PAGE NO.:
DATE: / /

Unit: 4 :-

Behavioral modelling

There are two structured procedure in verilog "always" and "initial". this two statement are basic statement in behavorial modelling. Activity flows in verilog in parallel. Each activity flow start at stimulation time zero.

* Initial statement :-

All statement inside initial statement constitute in a block. An initial blocks start at time 0, execute at exactly ones and then does not execute again. If there are multiple initial blocks each block starts of execution at time zero. Each block finishes execution independently of other blocks.

* Multiple behavioral statement must be group 'begin' and 'end' if there is only one behavioral statement then grouping is not necessary.

ex :-

```
module stimulus;
reg x,y,a,b,m;
initial
m = 1'b0; // single statement
```

PAGE NO.:
DATE: / /

```
initial
  begin
  #5      a = 1'b1; // multiple Statement
  #25     a = 1'b0; #
  end
initial
  begin
  #10     x = 1'b0;
  #25     y = 1'b1;
  end
initial
  #50     $finish;
endmodule
```

Time	Statement executed.
0	m = 1'b0;
5	a = 1'b1;
10	x = 1'b0;
30	b = 1'b0;
35	y = 1'b1;
50	$finesh;

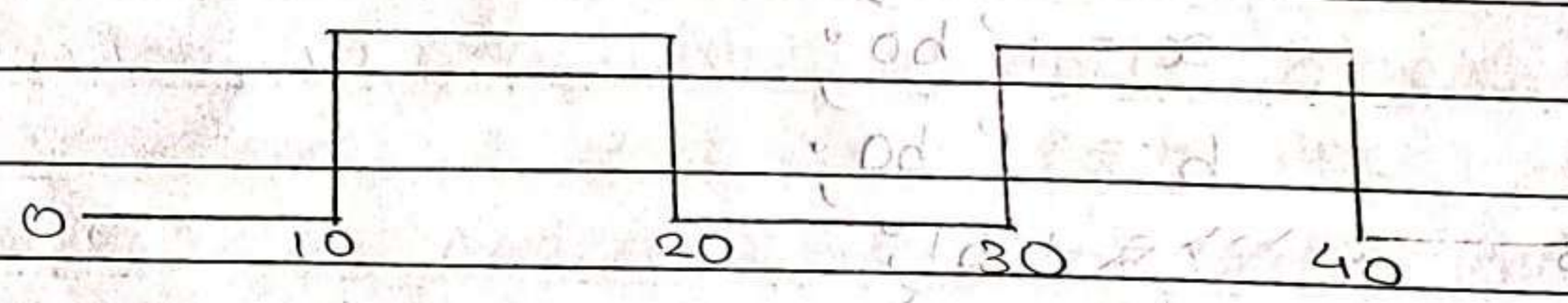
★ Always Statement :-

The always statement starts at time 0 and executes the statement in the always block in a looping fashion.

for example - clock generator module that toggles the clock signal every half cycle

ex :-

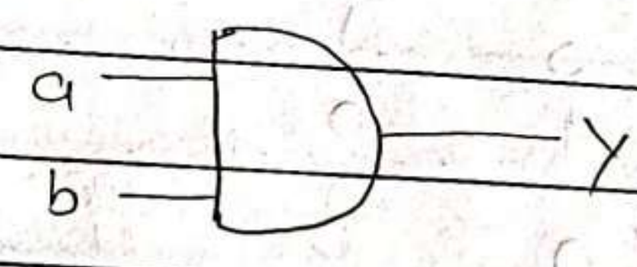
```
module clockgen (clock);
output clock;
reg clock;

// initialize clock at time zero
initial
  clock = 1'b0;

// initialize clock every halfcycle
always
  #10 clock = ~ clock;
initial
  # 1000 $ finish
endmodule.
```



PAGE NO.:
DATE: / /

* And gate :-

a	b	Y
0	0	0
0	1	0
1	0	0
1	1	1

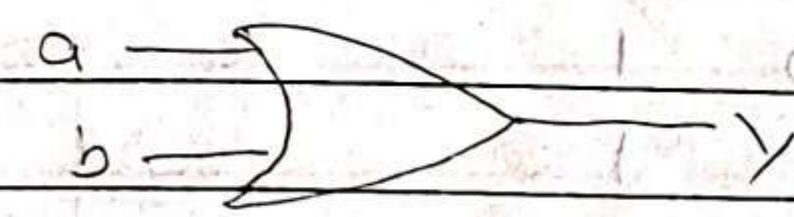


$Y = a \cdot b$

```
module and-gate (Y, a, b);
output Y;                    output reg Y;
reg Y;
input a, b;
always @ (a, b).   // only input no output no space
Y = a & b;
endmodule
```

* OR gate :-

a	b	Y
0	0	0
0	1	1
0	0	1
1	1	1

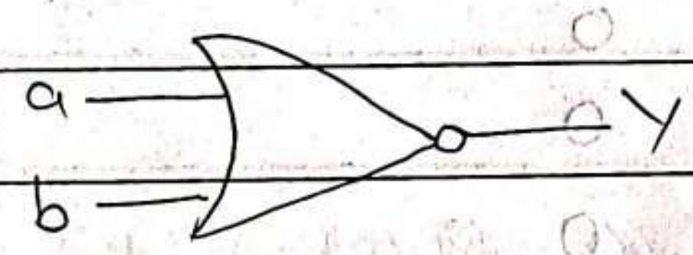


$Y = a + b$

```
module or-gate (Y, a, b);
output Y;
reg Y;
input a, b;
always @ (a, b)
Y = a | b;
end
```

* Nor gate :-

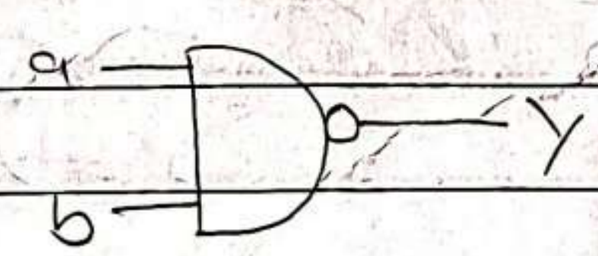
a	b	Y
0	0	1
0	1	0
1	0	0
1	1	0




```
module nor-gate (Y,a,b);
output Y;
reg Y;
input a,b;
always @(a,b)
Y= ~(a|b);
endmodule
```

* Nand gate :-

a	b	Y
0	0	1
0	1	1
1	0	1
1	1	0

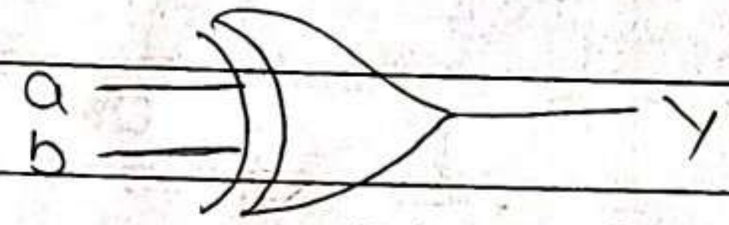



```
module nand-gate (Y,a,b);
output Y;
reg Y;
input a,b;
always@(a,b)
```

```
Y= ~(a & b);
assig
endmodule
```

* xor gates-

a	b	Y
0	0	0
0	1	1
1	0	1
1	1	0



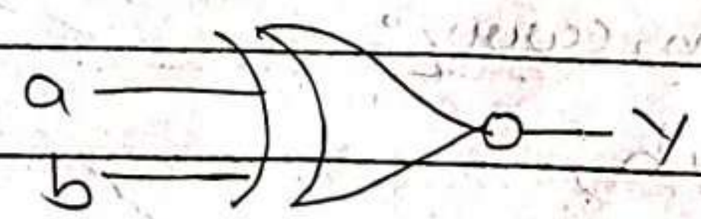
$Y = \bar{a}b + a\bar{b}$

$Y = a \oplus b$

```
module xor-gate(Y, a, b);
output Y;
reg Y;
input a,b;
always@(a,b);
Y= a^b;        // Y=((~a) & b) | (a & (~b))
endmodule
```

* xnor gates-

a	b	Y
0	0	1
0	1	0
1	0	0
1	1	1



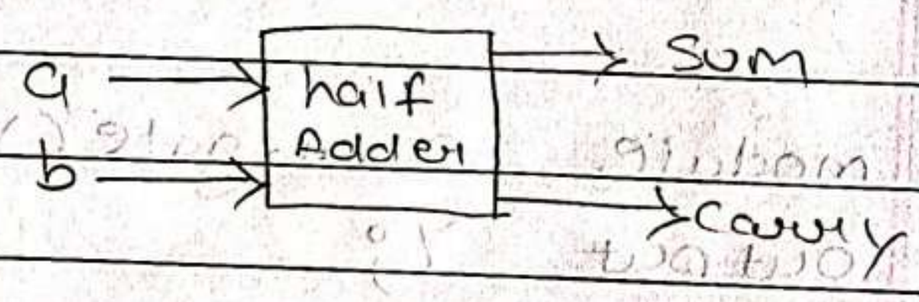
$Y = \bar{a}\bar{b} + ab$

$= \overline{a \oplus b}$

```
module xnor-gate(Y,a,b);
output Y;
reg Y;
input a,b;
always@(a,b)
Y = ~(a^b);        // Y = ((~a)&(~b)) | (a&b));
endmodule
```

* Half Adder:-

a	b	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Sum, $S = \bar{a}b + a\bar{b}$ Carry = $C = A \cdot B$

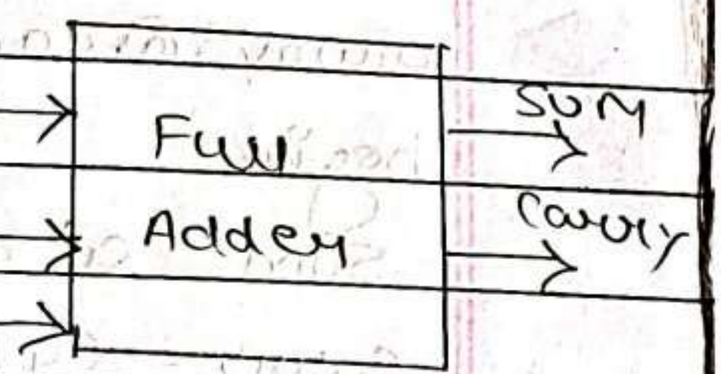
$S = a \oplus b$

```
module half-adder(sum,carry,a,b);
output sum,carry;
reg sum,carry;
input a,b;
always@(a,b)
begin
Sum = a^b;        // Sum = ((~a)&b)) | (a&(~b));
Carry = a&b;
end
endmodule
```

PAGE NO.:
DATE: / /

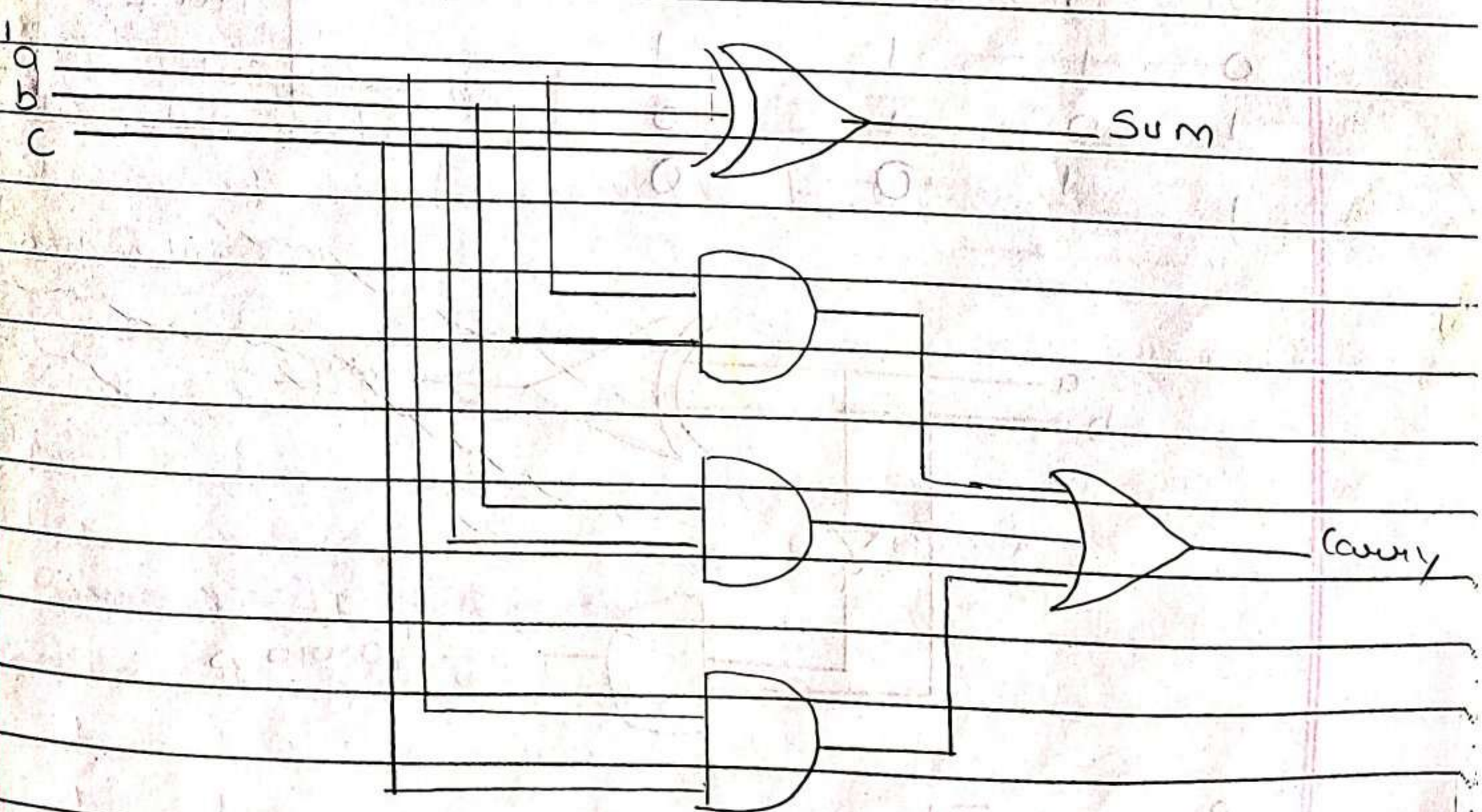
* Full Adder :-

a	b	c	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



$Sum = a \oplus b \oplus c$

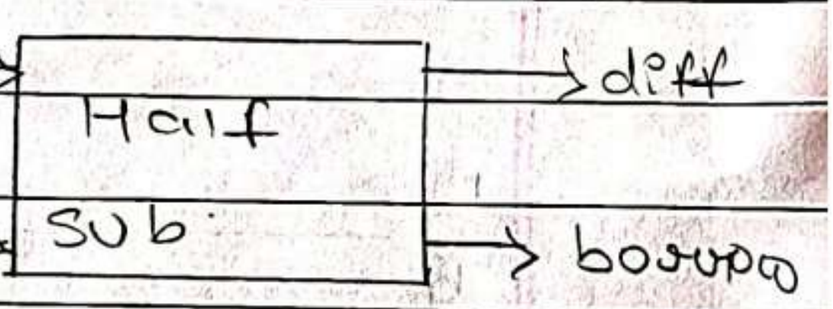
$Carry = ab + bc + ac$

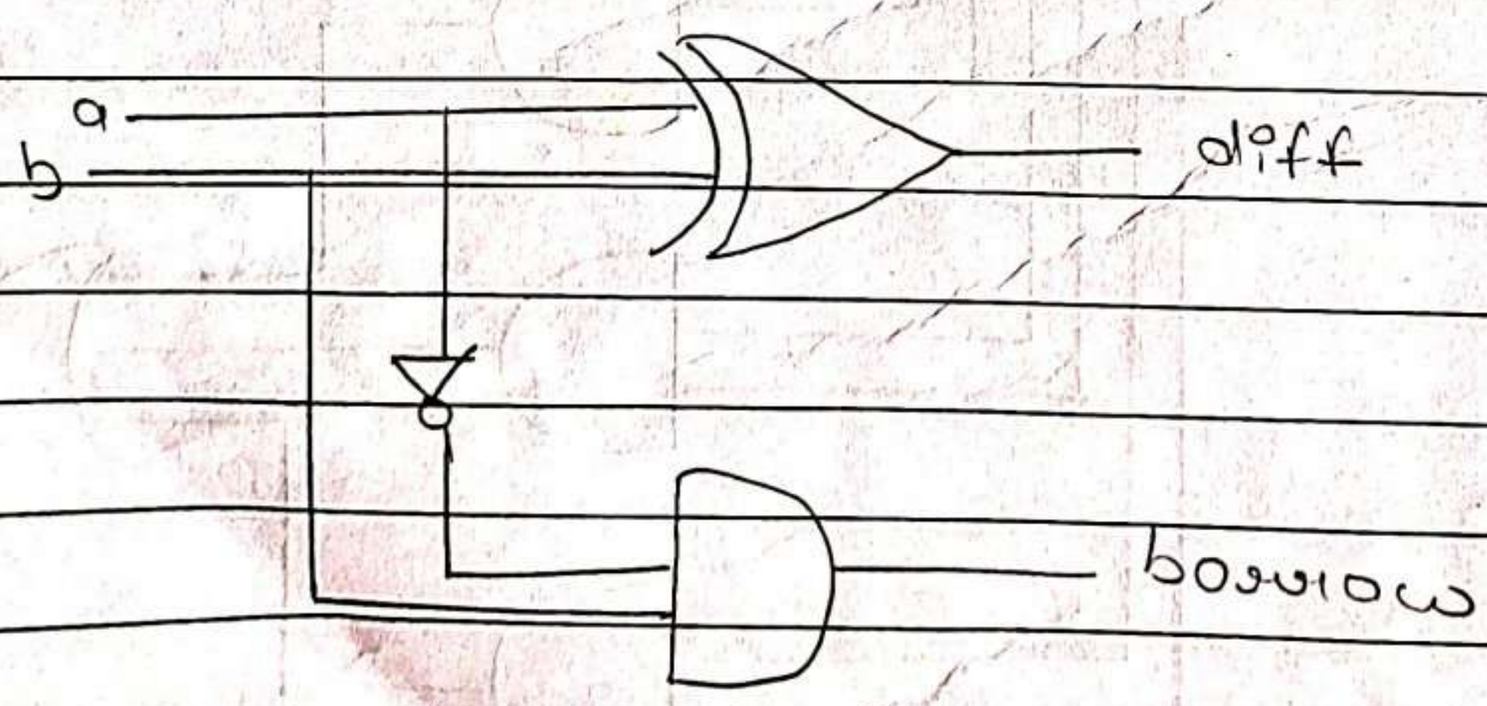


```
module full_adder (sum, carry, a, b, c);
output sum, carry;
reg sum, carry;
input a, b, c;
always @ (a, b, c)
begin
 sum = a^ b^ c;
 carry = (a&b) | (b&c) | (a&c);
end
endmodule
```

Half substractor :-

a	b	borrow	difference
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0





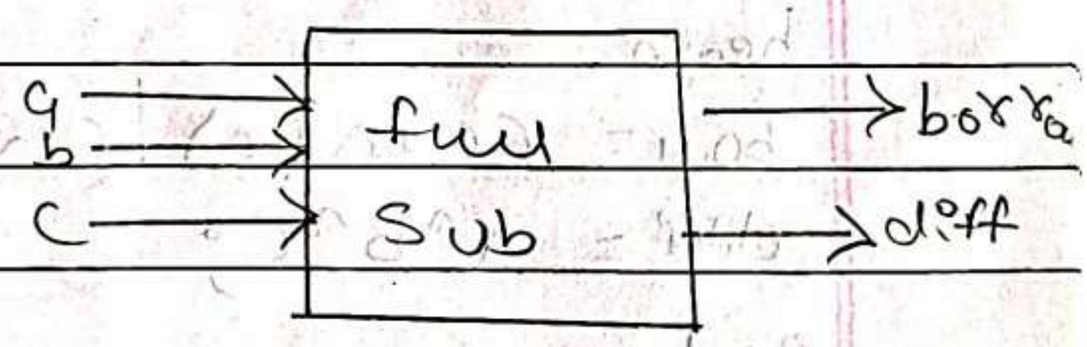
Borrow = $\bar{a}b$

~~Sub~~ diff = $a \oplus b$

```
module half-sub (borrow, diff, a, b);
output borrow, diff;
reg borrow, diff;
input a, b;
always @ (a,b)
begin
borrow = ((~a) & b);
diff = a ^ b;
end
endmodule
```

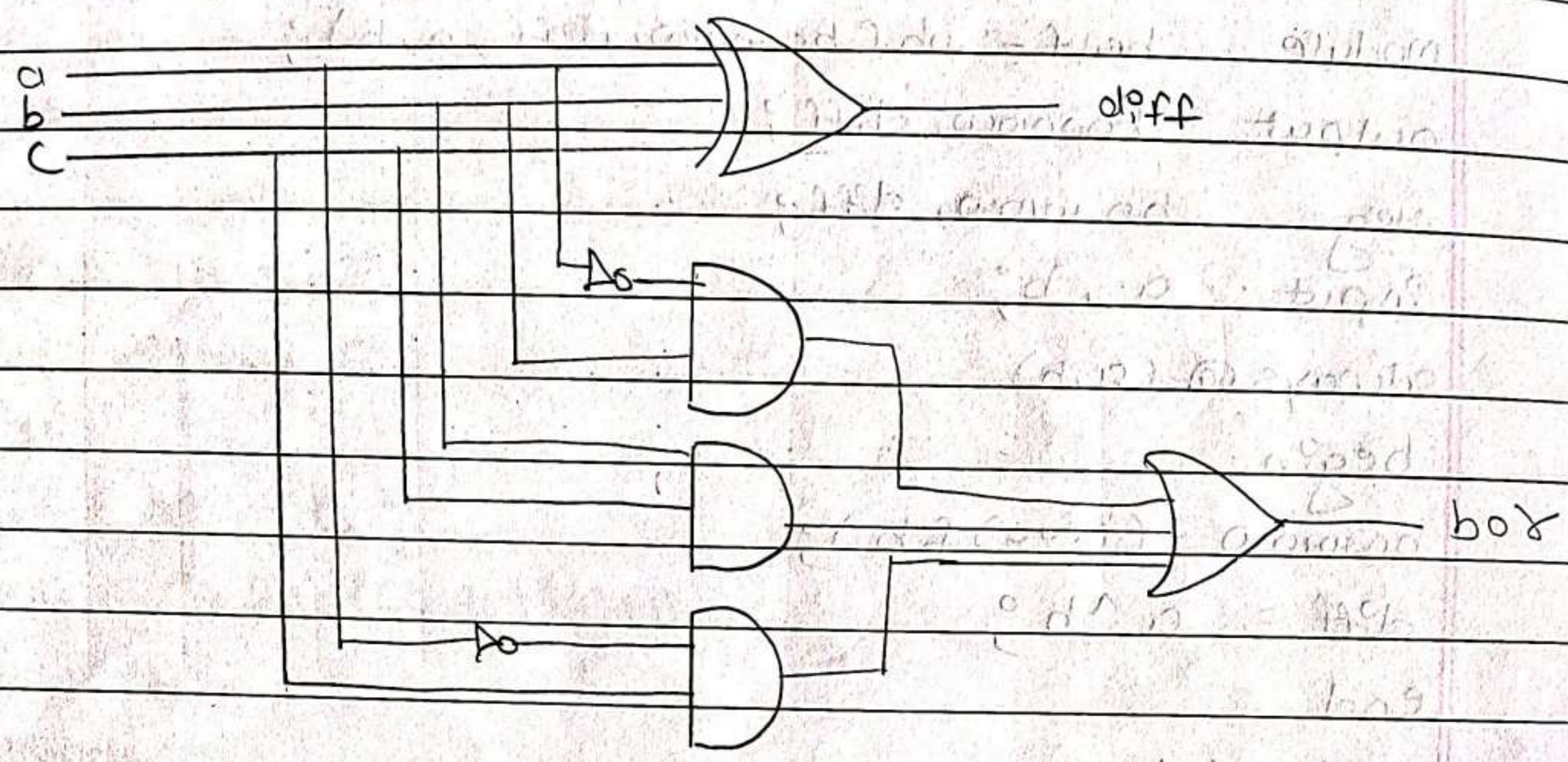
* Full Substractor :-

a	b	c	diff borrow	borrow diff
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



$Borrow = \bar{a}c + \bar{a}b + bc$

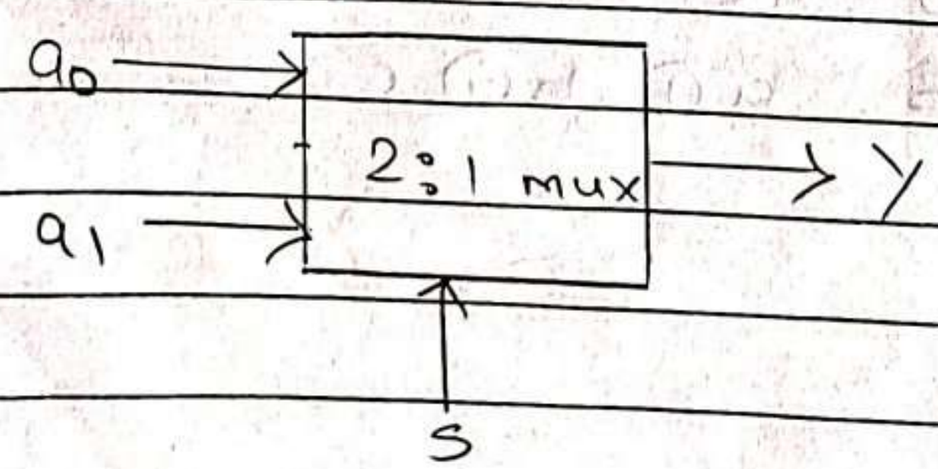
$diff = a \oplus b \oplus c$



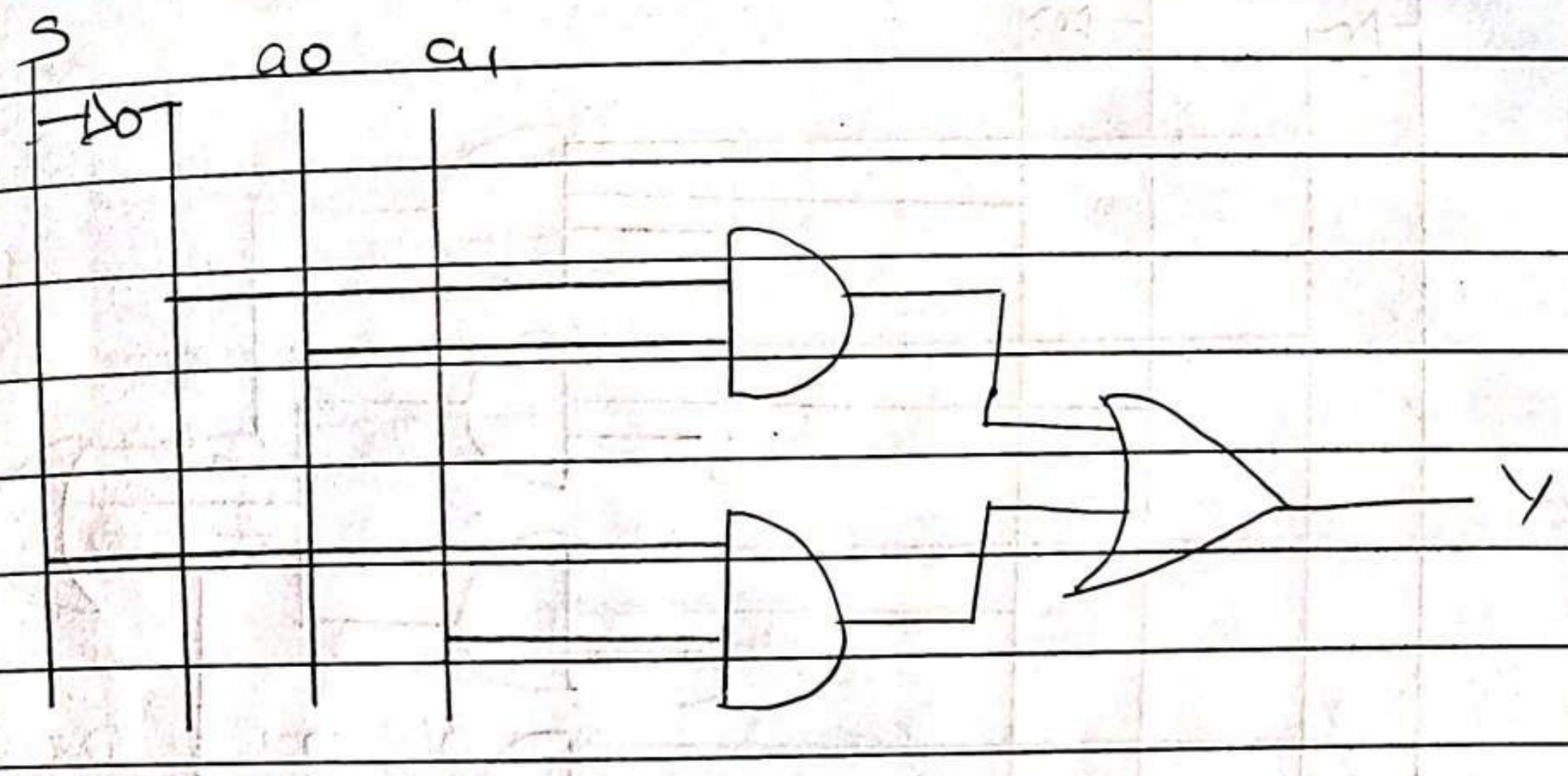

```
Module full-sub(bor, diff, a, b, c);
output bor, diff;
reg bor, diff;
input a, b, c;
always@(a,b,c)
begin
bor = ((~a)&c) | ((~a)&b) | (b&c);
diff = a^b^c;
end
endmodule
```

* 2:1 mux :-

I/P	O/P
S	Y
0	a_0
1	a_1



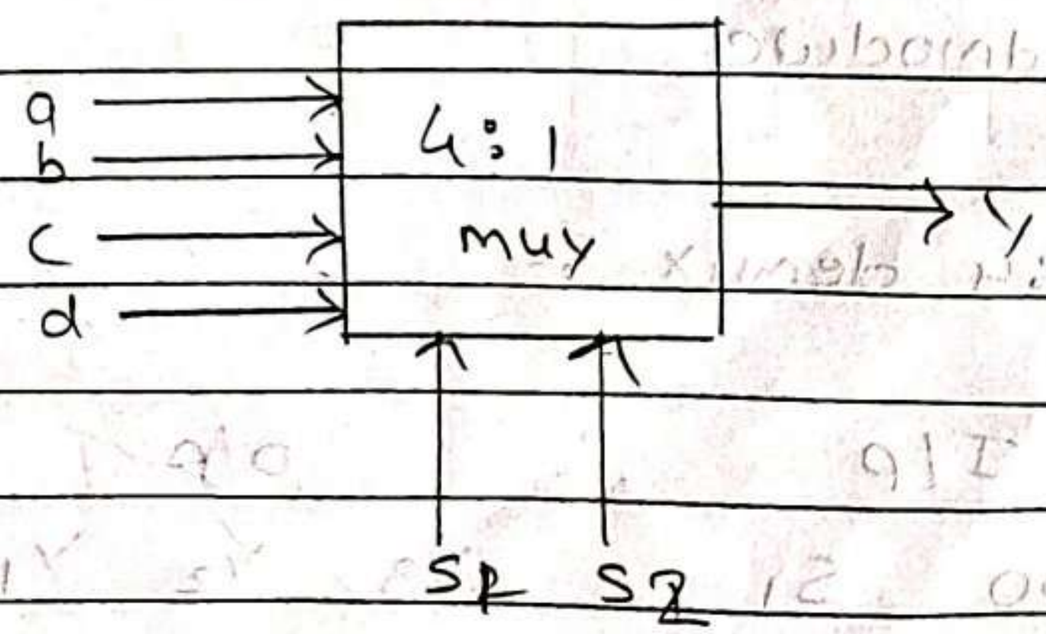
$Y = \bar{S}a_0 + Sa_1$



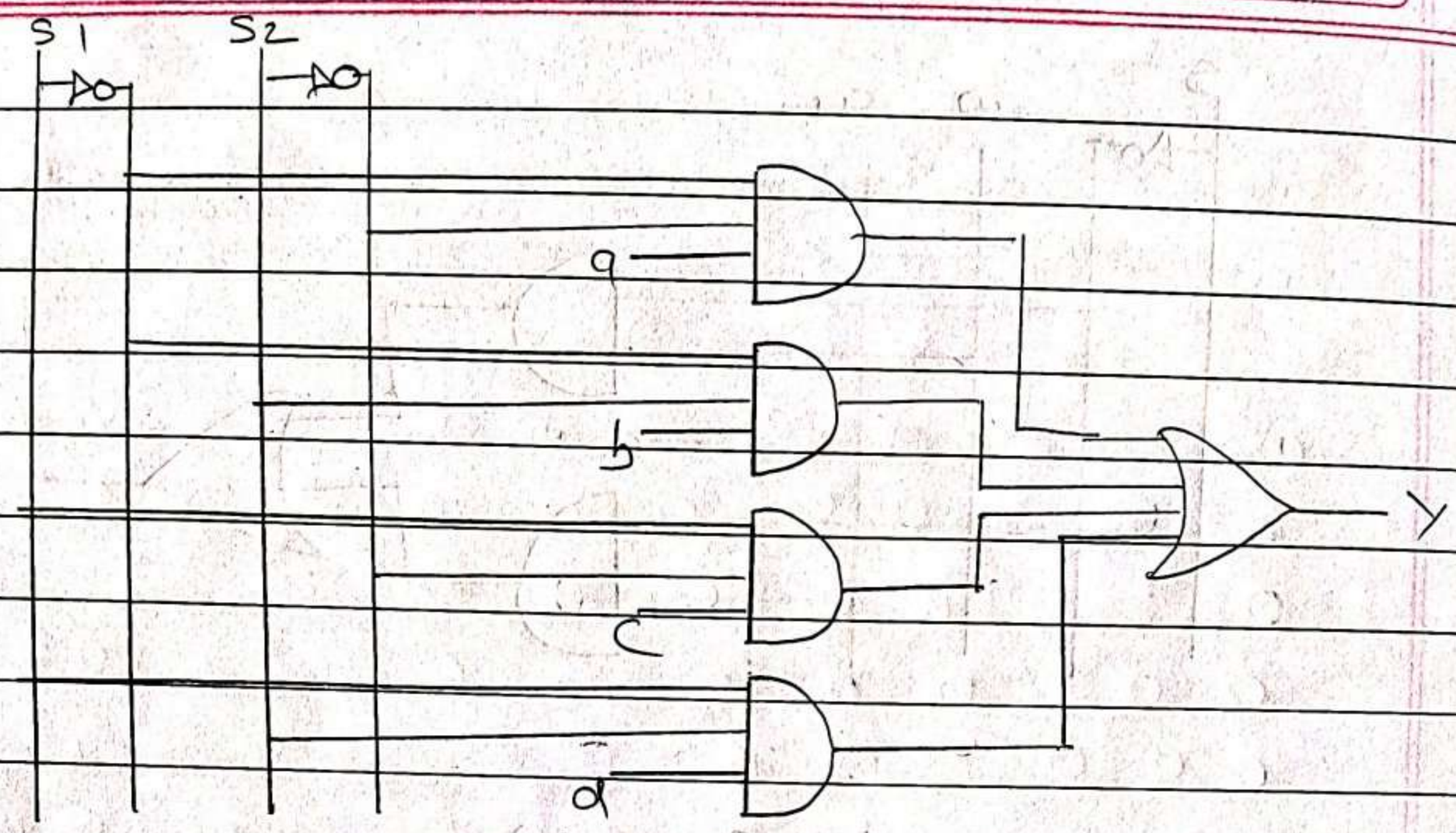

```
module   mux-21 (Y, S, a0, a1)
output   Y;
reg      Y;
input    S, a0, a1;
always@ (a0, a1, S)*
Y = ((~S) & a0) | (S & a1);
endmodule
```

★ 4:1 mux :-

S_1	S_2	Y
0	0	a
0	1	b
1	0	c
1	1	d



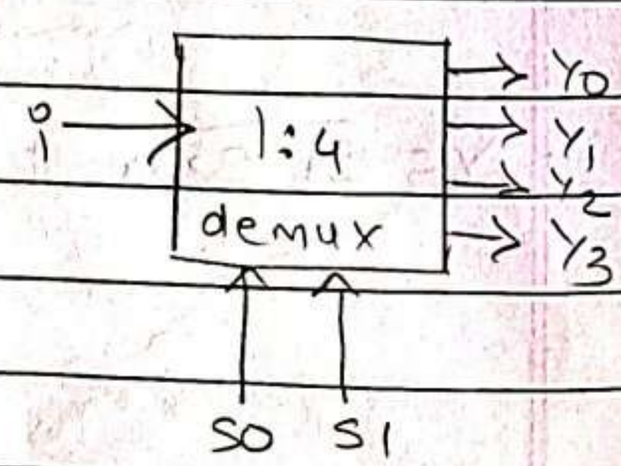
$$Y = \overline{S_1}\,\overline{S_2}\,a + \overline{S_1}\,S_2\,b + S_1\overline{S_2}\,c + S_1 S_2\,d$$

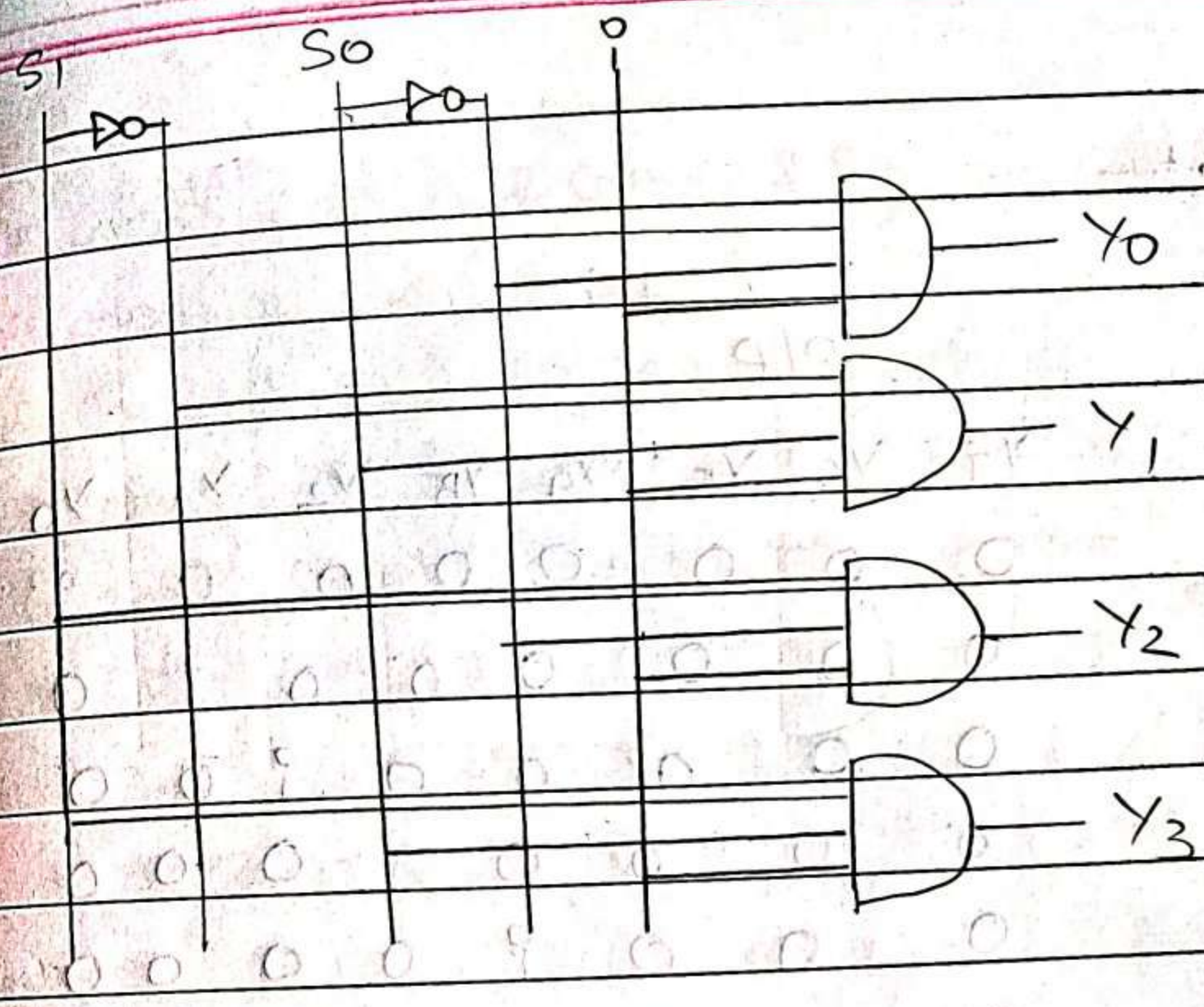



```
module MUX-41 (Y, S1, S2, a, b, c, d);
output Y;
reg Y;
input S1, S2, a, b, c, d;
always @ (S1, S2, a, b, c, d)
Y = ((~S1) & (~S2) & a) | ((~S1) & S2 & b) | (S1 & (~S2) & c) |
        (S1 & S2 & d);
endmodule
```

* 1:4 demux :-

I/P		O/P			
S0	S1	Y3	Y2	Y1	Y0
0	0	0	0	0	i
0	1	0	0	i	0
1	0	0	i	0	0
1	1	i	0	0	0





$Y_3 = S_1 S_0 i$

$Y_2 = S_1 \overline{S_0} i$

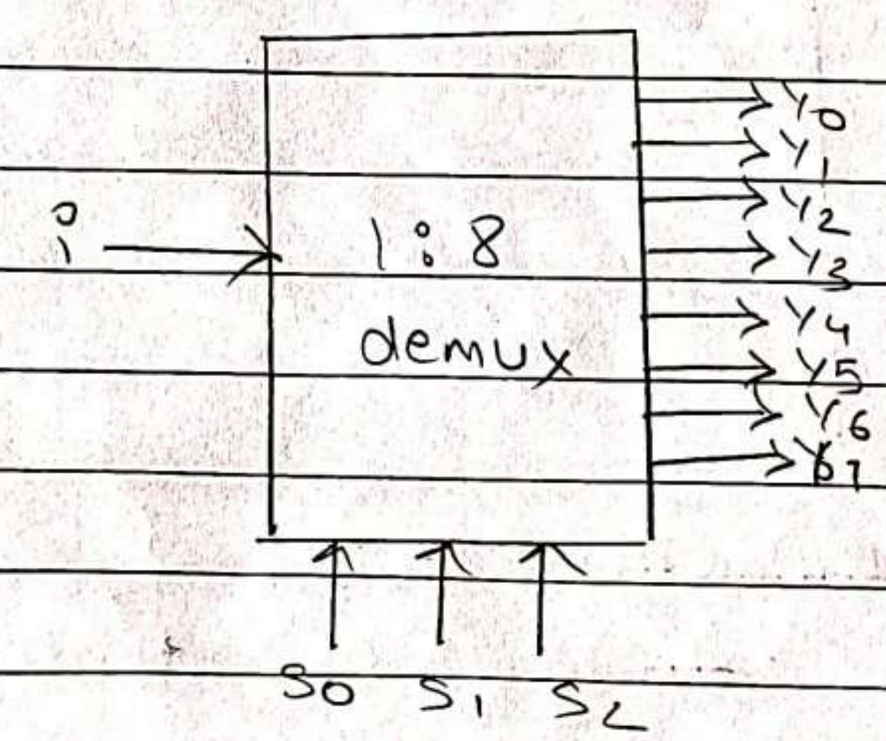
$Y_1 = \overline{S_1} S_0 i$

$Y_0 = \overline{S_1}\,\overline{S_0} i$

```
module demux-14 (Y3, Y2, Y1, Y0, i, S0, S1);
output Y3, Y2, Y1, Y0;
reg Y3, Y2, Y1, Y0;
input S0, S1, i;
always@ (S0, S1)
always@(S0, S1, i)
begin
Y0 = (~S0) & (~S1) & i;
Y1 = (~S0) & S1 & i;
Y2 = S0 & (~S1) & i;
Y3 = S0 & S1 & i;
end
endmodule
```

* 1:8 demux :-

I/P			O/P							
S_0	S_1	S_2	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	i
0	0	1	0	0	0	0	0	0	i	0
0	1	0	0	0	0	0	0	i	0	0
0	1	1	0	0	0	0	i	0	0	0
1	0	0	0	0	0	i	0	0	0	0
1	0	1	0	0	i	0	0	0	0	0
1	1	0	0	i	0	0	0	0	0	0
1	1	1	i	0	0	0	0	0	0	0



$Y_0 = \overline{S_0}\,\overline{S_1}\,\overline{S_2}\,i$

$Y_1 = \overline{S_0}\,\overline{S_1}\,S_2\,i$

$Y_2 = \overline{S_0}\,S_1\,\overline{S_2}\,i$

$Y_3 = \overline{S_0}\,S_1\,S_2\,i$

$Y_4 = S_0\,\overline{S_1}\,\overline{S_2}\,i$

$Y_5 = S_0\,\overline{S_1}\,S_2\,i$

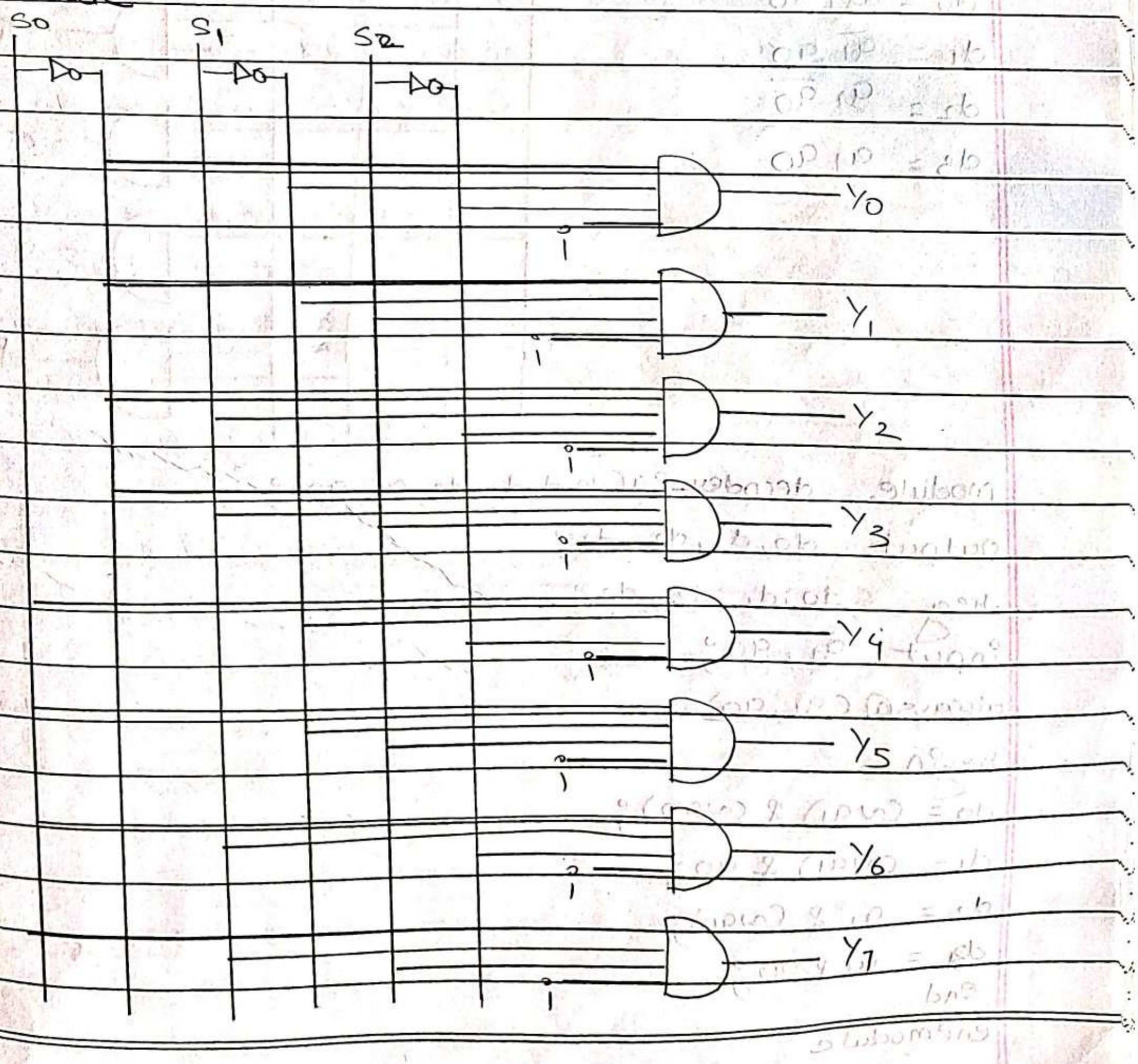
$Y_6 = S_0\,S_1\,\overline{S_2}\,i$

$Y_7 = S_0\,S_1\,S_2\,i$

```
module demux-18(Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7,S0,S1,S2,i);
output Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7;
reg Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7;
input S0,S1,S2,i;
always@(S0,S1,S2,i)
begin
```

PAGE NO.:
DATE: / /

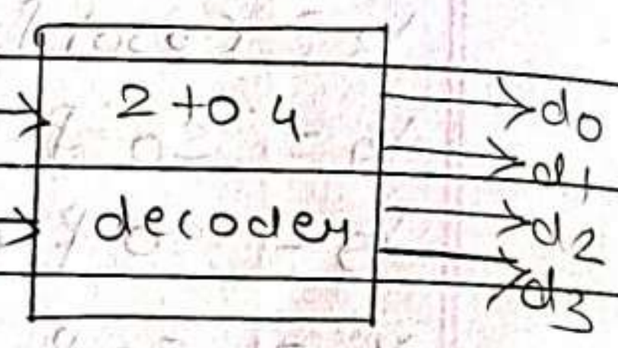
```
Y0 = (~S0) & (~S1) & (~S2) & i;
Y1 = (~S0) & (~S1) & S2 & i;
Y2 = (~S0) & S1 & (~S2) & i;
Y3 = (~S0) & S1 & S2 & i;
Y4 = S0 & (~S1) & (~S2) & i;
Y5 = S0 & (~S1) & S2 & i;
Y6 = S0 & S1 & (~S2) & i;
Y7 = S0 & S1 & S2 & i;
end
endmodule
```



PAGE NO.:
DATE: / /

* 2 to 4 decoder :-

I/P		O/P			
a_1	a_0	d_3	d_2	d_1	d_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

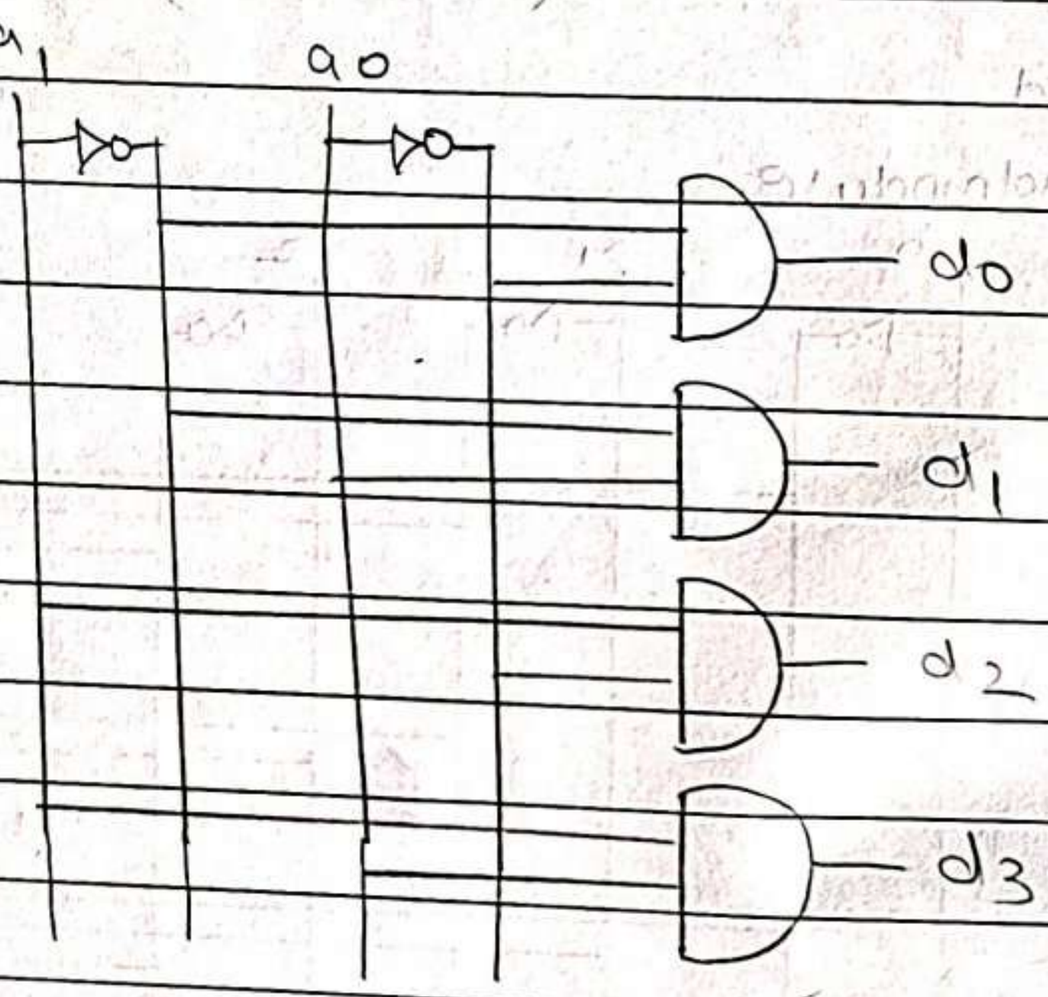


$d_0 = \overline{a_1}\,\overline{a_0}$

$d_1 = \overline{a_1}\,a_0$

$d_2 = a_1\,\overline{a_0}$

$d_3 = a_1\,a_0$




```
module decoder_24(d0,d1,d2,d3,a1,a0);
output d0,d1,d2,d3;
reg d0,d1,d2,d3;
input a1,a0;
always@(a1,a0)
begin
d0 = (~a1) & (~a0);
d1 = (~a1) & a0;
d2 = a1 & (~a0);
d3 = a1 & a0;
end
endmodule
```

PAGE NO.:
DATE: / /

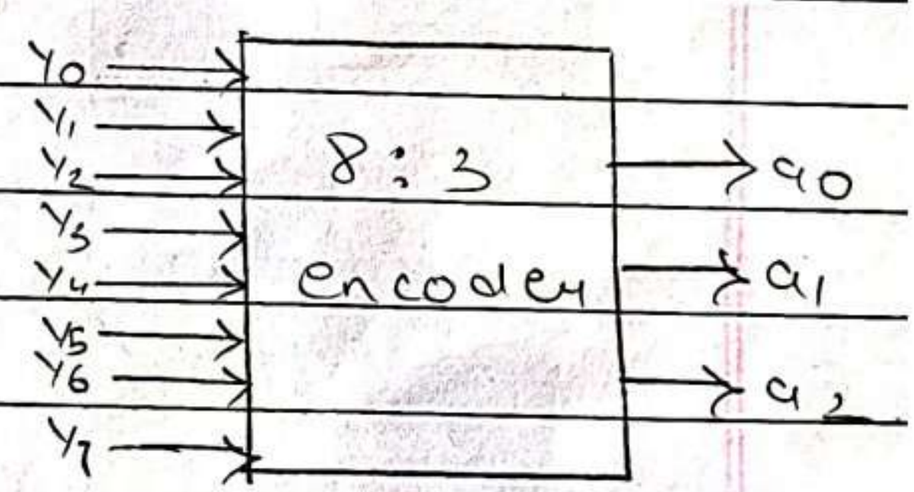
* 8:3 encoder :-

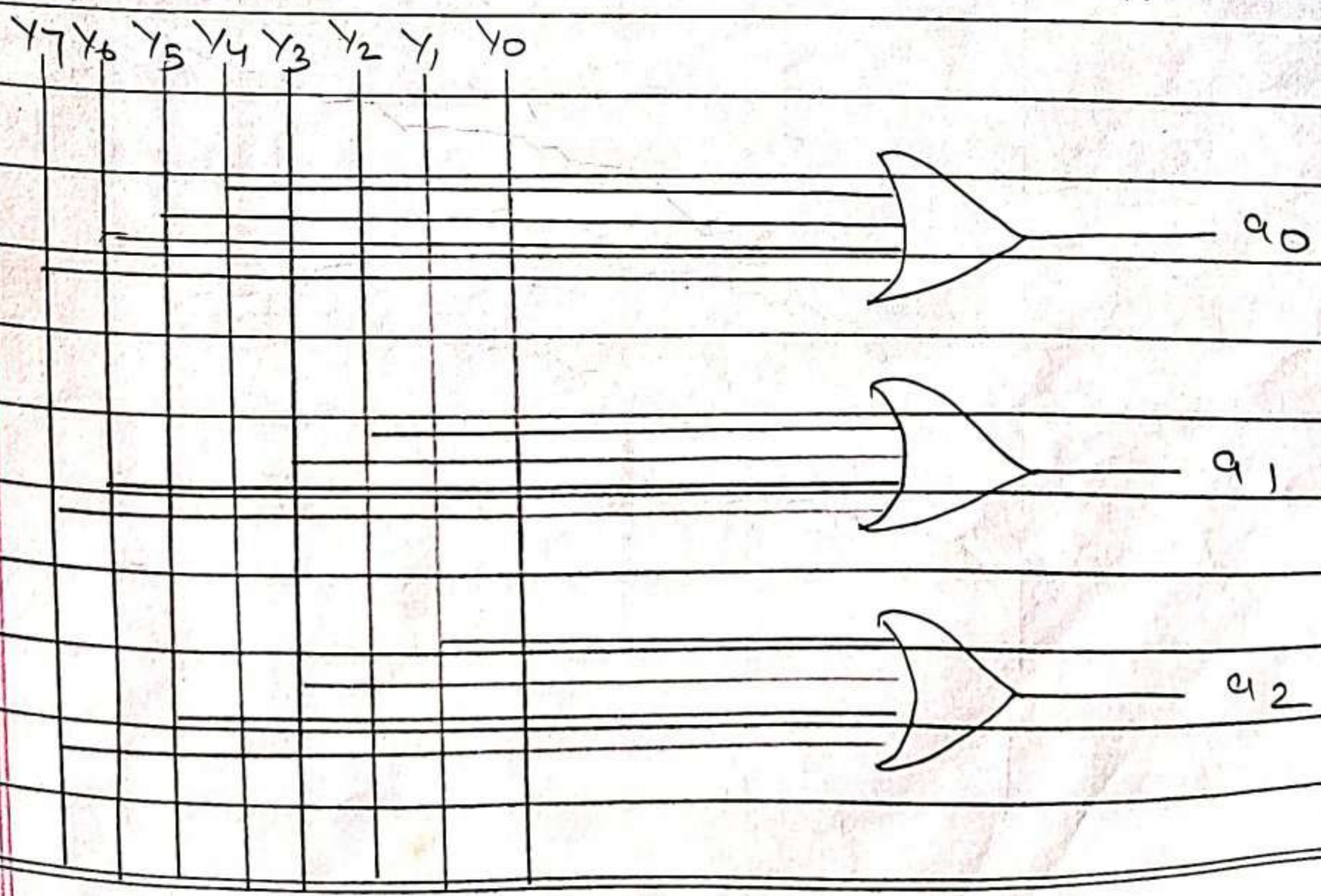
I/P								O/P		
Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	a_0	a_1	a_2
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

$a_2 = Y_1 + Y_3 + Y_5 + Y_7$

$a_1 = Y_2 + Y_3 + Y_6 + Y_7$

$a_0 = Y_4 + Y_5 + Y_6 + Y_7$





PAGE NO.:
DATE: / /

```
Module encoder-83 (q0, q1, q2, Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7);
output q0, q1, q2;
reg q0, q1, q2;
input Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7;
always@ (Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7)
begin
q0 = Y4 | Y5 | Y6 | Y7;
q1 = Y2 | Y3 | Y6 | Y7;
q2 = Y1 | Y3 | Y7 | Y7;
end
endmodule
```

Scanned with OKEN Scanner

PAGE NO.:
DATE: / /

* If, Else Statement :-

1. Type 1

```
~~if (p->max~~
if (p > max_count)
    p=0;
```

2. Type 2

```
if (sel)
    out = pno;
else
    out = pnt;
```

3. Type 3

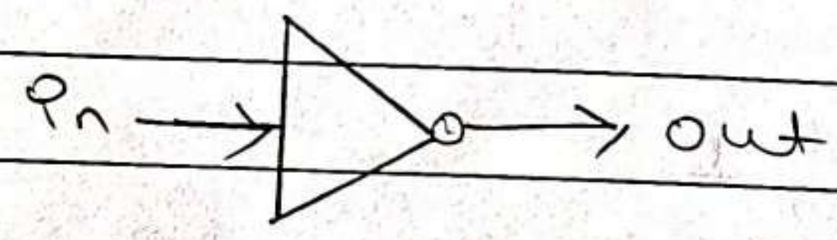
```
if (sel == 0)
    out = pno;
else if (sel == 1)
    out = pn1;
else if (sel == 2)
    out = pn2;
else
    out = 0;
```

Scanned with OKEN Scanner

PAGE NO.:
DATE: / /

* NOT gate :-

I/P	O/P
a	~~b~~
0	1
1	0



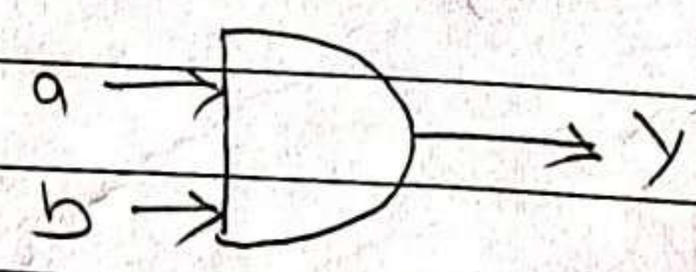
Out = in

Y = a

```
module not_gate (Y,a);
output Y;
reg Y;
input a;
always@(a)
begin
  if (a= = 1'b0)
      Y= 1'b1;
  else
      Y = 1'b0;
end
endmodule
```

* AND gate :-

a	b	Y
0	0	0
0	1	0
1	0	0
1	1	1



Y = ab

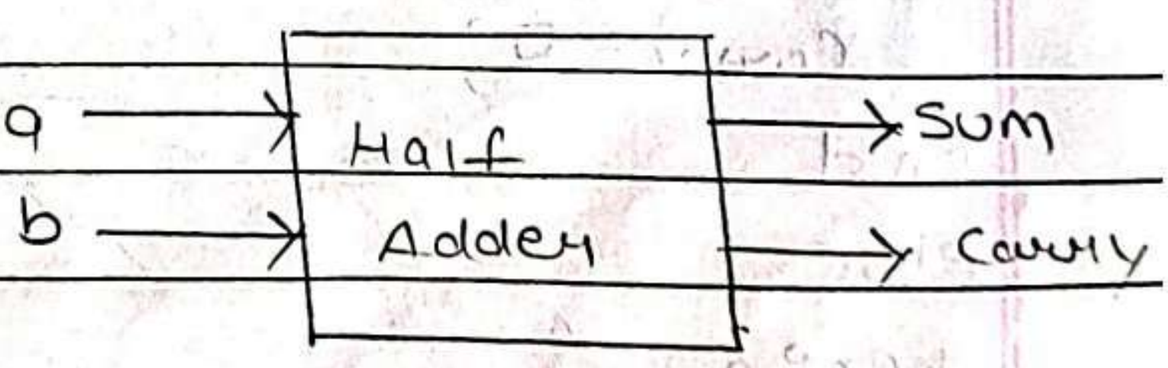
PAGE NO.:
DATE: / /

```
module and-gate (Y, a, b);
output Y;
reg Y;
input a, b;
always @ (a, b)
begin
if ((a==0) & (b==0))
    Y=0;
else if ((a==0) & (b==1))
    Y=0;
else if ((a==1) & (b==0))
    Y=0;
else
    Y=1;
end
endmodule
```

Imp

★ Half Adder :- 8:1

a	b	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Sum = $a \oplus b$

Carry = ab

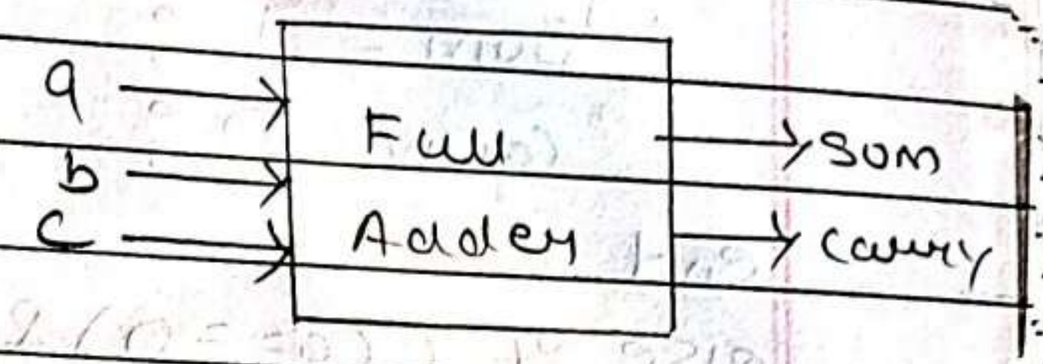
DATE : /

```
module half-adder(sum,carry,a,b);
output sum,carry;
reg sum,carry;
input a,b;
always@(a,b)
begin
if ((a==0) & (b==0))
begin
  sum= 0;
  carry= 0;
end
else if ((a==0) & (b==1))
begin
   sum = 1;
   carry = 0;
end
else if((a==1) & (b==0))
begin
   sum = 1;
   carry = 0;
end
else
begin
   sum= 0;
   carry= 1;
end
end
endmodule
```

PAGE NO.:
DATE: / /

* Full Adder :-

a	b	c	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



$sum = a \oplus b \oplus c$

$Carry = ab + bc + ac$

```
module full-adder (sum, carry, a, b, c);
output sum, carry;
reg sum, carry;
input a, b, c;
always@ (a, b, c)
begin
if ((a==0) & (b==0) & (c==0))
begin
  sum = 0;
  carry = 0;
end
else if ((a==0) & (b==0) & (c==1))
begin
  sum = 1;
  carry = 0;
end
```

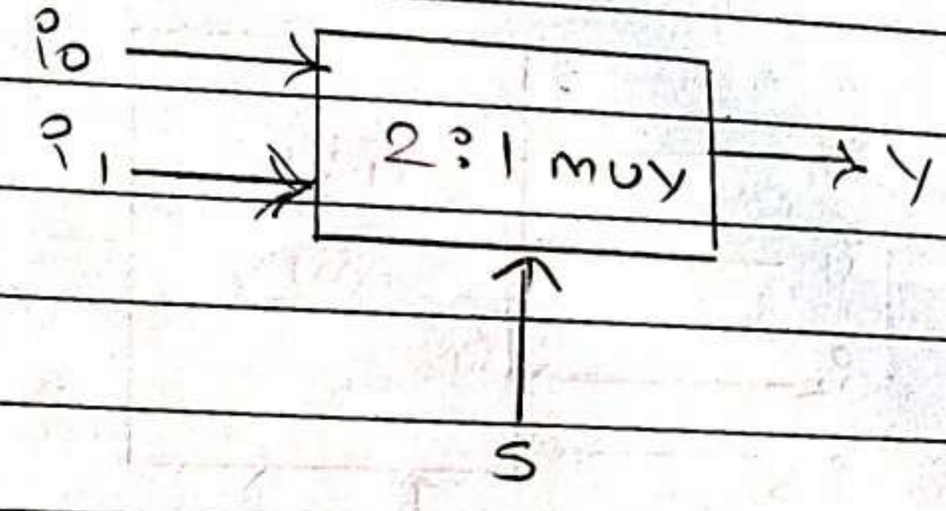
PAGE NO.:
DATE: / /

```
else if ((a==0) & (b==1) & (c==0))
begin
    Sum = 1;
    Carry = 0;
end
else if ((a==0) & (b==1) & (c==1))
begin
    Sum = 0;
    Carry = 1;
end
else if ((a==1) & (b==0) & (c==0))
begin
    Sum = 1;
    Carry = 0;
end
else if ((a==1) & (b==0) & (c==1))
begin
    Sum = 0;
    Carry = 1;
end
else if ((a==1) & (b==1) & (c==0))
begin
    Sum = 0;
    Carry = 1;
end
else
begin
    Sum = 1;
    Carry = 1;
end
end
```

PAGE NO.:
DATE: / /

★ 2:1 MUX :-

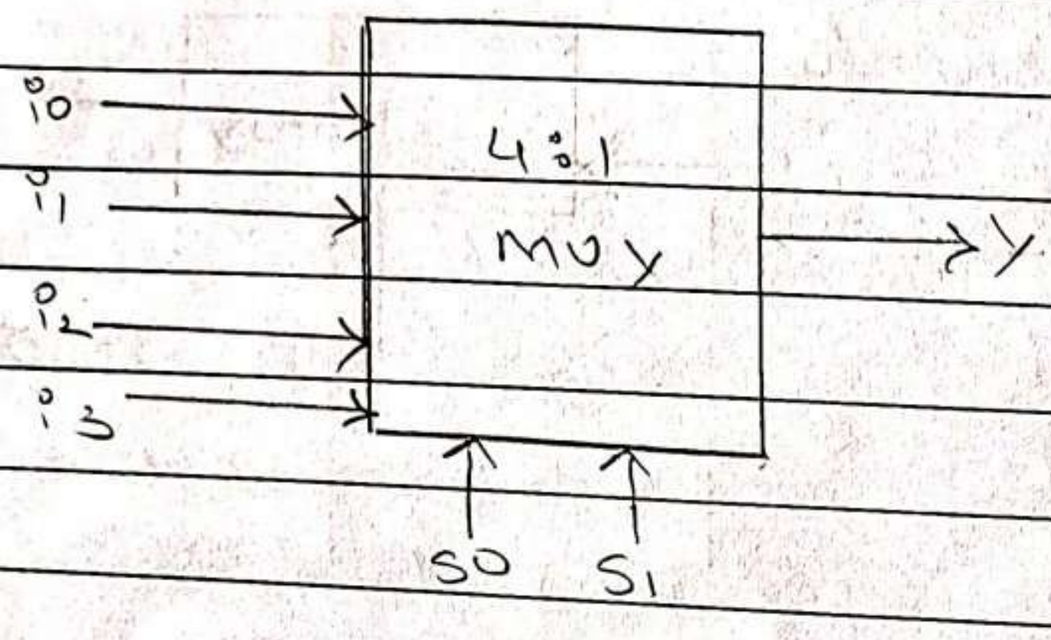
I/P	O/P
S	Y
0	i_0
1	i_1



$Y = \bar{S} i_0 + S i_1$

```
module mux-21(Y, S, i0, i1);
output Y;
reg Y;
input S, i0, i1;
always@(S, i0, i1)
begin
if (S==0)
  Y = i0;
else
  Y = i1;
end
endmodule.
```

* 4:1 mux

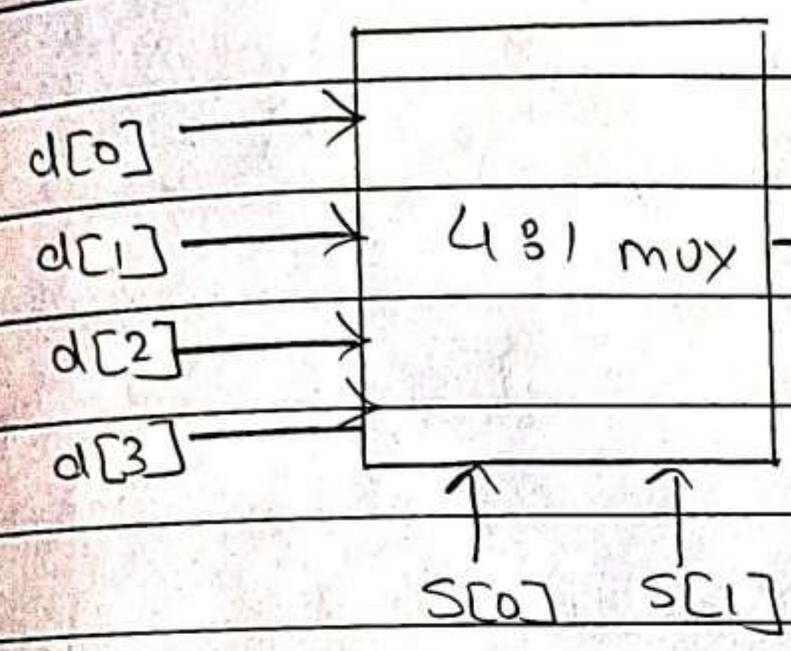


S0	S1	Y
0	0	i0
0	1	i1
1	0	i2
1	1	i3

```
module mux_41 (Y, i0, i1, i2, i3, S0, S1);
output Y;
reg Y;
input S0, S1, i0, i1, i2, i3;
always @(S0, S1, i0, i1, i2, i3)
begin
 if (S0 = = 0 & S1 = = 0)
   Y = i0;
 else if (S0 = = 0 & S1 = = 1)
   Y = i1;
 else if (S0 = = 1 & S1 = = 0)
   Y = i2;
 else if (S0 = = 1 & S1 = = 1)
   Y = i3;
end
endmodule
```

PAGE NO.:
DATE: / /

* 4:1 MUX using vector

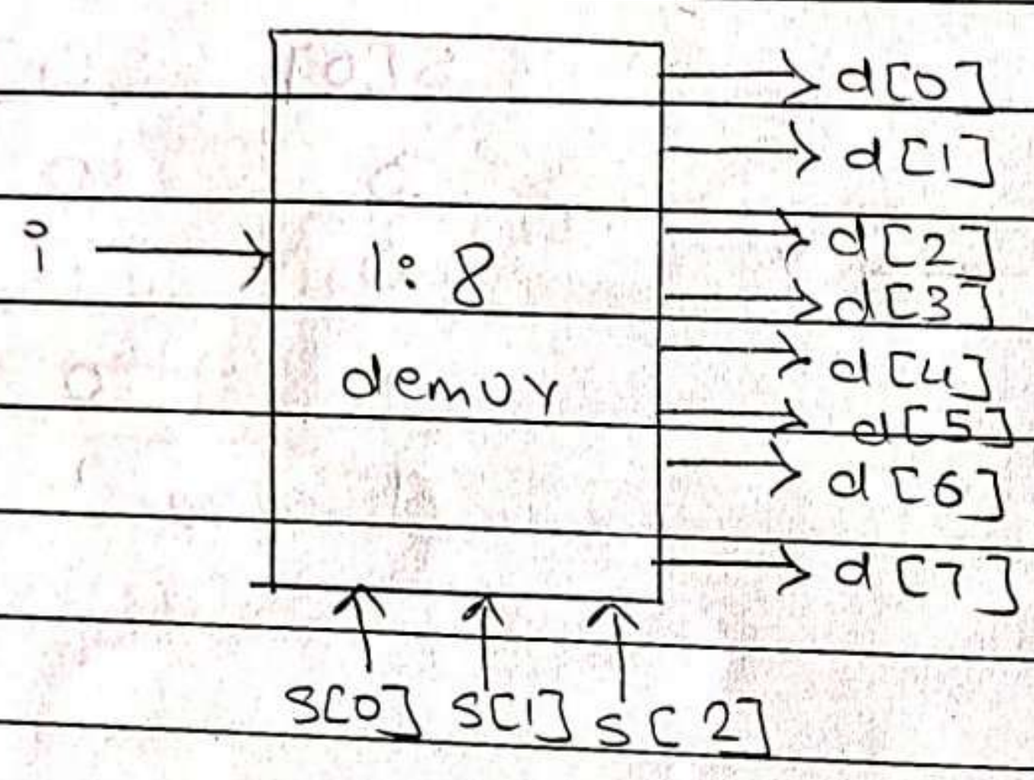


S[0]	S[1]	Y
0	0	d[0]
0	1	d[1]
1	0	d[2]
1	1	d[3]

```
module mux-41(Y, d,s);
output Y;
reg Y;
input [3:0] d;
input [1:0]s;
always@ (d,s)
begin
if (s[0] = = 0 & s[1] = = 0)
  Y = d[0];
else if (s[0] = = 0 & s[1] = = 1)
  Y = d[1];
else if (s[0] = = 1 & s[1] = = 0)
  Y = d[2];
else (s[0] = = 1 & s[1] = = 1)
  Y = d[3]
end
endmodule
```

PAGE NO.:
DATE: / /

* 1:8 demux using if else statement :-



S[0]	S[1]	S[2]	d[0]	d[1]	d[2]	d[3]	d[4]	d[5]	d[6]
0	0	0	i	0	0	0	0	0	0
0	0	1	0	i	0	0	0	0	0
0	1	0	0	0	i	0	0	0	0
0	1	1	0	0	0	i	0	0	0
1	0	0	0	0	0	0	i	0	0
1	0	1	0	0	0	0	0	i	0
1	1	0	0	0	0	0	0	0	i
1	1	1	0	0	0	0	0	0	0

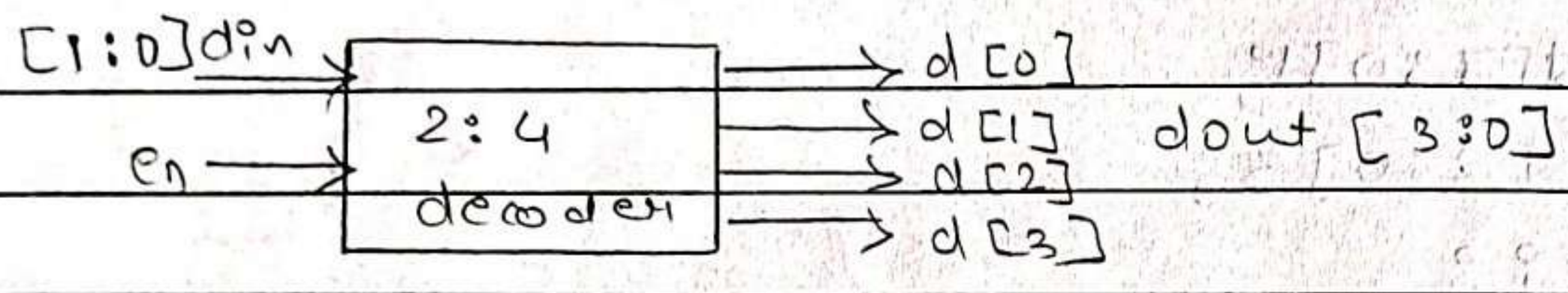
d[0] = S[0] S[1] S[2] i
d[1] = S[0] S[1] S[2] i
d[2] = S[0] S[1] S[2] i
d[3] = S[0] S[1] S[2] i
d[4] = S[0] S[1] S[2] i
d[5] = S[0] S[1] S[2] i
d[6] = S[0] S[1] S[2] i
d[7] = S[0] S[1] S[2] i

Scanned with OKEN Scanner

PAGE NO.:
DATE: / /

```
module demux-18 (i, d, s);
output [7:0]d;
reg d[7:0]d;
input [2:0]s;
input i;
always@ (i, s)
begin
if (s[0] = = 0 & s[1] = = 0 & s[2] = = 0)
    d[0] = i;
else if (s[0] = = 0 & s[1] = = 0 & s[2] = = 1)
    d[1] = i;
else if (s[0] = = 0 & s[1] = = 1 & s[2] = = 0)
    d[2] = i;
else if (s[0] = = 0 & s[1] = = 1 & s[2] = = 1)
    d[3] = i;
else if (s[0] = = 1 & s[1] = = 0 & s[2] = = 0)
    d[4] = i;
else if (s[0] = = 1 & s[1] = = 0 & s[2] = = 1)
    d[5] = i;
else if (s[0] = = 1 & s[1] = = 1 & s[2] = = 0)
    d[6] = i;
else [s[0] = = 1 & s[1] = = 1 & s[2] = = 1)
    d[7] = i;
end
endmodule
```

* 2:4 decoder using if else statement :-



I/P			O/P			
en	din[1]	din[0]	dout[3]	dout[2]	dout[1]	dout[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

```
module dec_24(dout, en, din);
output [3:0] dout;
reg [3:0] dout;
input en;
input [1:0] din;
always@ (en, din)
begin
if (en = = 1)
begin
if (din = = 2'b00)
dout = 4'b0001;
else if (din = = 2'b01)
dout = 4'b0010;
else if (din = = 2'b10)
dout = 4'b0100;
else
dout = 4'b1000;
end
endmodule
```

PAGE NO.:
DATE: / /

* Case Statement :-

```
case (expression)

alternative 1 : statement 1;
alternative 2 : statement 2;
alternative 3 : statement 3;
alternative 4 : statement 4;
    default : default_statement;
endcase
```

The keywords case, endcase, default are used in case statement. Each statement one to default can be a single statement or block of multiple statement.

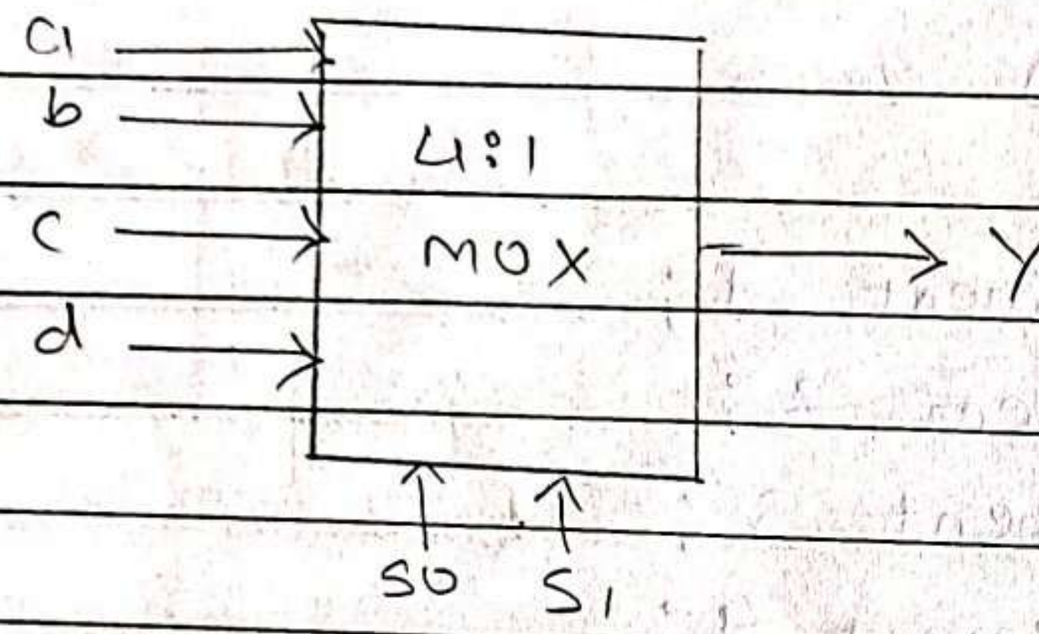
A block of multiple statement must be group of keywords begin and end. If none of alternative matches then default statement is executed.

Scanned with OKEN Scanner

Hardware is compulsory

PAGE NO.:
DATE: / /

★ 4:1 mux using ~~gate~~ case :-

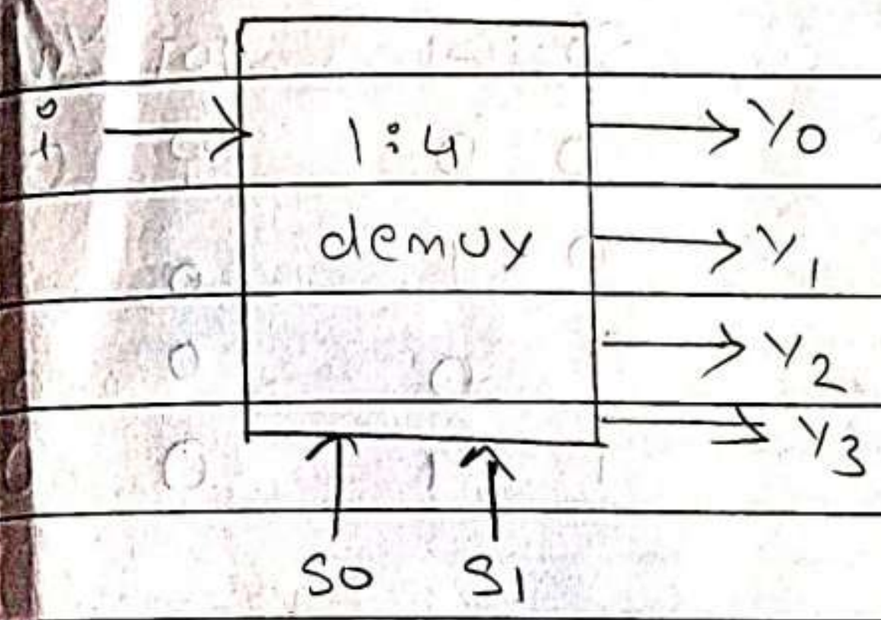


S0	S1	Y
0	0	a
0	1	b
1	0	c
1	1	d

```
module mux_41 (Y, s0, s1, a, b, c, d);
output Y;
reg Y;
input s0, s1, a, b, c, d;
always @(s0, s1, a, b, c, d)
begin
case ({s0, s1})
2'b00 : Y=a;
2'b01 : Y=b;
2'b10 : Y=c;
2'b11 : Y=d;
endcase
end
endmodule
```

PAGE NO.:
DATE: / /

* 1:4 demux using case :-

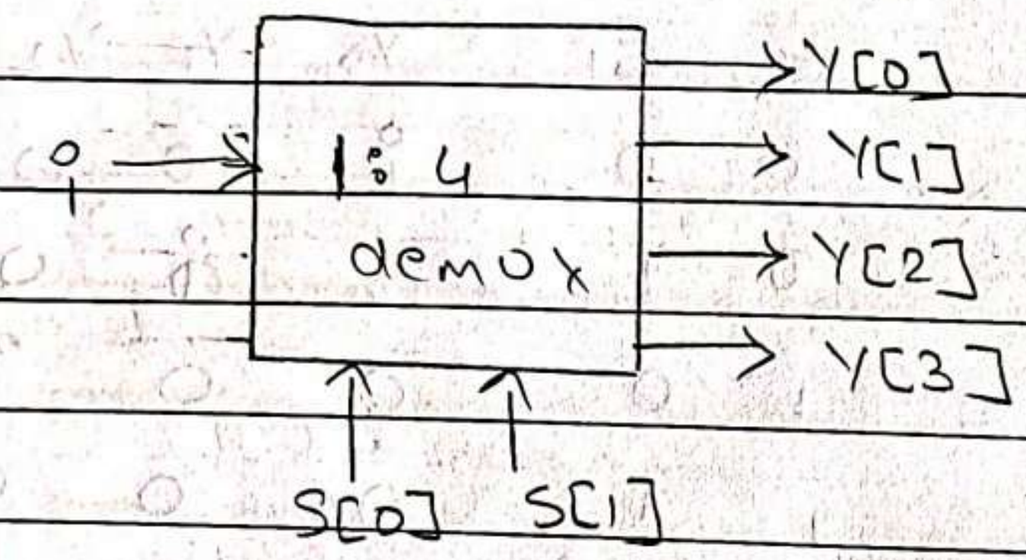


S0	S1	Y0	Y1	Y2	Y3
0	0	i	0	0	0
0	1	0	i	0	0
1	0	0	0	i	0
1	1	0	0	0	i

```
module demux-14 (Y0, Y1, Y2, Y3, S0, S1, i);
output Y0, Y1, Y2, Y3;
reg Y0, Y1, Y2, Y3;
input S0, S1, i;
always@ (S0, S1, i)
begin
case ({S0, S1})
2'b00 : begin
Y0 = i; Y1 = 0; Y2 = 0; Y3 = 0;
end
2'b01 : begin
Y0 = 0; Y1 = i; Y2 = 0; Y3 = 0;
end
2'b10 : begin
Y0 = 0; Y1 = 0; Y2 = i; Y3 = 0;
end
2'b11 : begin
Y0 = 0; Y1 = 0; Y2 = 0; Y3 = i;
end
endcase
end
endmodule
```

PAGE NO.:
DATE: / /

* 1:4 demux using case and vector:-

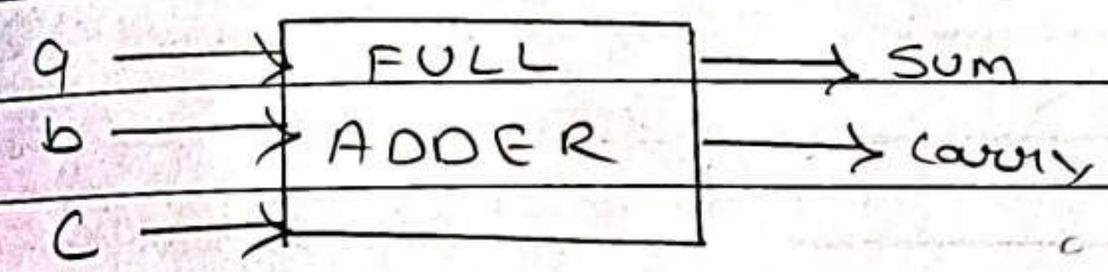


S0	S1	Y[0]	Y[1]	Y[2]	Y[3]
0	0	i	0	0	0
0	1	0	i	0	0
1	0	0	0	i	0
1	1	0	0	0	i

```
module demux-14(Y,s,i);
output [3:0]Y;
reg [3:0]Y;
input [1:0]s;
input i;
always@(s,i)
begin
case(s)
2'b00 : begin
  Y[0] = i; Y[3:1] = 0;
end
2'b01 : begin
Y[1] = i; Y[3:2] = 0; Y[0] = 0;
end
2'b10 : begin
Y[2] = i ; Y[1:0] = 0; Y[3] = 0;
end
2'b11 : begin
Y[3] = i ; Y[2:0] = 0;
end
endcase
end
endmodule
```

PAGE NO.:
DATE: / /

* Full Adder using Case :-



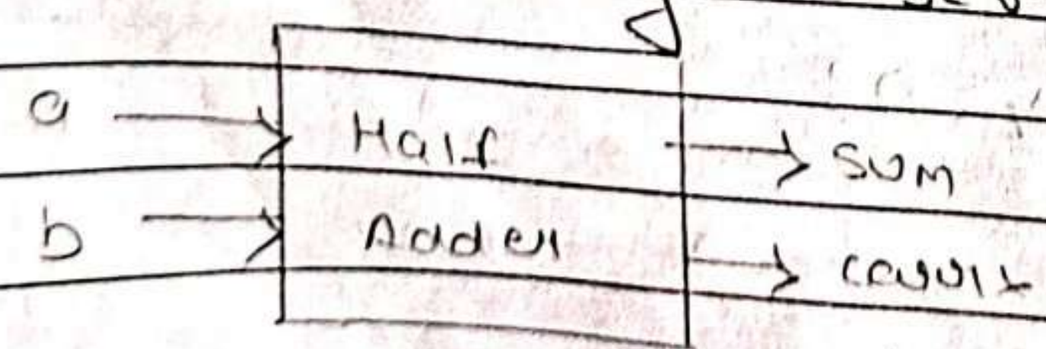
I/p O/p

a	b	c	sum	carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

```
module full-adder (sum, carry, a, b, c);
output sum, carry;
reg sum, carry;
input a, b, c;
always@ (a, b, c)
begin
case ({a, b, c})
3'b000 : begin
    sum = 0 ; carry = 0;
end
3'b001 : begin
    sum = 1; carry = 0;
end
```

```
3'b010 : begin
  sum = 1; carry = 0;
end
3'b011 : begin
  sum = 0; carry = 1;
end
3'b100 : begin
  sum = 1; carry = 0;
end
3'b101 : begin
  sum = 0; carry = 1;
end
3'b110 : begin
  sum = 0; carry = 1;
end
3'b111 : begin
  sum = 1; carry = 1;
end
endcase
end
endmodule
```

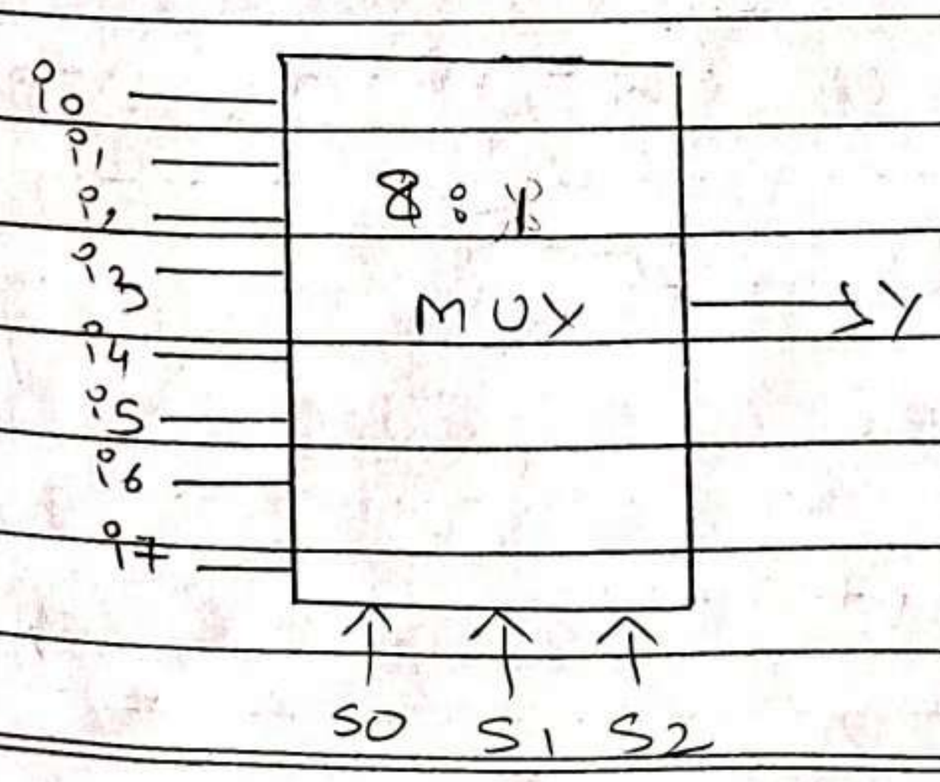
* Half Adder using case :-



a	b	sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

```
module half-adder (sum, carry, a, b);
output sum, carry;
reg sum, carry;
input a, b;
always@(a, b)
begin
case ({a, b})
2'b00 : begin
  sum = 0 ; carry = 0 ;
end
2'b01 : begin
  sum = 1 ; carry = 0 ;
end
2'b10 : begin
  sum = 1 ; carry = 0 ;
end
2'b11 : begin
  sum = 1 ; carry = 1 ;
end
endcase
end
endmodule
```

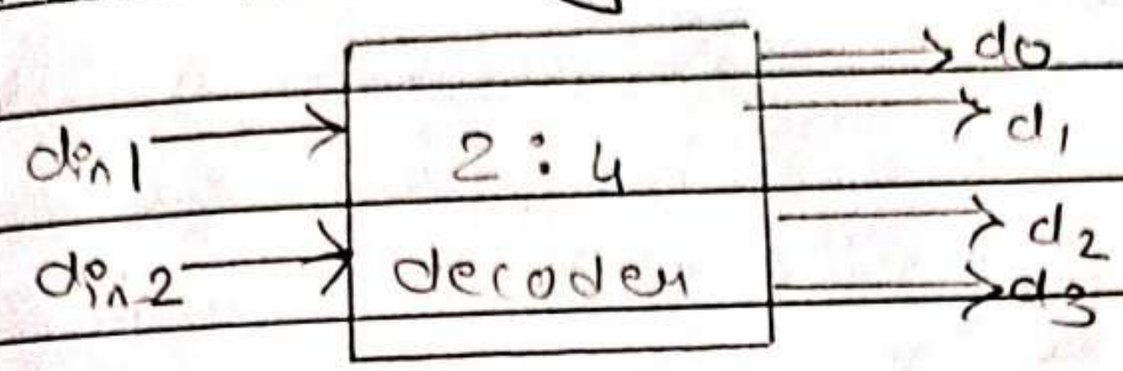
* 8:1 mux using case



S_0	S_1	S_2	Y
0	0	0	i_0
0	0	1	i_1
0	1	0	i_2
0	1	1	i_3
1	0	0	i_4
1	0	1	i_5
1	1	0	i_6
1	1	1	i_7

```
module mux-81 (Y, S0, S1, S2, i0, i1, i2, i3, i4, i5, i6, i7);
output Y;
reg Y;
input S0, S1, S2, i0, i1, i2, i3, i4, i5, i6, i7;
always @ (S0, S1, S2, i0, i1, i2, i3, i4, i5, i6, i7)
begin
case ({S0, S1, S2})
  3'000 : Y = i0;
  3'001 : Y = i1;
  3'010 : Y = i2;
  3'011 : Y = i3;
  3'100 : Y = i4;
  3'101 : Y = i5;
  3'110 : Y = i6;
  3'111 : Y = i7;
endcase
end
endmodule
```

* 2:4 decoder using case

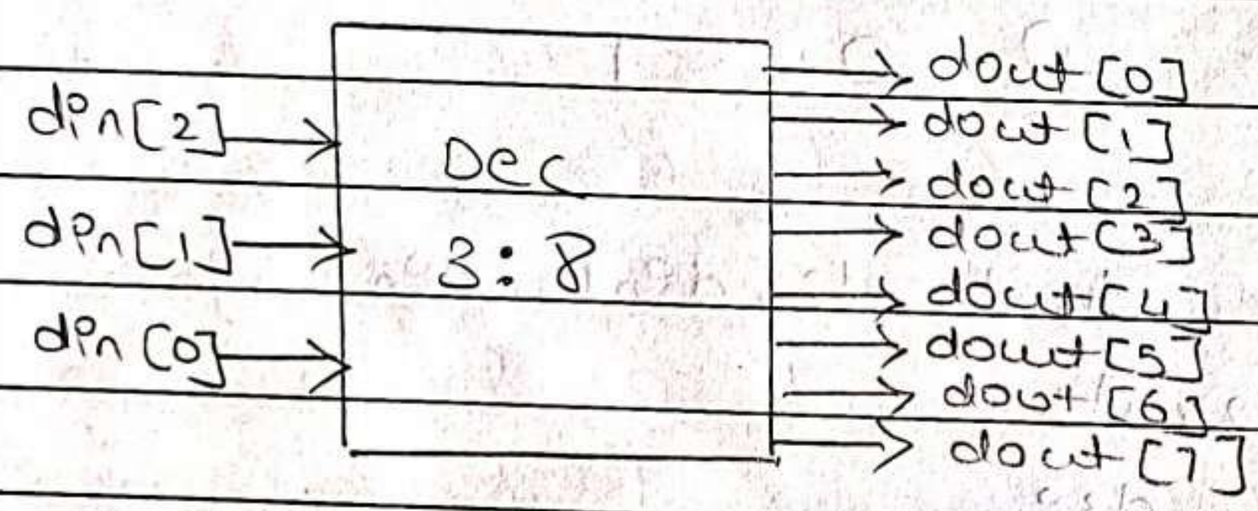


din1	din2	d0	d1	d2	d3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

```
module dec_24(d0, d1, d2, d3, din1, din2);
output d0, d1, d2, d3;
reg d0, d1, d2, d3;
input din1, din2;
always@(din1, din2)
begin
case({din1, din2})
2'b00: begin
  d0=1; d1=0; d2=0; d3=0;
end
2'b01: begin
  d1=1; d0=0; d2=0; d3=0;
end
2'b10: begin
  d2=1; d0=0; d1=0; d3=0;
end
2'b11: begin
```

```
    d3=1; d0=0; d1=0; d2=0;
end
endcase
end
endmodule
```

★ IMP 3:8 decoder using case statement :-



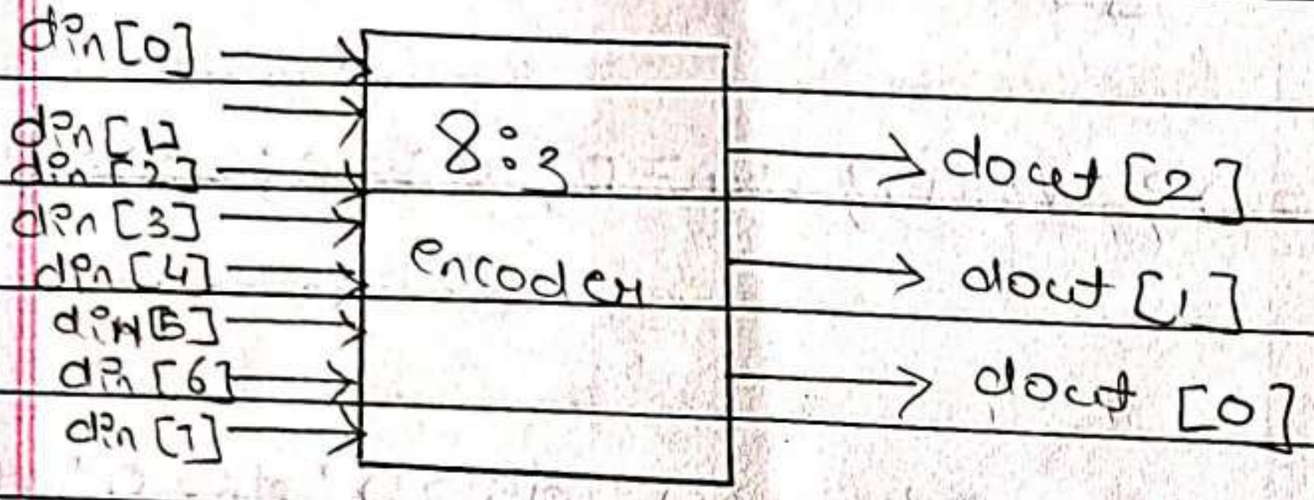
din 0	din 1	din 2	d[0]	d1	d2	d3	d4	d5	d6	d7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

```
module dec-38 (d0, d1, d2, d3, d4, d5, d6, d7, din0, din1, din2);
output d0, d1, d2, d3, d4, d5, d6, d7;
reg d0, d1, d2, d3, d4, d5, d6, d7;
input din1, din2, din0;
always @ (din0, din1, din2)
```

```
begin
case ({din0, din1, din2})
3'b000: begin
  d0=1; d1=0; d2=0; d3=0; d4=0; d5=0; d6=0; d7=0;
end
3'b001 : begin
  d1=1; d0=0; d2=0; d3=0; d4=0; d5=0; d6=0; d7=0;
end
3'b010 : begin
  d2=1; d0=0; d1=0; d3=0; d4=0; d5=0; d6=0; d7=0;
end
3'b011 : begin
  d3=1; d0=0; d1=0; d2=0; d4=0; d5=0; d6=0; d7=0;
end
3'b100 : begin
  d4=1; d0=0; d1=0; d2=0; d3=0; d5=0; d6=0; d7=0;
end
3'b101 : begin
  d5=1; d0=0; d1=0; d2=0; d3=0; d4=0; d6=0; d7=0;
end
3'b110 : begin
  d6=1; d0=0; d1=0; d2=0; d3=0; d4=0; d5=0; d7=0;
end
3'b111 : begin
  d7=1; d0=0; d1=0; d2=0; d3=0; d4=0; d5=0; d6=0;
end
endcase
end
endmodule
```

PAGE NO.:
DATE: / /

* 8:3 encoder using case.



I/P O/P

By using a vector

din7	din6	din5	din4	din3	din2	din1	din0	dout[2]	dout[1]	dout[0]
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

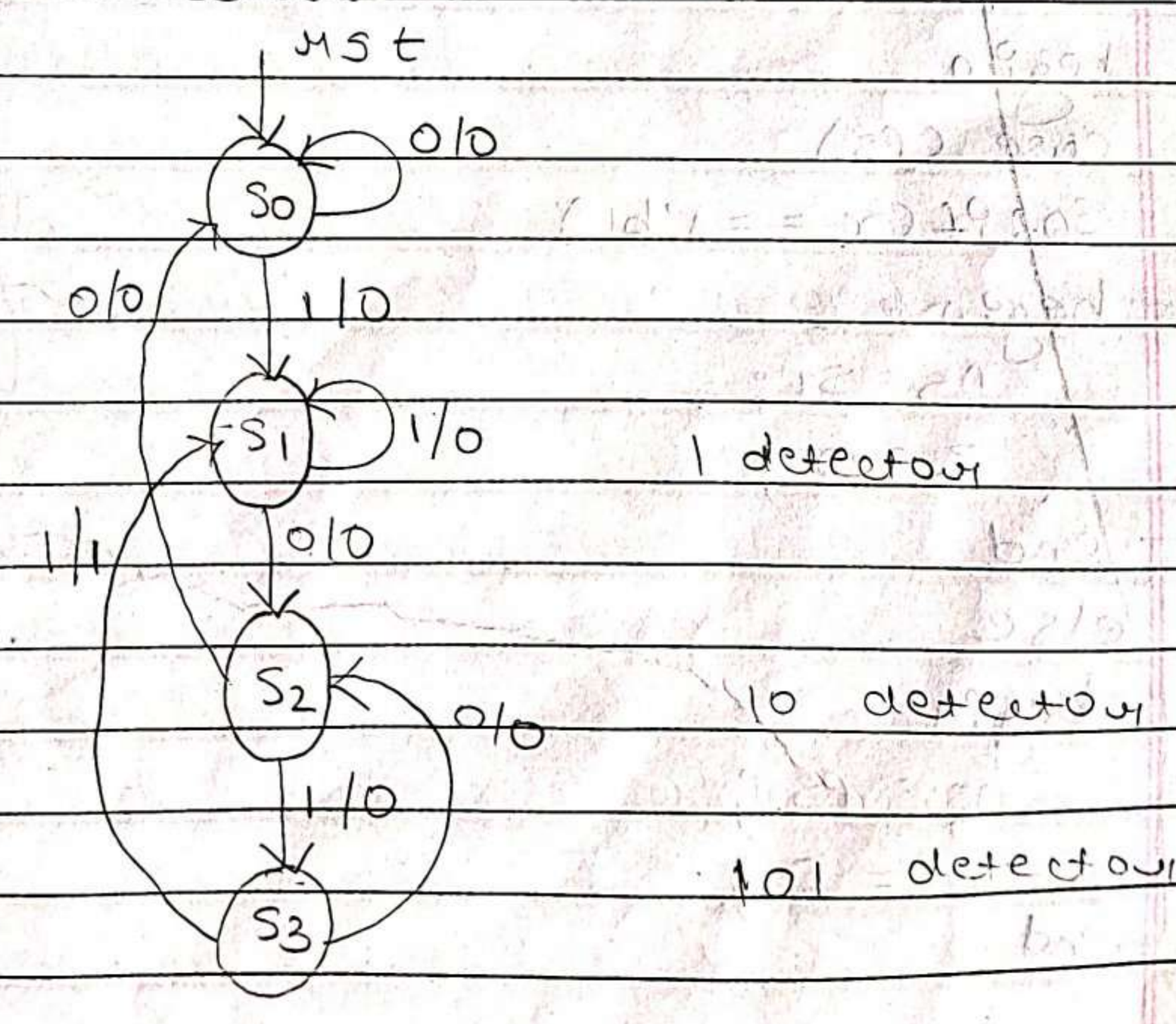
```
~~Module = enc~~
Module enc_83(dout, din);
output [2:0] dout;
reg [2:0] dout;
input [7:0] din;
always @(din)
begin
case(din)
8'b00000001 : dout = 3'b000;
8'b00000010 : dout = 3'b001;
8'b00000100 : dout = 3'b010;
```

```
8'b 00001000 : 3'b 011;
8'b 00010000 : 3'b 100;
8'b 00100000 : 3'b 101;
8'b 01000000 : 3'b 110;
8'b 10000000 : 3'b 111;
endcase
end
endmodule
```

* Finite State Machine [FSM] :-

1. Detect the input sequence 1011 using Mealy machine with overlapping allowed Method write behavioral verilog code for the sequence detector.

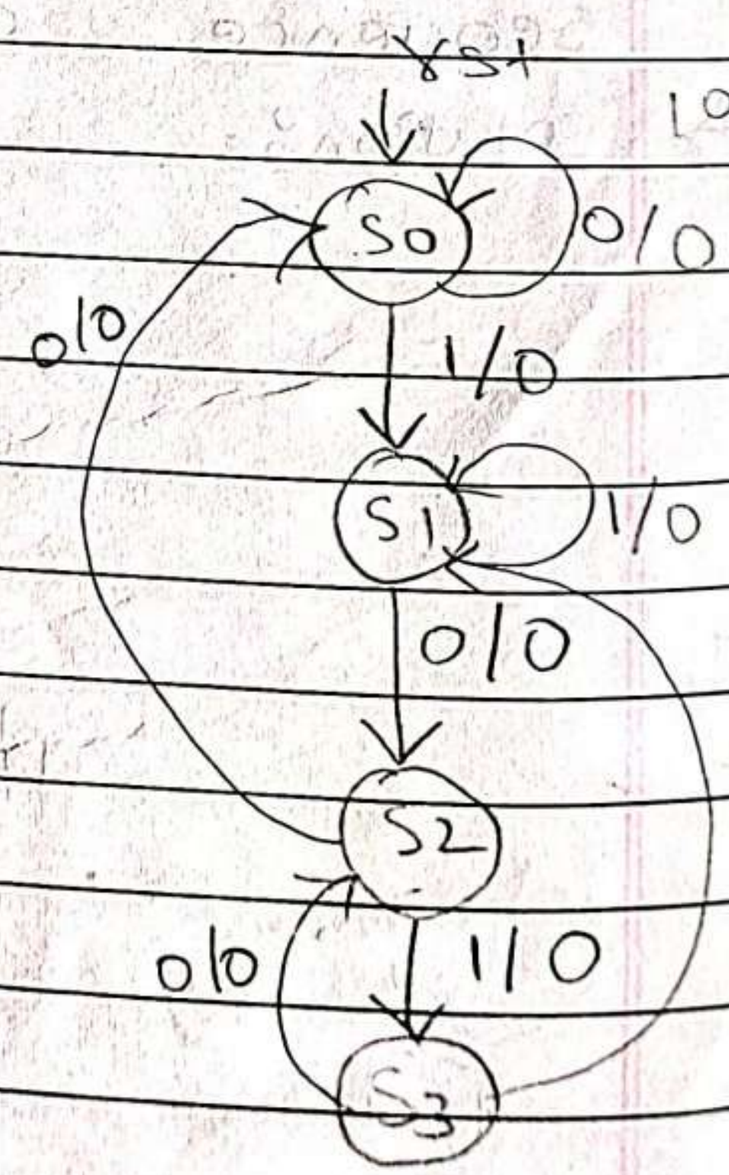
→ Solution :-



State table

present state [P.S]	Next State [NS] x=0	x=1	Z
S0	S0	S1	0
S1	S2	S1	0
S2	S0	S3	0
S3	S2	S1	1

```
Module fsm_1011 (z, rst, clk, x);
output reg z;
input x, rst, clk;
reg [1:0] ps, ns;
parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11
always@ (x, ps)
begin
case (ps)
S0: if (x == 1'b1)
  begin
    ns = S1;
    z = 0;
  end
else
  begin
    ns = ps;
    z = 0;
  end
```

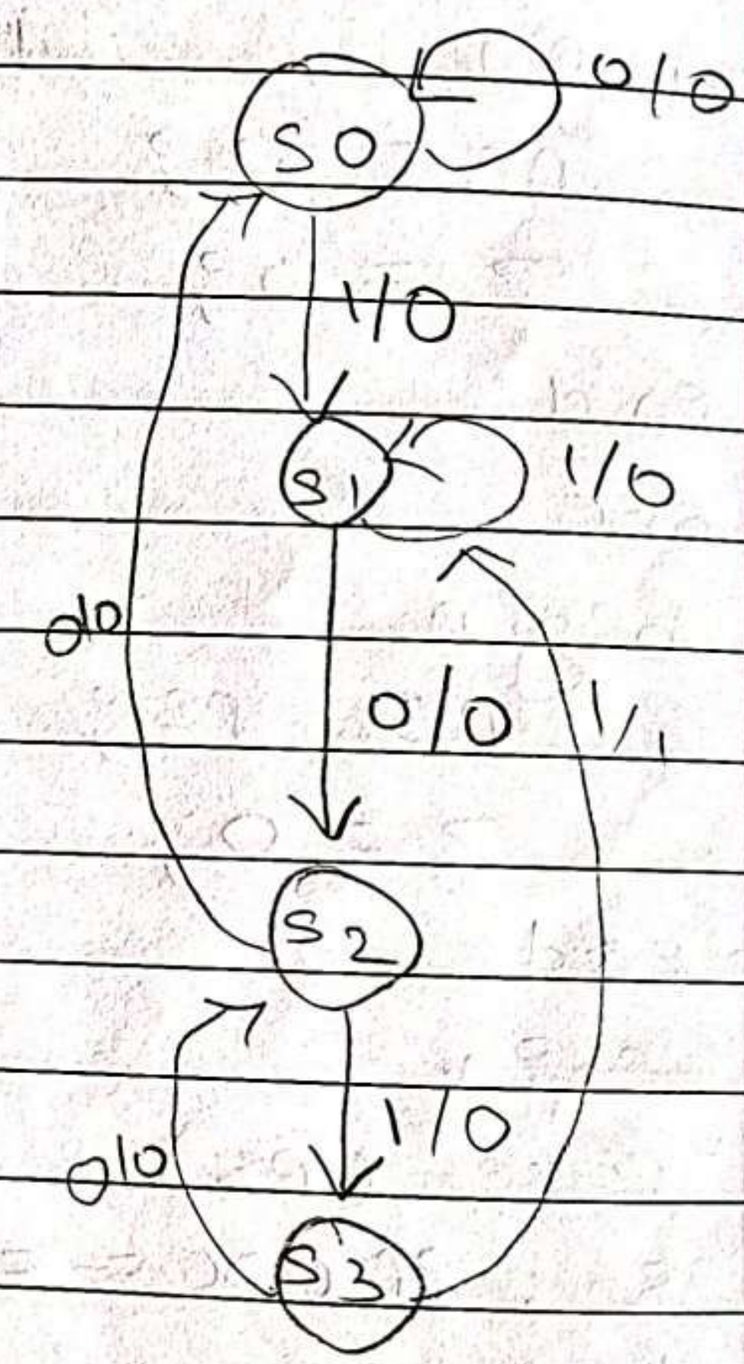


PAGE NO.:
DATE: / /

~~begin~~
~~case (PS)~~
S_1 : if (x == 1'b0)
begin
ns = S2;
z = 0;
end
else
begin
ns = ps;
z = 0;
end
~~begin~~
~~case (PS)~~
S_2 : if ~~(x == 1'b0)~~ (x == 1'b1)
begin
ns = S3;
z = 0
end
else
begin
ns = ~~ps~~ S0;
z = 0;
end
~~begin~~
~~case (PS)~~
S_3 : if (x == 1'b0)
~~begin~~

```
ns = s1;
z = 1;
end
else
begin
    ns = s2;
    z = 0;
end
endcase
end
endmodule
always@ (posedge clk)
begin
if (rst)
ps = s0;
else
  ps = ns;
end
endmodule.
```

rst = 1
(active high reset system reset)

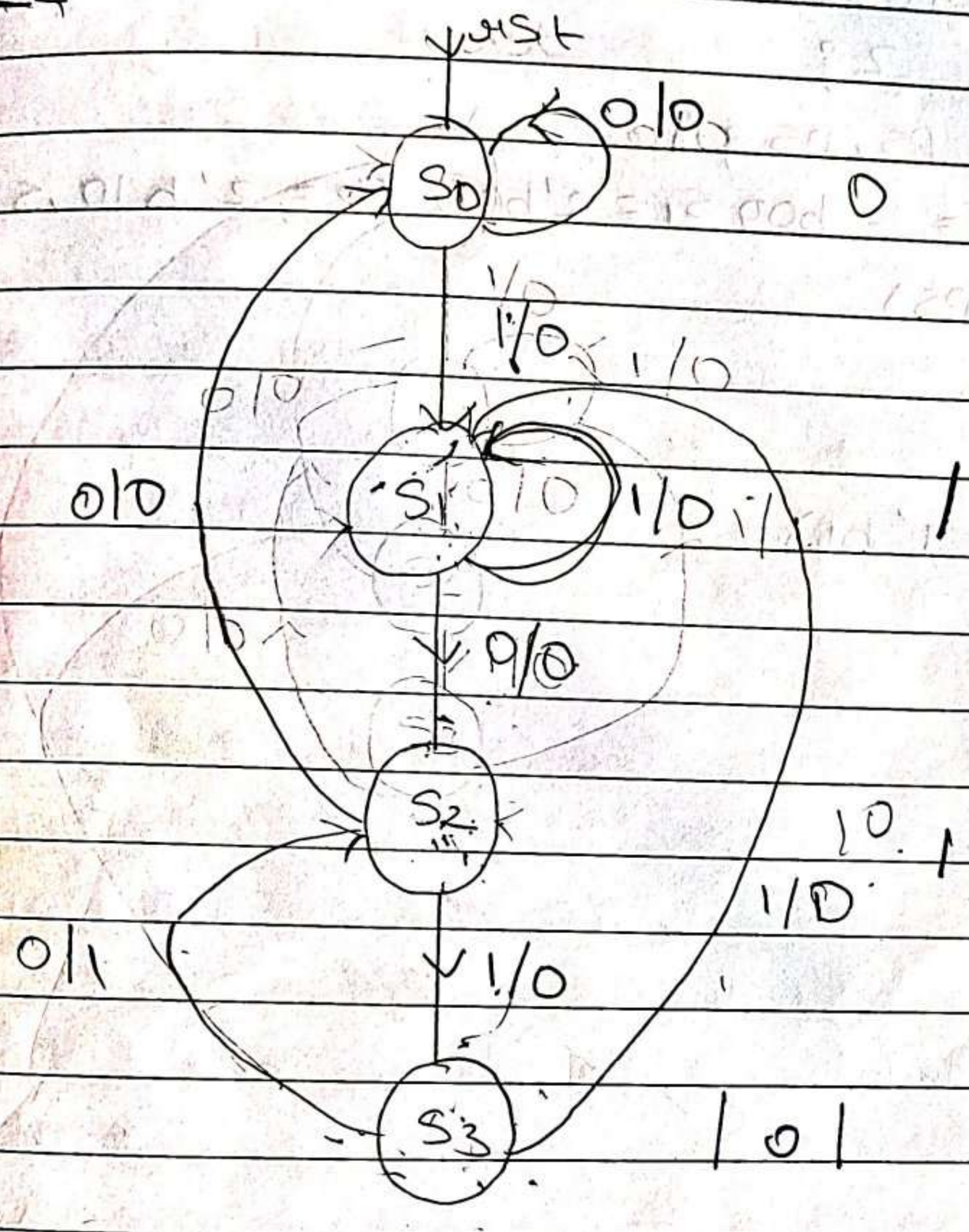


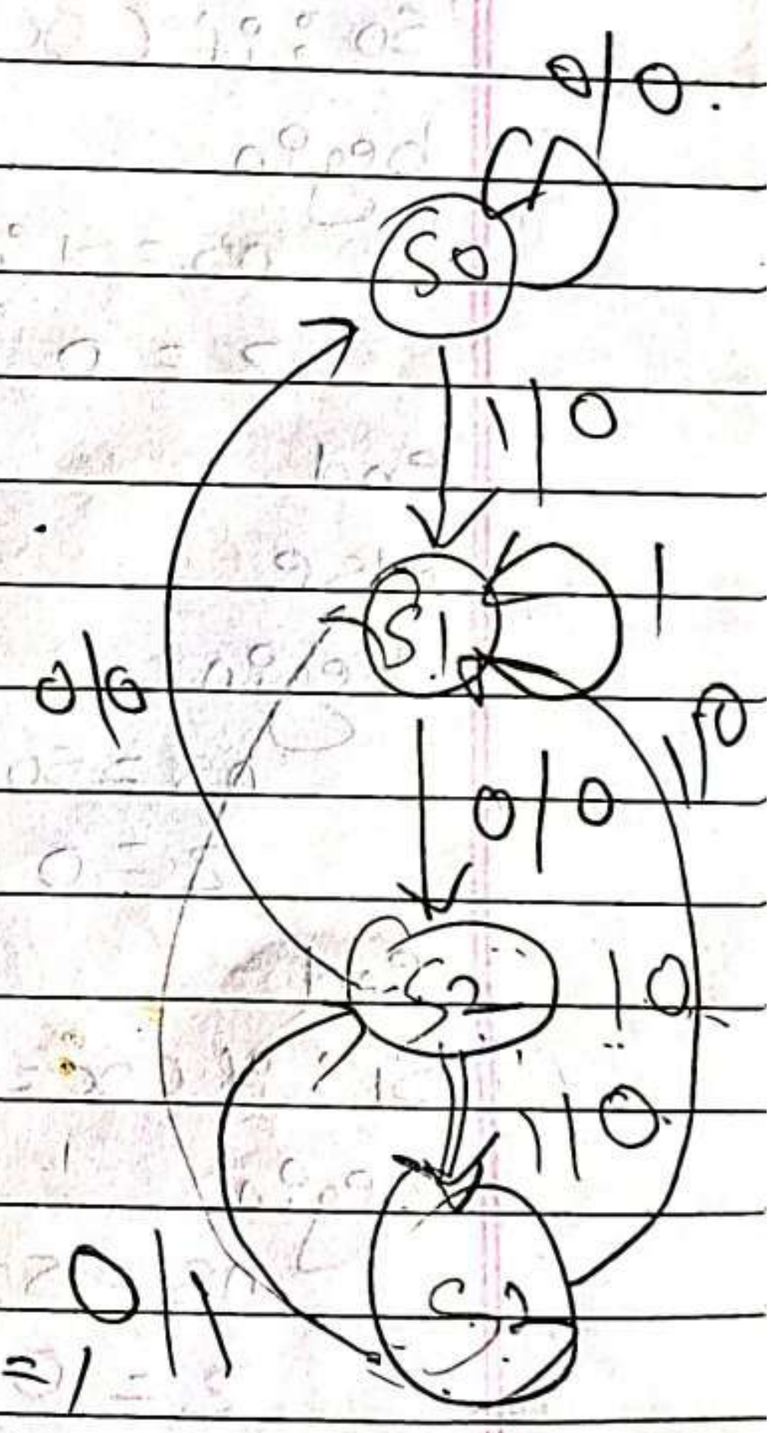
PS	NS x=0	NS x=1	Z
S0	S0	S1	
S1	S2	S1	
S2	S0	S3	
S3	S2	S1	

PAGE NO.:
DATE: / /

Imp

* 1010 detector





Present State [P.S]	Next state x=0	x=1	Z x=0	x=1
S0	S0	S1	0	0
S1	S2	S1	0	0
S2	S0	S3	0	0
S3	S2	S0	1	0

```
module fsm_1010 (z, rst, clk, x);
output reg z;
input [1:0] ps, ns;
parameter s0 = 2'b00, s1 = 2'b01, s2 = 2'b10, s3 = 3'b11
always @ (x, ps)
begin
case (ps)
s0 : if (x = = 1'b1)
begin
    ns = s1;
    z = 0;
end
else
begin
    ns = s0;
    z = 0;
end
s1 : if (x = = 1'b0)
begin
    ns = s2;
    z = 0;
end
else
begin
    ns = s1;
    z = 0;
end
```

```
S2: if (x == 1'b1)
begin
    ns = S3;
    z = 0;
end
else
begin
    ns = S0;
    z = 0;
end
S3: if (x == 1'b0)
begin
    ns = S2;
    z = 1;
end
else
begin
    ns = S0;
    z = 0;
end
endcase
end
always@ (posedge clk)
begin
if (rst)
ps = S0;
else
    ps = ns;
end
endmodule
```

PAGE NO.:
DATE: / /

* Loops :-

There are four types of looping statement system

1. while
2. for
3. Repeat
4. forever

All looping statement appears only inside an initial statement.

1. While loop :-

It executes until the while expression is not true. If the loop is entered until the while expression is not true the loop is not executed. If the multiple statement are to be executed in the loop then should be written in between begin and end.

ex :- // increment count from 0 to 127 & exit

```
// display the count variable integer
count
begin
  count = 0;
while (count <128) // execute loop till count
begin                                  is 127
$ display ("count = %d", count);
  count = count + 1;
end
end
```

PAGE NO.:
DATE: / /

2. Forever Loop :-

ex :- // clock generation
reg clock;
initial
begin
clock = 1' b0;
forever # 10 clock = ~ clock;
// clock with period in units
end

Instead of always block forever loop is written.

3. For loop :-

For loop contains three parts;

1. initial condition.
2. Check condition
3. If the terminating condition is true then procedural assignment changes value.

The initial condition and the incrementing procedural assignment are included in for loop and do not need to be specified separately.

eg :- // increment count from 0 to 127 and exit at 128
// display the count variable
integer count;
initial
begin

```
for (count = 0; count < 128; count = count + 1)
$ display ("count = %d", count);
end
```

4. Repeat loop :-

The Repeat construct must contain a number which can be a constant variable or a single value. If a number is variable it is evaluated only when the loops start.

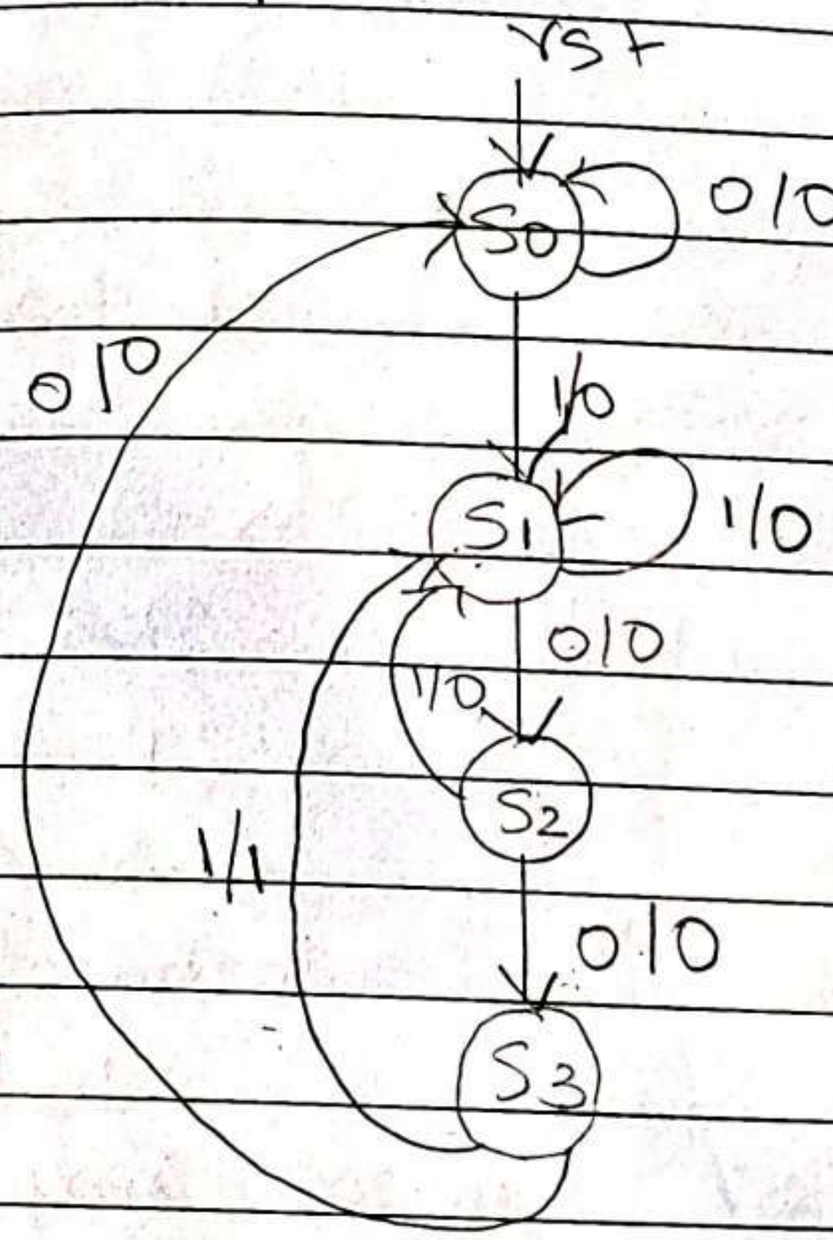
eg :- // Increment count from 0 to 127 and exit at 128.

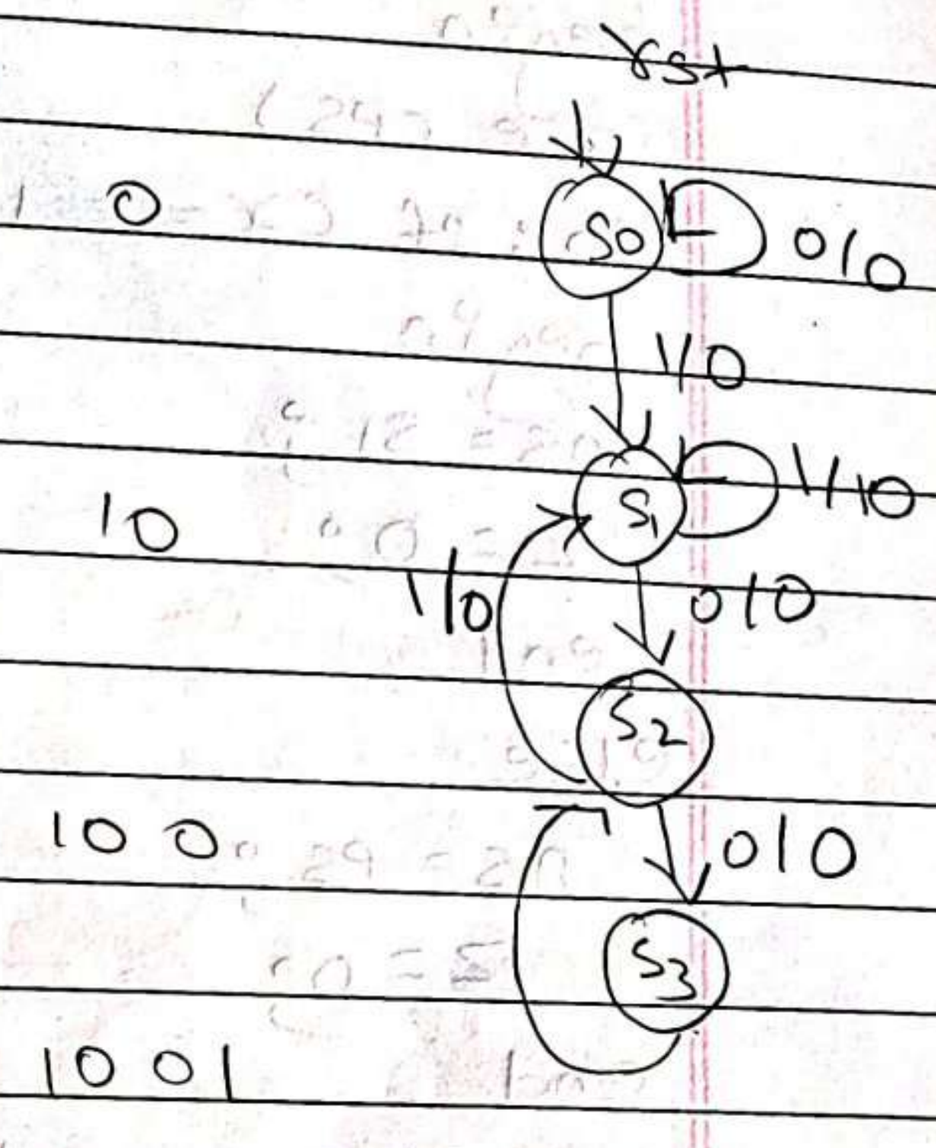
// display the count variable.

```
initial
begin
count = 0;
repeat (128)
begin
$ display ("count = %d", count);
count = count + 1;
end
end
```

DATE: / /

* FSM 1001





1 0

1 0 0

1 0 0 1

PS	NS		Z	
	x=0	x=1	x=0	x=1
S0	S0	S1	0	0
S1	S2	S1	0	0
S2	S3	S1	0	0
S3	S0	S1	0	1

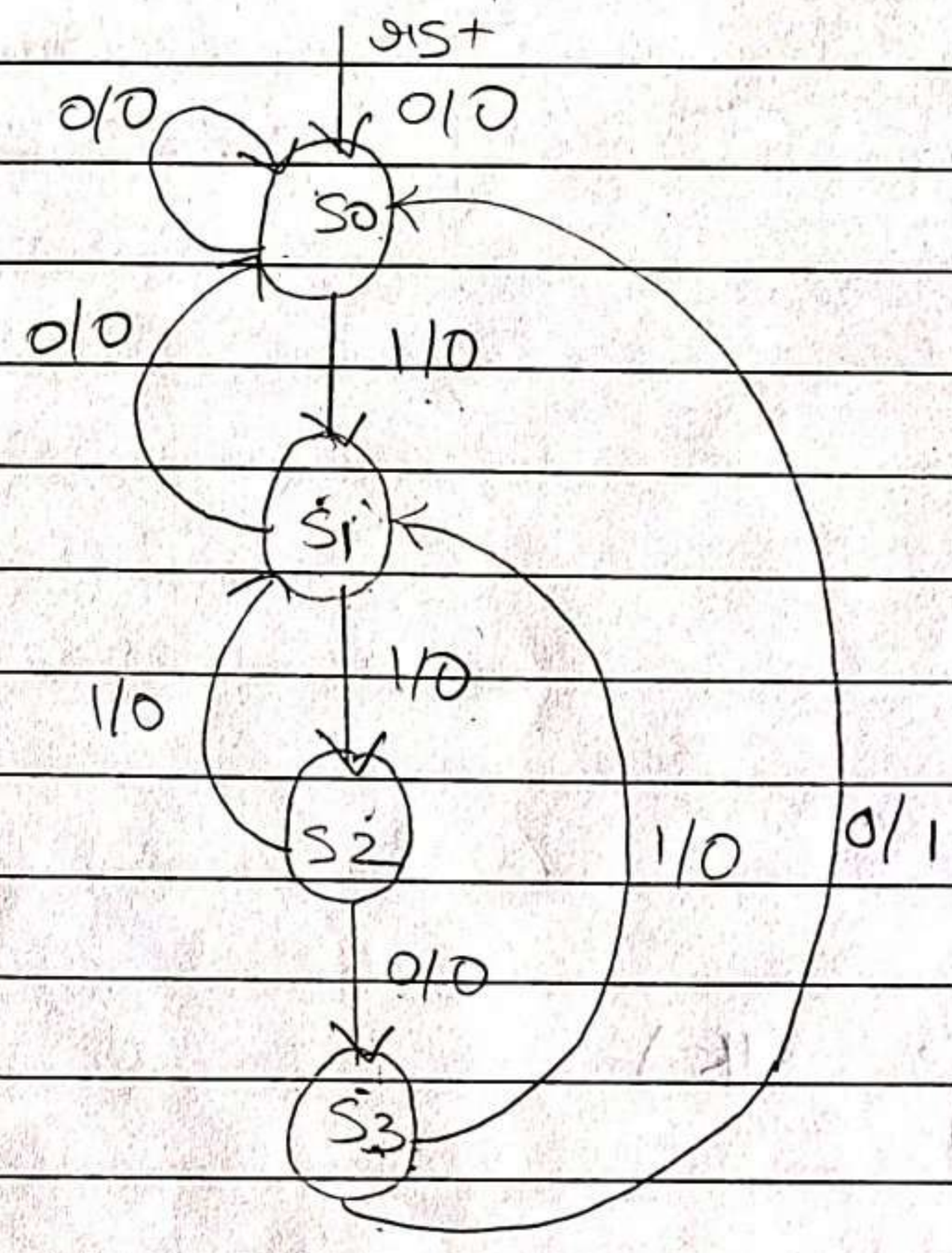
```
Module fsm_1001 (z, rst, clk, x);
output z;
reg z;
input x, rst, clk;
reg [1:0] ps, ns;
parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10,
          S3 = 2'b11;
always @ (x, ps)
```

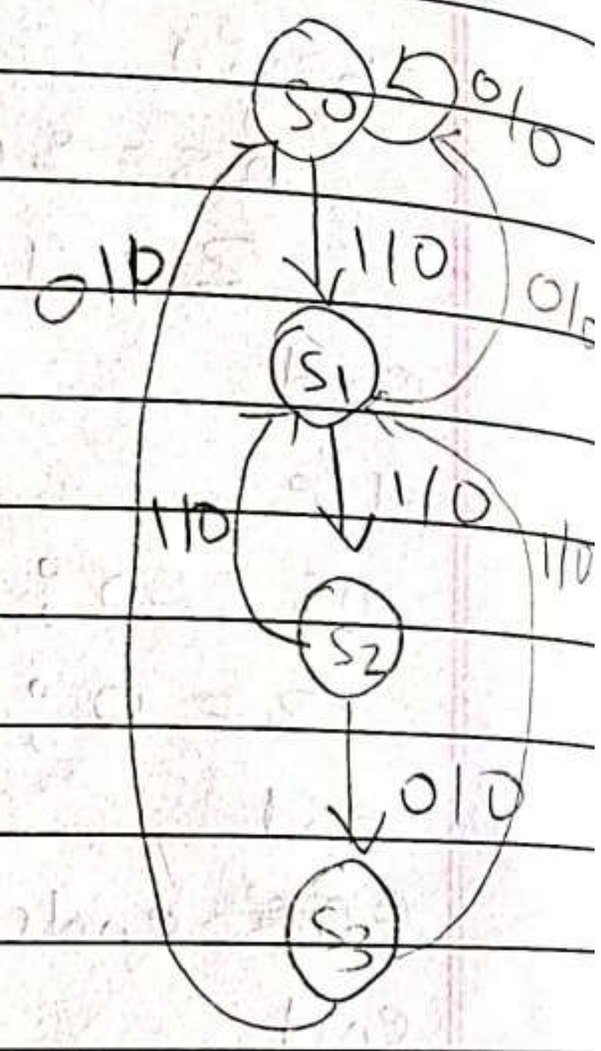
```
begin
case (PS)
S0: if (x==1'b1)
 begin
  NS=S1;
  Z=0;
 end
else
  NS=PS;
   Z=0;
end
S1: if (x==1'b0)
 begin
  NS=S2;
  Z=0;
 end
else
  NS=S1;
  Z=0;
end
S2: if (x==1'b0)
begin
   NS=S3;
   Z=0;
  end
else
  NS=S1;
  Z=0;
end
```

```
S3 : if (x == 1'b1)
begin
  ns = s1;
  z = 1;
end
else
  ns = s0;
  z = 0;
end
~~else~~ endcase
end
always@ (posedge clk)
begin
if (rst)
ps = s0;
else
ps = ns;
end
endmodule
```

Imp

* FSM 1100 mealy





PS	NS		Z	
	x=0	x=1	x=0	x=1
S0	S0	S1	0	0
S1	S0	S2	0	0
S2	S3	S1	0	0
S3	S0	S1	1	0

Scanned with OKEN Scanner

PAGE NO.:
DATE: / /

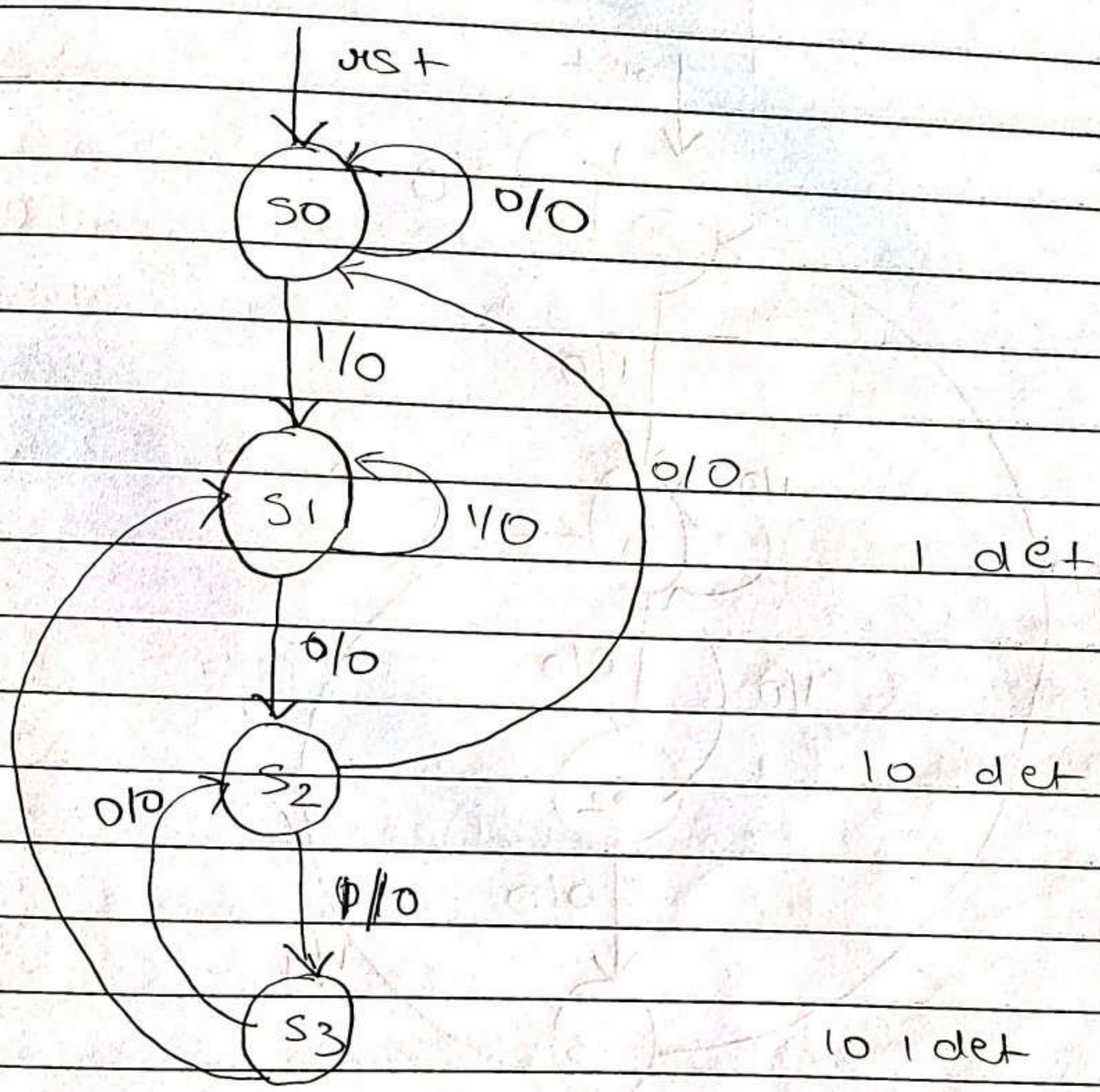
```
module fsm-100 (z, rst, clk, x);
output z;
reg z;
input x, rst, clk;
reg [1:0] ps, ns;
parameter s0 = 2'b00, s1 = 2'b01, s2 = 2'b10,
          s3 = 2'b11;
always @ (x, ps)
begin
case (ps)
s0: if (x = = 1'b1)
begin
  ns = s1;
  z = 0;
end
else
  ns = s0;
  z = 0;
end
s1: if (x = = 1'b1)
begin
  ns = s2;
  z = 0;
end
else
  ns = s0;
  z = 0;
end
```

Scanned with OKEN Scanner

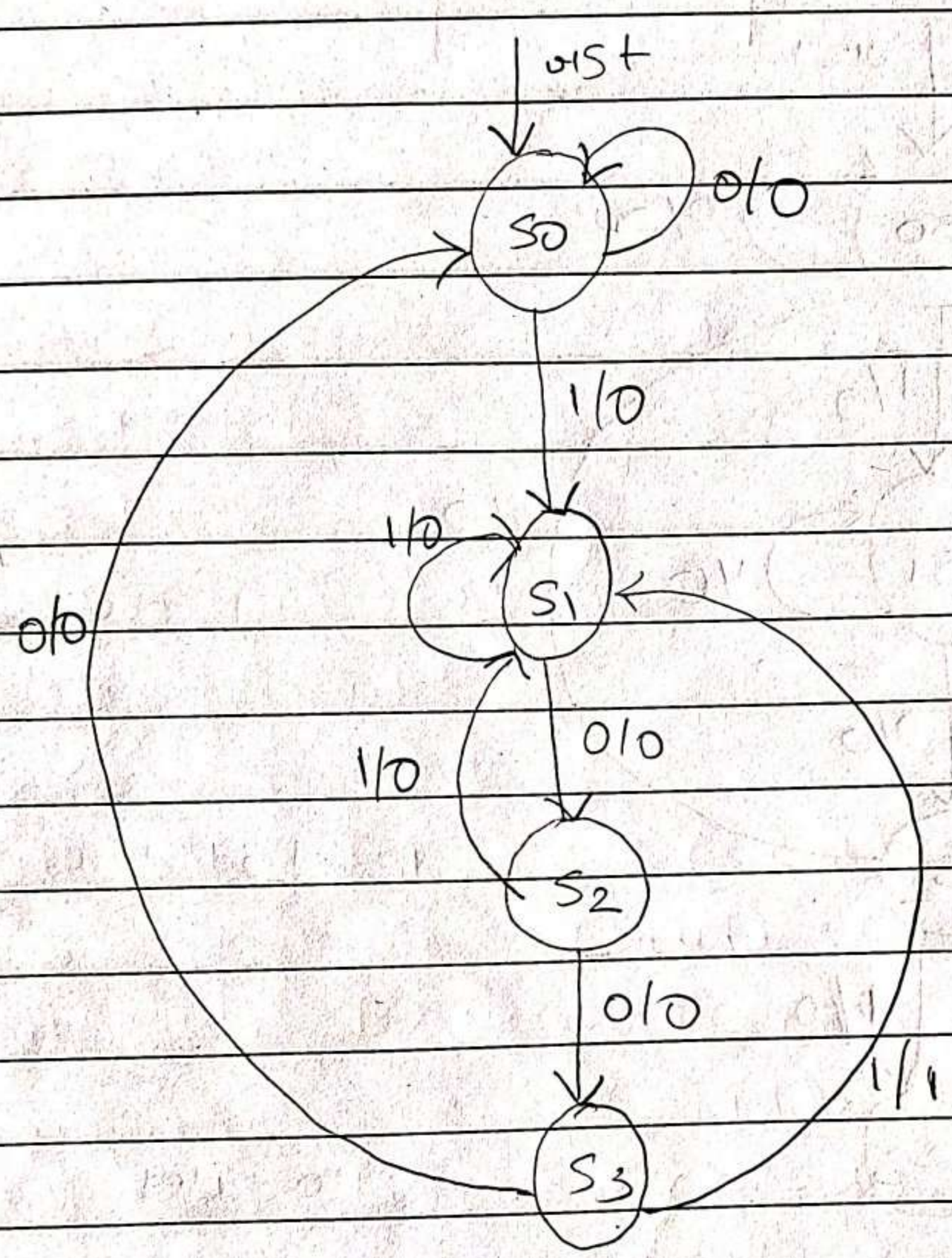
```
S2: if (x==1'b0)
begin
  ns = S3;
  z = 0;
end
else
  ns = S1;
  z = 0;
end
S3: if (x==1'b0)
 begin
  ns = S0;
  z = 1;
 end
else
 ns = S1;
 z = 0;
end
endcase
end
always@ (posedge clk)
begin
if (rst)
ps = 0
else
ps = ns
end
endmodule.
```

PAGE NO.:
DATE: / /

* 1011 fsm mealy :-



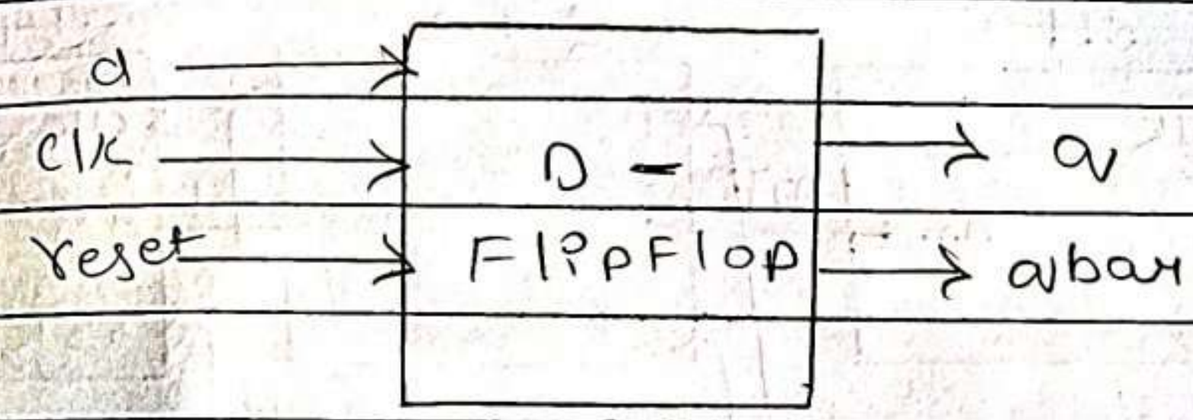
* 1001 fsm - mealy :-



Scanned with OKEN Scanner

PAGE NO.:
DATE: / /

* Behavioral Verilog Code for D-flipflop:

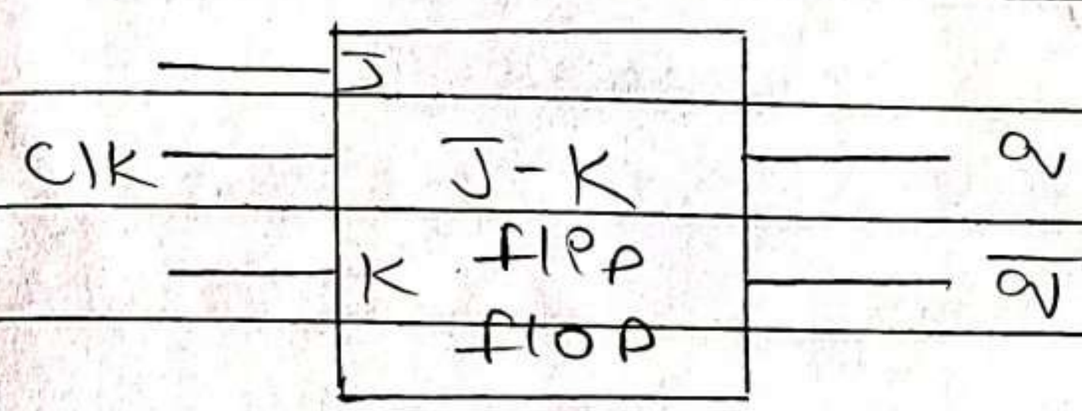


Truth table

rst	clk	d	q	qbar
1	Pos	0	0	1
1	Pos	1	1	0
0	Pos	0	0	1
0	Pos	1	1	0

```
module dff_beh (q, qbar, rst, clk, d);
output q, qbar;
reg q, qbar;
input clk, rst, d;
always@ (posedge clk)
begin
if (rst)
begin
q = 0;
qbar = 1;
end
else
begin
q = d
qbar = !d;
end
end
endmodule
```

* JK flip flop :-



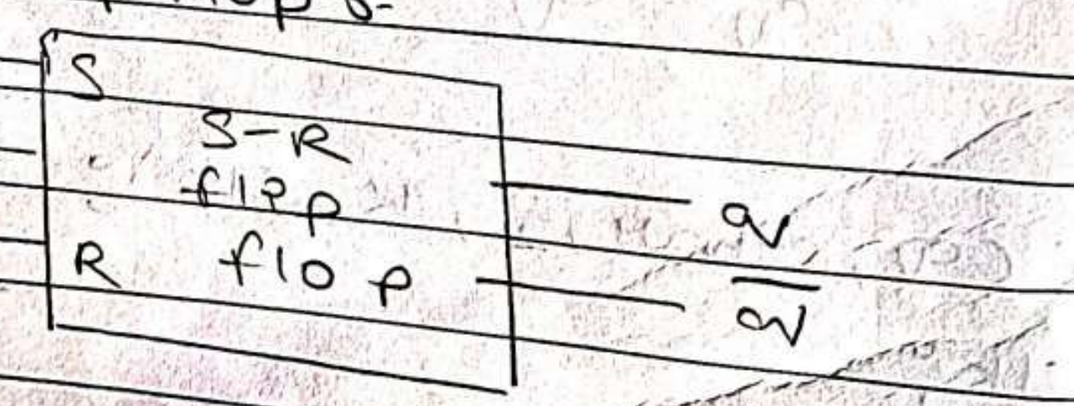
CLK	J	K	q	$\bar{q}$	state
+ve edge	0	0	q	$\bar{q}$	No change
+ve	0	1	0	1	Reset
+ve	1	0	1	0	set
+ve	1	1	$\bar{q}$	q	toggle

```
module JK-ff (q, qbar, J, K, CLK);
input J, K, CLK;
output q, qbar;
reg q, qbar;
always@ (posedge CLK)
begin
case ( {J, K} )
2'b00 : begin
        q = q;
```

```
        qbar = qbar;
        end
2'b01 : begin
        q=0;
        qbar=1;
        end
2'b10 : begin
        q=1;
        qbar=0;
        end
2'b11 : begin
        q = qbar;
        qbar=q;
        end
        endcase
        end
endmodule
```

Exp

SR flip flop :-



clk	S	R	q	$\overline{q}$	State
+ve	0	0	q	qbar	No change
+ve	0	1	0	1	Reset
+ve	1	0	1	0	Set
+ve	1	1	X	X	Invalid

```
module SR-FF (q, qbar, s, r, clk);
input s, r, clk;
output q, qbar;
reg q, qbar;
always @ (posedge clk)
begin
case ({s, r})
2'b00 : begin
        q = q;
        qbar = qbar;
        end
2'b01 : begin
        q = 0;
        qbar = 1;
        end
2'b10 : begin
        q = 1
        qbar = 0;
        end
2'b11 : begin
        q = 'z';
        qbar = 'z';
        end
endcase
end
endmodule
```

—X—