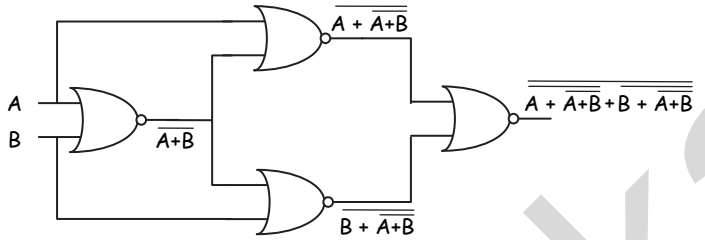
**Q.1(a) Implement Ex-NOR gate using NOR gates.** **[5]**

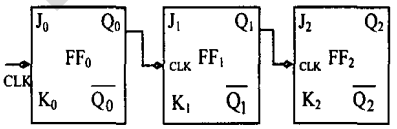
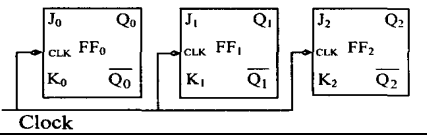
**(A)**




$$\overline{(A+\overline{A+B})+(B+\overline{A+B})} = (A+\overline{A+B})(B+\overline{A+B}) = (A+\overline{A}\,\overline{B})(B+\overline{A}\,\overline{B})$$
$$= AB+A\overline{A}\,\overline{B}+\overline{A}\,\overline{B}B+B\overline{B} = AB+\overline{A}\,\overline{B}$$
$$= \overline{A\oplus B}$$

**Q.1(b) Compare asynchronous counter with synchronous counter.** **[5]**

**(A)**

| | Asynchronous Counter | Synchronous Counter |
|---|---|---|
| 1. | In an Asynchronous Counter the output of one Flip Flop acts as the clock Input of the next Flip Flop. | In a Synchronous Counter all the Flip Flop's are connected to a common clock signal. |
| 2. | Speed is High. | Speed is Low. |
| 3. | Only JK or T Flip Flop can be used to construct Asynchronous Counter. | Synchronous Counter can be designed using JK, RS, T and D Flip Flop. |
| 4. | Problem of Glitch arises. | Problem of Lockout. |
| 5. | Only serial count either up or down is possible. | Random and serial counting is possible. |
| 6. | Settling time is more. | Settling time is less. |
| 7. | Also called as serial counter. | Also called as Parallel Counter. |
| 8. |  |  |

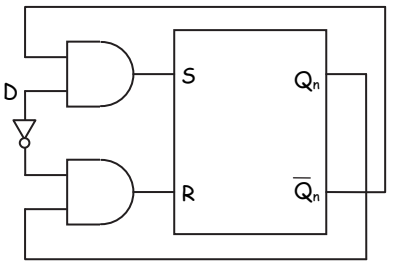
**Q.1(c) Convert SR Flip-Flop into D Flip-Flop and T Flip-Flop.** **[5]**

**(A)** Excitation table of SR, D and T flip-flops are :

| $Q_n$ | $Q_{n+1}$ | D | T | S | R |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | × |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | × | 0 |

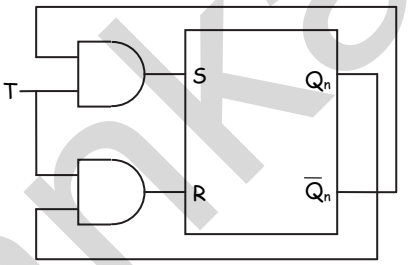
1) $S = \bar{Q}_n D$, $R = Q_n \bar{D}$

SR to D




2) $S = \bar{Q}_n T$, $R = Q_n T$

SR to T




**Q.1(d) Implement full adder using 8:1 multiplexers.** **[5]**

**(A)** A full-adder circuit is an arithmetic circuit block that is capable of adding three binary digits to produce a sum and carry output.

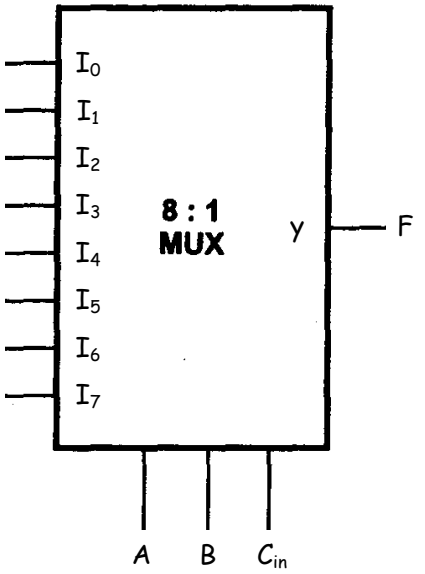
**Truth Table :**

| A | B | $C_{in}$ | Sum (S) | $C_{out}$ |
|---|---|---|---|---|
| 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 |

The Boolean expression for output variables are

$$S = \bar{A} \cdot \bar{B} \cdot C_{in} + \bar{A} \cdot B \cdot \bar{C}_{in} + \bar{\bar{A}} \cdot \bar{B} \cdot \bar{C}_{in} + A \cdot B \cdot C_{in}$$

$$C_{out} = \bar{A} \cdot B \cdot C_{in} + A \cdot \bar{B} \cdot C_{in} + A \cdot B \cdot \bar{C}_{in} + A \cdot B \cdot C_{in}$$




| A | B | $C_{in}$ | F |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

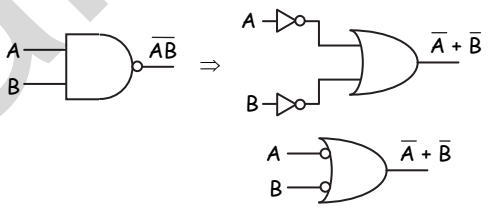
**Q.2(a) State and prove De-Morgan theorem? Also prove Boolean algorithm laws.** **[10]**

**(A)** **Theorem :** $\overline{(A \cdot B)} = \overline{A} + \overline{B}$

Compliment of product equal to sum of their compliments.

Proof :

| A | B | $\overline{A}$ | $\overline{B}$ | $\overline{AB}$ | $\overline{A+B}$ |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 |

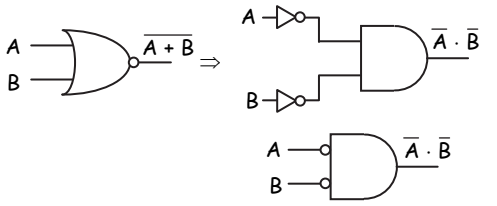



**Theorem :** $\overline{(A+B)} = \overline{A} \cdot \overline{B}$

Compliment of sum equal to product of their compliments.

Proof :

| A | B | $\overline{A}$ | $\overline{B}$ | $\overline{AB}$ | $\overline{A+B}$ |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 |



**Boolean Algebra :**

The behavior of binary numbers using the switching algebra was developed by the British mathematician George Boole in the nineteenth century. The branch of mathematics is contained within the concepts of Boolean algebra, and provides the basis for modern logic design.

| OR Identities | AND Identities |
|---|---|
| $A + 0 = A$ | $A \cdot 0 = 0$ |
| $A + 1 = 1$ | $A \cdot 1 = A$ |
| $A + A = A$ | $A \cdot A = A$ |
| $A + \overline{A} = 1$ | $A \cdot \overline{A} = 0$ |
| $\overline{\overline{A}} = A$ | |
| $A + B = B + A$ | $A \cdot B = B \cdot A$ |
| $A + (B + C) = (A + B) + C$ | $A \cdot (B \cdot C) = (A \cdot B) \cdot C$ |
| $A \cdot (B + C) = A \cdot B + A \cdot C$ | $A + (B \cdot C) = (A + B) \cdot (A + C)$ |
| $\overline{(A + B)} = \overline{A} \cdot \overline{B}$ | $\overline{(A \cdot B)} = \overline{A} + \overline{B}$ |
| $A + A \cdot B = A$ | $A + \overline{A} \cdot B = A + B$ |

**Algebraic Laws**

**Commutative Laws :** The laws of commutation allow us to arrange variables in any order without changing the result. With two variables A and B, these are given by $A + B = B + A$

$A \cdot B = B \cdot A$

**Associative Laws :** Associative laws define the order in which the operations are performed. In both the OR and the AND operations, the grouping does not affect the result. This gives identities such as

$A + B + C = (A + B) + C = A + (B+C)$

for the OR operation, and identities such as

$A\ B \cdot C = (A\ B) \cdot C = A \cdot (B\ C)$

for the AND expressions. Since the associative law applies to both the OR and the AND operations, we usually omit parentheses when writing logic expressions.

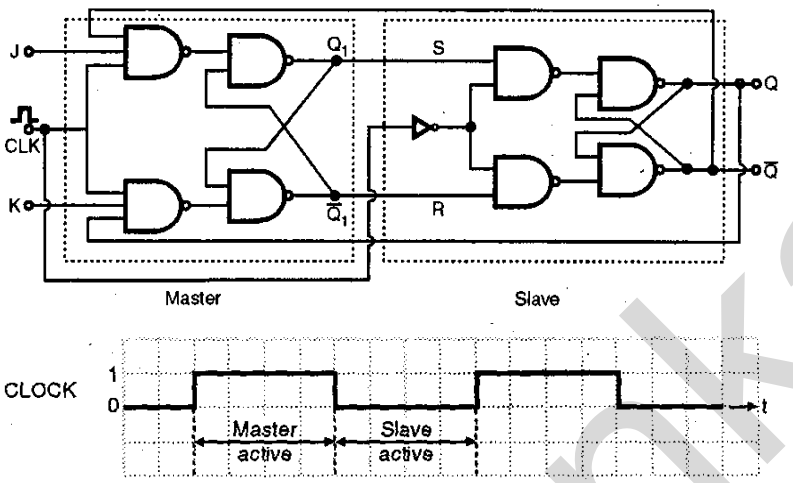
**Distributive Laws :** The distribution of AND and OR operations is governed by the following laws. It is important to remember the rule of precedence that, within a grouping, the AND operation always precedes the OR. The two important distributive laws are given by

$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$

$A + (B \cdot C) = (A + B) \cdot (A + C)$

**Q.2(b) Explain how master slave JK Flip-Flop avoids race around condition. [10]**

**(A) Master Slave JK Flip Flop**




Master is positive level triggered. But due to the presence of the inverter in the clock line, the slave will respond to the negative level. Hence when the clock = I (positive level) the master is active and the slave is inactive, whereas when clock = 0 (low level) the slave is active and the master is inactive.

**Case I :** Clock = X, J = K = 0

(i) For clock = 1, the master is active, slave inactive. As J = K = 0. Therefore, output of master, i.e., $Q_1$ and $\bar{Q}_1$ will not change. Hence the S and R inputs to the slave will remain unchanged.

(ii) As soon as clock = 0, the slave becomes active and master is inactive. But since the S and R inputs have not changed, the slave outputs will also remain unchanged.

$\therefore$ The outputs won't change if J = K = 0.

**Case II :** Clock = ⎍, J = K = 0.

This condition has been already discussed in case f.

**Case III :** Clock = ⎍, J = 0 and K = 1.

- Clock = 1 : Master active, slave inactive.
- Therefore, outputs of the master become $Q_1 = 1$ and $\bar{Q}_1 = 0$.

  That means S = 0 and R = 1.
- Clock = 0 : Slave active, master inactive.
- Therefore, outputs of the slave become Q = 0 and $\bar{Q} = 1$.

- Again if clock = 1 : Master active, slave inactive.
  Therefore, even with the changed outputs Q = 0 and $\bar{Q}$ = 1 feedback to master, its outputs will $Q_1$ = 0 and $\bar{Q}_1$ = 1. That means S = 0 and R = 1.
- Hence with clock = 0 and slave becoming active, the outputs of slave will remain Q = 0 and $\bar{Q}$ = 1.
- Thus we get a stable output from the Master Slave.

**Case IV :** Clock = ⎍, J = 1, K = 0.

- Clock = 1 : Master active, slave inactive.
  Therefore, outputs of master become $Q_1$ = 0 and $\bar{Q}_1$ = 0, i.e., S = 1, R = 0.
- Clock = 0 : Master inactive, slave active.
  Therefore, outputs of slave become Q = 1 and $\bar{Q}$ = 0.
- Again if clock = 1 then it can be shown that the outputs of the slave are stabilized to Q = 1 and $\bar{Q}$ = 1.

**Case V :** Clock = ⎍, J = 1, K = 1.

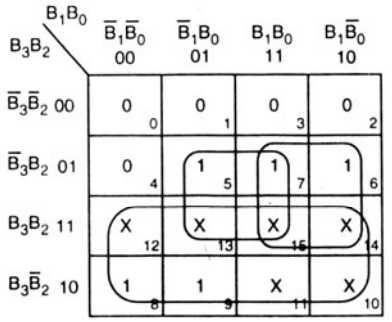
- Clock = 1 : Master active, slave inactive.
  Therefore, outputs of master will toggle. So S and R also will be inverted.
- Clock = 0 : Master inactive, slave active.
  Therefore, outputs of the slave will toggle.
- These changed output are returned back to the master inputs.
- But since clock = 0, the master is still inactive. So it does not respond to these changed outputs.
- This avoids the multiple toggling which leads to the race around condition. Thus the master slave flip flop will avoid the race around condition.

**Q.3(a) Design BCD to Ex-3 code converter.** **[10]**

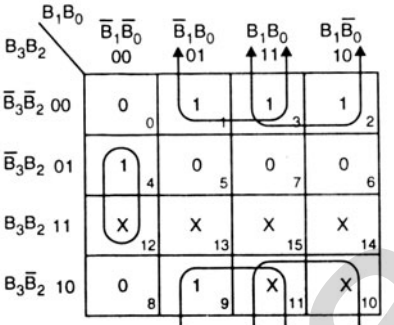
**(A)** BCD is valid upto $(9)_{10}$ only. But as we are taking 4 bits, states will be 0 to 15. Therefore states from $(10)_{10}$ to $(15)_{10}$ the output will be 'X' (don't care) because we know that for valid BCD, these codes (10–15) will not occur. Table below shows the truth table.

| Decimal | $B_3$ | $B_2$ | $B_1$ | $B_0$ | $E_3$ | $E_2$ | $E_1$ | $E_0$ |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 5 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 6 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 7 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| 8 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 9 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |

Now draw K–maps :

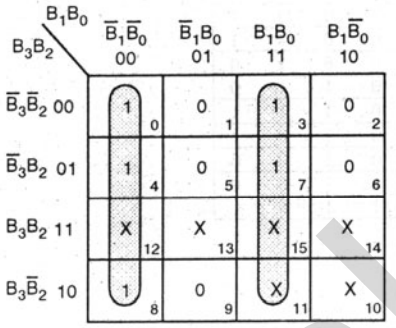



$E_3 = B_3 + B_2 (B_0 + B_1)$



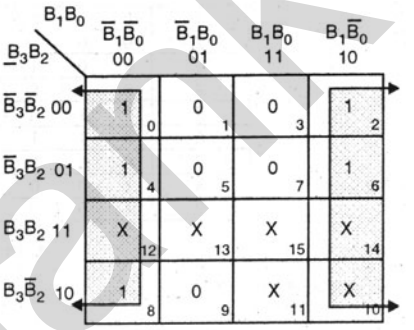

$E_2 = B_2\bar{B}_1\bar{B}_0 + \bar{B}_2B_0 + \bar{B}_2B_1$

$= B_2\bar{B}_1\bar{B}_0 + \bar{B}_2(B_0+B_1)$



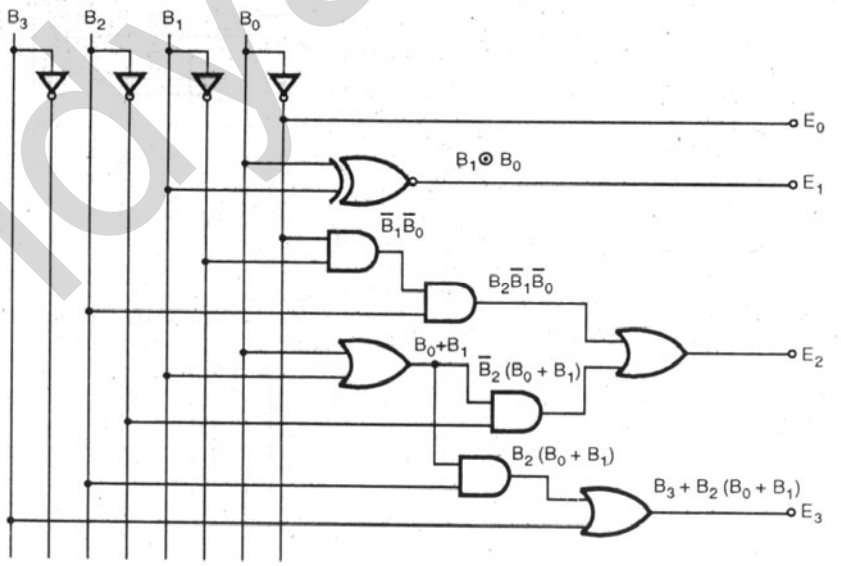

$E_1 = \bar{B}_1\bar{B}_0 + B_1B_0$

$= B_1 \odot B_0$



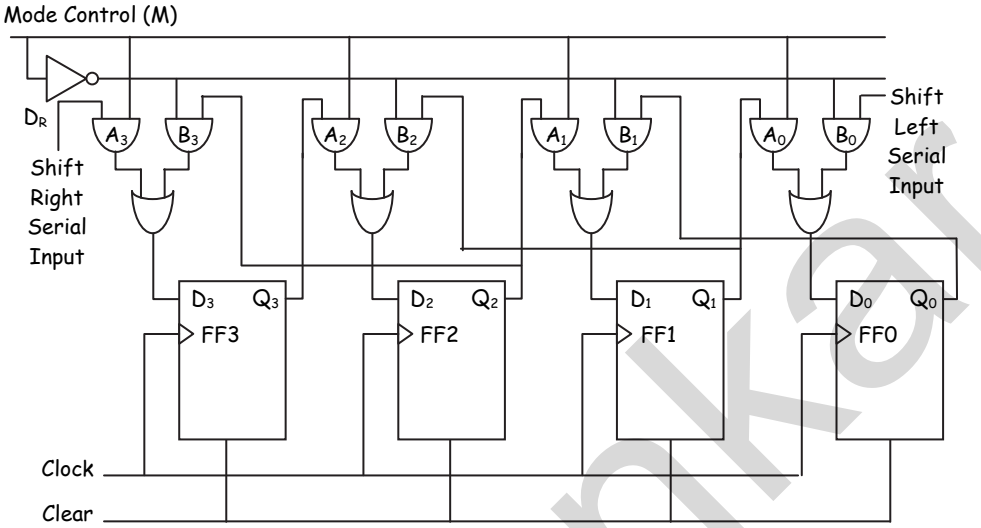

$E_0 = \bar{B}_0$

**Circuit :**



**Q.3(b) Explain 4 bit bidirectional shift register.** **[10]**

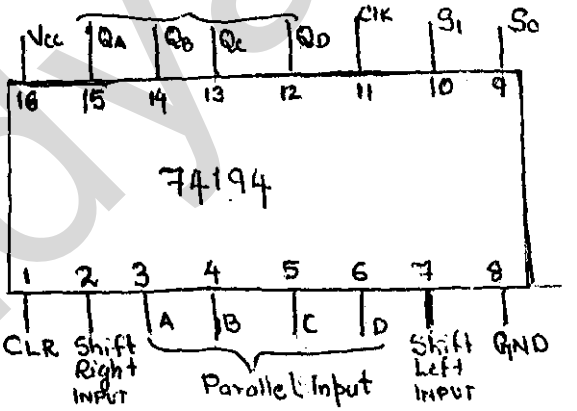
**(A) Bi–Directional Shift Register**

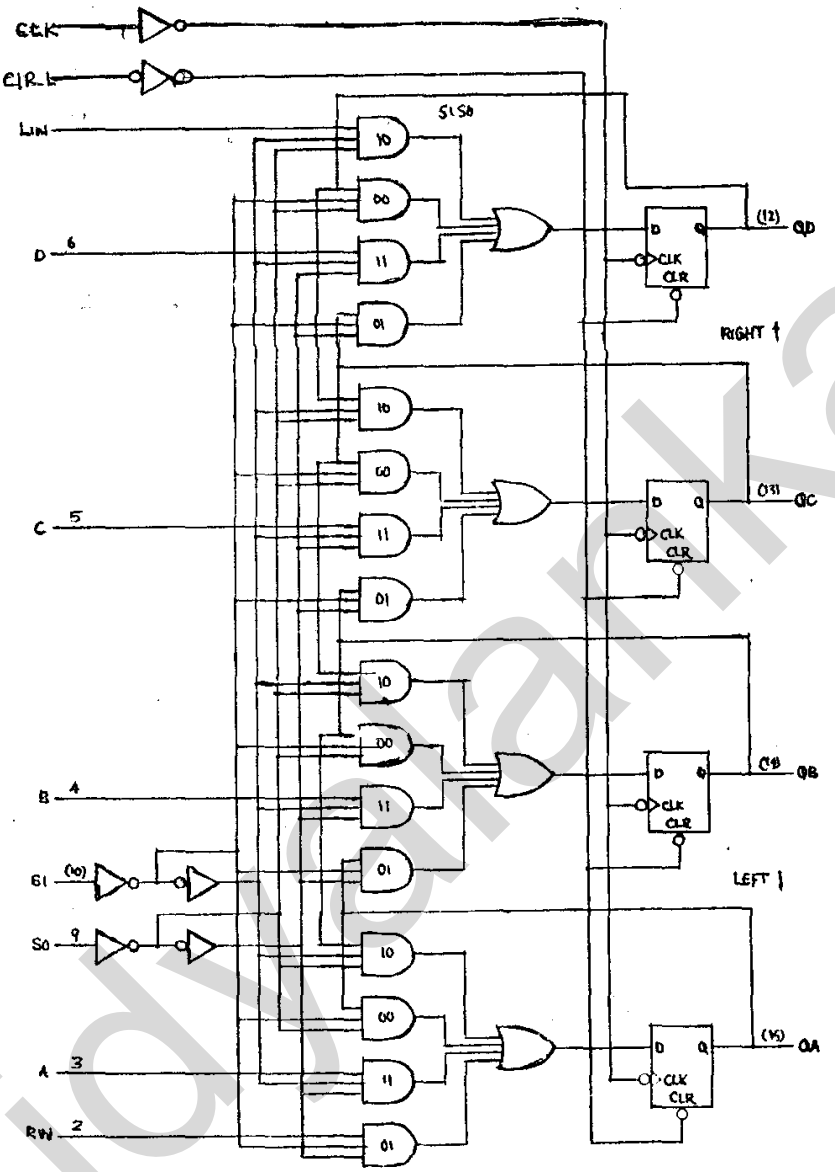



When Mode Control M = 1 all the 'A' AND gates are enabled and DR is shifted to lie right, when the clock pluses are applied.

When Mode Control M =0 all the 'B' AND gates are enabled and DL is shifted to the left.

**Bidirectional Universal Shift Register**






Internal Schematic of IC74194

| Functions | INPUTS | | Next State | | | |
|---|---|---|---|---|---|---|
| | S1 | S0 | Qa* | Qb* | Qc* | Qd* |
| Hold | 0 | 0 | Qa | Qb | Qc | Qd |
| Shift Right | 0 | 1 | Rin | Qa | Qb | Qc |
| Shift Left | 1 | 0 | Qb | Qc | Qd | Lin |
| Load | 1 | 1 | A | B | C | D |

**Q.4(a) Reduce the following function using Quine Mc-Clusky method : [10]**

**$f(A,B,C,D) = \sum m(0,2,5,7,8,10,12,15)$**

**(A)** To minimize the following expression using Quice-McClusky method

$f(A, B, C, D) = \Sigma\, m(0, 2, 5, 7, 8, 10, 12, 15)$

| | | |
|---|---|---|
| 0 | 0000 | ✓ |
| 1 | 0001 | × |
| 2 | 0010 | ✓ |
| 3 | 0011 | × |
| 4 | 0100 | × |
| 5 | 0101 | ✓ |
| 6 | 0110 | × |
| 7 | 0111 | ✓ |
| 8 | 1000 | ✓ |
| 9 | 1001 | × |
| 10 | 1010 | ✓ |
| 11 | 1011 | × |
| 12 | 1100 | ✓ |
| 13 | 1101 | × |
| 14 | 1110 | × |
| 15 | 1111 | ✓ |

| | | | |
|---|---|---|---|
| group 0 | 0 | 0000 | ✓ |
| group 1 | 2 | 0010 | ✓ |
| | 8 | 1000 | ✓ |
| group 2 | 5 | 0101 | ✓ |
| | 10 | 1010 | ✓ |
| | 12 | 1100 | |
| group 3 | 7 | 0111 | ✓ |
| group 4 | 15 | 1111 | ✓ |

| Grouping Column | |
|---|---|
| 0,2 | 00–0 |
| 0,8 | –000 |
| 2,10 | –010 |
| 8,10 | 10–0 |
| 5,7 | 01–1 |
| 7,15 | –111 |

Dividing the grouping columns, further into groups, we get

| Group column | | |
|---|---|---|
| 0,2 | 00–0 | ✓ |
| 0,8 | –000 | ✓ |
| 2,10 | –010 | ✓ |
| 8,10 | 10–0 | ✓ |
| 5,7 | 01–1 | |
| 7,15 | –1,1,1 | |

Column 2

0,2,8,10 –0–0

~~0,8,2,10 –0–0~~ ← Eliminating redundancy

Consider the unchecked terms

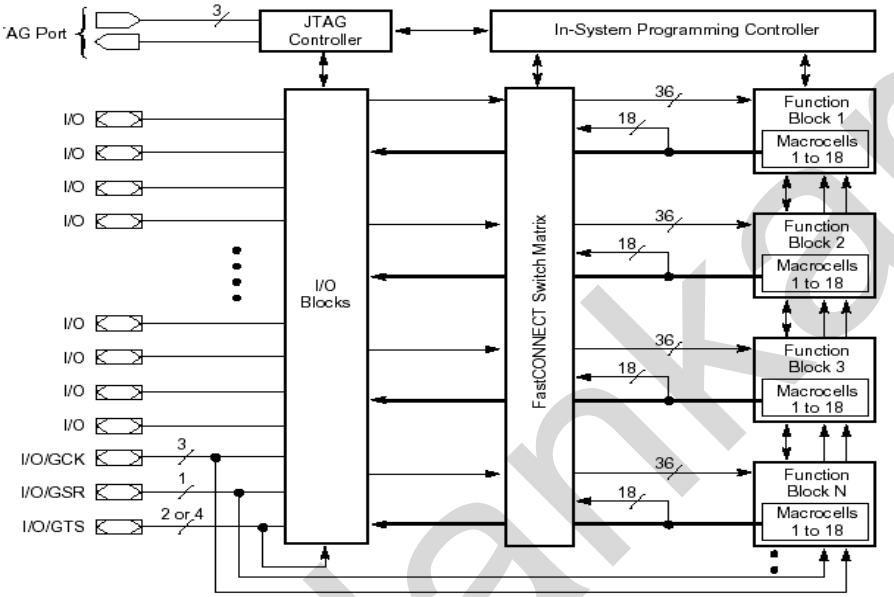
$\therefore \quad f = \bar{A}BD + BCD + B\bar{D}$

| AB | CD: 00 $\bar{C}\bar{D}$ | 01 $\bar{C}D$ | 11 $CD$ | 10 $C\bar{D}$ |
|---|---|---|---|---|
| 00 $\bar{A}\bar{B}$ | 1 | | | 1 |
| 01 $\bar{A}B$ | | 1 | 1 | |
| 11 $AB$ | | | 1 | |
| 10 $A\bar{B}$ | 1 | | | 1 |

$f = B\bar{D} + ABCD + \bar{A}BD$

**Q.4(b) Draw block diagram of Internal architecture of XC 9500 family CPLD and explain. [10]**

**(A) Architecture of Xilinx XC9500 CPLD Family**




**Architectural Description**

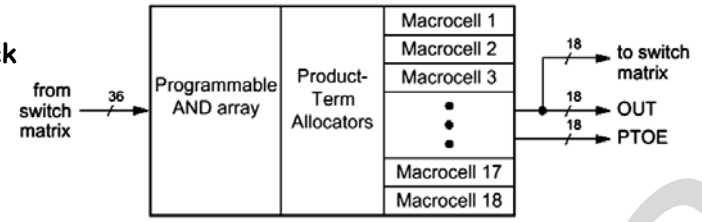
Each external I/O pin can be used as an input, an output, or a bidirectional pin according to device programming. The I/O pins at the bottom are also used for special purposes. Any of the 3 pins can be used as "Global Clocks" (GCK).Each macrocell can be programmed to use a selected clock input. One pin can be used as a "Global Set/Reset" (GSR).Each macrocell can use this signal as an asynchronous Preset or Clear. Two or Four pins depending on the devices can be used as "Global Three State Controls" (GTS).One of the signals can be selected in each macrocell to output enable the corresponding output driver when the macrocell's output is hooked to an external I/O pin.

Only four Functional Blocks (FB) are shown but XC9500 scales to accommodate 16 FB's in the XC95288.Regardless of the specific family member each FB programmable receives 36 signals from the switch matrix. The inputs to the switch matrix are the 18 macrocell outputs from each of the functional blocks and the external inputs from the I/O pins.

Each Functional block also has 18 outputs that run under the switch matrix and connect to the I/O blocks. These are the output-enable signals for the I/O block output drives; they're used when FB macrocells's output is hooked up to an external I/O pin. Each Functional Block has programmable logic

capability with 36 inputs and 18 outputs. Fast Connect Switch Matrix connects all Functional Block outputs to the I/O blocks and the input signals from the I/O block to the Functional Block.

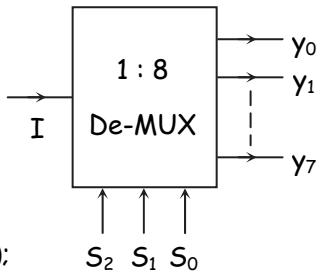
**Function block**




The Functional Block consists of 18 independent macrocells capable of implementing combinatorial or a registered function.. Functional Block also receives global clock, output enable, set/reset signals. The functional block generates 18 outputs that drive FastCONNECT switch matrix. These 18 outputs and their corresponding 18 output enable signals also drive the I/O Block. 36 inputs provides 72 true and complement signals into programmable AND-array to form 90 product terms. Product term allocator allocates any number of these 90 product terms to each macrocell. Each function block supports local feedback paths that allow any number of Functional Outputs to drive its own Programmable AND array without going outside the Functional Block. XC9500 and other CPLD's have product-term allocators that allow a macrocell's unused product terms to be used by other nearby macrocells in the same Functional Block.

**Q.5(a) Write VHDL code of 1:8 De-mux.** **[10]**

**(A)**




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity demux1_8 is
  Port ( I : IN STD_LOGIC;
       S : IN STD_LOGIC_VECTOR(2 downto 0);
       Y : OUT STD_LOGIC_VECTOR(7 downto 0));
END demux1_8;

ARCHITECTURE behavioral OF demux1_8 IS

BEGIN
PROCESS(s)
BEGIN
If (I = 1) then
CASE s IS
      WHEN "000"=>y<="00000001";
```

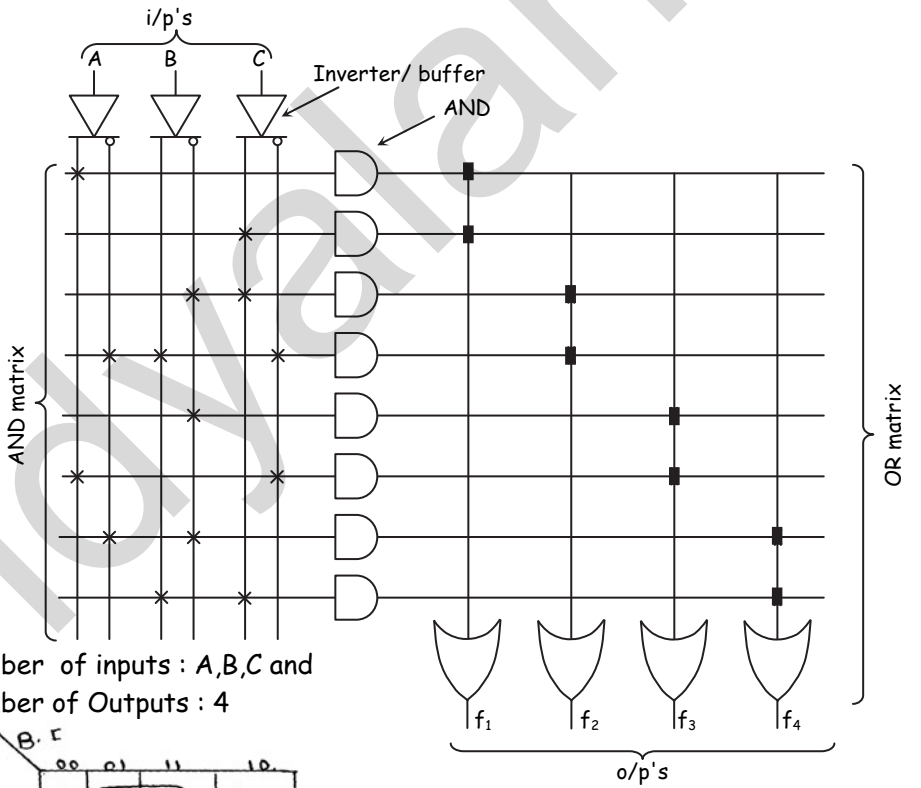
```
                WHEN "001"=>y<="00000010";
                WHEN "010"=>y<="00000100";
                WHEN "011"=>y<="00001000";
                WHEN "100"=>y<="00010000";
          WHEN "101"=>y<="00100000";
                WHEN "110"=>y<="01000000";
                WHEN OTHERS =>y<="10000000";
                Else y < = "00000000"
    END CASE;
    END PROCESS;
    END behavioral;
```

**Q.5(b) Design a circuit with optimum utilization of PLA to implement following functions : [10]**

**$f_1 = \sum m\ (1,3,4,5,6,7)$** **$f_2 = \sum m\ (1,2,5)$**

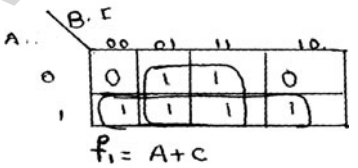
**$f_3 = \sum m\ (0,1,4,5,6)$** **$f_4 = \sum m\ (0,1,3,7)$**

**(A)**




Number of inputs : A,B,C and

Number of Outputs : 4



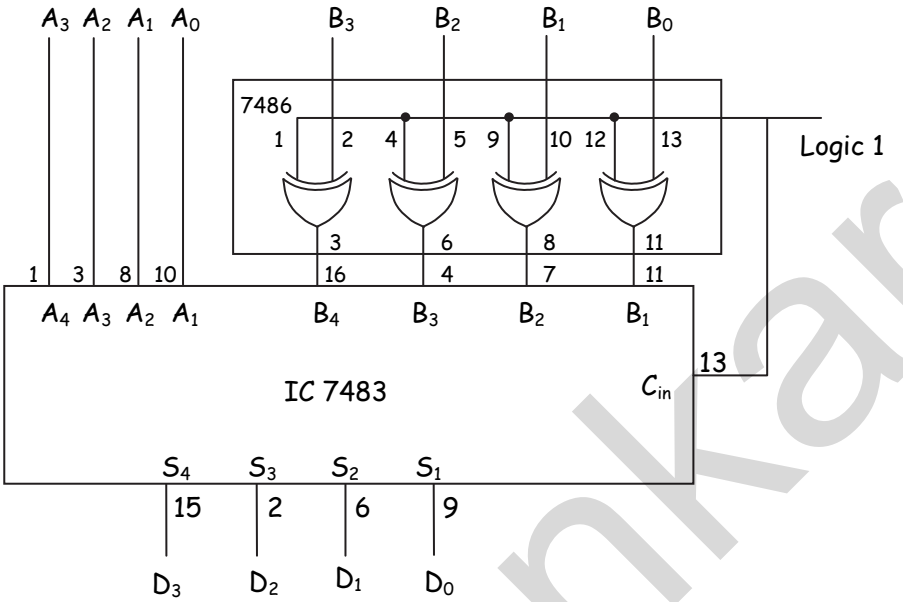

Similarly $f_2 = \bar{B}C + \bar{A}B\bar{C}$ $f_3 = \bar{B} + A\bar{C}$ $f_4 = \bar{A}\bar{B} + BC$

**Q.6(a) Explain BCD adder with suitable example. [8]**

**(A) BCD adder using IC 7483**

A BCD adder is used to perform the addition of BCD numbers. A BCD digit can have any of the ten possible 4-bit binary representations, i.e. 0000, 0001, … 1001, equivalent of decimal numbers 0, 1, …9 when we set out to add tow BCD digits and we assume that there is an input carry too, the highest binary number that we can get is the equivalent of decimal number 19 (9 + 9 + 1). So the binary number is $(10011)_2$. If we do BCD addition, the expected answer would be $(0001\ 1001)_{BCD}$. If we restrict the output bits to the minimum required, the result in BCD would be $(1001)_{BCD}$.

| Decimal Sum | Binary Sum | | | | | BCD Sum | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | K | $Z_3$ | $Z_2$ | $Z_1$ | $Z_0$ | C | $S_3$ | $S_2$ | $S_1$ | $S_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 4 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 5 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 6 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 7 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 8 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 9 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 10 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 11 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 12 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 13 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 14 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 15 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 16 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 17 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| 18 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 19 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |




**Q.6(b) Explain static RAM.** **[6]**

**(A) Static MOS RAM Cell**

They are volatile, the data or information is lost if the power is switched off.

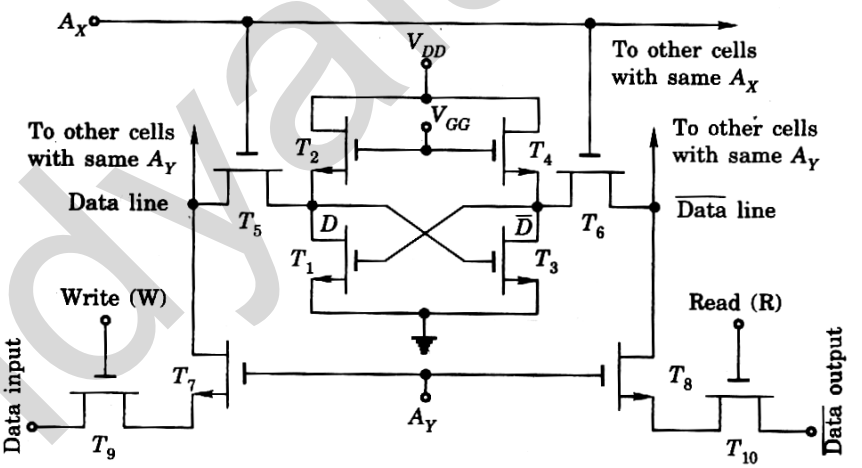



**Fig.** : Static MOS RAM cell

It is addressed by setting $A_x = 1$

When $A_x = A_y = 1$, cell is connected to data and $\overline{data}$ line. When $A_y = 1$ $T_7$, $T_8$ are ON.

*Write operation* : Set W = 1, $T_9$ is ON

If data input = 1, making $T_3$ ON, and $\overline{D} = 0$

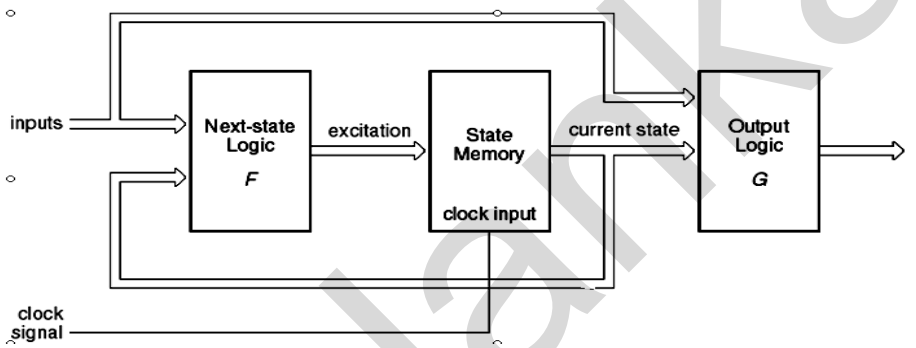
Similarly if data input = 0, making $T_3$ OFF, level at $\overline{D} = 1$

*Read operation :* Set R = 1, this connect data output to $\overline{D}$.
Hence data is read as $\overline{Data}$ output.
Power consumption can be reduced by using $V_{GG}$ supply only during Read /write operation.
It is called static, because the data remains stored unless it is changed by new entry or power is shutdown.

***Note :*** *$T_2$, $T_4$ are off during normal operation.*

**Q.6(c) Explain Mealy and Moore machines. [6]**

**(A) Mealy Machine**

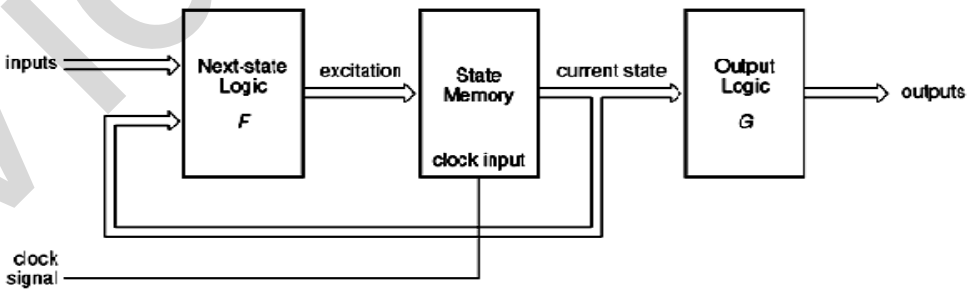



From the figure we observe the following :
Next logic F is function of inputs of current state. The current state is stored in a set of 'n' flip-flops and has '2n' distinct states.

Next state = F (Current state, input)
Output = G (Current state, input)

Such machine structures are called as "MEALY MECHINE"

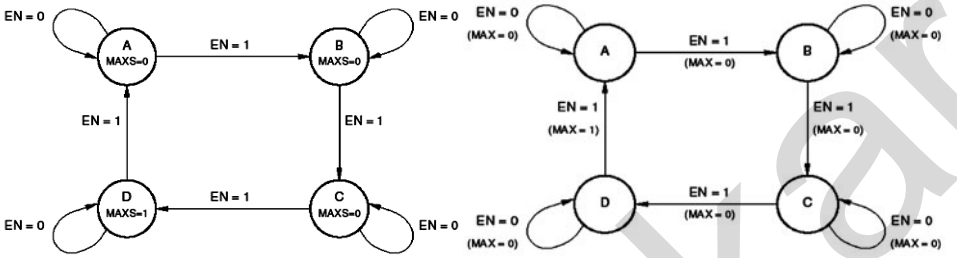
**Moore Machine**




A sequential circuit whose output depends only on the current state is called as " MOORE MACHINE "

Output = G [Current State]

Many state machines are called as Mealy machines because they have one or more Mealy type outputs that depend on inputs as well as state. Many of the same machines also have one or more Moore-type outputs that depend only on state. In the Moore state machine the transition is dependent on the state i.e. the reason why we find only one variable on the transition arrow.




❑ ❑ ❑ ❑ ❑