode is selected, counter counts from 0 to 15 and then again from 0 to 15.
//When Down mode is selected, counter counts from 15 to 0 and then again from 15 to 0.
//Changing mode doesn't reset the Count value to zero.
//You have apply high value to reset, to reset the Counter output.

module upordown_counter(
   Clk,
   reset,
   UpOrDown,  //high for UP counter and low for Down counter
   Count
   );


   //input ports and their sizes
   input Clk,reset,UpOrDown;
   //output ports and their size
   output [3 : 0] Count;
   //Internal variables
   reg [3 : 0] Count = 0;

    always @(posedge(Clk) or posedge(reset))
    begin
      if(reset == 1)
         Count <= 0;
      else
         if(UpOrDown == 1)   //Up mode selected
            if(Count == 15)
               Count <= 0;
            else
               Count <= Count + 1; //Incremend Counter
         else  //Down mode selected
            if(Count == 0)
               Count <= 15;
            else
               Count <= Count - 1; //Decrement counter
   end

endmodule
```

RTL Schematic:



Truth table:

Clk	Rst	bin_out (3)	bin_out (2)	bin_out (1)	bin_out (0)
↑	1	0	0	0	0
↑	0	0	0	0	1
↑	0	0	0	1	0
↑	0	0	0	1	1
↑	0	0	1	0	0
↑	0	0	1	0	1
↑	0	0	1	1	0
↑	0	0	1	1	1
↑	0	1	0	0	0
↑	0	1	0	0	1
↑	0	0	0	0	0
↑	0	0	0	0	1
↑	0	0	0	1	0

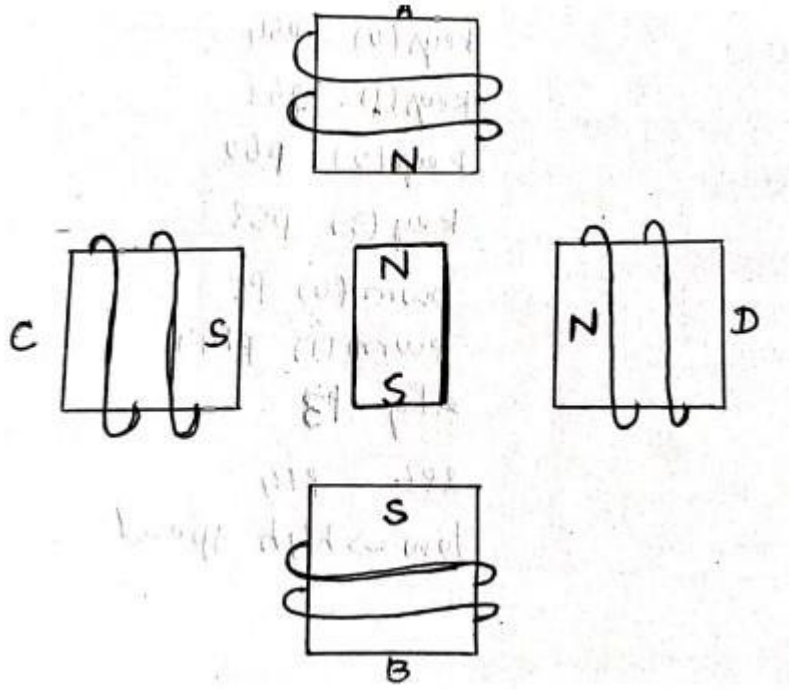
Verilog Code to design 4 –bit BCD Synchronous reset counter

```
module bcd_syn( clk, rst, bcd_out);
input clk, rst;
output [3:0] bcd_out;
reg [3:0] bcd_out;
reg [22:0] div;
reg clkdiv;
always @ (posedge clk)
begin
        div = div+1'b1;
        clkdiv = div[22];
end
always @ (posedge clkdiv)
begin
        if (rst)
                bcd_out=4’d0;
       else if(bcd_out<4’d9)
                bcd_out=bcd_out+4’d1;
        else
                bcd_out=4’d0;
end
               endmodule
```

PART B
Demonstration Experiments

(For CIE only – not to be included for SEE)

Hardware Details:



Result:

Experiment No 9: **Date: __/_/_____**

Verilog Code to control direction of Stepper Motor

```
module step_mot(clk,reset,dir,  stepout);
input clk,reset,dir;
output [3:0] stepout;
reg [25:0] div;
reg [3:0] shift_reg;

always @ (posedge clk)
begin
        div = div + 1'b1;
end

always @ (posedge div[15])
begin
          if (reset==1'b1)
             shift_reg="0001";
          else
            begin

                    if (dir==1'b0)

                          shift_reg= {shift_reg[2:0],shift_reg [3]};
                    else
                           shift_reg= {shift_reg[0],shift_reg [3:1]};
             end
end

assign stepout = shift_reg;

endmodule
```

Procedure:

1. Make the connection between FRC9 of the FPGA board and steppermotor connector of interfacing card
2. Make the connection between FRC1 of the FPGA board and DIP switch connector of the GPIOcard-2.
3. Assign appropriate pins to input and output.
4. Connect USB cable and power supply to the FPGA board.

Experiment No 10 : Date: __/__/_____

Verilog programs to interface a Relay or ADC to the FPGA/CPLD and demonstrate its working.

```
module relays ( light, control1,control2);
input control1,control2;
output light;
always @ (cntrl1,cntrl2)
begin
light&lt;= cntrl1 OR cntrl2 ;
end
endmodule
```

UCF file(User constraint)

```
NET &quot;cntrl1&quot; LOC = &quot;P74&quot;
NET &quot;cntrl2&quot; LOC = &quot;P76&quot;
NET &quot;light&quot; LOC = &quot;P5&quot;
```

Procedure:

1. Make the connection between FRC9 of the FPGA board to the External light connector of the VTU card2.
2. Make the connection between FRC1 of the FPGA board to the Dip switch connector of the VTU card2.
3. Connect the downloading cable and power supply to the FPGA board.
4. Then open the Xilinx iMPACT software (refer ISE flow) select the slave serial mode and select the respective BIT file and click program.
5. Make the reset switch on (active low) and analyze the data.

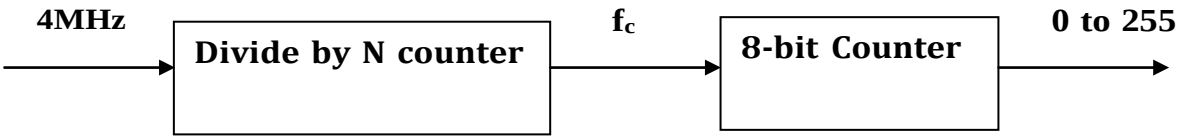
Design:

Given frequency, [f_m]=2 KHz

System frequency, [f_s]=4 MHz

Time, [T] =1/fm=0.5ms

Duty Cycle, [DC]=50%



- Count*1/f_c= T

  256*1/f_c=0.5ms

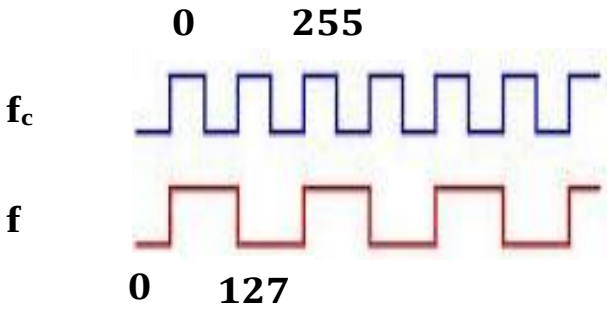
- f_c=512 KHz
- 2^N =f_s/f_c

  =4 MHz/512 KHz =8

- N=3
- Counter value =DC*Count

  0.50*256

  128



Experiment No 11: **Date: __/__/_____**

a. Verilog Code to generate Square wave using DAC

```
module square_wave(clk,rst,dac_out);input
clk;
input rst;
output [0:7] dac_out;
reg [7:0] div;
reg [0:7] counter;
reg [0:7] dac_out;


 always @ (posedge clk)
  begin
      div= div + 1'b1;
 end

always @ (posedge div[3])
begin
         if (rst)
                counter =8'b0000000;
         else
                counter<=counter + 1'b1;
end

always @(counter)
begin
         if(counter<=127)
                dac_out=8'd1;
         else
                dac_out=8'd0;
end
endmodule
```

Procedure:

1. Make the connection between FRC5 of the FPGA board and DAC connector of GPIOcard-2

2. Make the connection between FRC1 of the FPGA board and DIPswitch connector of the GPIOcard-2.

3. Assign appropriate pins to input and output.

4. Connect USB cable and power supply to the FPGA board.

Result:

b. Verilog Code to generate Triangular wave using DAC

```
module triangularwave(clk,rst,  dac_out);
input clk,rst;
output [0:7] dac_out;
reg [3:0] div;
reg [0:8] counter;
reg [0:7]  dac_out; always
@ (posedge clk)begin
        div = div +1'b1;
end

always @ (posedge div[2])
begin
         if (rst ==1'b1)
         begin
                counter = 9'b00000000;
         end
        else
        begin
                    counter = counter  +  1'b1;if
                    (counter[0] == 1 )
                    begin
                         dac_out = counter[1:8];end
                     else
                     begin
                        dac_out = ~(counter[1:8]);end


          end
end
endmodule
```

Result:

Experiment No 12: **Date: __/__/_____**

Verilog programs to interface Switches and LEDs to the FPGA/CPLD and demonstrate its working.

```
module switch_led(output led, input switch1);
reg led_out;
always @ (switch1)
begin
if(switch==1)
begin
led_out= 1'b1;
end
else
begin
led_out=1'b0;
end

end
assign led=led_out;
endmodule
```

UCF:

```
NET "led" LOC=P119;
NET "switch1" LOC=P124;
```

Question Bank

1. Verilog program to simplify the given Boolean expressions.
2. Verilog program for the following combinational design to verify the design of 3 to 8 decoder realization (behavioral model).
3. Verilog program for the following combinational design to verify the design of 8 to 3 encoder with priority and without priority (behavioral model).
4. Verilog program for the following combinational design to verify the design of 8 to 1 multiplexer using behavioral model.
5. Verilog program for the following combinational design to verify the design of 4-bit binary to gray converter using behavioral model.
6. Verilog program for the following combinational design to verify the design of 4-bit gray to binary converter using behavioral model.
7. Verilog program for the following combinational design to verify the design of 4-bit binary to excess3 converter using behavioral model.
8. Verilog program for the following combinational design to verify the design of 4-bit excess3 to binary converter using behavioral model.
9. Verilog program for the following combinational design to verify the design of 1 to 8 demultiplexer using behavioral model.
10. Verilog program for the following combinational design to verify the design of 3 to 8 decoder.
11. Verilog program for the following combinational design to verify the design of 2-bit comparator.
12. Verilog code for a half adder/full adder using data flow description.
13. Verilog code for a half subtractor/full subtractor using data flow description.
14. Verilog code for 4-bit ALU which performs different functionalities.
15. Verilog code for SR/D flip flop using behavioral model.
16. Verilog code for JK/T flip flop using behavioral model.
17. Verilog code for up/down BCD and binary counter using behavioral model.
18. Verilog code to interface a Stepper motor to FPGA and to control the Stepper motor rotation in specified direction by N steps.
19. Verilog programs to interface a Relay or ADC to the FPGA/CPLD,demonstrate its working.
20. Verilog programs to interface DAC to the FPGA/CPLD for Waveform generation.
21. Verilog programs to interface Switches and LEDs to the FPGA/CPLD and demonstrate its working.

VIVA QUESTIONS

1. What is Verilog language?
2. Which block describes a design's interface?
3. Which block describes a design's behavior?
4. What is the difference between simulation and synthesis?
5. Which data type defines a single logic signal?
6. Which data type describes a bus?
7. What two ways can a vector's range be described?
8. What are the only two values for a Boolean type?
9. What are the numerical data types?
10. What type is use to create a user data type?
11. What reserved word is used to declare a user data type?
12. Which data type includes time units as values?
13. Create the module block for a three input XOR gate.
14. Which keyword is used to end Verilog program?
15. What part of a port declaration defines signals in or out direction?
16. Create the integer constant included in a module BUS_SIZE and assigns it a value of 32?
17. Which symbols are used as an assignment operator to assign a literal to an identifier name?
18. Which symbols are used to assign an expression's result to an output interface signal?
19. What are the rules used to define an identifier name?
20. What symbols define a comment line?
21. How does a transport delay differ from an inertial delay?
22. What is the purpose of a SIGNAL declaration?
23. Where are SIGNAL declarations placed in the design?
24. Write an assignment statement that assigns the contents of S(5) to t(2).
25. What is the purpose of a process' sensitivity list?
26. Under what conditions is a process run?
27. What is an EVENT? What is the difference between event and non-event driven process?
28. Which symbols are used to differentiate between logic 1 and an integer 1?
29. In an if..then..else construct, which statements are executed if the condition is TRUE and which if it is FALSE?
30. What is the purpose of a for loop?
31. What are the requirements for a for loop?
32. What is meant by instantiating a component?
33. How do signal declarations differ from port interface declarations?
34. What is the prime use of signals?
35. How many parameters can be passed into a function?
36. Write a function that returns the sum of two 8-bit words.
37. How are functions called?
38. How do procedures differ from functions?