## Experin. - 4(a) :- Verilog HDL behavioural model program based on various operations.

**Aim:-** To write a program to verify HDL using two 4 bit inputs and one 8-bit output for any 12 arithmetic logical relation operations with output verific.

**Components req. :-** Model Sim software.

**Procedure :-**

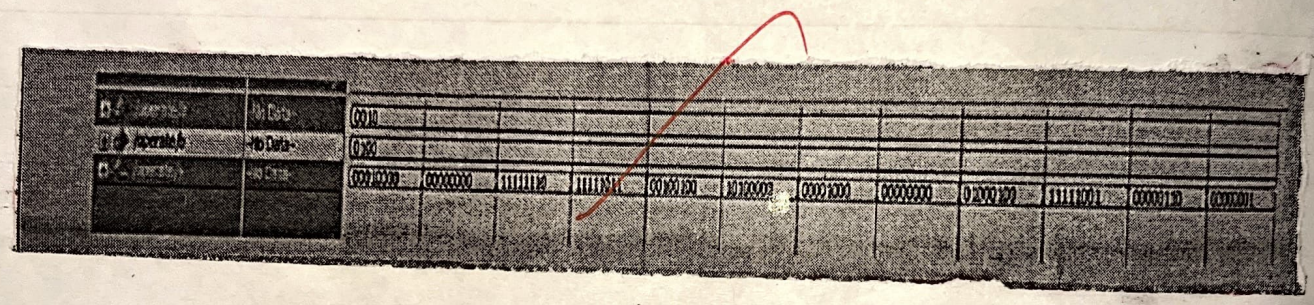
1. Open model sim, go to layout, click on reset.
2. File → new, select folder & name it.
3. File → new select project, click on yes. Name the project & locate it to the file location you want.
4. A new window appears, click on create new file on it. Name the file {shouldn't be the same as project name}. choose file type as verilog, click ok.
5. Double click on the file shown in modelsim. Type your code based on var. applications using behavioural model and save as in work folder.
6. Compile it using compile selected. "Compilation successful" should appear in transcript.
7. Simulate → start simulation.

Teacher's Signature _____

# Behavioural.

```
module operate(y,a,b);
 output reg [7:0] y;
 input [3:0] a,b;
always @(*) begin
 if (a!=4'b0000)
 y=a**b;
 else
 y=8'b00000000;
 #10 y=a&b;
 #10 y=a-b;
 #10 y=-b;
 #10 y={a,b};
 #10 y={4'b1010,a>>4};
 #10 y=a*b;
 #10 y={a==b};
 #10 y={2{b}};
 #10 y=a^b;
 #10 y=a|b;
 #10 y=(a<=b)?8'b00000001:8'b00000000;
end
endmodule
```

always @(a or b) begin



8. Move to blue window, right click on each input and add wire. Later right click and select force of each input. Update input values & run.

9. Values of the output changes upon diff. operations being used which is visible on wave.

Result :-

Outputs obtained from simulation are verified with outputs from manual calculation.

• Manual output:-

I/P : A, B

O/P = Y

Let a = 0010 = 2

b = 0100 = 4

(1.) y = a ** b

= 2 ** 4 = 16.

y = 8'b00010000

(2.) y = a & b

= a . b

= (4'b0010) . (4'b0100)

= 8'b00000000

(3.) y = a - b

= 4'b0010 - 4'b0100

y = 8'b11111110

(4.) y = ~b

= ~4'b0100

= ~8'b00000100

= 8'b11111011

(5.) y = {a, b}

= 8'b00100100

(6.) y = {4'b1010, a >> 4}

= 8'b10100000

(7.) y = a * b

= 8'b00010000

(8.) y = (a == b)

y = False

y = 8'b00000000

(9.) y = {2{b}}

= 8b'01000100

(10.) y = a ~^ b

y = 8b'11111001

(11.) y = a + b

y = 8'b00000110

(12.) y = (a < = b)

y = 8'b00000001

## Experim. - 4(b) : Verilog HDL program for the implementation of a boolean function using decoder & an ext. OR gate.

**Aim:** To implement a boolean fxn. using decoder & an ext. OR gate using verilog HDL.

**Components:** Model Sim software.

**Procedure:**

1. Open model sim ⟶ layout ⟶ RESET.
2. File ⟶ new ⟶ select folder ⟶ name it.
3. Again go to file ⟶ new ⟶ select project ⟶ yes. name it & locate it.
4. A new window appears, click on create new file on it. Name the file { different from project name }. Choose file type as verilog. click on ok.
5. Double click on the file in model sim. Type your work and save in work folder.
6. Compile it using compile selected. "Config. was successful" should appear in transcript.
7. Go to simulate & start simulation.

Teacher's Signature _____

$F(x,y,z) = \bar{x}\bar{y}z + xy$

$F = \bar{x}\bar{y}z + xyz + xy\bar{z}$

001 111 110

$F = \sum_m (1, 6, 7)$

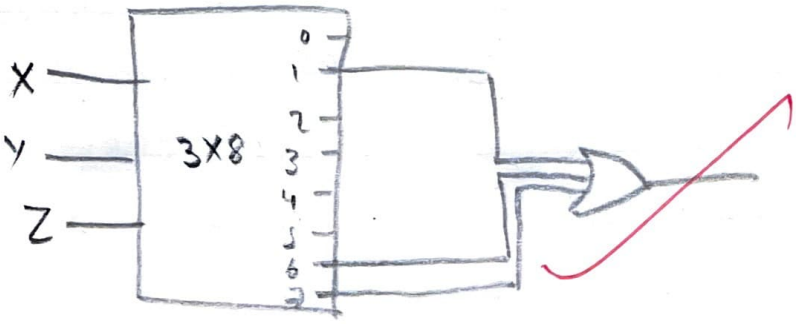
no of I/P = 3

we use 3:8 decoder

Truth table :-

| X | Y | Z | $d_1$ | $d_6$ | $d_7$ | $F = d_1 + d_6 + d_7$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 |

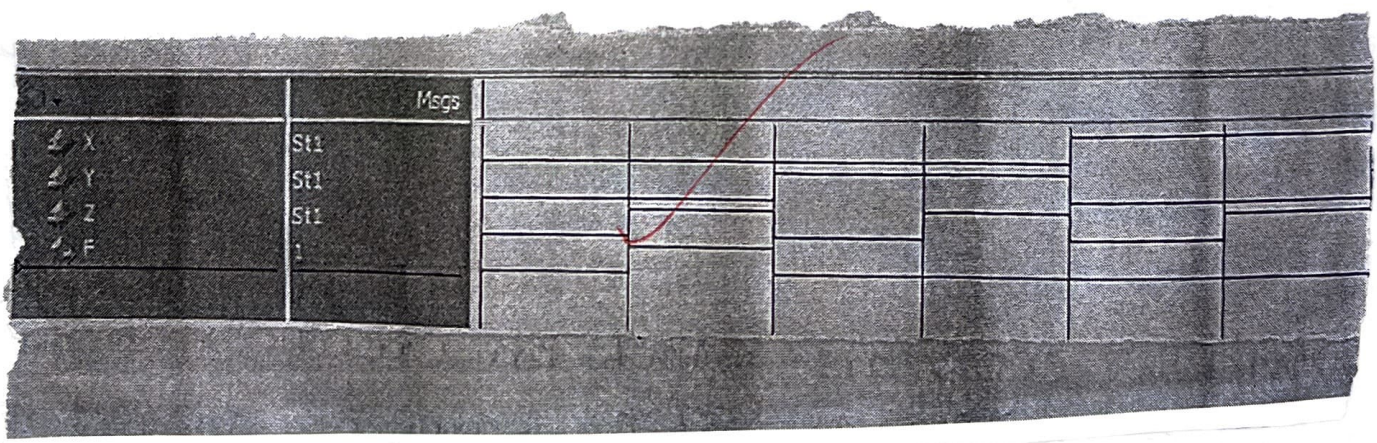
Logic diagram :-



Behavioural.

```
module decoder3to8 (D,x,y,z);
 input wire x, y, z;
 output reg [7:0] D;
always @(*) begin
        D = 8'b00000000;
        case ({x, y, z})
            3'b000: D = 8'b00000001;
            3'b001: D = 8'b00000010;
            3'b010: D = 8'b00000100;
            3'b011: D = 8'b00001000;
            3'b100: D = 8'b00010000;
            3'b101: D = 8'b00100000;
            3'b110: D = 8'b01000000;
            3'b111: D = 8'b10000000;
        endcase
    end
endmodule

module Implement (F,x,y,z);
 input wire x, y, z;
 output reg F;
 wire [7:0] D;
 decoder3to8 dec(D,x,y,z);
always @(*) begin
        F = D[1] | D[6] | D[7];
end
endmodule
```



## expt. – 5(a) : Implem. of Boolean exp. using 4x1 MUX.

**Aim:-** To design & implem. a boolean expression using 4x1 mux & verify using Verilog HDL and waveform.

**Boolean exp. :-** $F = A\bar{B}C + AB\bar{C} + \bar{A}C$

$F = \Sigma m(1, 3, 5, 6)$.

**Truth table:-**

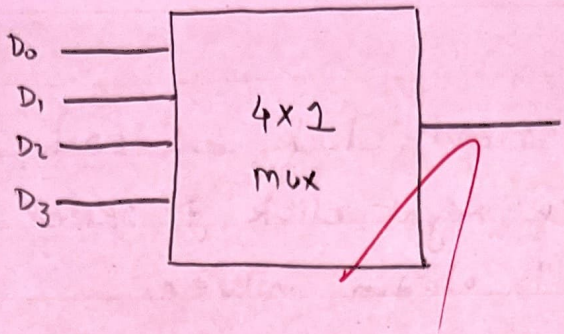
| A | B | C | $F = A\bar{B}C + AB\bar{C} + \bar{A}C$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

**Procedure :-**

1. Open modelsim.

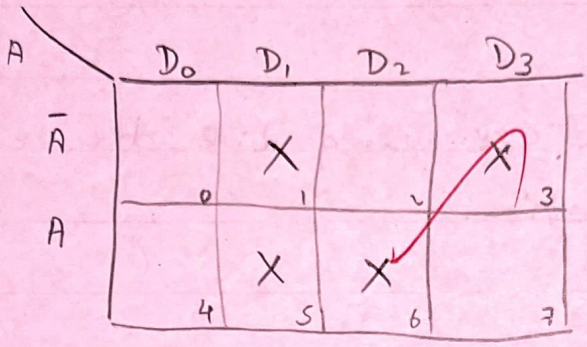
Teacher's Signature

## mux diagram:-



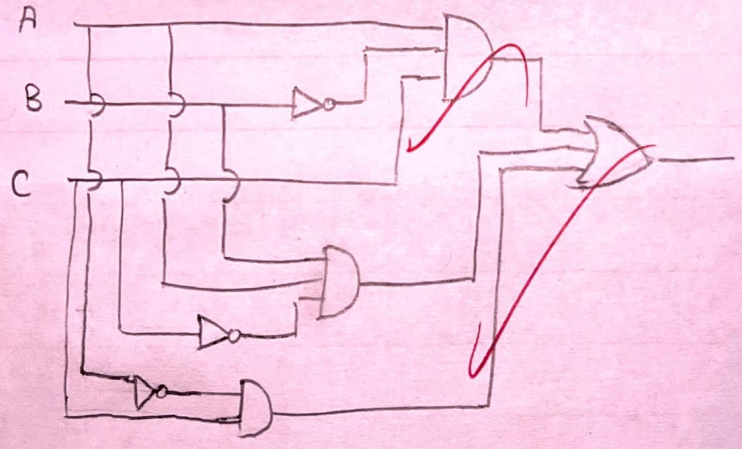

$F = A\bar{B}C + AB\bar{C} + \bar{A}C$

## multiplexer box method :-




output :- 0 1 A Ā

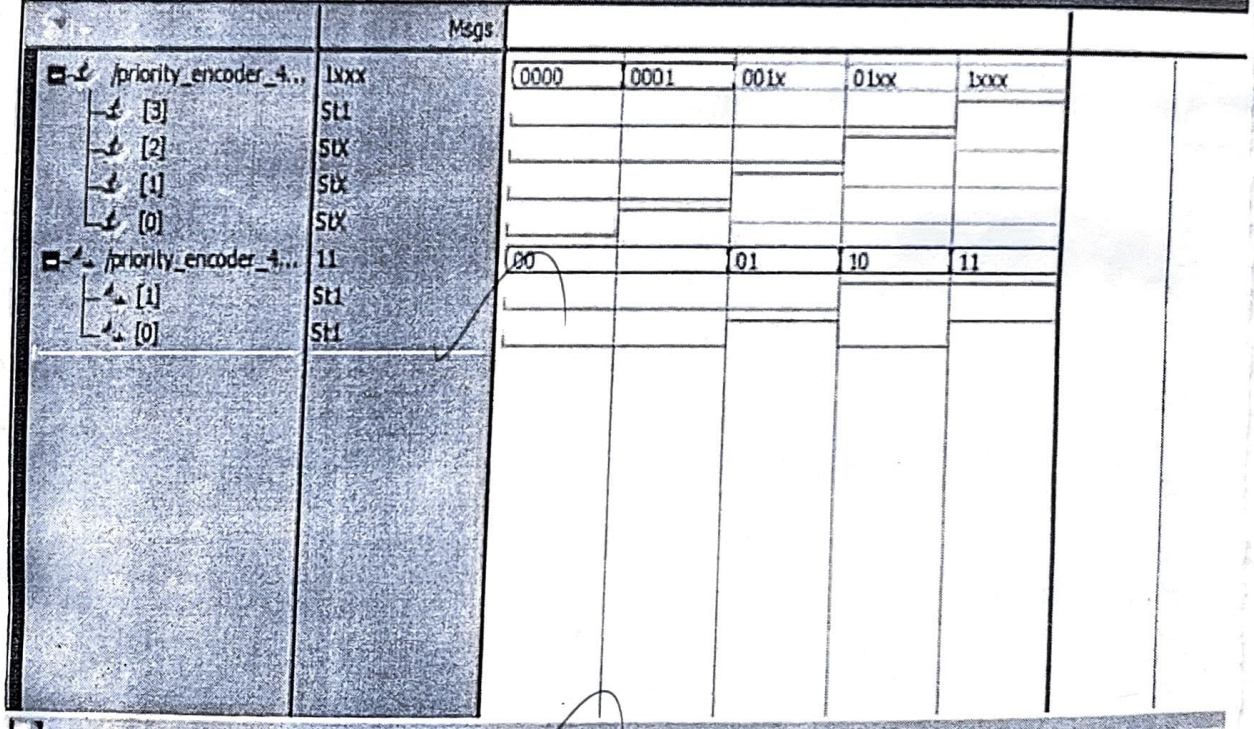
## logic diagram :-



C:/intelFPGA/priority_encoder_4_2_beh.v (/priority_encoder_4_2_beh) - Default

```
Ln#
 1  module priority_encoder_4_2_beh(
 2      input [3:0] D,
 3      output reg [1:0] Y
 4  );
 5
 6  always @(*) begin
 7      casez(D)
 8          4'b1xxx: Y = 2'b11;
 9          4'b01xx: Y = 2'b10;
10          4'b001x: Y = 2'b01;
11          4'b0001: Y = 2'b00;
12          default: Y = 2'b00;
13      endcase
14  end
15
16  endmodule
17
18
```

Wave - Default




C:/intelFPGA/priority_encoder_4_1_dataflow.v (/priority_encoder_4_2_dataflow) - Default

```
Ln#
1  module priority_encoder_4_2_dataflow(
2   input [3:0] D,
3   output [1:0] Y
4  );
5  assign Y[1] = D[3] | D[2];
6  assign Y[0] = D[3] | (~D[2] & D[1]);
7  endmodule
8
```

## expt.-5(b): Design & implem. of priority encoder using Verilog HDL:-

**Aim:-** To design & implem. a priority encoder in verilog HDL & verify using truth table and logic diagram.

**Software req.:-** ModelSim intel FPGA Starter 18.1

**Truth table:-**

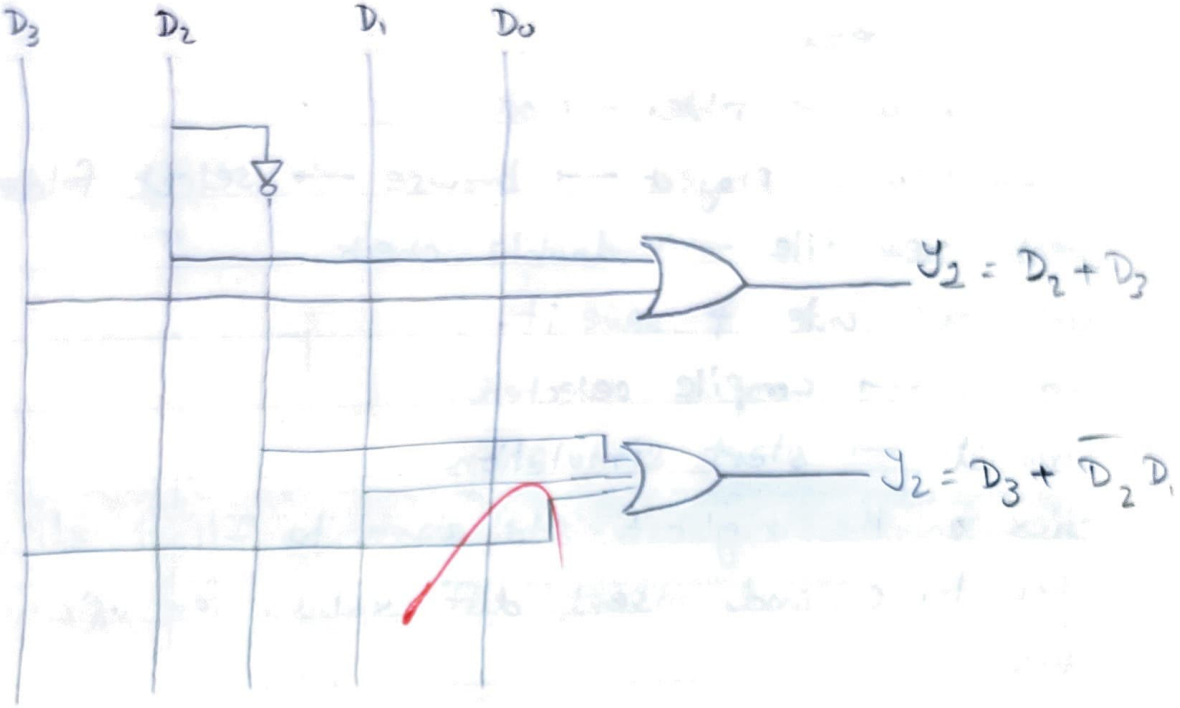
| $I_3$ | $I_2$ | $I_1$ | $I_0$ | $A_1$ | $A_0$ | V |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | X | 0 | 1 | 1 |
| 0 | 1 | X | X | 1 | 0 | 1 |
| 1 | X | X | X | 1 | 1 | 1 |

**Procedure:-**

1. Open modelsim
2. Layout → reset.
3. File → new → folder → ok.
4. File → new → project → browse → select folder.

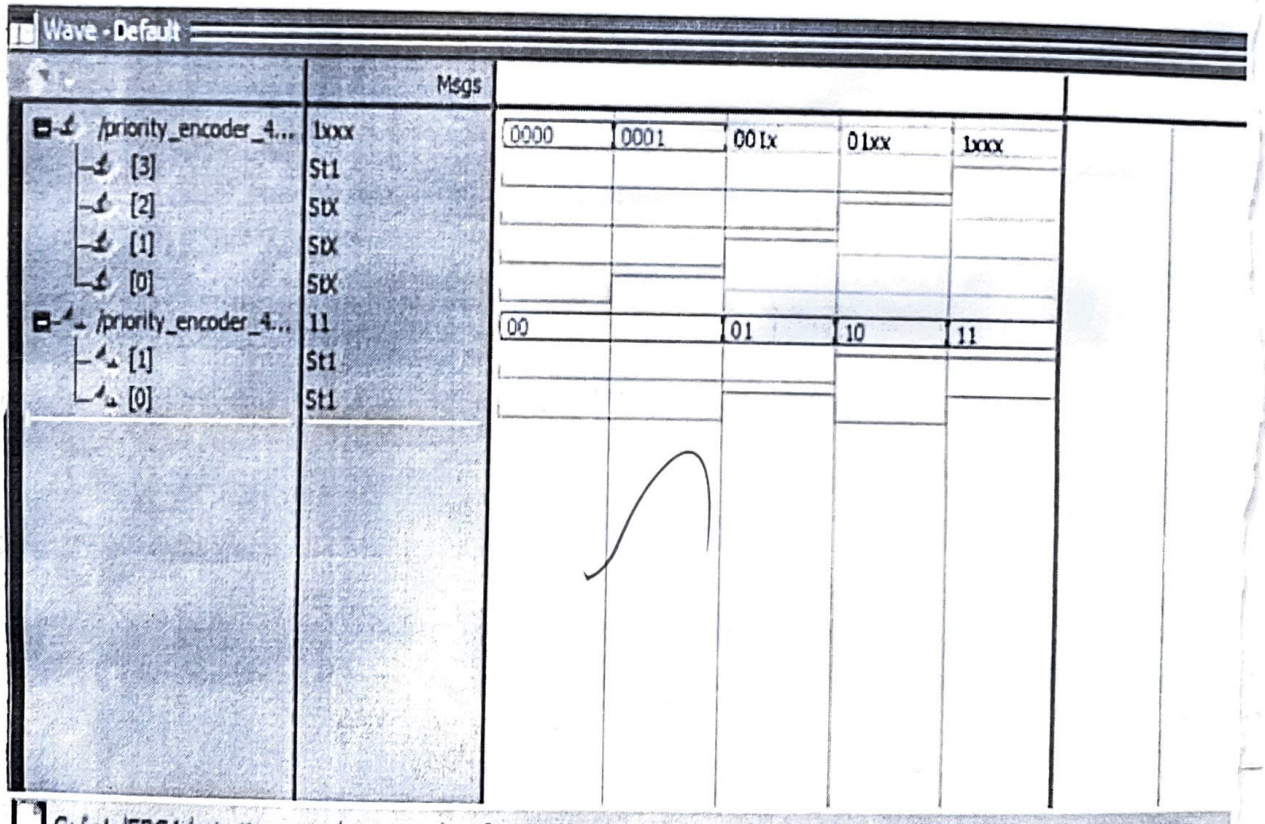
Teacher's Signature _____

Logic diagram :-



C:/intelFPGA/priority_encoder_4_2_beh.v (/priority_encoder_4_2_beh) - Default

```
Ln#
 1  module priority_encoder_4_2_beh(
 2      input [3:0] D,
 3      output reg [1:0] Y
 4  );
 5
 6  always @(*) begin
 7      case(D)
 8          4'b1xxx: Y = 2'b11;
 9          4'b01xx: Y = 2'b10;
10          4'b001x: Y = 2'b01;
11          4'b0001: Y = 2'b00;
12          default: Y = 2'b00;
13      endcase
14  end
15
16  endmodule
17
18
```




C:/intelFPGA/priority_encoder_4_1_dataflow.v (/priority_encoder_4_2_dataflow) - Default

```
Ln#
1  module priority_encoder_4_2_dataflow(
2  input [3:0] D,
3  output [1:0] Y
4  );
5  assign Y[1] = D[3] | D[2];
6  assign Y[0] = D[3] | (~D[2] & D[1]);
7  endmodule
8
```

Expt. No.: 6(2) : Design and Implem. Date: _____ Page No.: 39

of a 2-bit comparator (Hardware).

**Aim:-** To design & implem. a 2-bit comparator and verify it's functionality using Truth tables and Logic diagrams.

**Components req. :-**

| S.no. | Component | Specialization | Quantity |
|---|---|---|---|
| 1. | Digital trainer kit | — | 1 |
| 2. | Connecting wires | — | as much as required |
| 3. | 3-input AND gate | 74LS11 | 2 |
| 4. | 3-input OR gate | 74LS51 | 2 |
| 5. | 2-input NOT gate | 7404 | 1 |
| 6. | 2-input AND gate | 7408 | 3 |
| 7. | 2-input XNOR gate | 7486 | 2 |

**Expressions :-**

we can get,

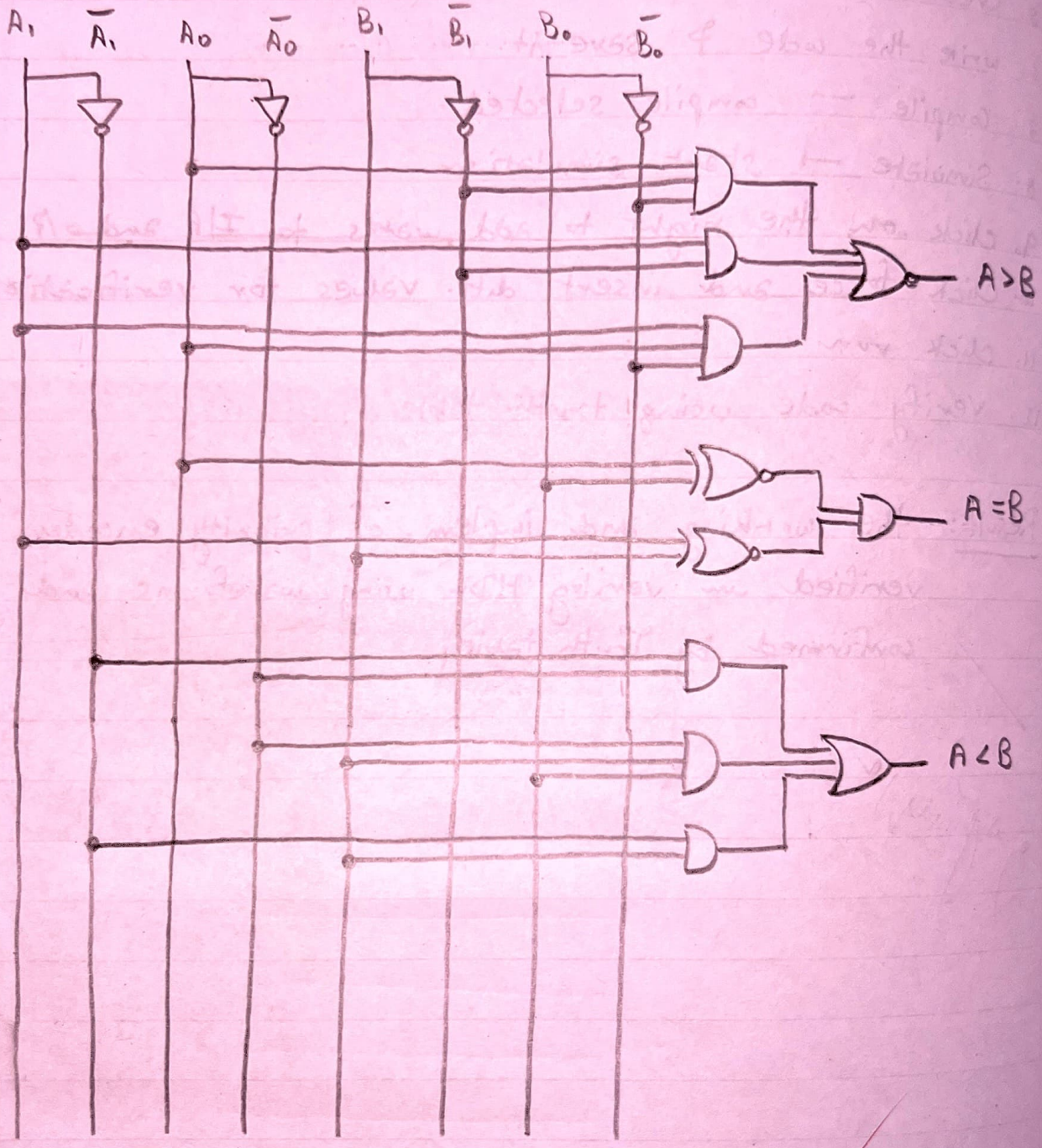
i) $A > B \Rightarrow A_1 \bar{B}_1 + A_0 \bar{B}_1 \bar{B}_0 + A_0 A_1 \bar{B}_0$

ii) $A = B \Rightarrow \overline{(A_1 \oplus B_1)} \cdot \overline{(A_0 \oplus B_0)}$

iii) $A < B \Rightarrow \bar{A}_1 B_1 + \bar{A}_0 B_1 B_0 + B_0 \bar{A}_0 \bar{A}_1$

Teacher's Signature _____

## Logic diagram:-



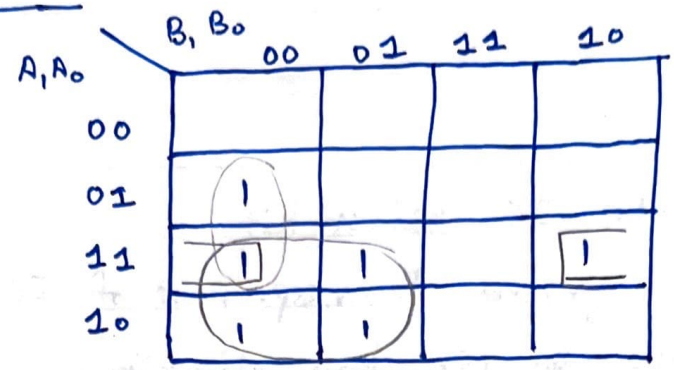
Truth table:-

| A | B | A < B | A = B | A > B |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |

| $A_1$ | $A_0$ | $B_1$ | $B_0$ | A < B | A = B | A > B |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 |

## K-map :-

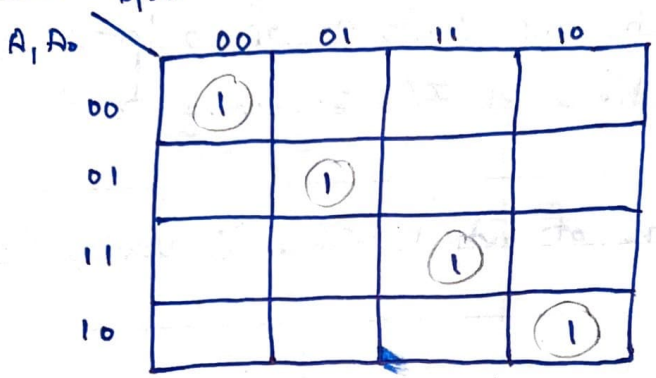
(a.) For A > B.




expression :-

$A_1\bar{B_1} + A_0\bar{B_1}\bar{B_0} + A_0A_1\bar{B_0}$

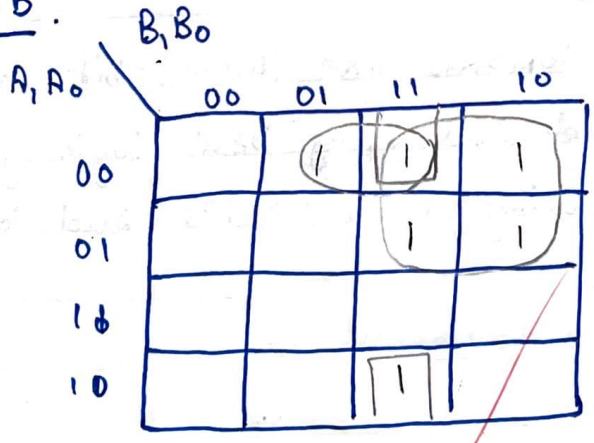
(b.) for A = B.




expression :-

$\overline{(A_1 \oplus B_1)} \cdot \overline{(A_0 \oplus B_0)}$

(c.) for A < B.




expression :-

$\bar{A_1}B_1 + \bar{A_0}B_1B_0 + B_0\bar{A_0}\bar{A_1}$.

**Aim:-** To design the working of a 4-bit magnitude comparator using Verilog HDL and verify using truth table.

**Software req. :-** Modelsim Intel FPGA Starter edition 18.1

**Theory :-**

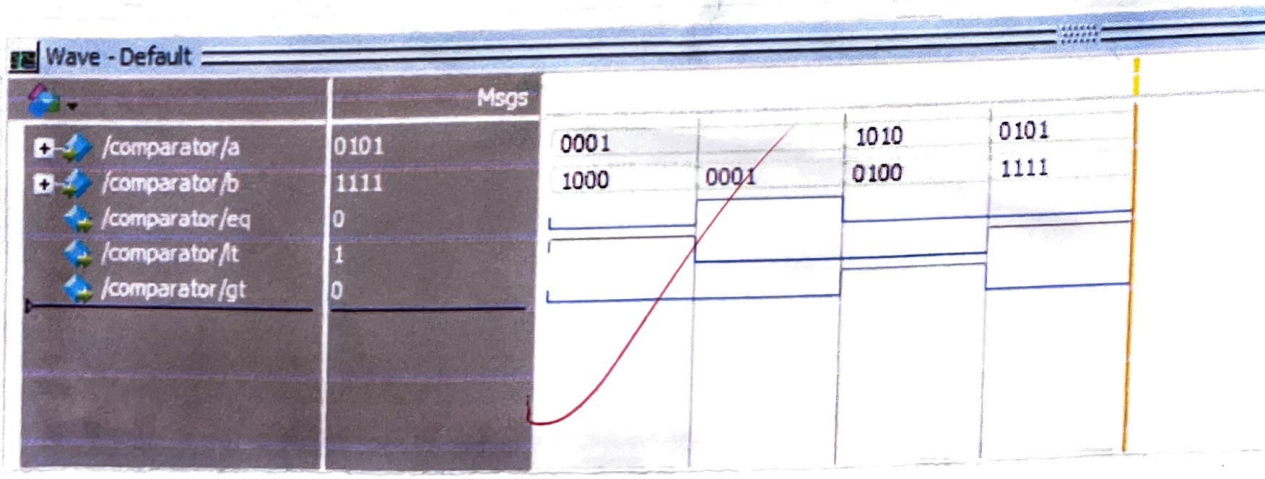
1) **Truth table :-**

| MSB | | | LSB | | | |
|---|---|---|---|---|---|---|
| $A_3\ B_3$ | $A_2\ B_2$ | $A_1\ B_1$ | $A_0\ B_0$ | $A > B$ | $A = B$ | $A < B$ |
| $A_3 > B_3$ | x | x | x | 1 | 0 | 0 |
| $A_3 < B_3$ | x | x | x | 0 | 0 | 1 |
| $A_3 = B_3$ | $A_2 > B_2$ | x | x | 1 | 0 | 0 |
| $A_3 = B_3$ | $A_2 < B_2$ | x | x | 0 | 0 | 1 |
| $A_3 = B_3$ | $A_2 = B_2$ | $A_1 > B_1$ | x | 1 | 0 | 0 |
| $A_3 = B_3$ | $A_2 = B_2$ | $A_1 < B_1$ | x | 0 | 0 | 1 |
| $A_3 = B_3$ | $A_2 = B_2$ | $A_1 = B_1$ | $A_0 > B_0$ | 1 | 0 | 0 |
| $A_3 = B_3$ | $A_2 = B_2$ | $A_1 = B_1$ | $A_0 < B_0$ | 0 | 0 | 1 |
| $A_3 = B_3$ | $A_2 = B_2$ | $A_1 = B_1$ | $A_0 = B_0$ | 0 | 1 | 0 |

Teacher's Signature

Behavioural.

```
Ln#
 1  module comparator(a,b,eq,lt,gt);
 2
 3  input [3:0] a,b;
 4  output reg eq,lt,gt;
 5
 6  always @(a,b)
 7  begin
 8   if (a==b)
 9   begin
10    eq = 1'b1;
11    lt = 1'b0;
12    gt = 1'b0;
13   end
14   else if (a>b)
15   begin
16    eq = 1'b0;
17    lt = 1'b0;
18    gt = 1'b1;
19   end
20   else
21   begin
22    eq = 1'b0;
23    lt = 1'b1;
24    gt = 1'b0;
25   end
26  end
27  endmodule
28
```



# Theorehical calculation

3 * 5

M = 3 = 0011

Q = 5 = 0101

1st of M = 1100

2nd of M = 1100
+1
→ M = 1101

1110
+0011
10001

| Cycle | A | Q | $Q_{n-1}$ |
|---|---|---|---|
| initial value | 0000 | 0101 | 0 |
| 1st. | 1101 | 0101 | 0 |
| A = A-m | | | |
| right shift | 1110 | 1010 | 1 |
| 2nd. | 0001 | 1010 | 1 |
| A = A+m | | | |
| right shift | 0000 | 1101 | 0 |
| 3rd. | 1101 | 1101 | 0 |
| A = A-m | 1110 | 1110 | 1 |
| right shift | | | |
| 4th. | | | |
| A = A+m | 0001 | 1110 | 1 |
| right shift | 0000 | 1111 | 0 |

15.

```
Ln#
 1  module booth_multiplication (
 2      input signed [3:0] multiplicand,
 3      input signed [3:0] multiplier,
 4      output reg signed [7:0] product
 5  );
 6
 7      reg signed [4:0] A;
 8      reg signed [3:0] Q, M;
 9      reg Q_1;
10      integer i;
11
12      always @(*) begin
13          A = 5'b00000;
14          Q = multiplier;
15          M = multiplicand;
16          Q_1 = 0;
17
18          for (i = 0; i < 4; i = i + 1) begin
19              if (Q[0] == 1 && Q_1 == 0) begin
20                  A = A - M;
21              end else if (Q[0] == 0 && Q_1 == 1) begin
22                  A = A + M;
23              end
24
25
26              {A, Q, Q_1} = {A[4], A, Q, Q_1} >>> 1;
27          end
28
29          product = {A, Q};
30      end
31
32  endmodule
```

```
VSIM 15> run
# Multiplicand=  3, Multiplier=  5, Product=  15
# Multiplicand= -4, Multiplier=  3, Product= -12
# Multiplicand= -6, Multiplier= -2, Product=  12
# Multiplicand=  6, Multiplier= -3, Product= -18
# ** Note: $finish    : C:/intelFPGA/18.1/24bail508/booths.v(65)
#    Time: 40 ps  Iteration: 0  Instance: /tb_booth_multiplication
```

```
Ln#
33
34  module tb_booth_multiplication;
35      reg signed [3:0] multiplicand;
36      reg signed [3:0] multiplier;
37      wire signed [7:0] product;
38
39      booth_multiplication uut (
40          .multiplicand(multiplicand),
41          .multiplier(multiplier),
42          .product(product)
43      );
44
45      initial begin
46          $monitor("Multiplicand=%d, Multiplier=%d, Product=%d", multiplicand, multiplier, product);
47
48          multiplicand = 4'sb0011;
49          multiplier = 4'sb0101;
50          #10;
51
52          multiplicand = 4'sb1100;
53          multiplier = 4'sb0011;
54          #10;
55
56          multiplicand = 4'sb1010;
57          multiplier = 4'sb1110;
58          #10;
59
60          multiplicand = 4'sb0110;
61          multiplier = 4'sb1101;
62          #10;
63
64          $finish;
65      end
66  endmodule
67
```

Expt. No.: 8 : Design & implem. of SR, JK, T & D flip flop. Date: _____ Page No.: 45

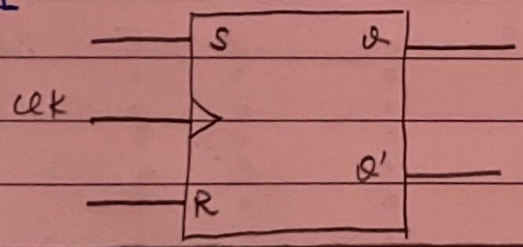
**Aim:** To design & implement SR, JK, T & D flip-flops.

**Components required:**

| S.no. | Component | Specification | Quantity |
|---|---|---|---|
| 1. | 2-i/p Nand | IC 7400 | 1 |
| 2. | 1-i/p Not | IC 7404 | 1 |
| 3. | trainer kit | — | 1 |
| 4. | connecting wires | — | few |

**Flip-flops:-**

(a.) **SR flip flop.**



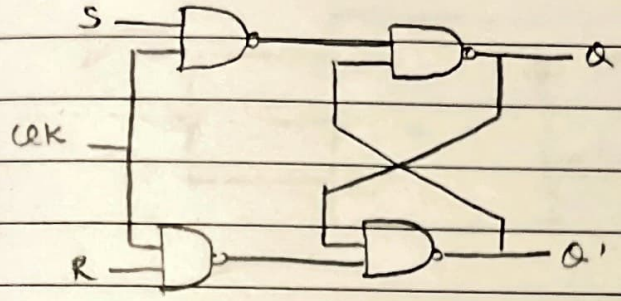

truth table:

| clk | S | R | Qn | Qn+1 | state |
|---|---|---|---|---|---|
| ↑ | 0 | 0 | 0 | 0 | no change |
| | 0 | 0 | 1 | 1 | |
| ↑ | 0 | 1 | 0 | 0 | reset |
| | 0 | 1 | 1 | 0 | |
| ↑ | 1 | 1 | 0 | 1 | set |
| | 0 | 0 | 1 | 1 | |
| ↑ | 1 | 1 | 0 | X | indeterminate |
| | 1 | 1 | 1 | X | |
| ↓ | X | X | 0 | 0 | no change. |
| | X | X | 1 | 1 | |

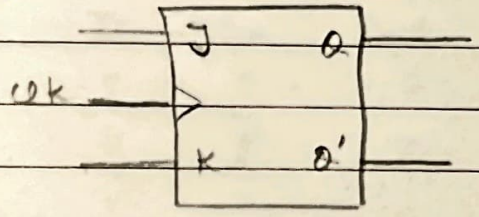
Teacher's Signature _____

MY CHOICE

logic diagram:-



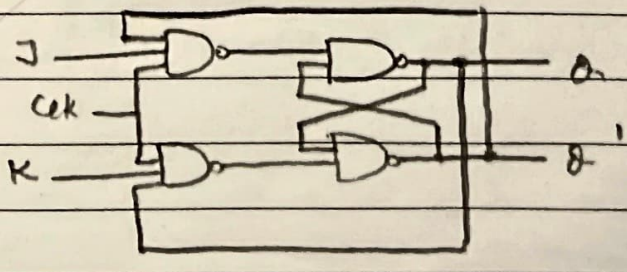

(b.) JK - flip flop.




truth table:

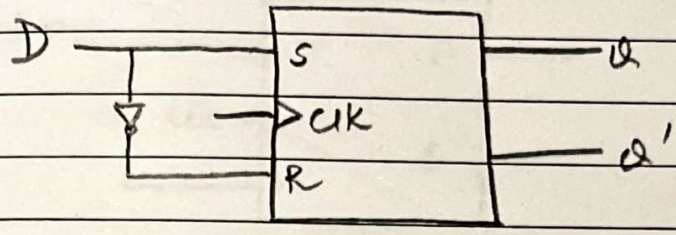
| clk | J | K | $Q_n$ | $Q_{n+1}$ | state |
|---|---|---|---|---|---|
| ↑ | 0 | 0 | 0 | 0 | |
| | 0 | 0 | 1 | 1 | No change |
| ↑ | 0 | 1 | 0 | 0 | |
| | 0 | 1 | 1 | 0 | Reset |
| ↑ | 1 | 0 | 0 | 1 | |
| | 1 | 0 | 1 | 1 | Set |
| ↑ | 1 | 1 | 0 | 1 | |
| | 1 | 1 | 1 | 0 | Toggle |
| ↓ | X | X | 0 | 0 | |
| | X | X | 1 | 1 | No change |

logic diagram:-




Teacher's Signature _____

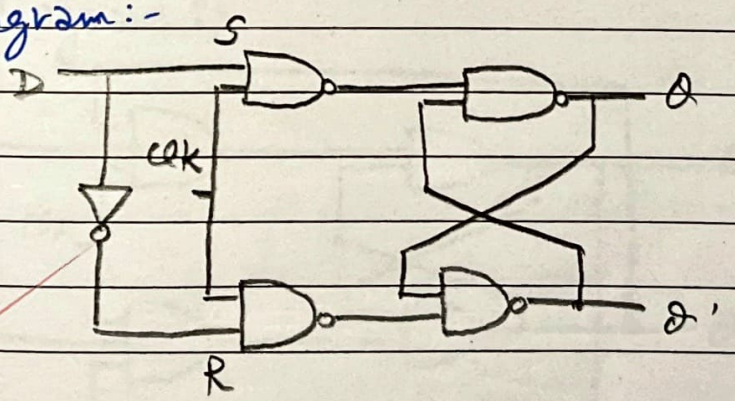
(c.) D- Flip flop.




truth table:-

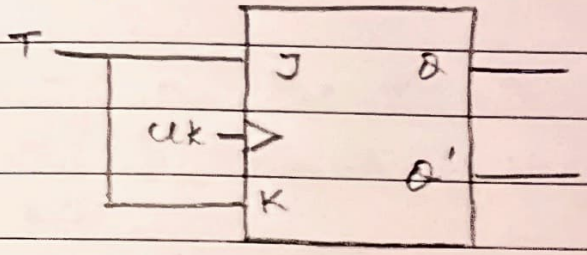
| Clk | D | $Q_n$ | $Q_{n+1}$ | State |
|---|---|---|---|---|
| ↑ | 0 | 0 | 0 | Reset |
| ↑ | 0 | 1 | 0 | Reset |
| ↑ | 1 | 0 | 1 | Set |
| ↑ | 1 | 1 | 1 | Set |
| ↓ | X | $Q_n$ | $Q_n$ | No change |

logic diagram:-




Teacher's Signature _____

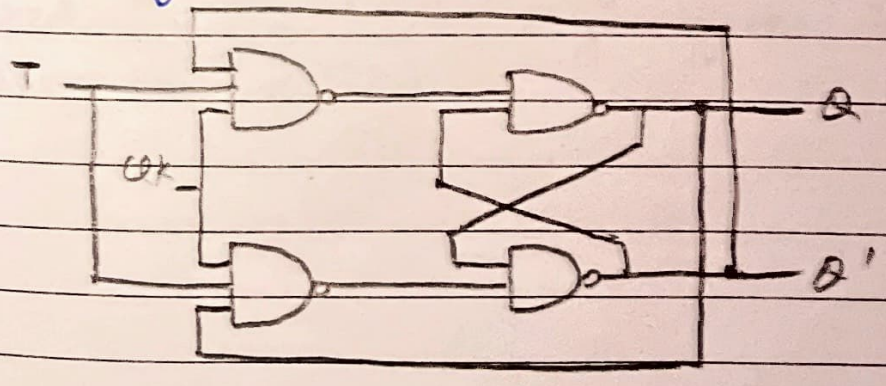
(d.) T- Flip flop.




Truth table

| Clk | T | $Q_n$ | $Q_{n+1}$ | State |
|---|---|---|---|---|
| ↑ | 0 | 0 | 0 | no change |
| ↑ | 0 | 1 | 1 | no change |
| ↑ | 1 | 0 | 1 | toggle |
| ↑ | 1 | 1 | 0 | toggle |
| ↓ | X | $Q_n$ | $Q_n$ | no change. |

logic diagram:-




Teacher's Signature _____

Expt. No.: 9 : Design and implem. Date: _____ Page No.: 50

of 4-Bit Shift registers
[SISO, SIPO, PIPO]

**Aim:** To design & implement 4-Bit Shift registers such as SISO, SIPO, PIPO.

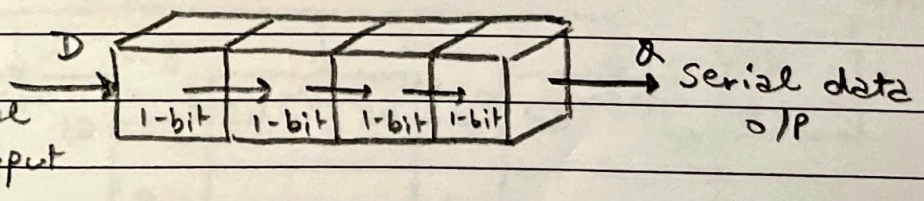
**Components req. :-**

| S.no. | Component | Specification | quantity |
|---|---|---|---|
| 1. | D- flip flop | IC 7474 | 2 |
| 2. | Digital tr. kit | — | 1 |
| 3. | Connecting wires | — | few. |

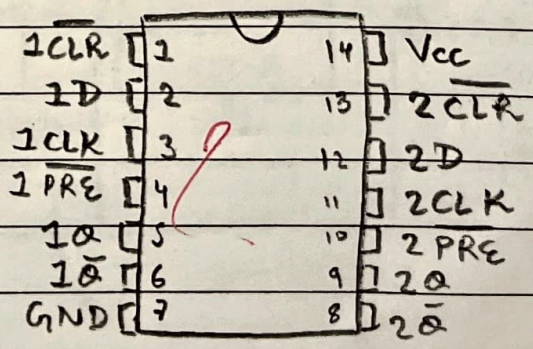
**Theory :-**

**(a) SISO (Serial in - Serial out).**

**Block diagram :-**




**IC 7474 pin diagram :-**



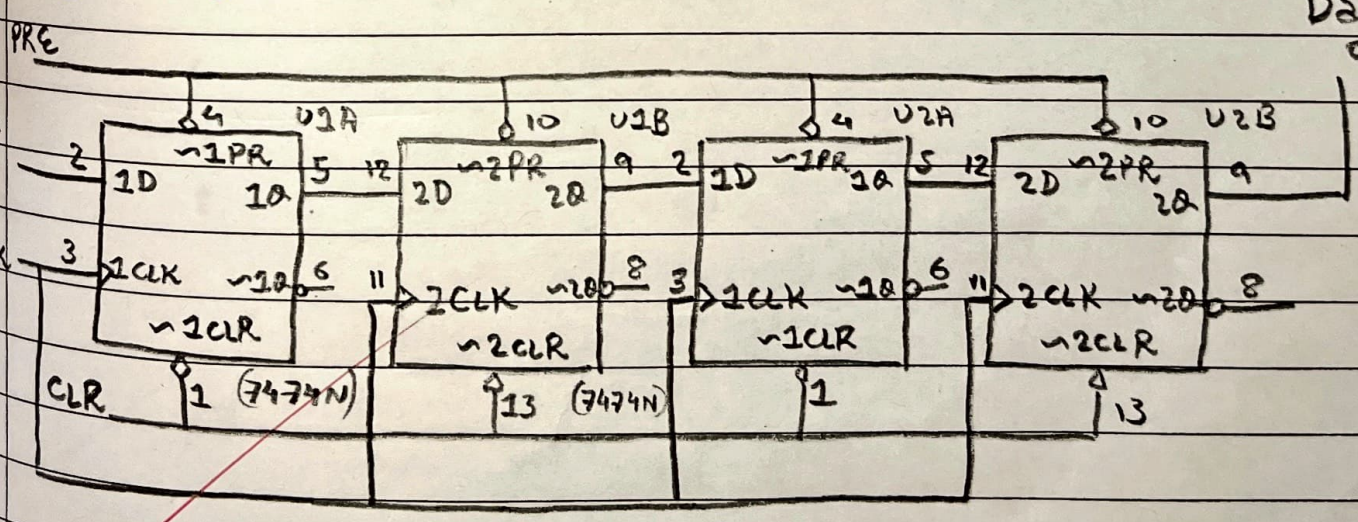

Teacher's Signature _____

MY CHOICE

## Truth table :-

| Clock pulse no. | D | $Q_3$ | $Q_2$ | $Q_1$ | $Q_0$ | Q |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 1 | 1 | 1 | 0 | 0 | 0 |
| 3 | 0 | 0 | 1 | 1 | 0 | 0 |
| 4 | 1 | 1 | 0 | 1 | 1 | 1 |
| 5 | 0 | 0 | 1 | 0 | 1 | 1 |
| 6 | 0 | 0 | 0 | 1 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 1 | 1 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 |

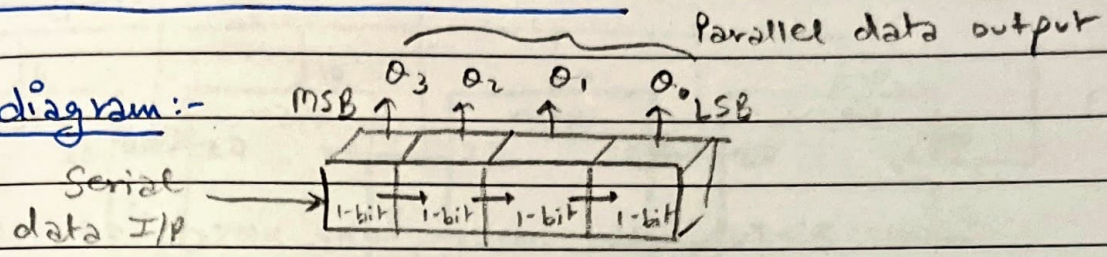
## Logic diagram :-




Teacher's Signature _____

(b) SIPO (Serial in - Parallel out). :-

Block diagram :-



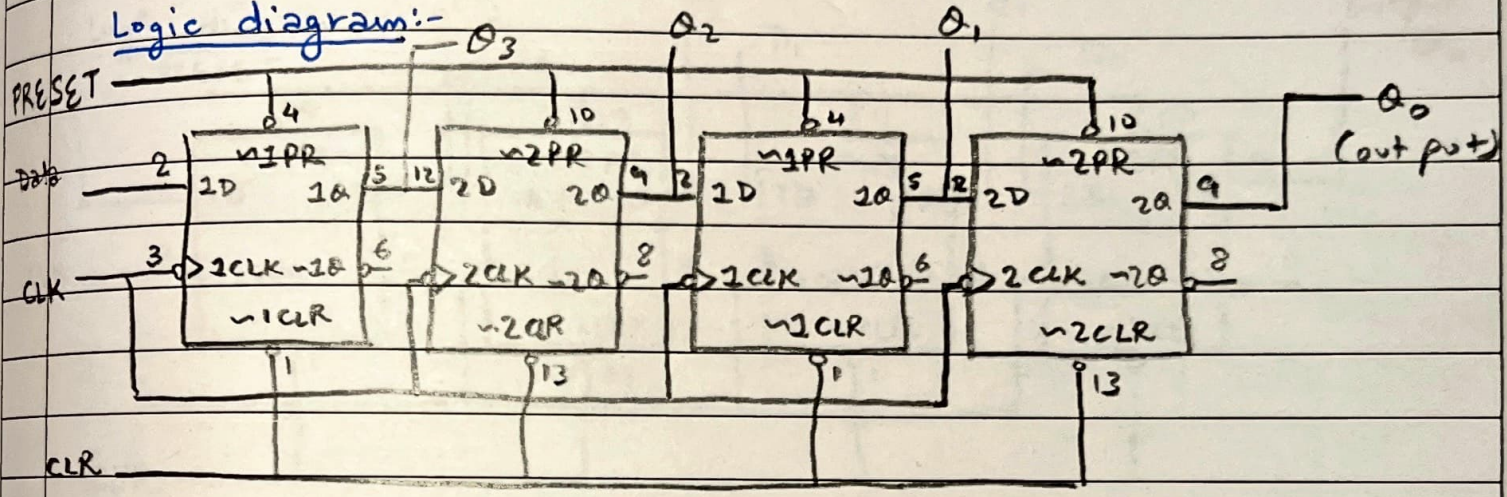

Truth table :-

| Clock pulse no. | D | $Q_3$ | $Q_2$ | $Q_1$ | $Q_0$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |
| 2 | 1 | 1 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 1 | 0 |
| 4 | 1 | 1 | 0 | 1 | 1 |
| 5 | 0 | 0 | 1 | 0 | 1 |
| 6 | 0 | 0 | 0 | 1 | 0 |
| 7 | 0 | 0 | 0 | 0 | 1 |
| 8 | 0 | 0 | 0 | 0 | 0 |

Teacher's Signature _____

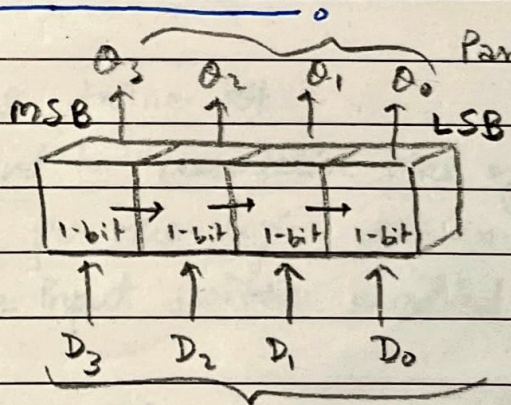
MY CHOICE

## Logic diagram:-




## (c.) PIPO (Parallel in - Parallel out) :-

## Block diagram:-



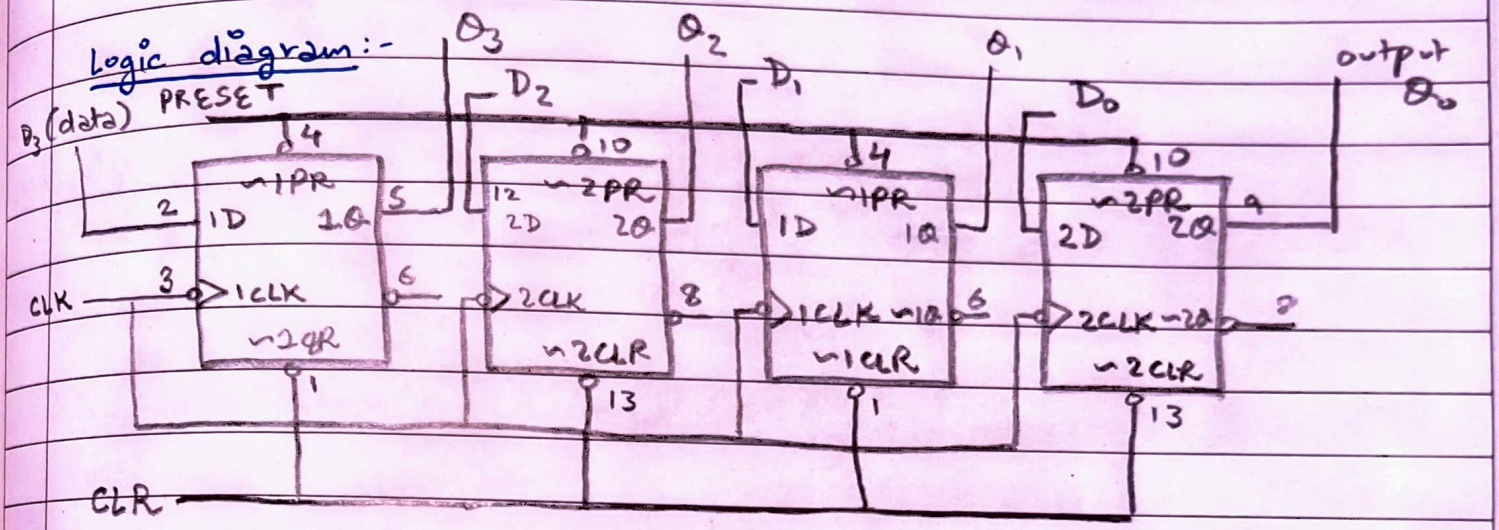

## Truth table :-

| Clock pulse no. | $D_3$ | $D_2$ | $D_1$ | $D_0$ | $Q_3$ | $Q_2$ | $Q_1$ | $Q_0$ |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 2 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 3 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 4 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |

Teacher's Signature _____

MY CHOICE

Logic diagram:-




Procedure :-

1). Place the IC on the IC trainer kit
2) Connect the Vcc and ground to respective pins of IC trainer kit.
3). Connections are given as per the logic diagram.
4). Connect the inputs to the input switches provided in the IC trainer kit.
5). Connect the outputs to the switches of output LEDs
6). Apply various combinations of input according to truth table.
7) Observe the condition of output LEDs and verify the truth table.

Result

The 4 bit shift registers as SIPO, SISO, PIPO are constructed using D flipflop IC(7474) and their truth tables are verified.

Teacher's Signature _____

Expt. No.: 10 (a) Date: Page No.: 55

MY CHOICE

## Implementation of Consecutive three ones detector using Verilog HDL.

**Aim:-** To create a simple FSM that can detect three ones using verilog HDL and verifying it using truth tables and the diagram.

**Software Reqaired :-** Model Sim FGPA Starter Edition 18.1 Intel

**Theory**

Truth Table.

| $A_n$ | $B_n$ | C (Input) | D (Output) | $A_{n+1}$ | $B_{n+1}$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 |

Teacher's Signature

MY CHOICE

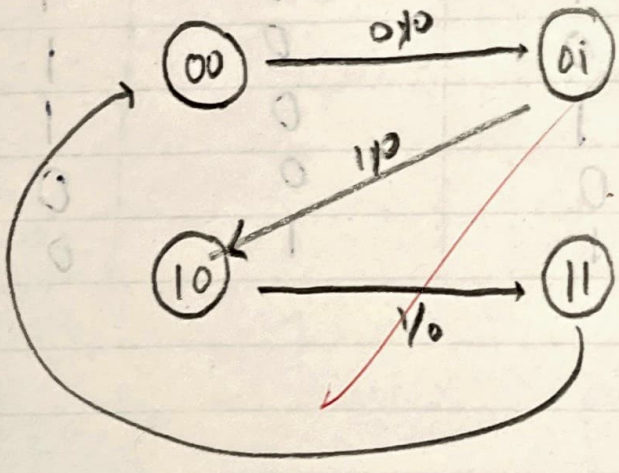
Procedure :-

1). Open Modelsim
2). Layout → Reset
3). File → New → Folder → Name Folder → ok
4). File → New Project → Select Folder → ok
5). Create new file
6) Write the programs.
7). Compile → Compile Selected
8) Simulate → Start Simulation
9). Click each of input and force required values.
10). Verify with truth table.

Result :- The FSM based detector is verified and with truth table and FSM diagram.

Teacher's Signature _____

FSM diagram (Mealy).




S.T

Expt. No.: 10(b). Date: Page No.: 52

# Automatic Traffic Light using Verilog HDL.

Aim - To create an automatic traffic light which gives colors in the waveform using verilog HDL.

Software Requried :- ModelSim Intel 18:1 FGPA Starter Edition.

Theory

Truth Table

| Present State | | Input | Output | | | Next State | |
|---|---|---|---|---|---|---|---|
| A | B | C | G | Y | R | $A_N$ | $B_N$ |
| 0 | 0 | 0 | x | x | x | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | x | x | x | x | x |
| 1 | 1 | 1 | x | x | x | x | x |

11 condition isn't there acording to FSM logic.

Teacher's Signature

Procedure :-

⇒ Open Model Sim
⇒ Layout → Reset
⇒ File → New → Folder → Name Folder → Ok
⇒ File → New Project → Name → Browse → Select the file
⇒ Create new file
⇒ Write the program
⇒ Compile selected and start simulation
⇒ Check for output while forcing values.
⇒ Verify using truth tables.

Result :- The creation of an automatic traffic light system using FSM logic in verilog HDL and verified using truth table and diagram.

C:/intelFPGA/18.1/traffic.v (/traffic_light) - Default

```
Ln#
 1  module traffic_light (clk,reset,in,red,yellow,green);
 2
 3  input wire clk,reset,in;
 4  output reg red,yellow,green;
 5
 6  parameter IDLE=2'b00;
 7  parameter ONE=2'b01;
 8  parameter TWO_ONES=2'b10;
 9  parameter THREE_ONES=2'b11;
10
11  reg [1:0] current_state,next_state;
12
13  always @ (posedge clk or posedge reset)
14  begin if (reset)
15  begin
16     current_state<=IDLE;
17  end
18  else begin
19     current_state<=next_state;
20  end
21  end
22
23  always @ (*)
24  begin
25  case (current_state)
26
27  IDLE:begin
28  if (in==1) begin
29     next_state = TWO_ONES;
30     green=1'b1;
31     red=1'b0;
32     yellow=1'b0;
33  end
34  else begin
35     next_state=IDLE;
36     green=1'bx;
37     red=1'bx;
38     yellow=1'bx;
39  end
40  end
41
42  ONE:begin
43  if (in==0) begin
44  next_state = IDLE;
45     green=1'b0;
46     red=1'b0;
47     yellow=1'b1;
48  end
49  else begin
50     next_state=IDLE;
51  end
52  end
```

C:/intelFPGA/18.1/ADD.v (/detect_three_ones) - Default *

```
Ln#
 1  module detect_three_ones(clk,reset,in,out);
 2
 3  input wire clk,reset,in;
 4  output reg out;
 5
 6  parameter IDLE=2'b00;
 7  parameter ONE=2'b01;
 8  parameter TWO_ONES=2'b10;
 9  parameter THREE_ONES=2'b11;
10
11  reg [1:0] current_state,next_state;
12
13  always @ (posedge clk or posedge reset)
14    begin if (reset)
15      begin
16        current_state<=IDLE;
17        out<=0;
18      end
19    else begin
20        current_state<=next_state;
21    end
22  end
23
24  always @ (*)
25  begin
26    case(current_state)
27      IDLE:begin
28      if (in==1) begin
29        next_state=ONE;
30      end
31      else begin
32        next_state=IDLE;
33      end
34      out=1'b0;
35      end
36
37      ONE:begin
38      if (in==1) begin
39        next_state=TWO_ONES;
40      end
41      else begin
42        next_state=IDLE;
43      end
44      out=1'b0;
45      end
```

```
      TWO_ONES:begin
      if (in==1) begin
        next_state=THREE_ONES;
      end
      else begin
        next_state=IDLE;
      end
      out=1'b0;
      end

      THREE_ONES:begin
      if (in==1) begin
        next_state=THREE_ONES;
      end
      else begin
        next_state=IDLE;
      end
      out=1'b1;
      end

      default:begin
        next_state=IDLE;
        out=1'b0;
      end
  endcase
end

endmodule
```