INDEX

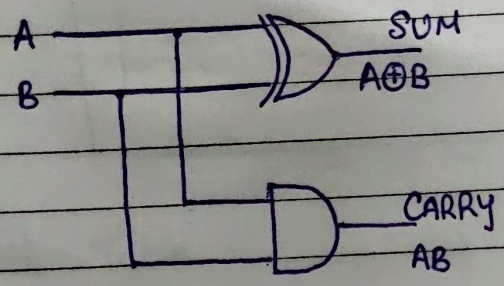
NAME: Raghuvir STD.: SEC.: ROLL NO.: SUB.:

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1.		Practical 1 :- half adder and full adder	6	
2.		Practical 2 :- MUX and DMUX	6	
3.		Practical 3 :- Flip Flop	6	
4.		Practical 4 :- Counter	6	
5.		Practical 5 :- Shift register	6	

Practical - 1

Aim :- To study half adder and full adder and write verilog code for the same.

1. Half adder :-



Truth table :-

Inputs		Outputs	
A	B	Sum (S)	Carry (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

* Behaviour - level modeling :-

```
module halfadder (s,c,a,b);
input a,b;
output s,c;
reg s,c;
always @(a or b)
begin
    s = a^b;
    c = a & b;
```

```
    end
endmodule
```

★ **Dataflow modeling**

```
module halfadder (s,c,a,b);
input a,b;
output s,c;
assign s=a^b;
assign c=a&b;
endmodule
```

★ **Gate-level modeling**

```
module halfadder (s,c,a,b);
input a,b;
output s,c;
xor G1 (s,a,b);
and G2 (c,a,b);
endmodule
```

★ K-Map

for carry

A \ B	0	1
0	0	0
1	0	1

$C = AB$

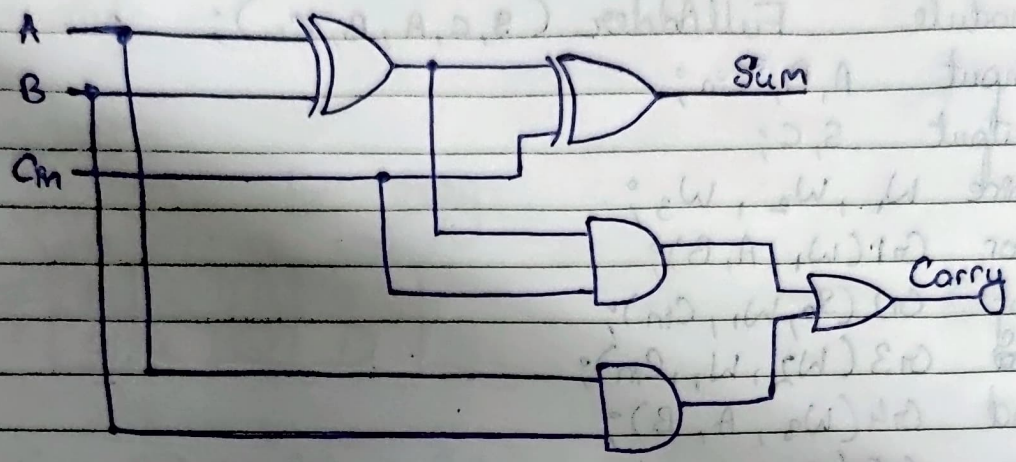
for SUM

A \ B	0	1
0	0	1
1	1	0

$C = AB' + BA' = A \oplus B$

classmate
Date
Page

★ Full adder



★ Truth Table

Input			Output	
A	B	Cin	Sum (s)	Carry (c)
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

★ Gate-level modeling

```
module FullAdder (S, C, A, B, Cin);
input A, B, Cin;
output S, C;
wire W1, W2, W3;
xor G1(W1, A, B);
xor G2(S0, W1, Cin);
and G3(W2, W1, Cin);
and G4(W3, A, B);
or G5(C, W2, W3);
endmodule
```

★ Dataflow modeling

```
module FullAdder (S, C, A, B, Cin);
input A, B, Cin;
output S, C;
assign C = A & B | A & Cin | B & Cin;
assign S = ~A & ~B & Cin | ~A & B & ~Cin | A & ~B & ~Cin | A & B & Cin;
endmodule
```

★ Behaviour level

```
module FullAdder (S, C, A, B, Cin);
input A, B, Cin;
output S, C;
reg S, C;
always @ (A or B or Cin);
  begin
    S = ~A & ~B & Cin | ~A & B & ~Cin | A & ~B & ~Cin | A & B & Cin;
```

```
    C = A&B | A&Cin | B&Cin ;
end
endmodule
```

★ **Kr MAP**

For carry

A \ BCin	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$C = AB + AC_{in} + BC_{in}$$

For Sum

A \ BCin	00	01	11	10
0	0	1	0	1
1	1	0	1	0

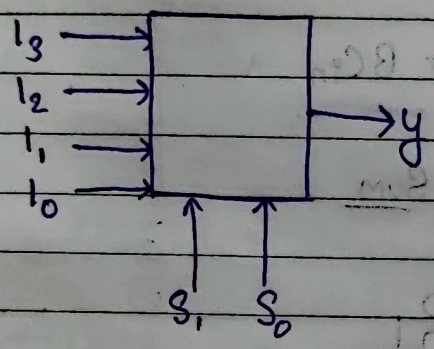
$$S = A'B'C_{in} + A'BC_{in}' + AB'C_{in}' + ABC_{in}$$

Practical - 2

Aim :- 4×1 Multiplexer

Theory :- 4×1 multiplexer has 4 inputs I_3, I_2, I_1 and I_0, 2 selection lines S_1 & S_0 and one output.

Block diagram :-



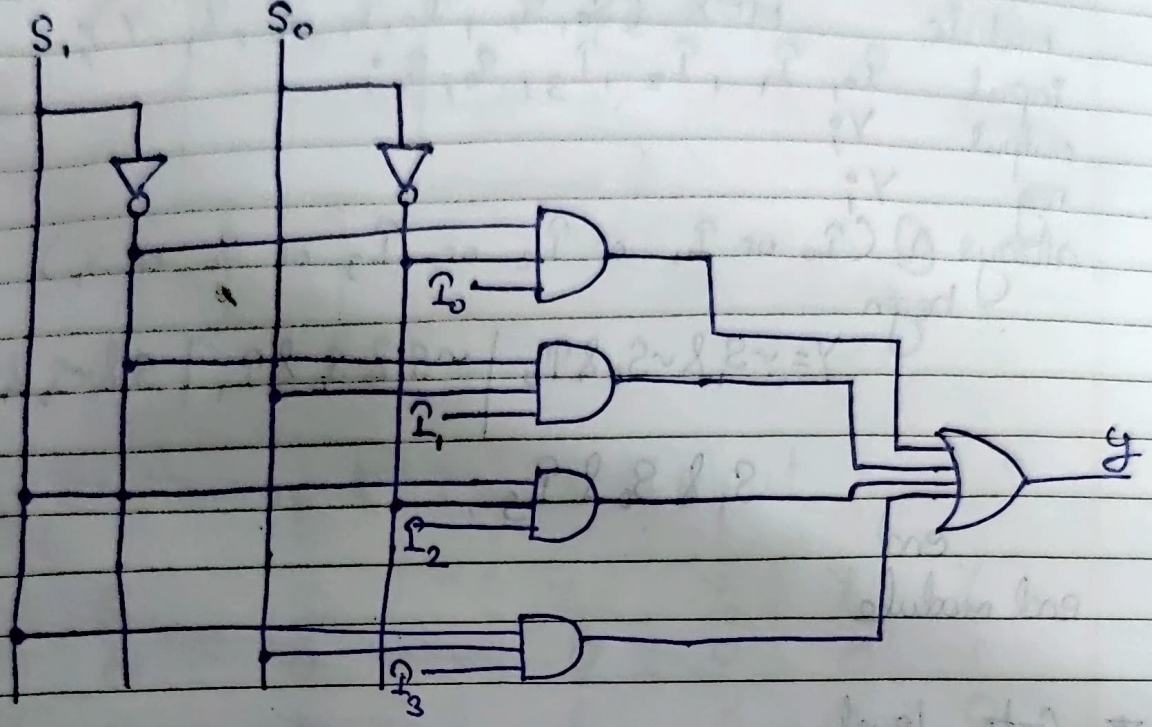
Truth Table :-

Selection lines		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

From the truth table

$$Y = \bar{S}_1 \bar{S}_0 I_0 + \bar{S}_1 S_0 I_1 + S_1 \bar{S}_0 I_2 + S_1 S_2 I_3$$

* Circuit diagram



* Dataflow modeling

```
module    MUX (I0, I1, I2, I3, S0, S1, Y);
input     I0, I1, I2, I3, S0, S1;
output    Y;
assign    Y = ~S1 & ~S0 & I0 | ~S1 & S0 & I1 | S1 & ~S0 & I2 |
                S1 & S0 & I3;
endmodule
```

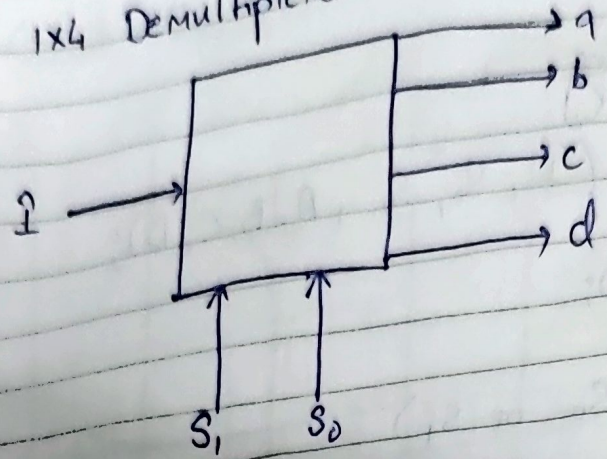
★ Behaviour modeling

```
module MUX (S0, S1, I0, I1, I2, I3, Y);
input I0, I1, I2, I3, S0, S1;
output Y;
reg Y;
always @ (I0 or I1 or I2 or I3 or S0 or S1)
  begin
    Y = ~S1 & ~S0 & I0 | ~S1 & S0 & I1 | S1 & ~S0 & I2 |
        S1 & S0 & I3;
  end
endmodule
```

★ Gate level

```
module MUX (S0, S1, I0, I1, I2, I3, Y);
input I0, I1, I2, I3, S0, S1;
output Y;
wire W1, W2, W3, W4, W5, W6;
not G1 (W1, S1);
not G2 (W2, S0);
and G3 (W3, W1, W2, I0);
and G4 (W4, W1, S0, I1);
and G5 (W5, S1, W2, I2);
and G6 (W6, S1, S0, I3);
or G7 (Y, W3, W4, W5, W6);
endmodule
```

1x4 Demultiplexer



Selection input		Output			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

$Y_3 = S_1 S_0 I$
$Y_2 = S_1 \bar{S_0} I$
$Y_1 = \bar{S_1} S_0 I$
$Y_0 = \bar{S_1} \bar{S_0} I$

* Dataflow modeling

```
module DMUX(I, S0, S1, A, B, C, D);
input I, S1, S0;
output A, B, C, D;
assign D = S1 & S0 & I;
assign C = S1 & ~S0 & I;
assign B = ~S1 & S0 & I;
assign A = ~S1 & ~S0 & I;
```

endmodule

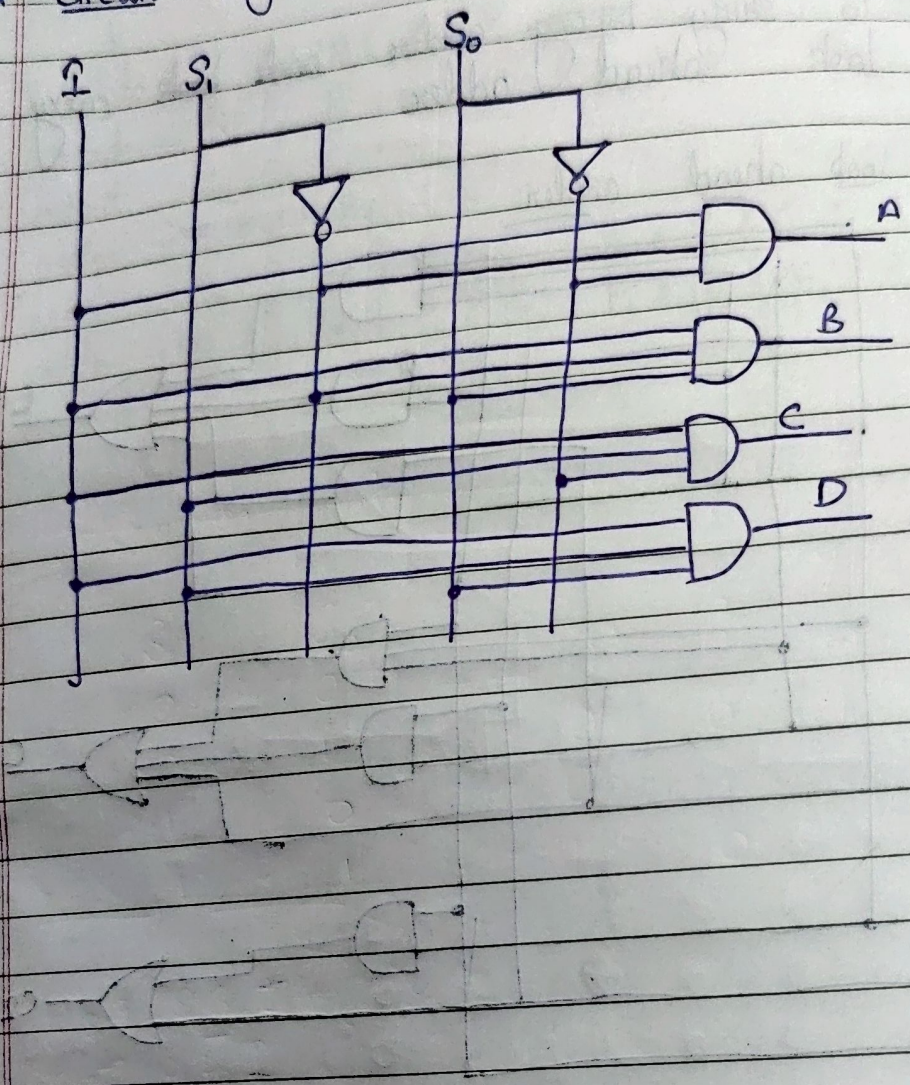
✡ Behaviour modeling

```
module    DMUX (I, S0, S1, A, B, C, D);
input     I, S0, S1;
output    A, B, C, D;
reg       A, B, C, D;
always @(I or S0 or S1)
      begin
            D = S1 & S0 & I;
            C = S1 & ~S0 & I;
            B = ~S1 & S0 & I;
            A = ~S1 & ~S0 & I;
      end
endmodule
```

✡ Gatelevel modeling

```
module    DMUX (I, S0, S1, A, B, C, D);
input     I, S0, S1;
output    A, B, C, D;
wire      W1, W2;
not       G1 (W1, S1);
not       G2 (W2, S0);
and       G3 (D, S1, S0, I);
and       G4 (C, S1, W2, I);
and       G5 (B, W1, S0, I);
and       G6 (A, W1, W2, I);
or        G7 (
```

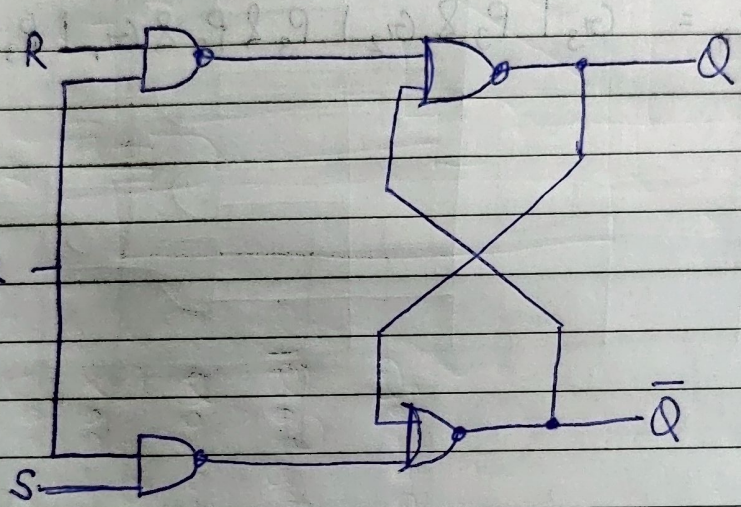
★ Circuit Diagram



Practical - 3

Aim :- To study SR Flip flop, JK Flip flop, T-Flip flop

* SR Flip-flop



Q_n	S	R	Q_{n+1}	
0	0	0	0	Reset
0	0	1	0	
0	1	0	1	
0	1	1	intermidiate / x	
1	0	0	1	Toggle
1	0	1	0	
1	1	0	1	
1	1	1	intermidiate / x	

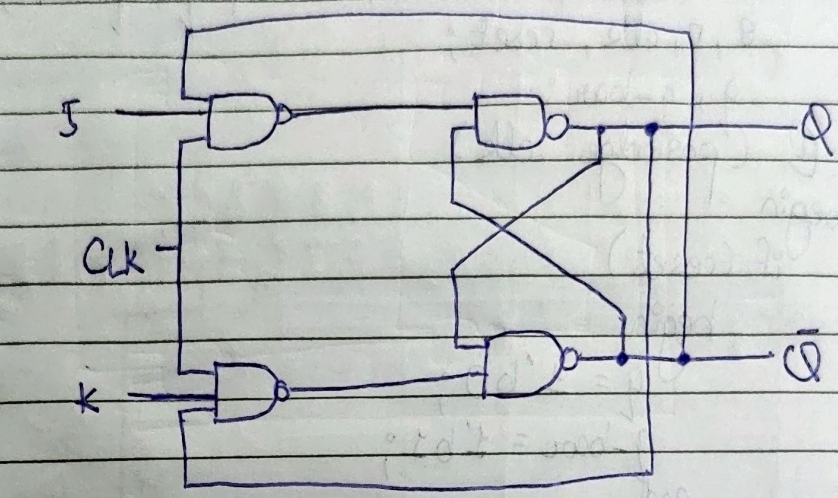
Behavioural

```
module SR (s, r, clk, reset, q, q_bar);
input s, r, clk, reset;
output q, q_bar;
always @ (posedge clk)
  begin
    if (reset)
      begin
        q = 1'b0;
        q_bar = 1'b1;
      end
    else
      begin
        case ({s,r})
          {1'b0, 1'b0} :
            begin
              q = q;
              q_bar = q_bar;
            end
          {1'b0, 1'b1} :
            begin
              q = 1'b0;
              q_bar = 1'b1;
            end
          {1'b1, 1'b0} :
            begin
              q = 1'b1;
              q_bar = 1'b0;
            end
          {1'b1, 1'b1} :
            begin
              q = x;
              q_bar = x;
```

O/p verified for SR, JK, T, D

2185211118

JK Flip-Flop



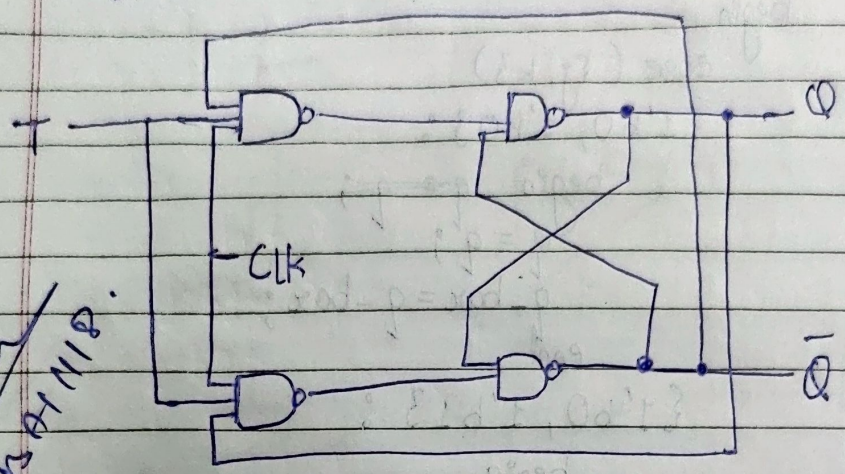
C	J	K	Q_n	$\bar{Q}_n$	Q_{n+1}	
↑	0	0	0	1	0	No change
↑	0	0	1	0	1	
↑	0	1	0	1	0	Reset
↑	0	1	1	0	0	
↑	1	0	0	1	1	Set
↑	1	0	1	0	1	
↑	1	1	0	1	1	Toggle
↑	1	1	1	0	0	

Behavioural

```
module    jff( j,k, clk, reset, q, q-bar);
input     j,k, clk, reset;
output    q, q-bar;
reg       q, q-bar;
always @ (posedge clk)
    begin
        if (reset)
            begin
                q = 1'b0;
                q-bar = 1'b1;
```

```
    end
  else
    begin
      case ({j, k})
      {1'b0, 1'b0}:
          begin q = q;
            q = q;
            q_bar = q_bar;
          end
      {1'b0, 1'b1}:
          begin
            q = 1'b0;
            q_bar = 1'b1;
          end
      {1'b1, 1'b0}:
          begin
            q = 1'b1;
            q_bar = 1'b0;
          end
      {1'b1, 1'b1}:
          begin
            q = ~q;
            q_bar = ~q_bar;
          end
      endcase
    end
  end
endmodule
```

* T flip-flop



T	Q_n	Q_{n+1}
0	0	0
1	1	0
0	1	1
1	0	1

Behavioural

```
module tF (t, clk, reset, q, q-bar);
input t, clk, reset;
output q, q-bar;
reg q, q-bar;
always @ (posedge clk)
    begin
        if (reset)
            begin
                q = 1'b0;
                q-bar = 1'b1;
            end
        else
```

```
begin
case {s,r}
{1'b0}
    q = q;
    q_bar = q_bar;
{1'b1}
    q = ~q;
    q_bar = ~q_bar;
end
end
endmodule
```

* Test bench (sr)

```
module sr_tb;
reg s, r, clk, reset;
wire q, q_bar;

sr uut (
    .s(s),
    .r(r),
    .clk(clk),
    .reset(reset),
    .q(q),
    .q_bar(q_bar)
);
initial begin
s = 1'b0;
r = 1'b0;
reset = 1;
clk = 1;
```

```
#100
reset = 0;
s = 1'b0;
r = 1'b1;

#100
reset = 0;
s = 1'b1;
r = 1'b0;

#100
reset = 0;
s = 1'b1;
r = 1'b1;

end
always #5 clk <= ~clk;
endmodule
```

* Test bench (jk)

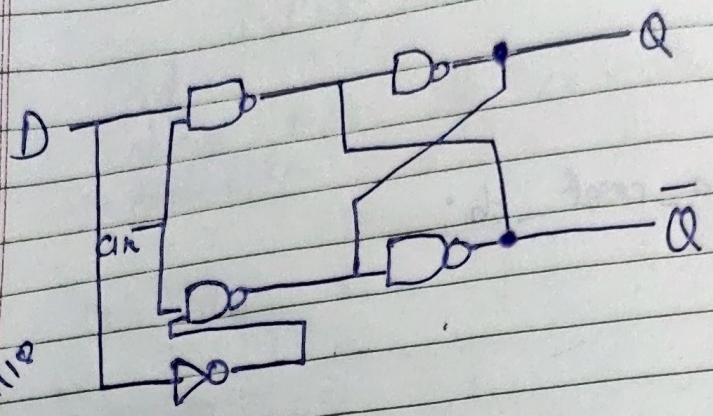
```
module jk_tb;
reg j, k, clk, reset;
wire q, q_bar;

jk uut(
    .j(j);
    .k(k);
    .clk(clk);
    .reset(reset);
    .q(q);
```

```
    ); q_bar (q_bar);
    initial begin
j = 1'b0;
k = 1'b0;
reset = 1;
clk = 1;

#100
reset = 0;
j = 1'b0;
k = 1'b1;

#100
reset = 0;
j = 1'b1;
k = 1'b0;

#100
reset = 0;
j = 1'b1;
k = 1'b1;
  end
always #5 clk <= ~clk;
endmodule
```

* Test bench (t)

```
module tbF_tb;
reg t, clk, reset;
wire q, q_bar;

t uut (
    t(t);
    clk(clk);
    reset(reset);
    q(q);
    q_bar(q_bar);
);
initial begin
t = 1'b0;
reset = 1;
clk = 1;

#100
t = 1'b1;


always #5 clk <= ~clk;

endmodule
```

* D - Flip Flop



Clk	D	Present state	Next state	State
↑	0	0	0	Reset
↑	0	1	0	Reset
↑	1	0	1	Set
↑	1	1	1	Set
↓	x	Q_n	Q_n	No change

* Behavioural

```
module dff_ async_ reset (data, clk, reset, q);
input data, clk, xreset;
output q;
reg q;
always @ (posedge clk or negedge reset)
    if (~rest)
        begin
        q <= 1'b0;
        end
        else
            begin
```

```
    q <= data;
  end
endmodule
```

* Test bench

```
module dff_asyn_reset_tb;
reg clk;
reg reset;
reg d;

wire q;
wire qb;

dff_asyn_reset dFlipFlop (.clk(clk), .reset(reset),
                          .d(d), .q(q), .q_bar(qb));

initial
  begin
  $monitor(clk, q, d, qb, reset);
  q = 1'b0;
   reset = 1;
   clk = 1;
   #10
   reset = 0;
   d = 1'b1;
   #100
   reset = 0;
   d = 1'b0;
   #100
   reset = 0;
   d = 1'b1;
```

#100
reset = 1;
d = 1'b1;
end
always #25 clk <= ~clk;
endmodule

Practical-4

Aim :- To design and simulate up/down counter

=> Truth table

Clock Pulse No.	Counter output at Q			
	Q_D	Q_C	Q_B	Q_A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

* Behavioural :-

```
module   UP_DOWN_COUNTER ( COUNT, CLK, UP_DOWN, RESET);
input   CLK, UP_DOWN, RESET;
output [3:0] COUNT;
reg  [3:0] COUNT = 0;
always @ (posedge CLK or posedge RESET)
begin
      if (RESET == 1)
            COUNT <= 0;
      else
            if(UP_DOWN == 1)
               if(COUNT == 15)
                     COUNT <= 0;
               else
                     COUNT <= COUNT + 1;
            else
               if( COUNT == 0)
                     COUNT <= 15;
               else
                     COUNT <= COUNT - 1;

end
endmodule
```

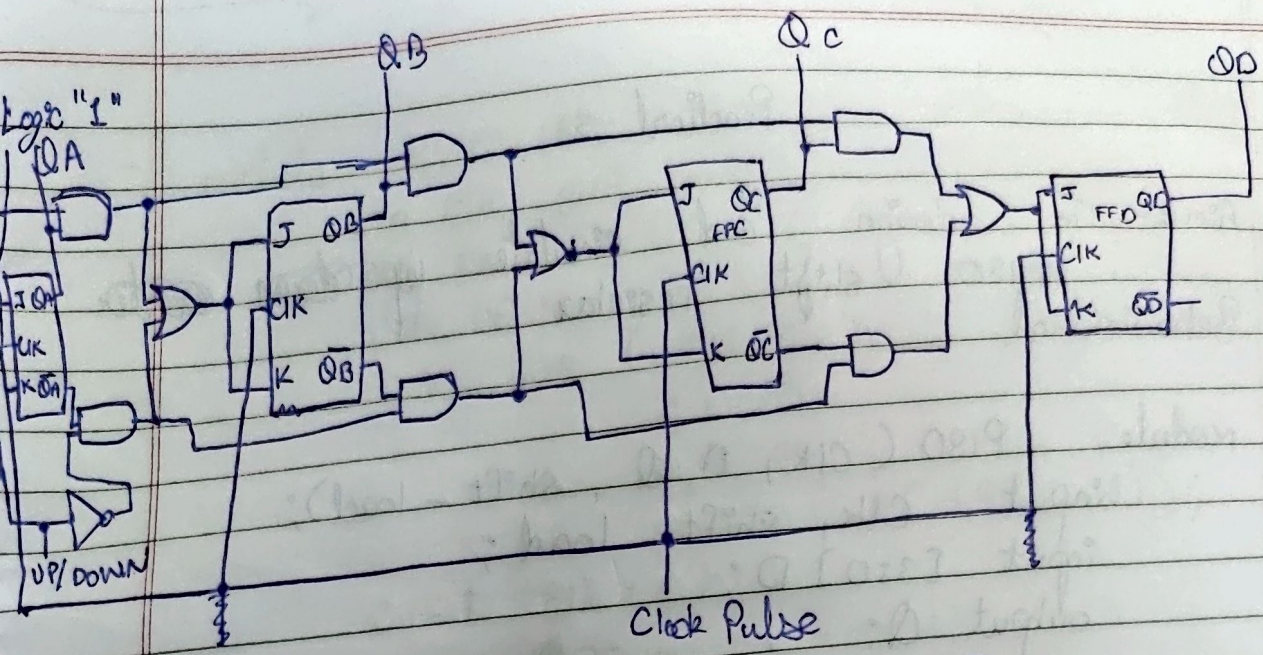
o/p
verified

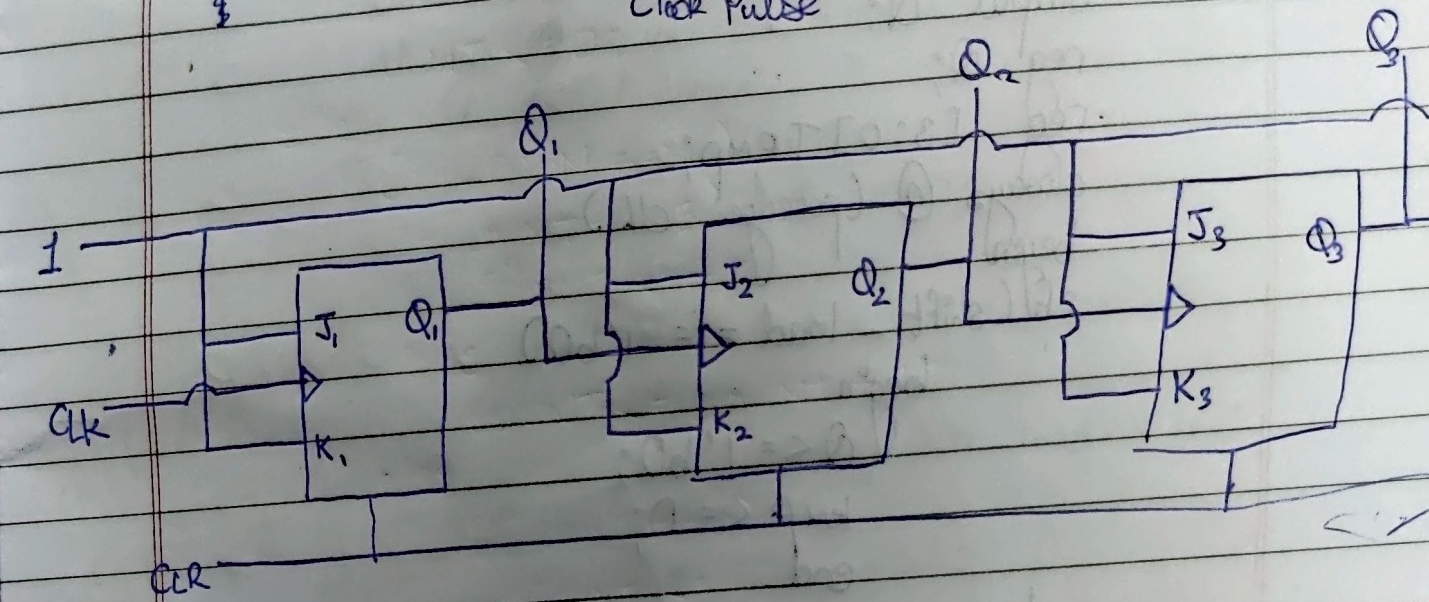
12.10.22

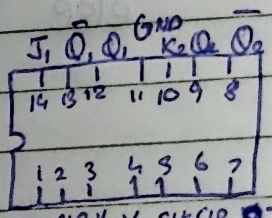
21BAA 1118

☆ Test Bench

```
module UP_DOWN_COUNTER_TB;
reg CLK, RESET;
reg UP_DOWN;
reg ;
wire [3:0] COUNT;
UP_DOWN_COUNTER uut (
        .COUNT(COUNT),
        .CLK(CLK),
        .UP_DOWN(UP_DOWN),
        .RESET(RESET));
initial CLK=0;
  always #10 CLK = ~CLK;
  initial begin
  RESET = 0; UP_DOWN = 1;
  #400
  UP_DOWN = 0;
  #400
  end
  end module
```

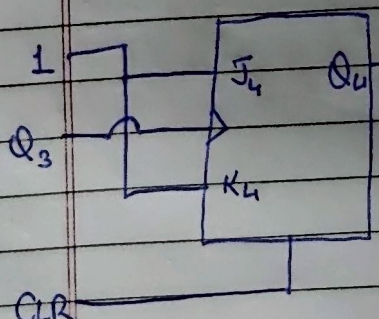

Logic "1"
QA
QB
QC
QD
J
CLK
K
FFC
FFD
UP/DOWN
Clock Pulse


1
CLK
CLR
Q_1
Q_2
Q_3
J_1
K_1
J_2
K_2
J_3
K_3


J_1 $\bar{Q}_1$ Q_1 GND K_2 Q_2 $\bar{Q}_2$
14 13 12 11 10 9 8
1 2 3 4 5 6 7
CLK, CLR, K_1, V_{CC} CLK, CLR, J_2

7473

(SU)


1
Q_3
CLR
J_4
Q_4
K_4
Q_4

Practical -3

Aim :- To design and simulate PISO shift resistar

* Behavioural

```
module PISO (clk, D, Q, shift_load);
    input clk, shift_load;
    input [3:0] D;
    output Q;
    reg Q;
    reg [3:0] Temp;
    always @ (posedge clk)
    begin
    if (shift_load == 1'b0)
        begin
        Q <= 1'b0;
        temp <= 0;
        end
    else
        begin
        Q <= temp[0];
        temp <= temp >> 1'b1;
        end
    end
    endmodule
```

```
module PISO_tb;
  reg Clk, shift_load;
  reg [3:0]D;
  wire Q;
  PISO uut (.Clk(Clk),
            .D(D),
            .Q(Q),
            .shift_load(shift_load));
initial clk = 1'b1;
always #100 Clk = ~Clk;
  initial begin
    D = 4'b1101;
    shift_load = 1'b0;
    #200;
    shift_load = 1'b1;
    #1200;
    D = 4'b1001; shift_load = 1'b0;
    #200
    shift_load = 1'b1;
    #1000 $stop;
```

21BEA01112

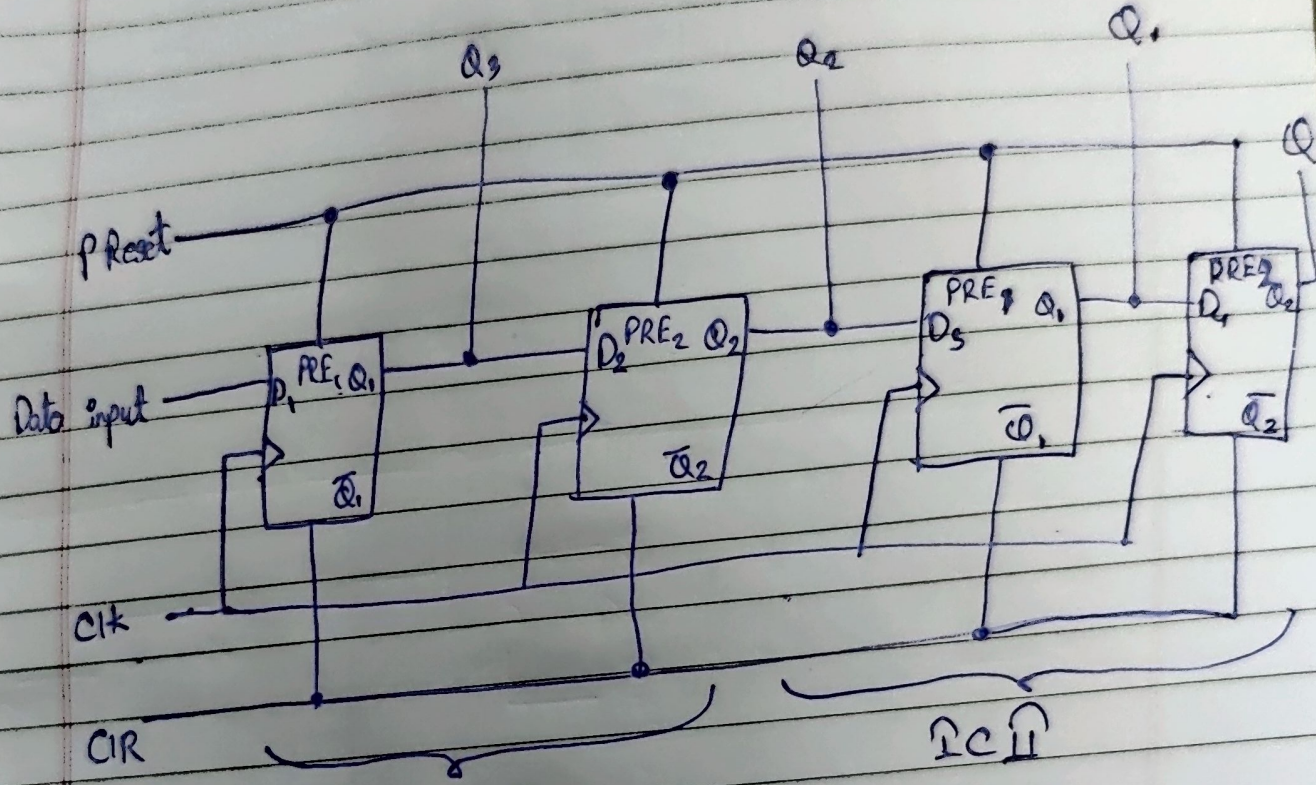
Op Verify shift register

load

end module

09/10/24

Clock pulse number	Shift/load	Parallel data input				serial data
		D_3	D_2	D_1	D_0	Q
1	0	1	1	0	1	0
2	1	1	1	0	1	1
3	1	0	1	1	0	0
4	1	0	0	1	1	1
5	1	0	0	0	1	1
6	1	0	0	0	0	0
7	1	0	0	0	0	0
8	0	1	0	0	1	0
9	1	1	0	0	1	1
10	1	0	1	0	0	0
11	1	0	0	1	0	0
12	1	0	0	0	1	1
13	1	0	0	0	0	0


Q3
Q2
Q1
Q
PReset
Data input
Clk
CIR
PRE1 Q1
D1
Q1
PRE2 Q2
D2
Q2
PRE1 Q1
D3
Q1
PRE2 Q2
D4
Q2
IC I
7474
IC II
7474

SIPO