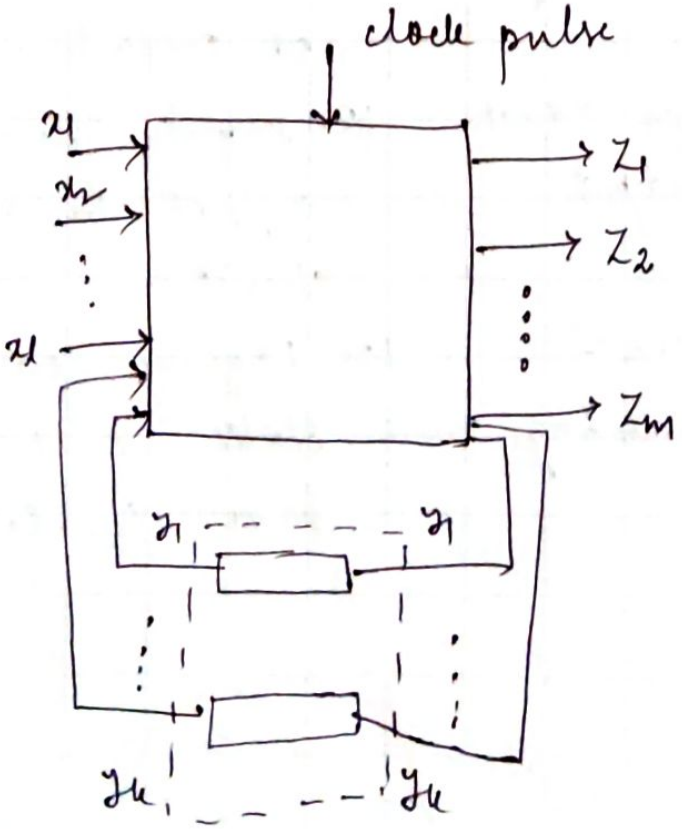
Unit-4

FINITE STATE MACHINES

A finite state machine is an abstract model describing the Synchronous sequential machine.

Sequence of events that occur at discrete instants describes the behaviour of FSM.

In practice, it is impossible to implement machines which have infinite storage capabilities. Machines whose past histories can affect their future behaviour only in a finite number of ways. (are considered) These are called finite state machines, that is, machines with a fixed number of states.



Capabilities and limitations of FSM:

1. Periodic sequence of finite states
2. No infinite sequence.
3. Limited Memory

FSM are of two types. They differ in the way of output is generated.

① Mealy type model.
② Moore type model.

Moore machine

(1) Its o/p is a function of present state only

(2) Input changes do not affect the output

(3) Requires more no: of states for implementing same function

Mealy machine.

(1) Its o/p is a functio of present state as we as present input

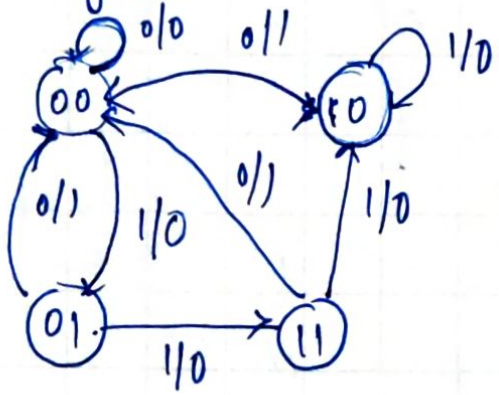
(2) Input changes ma affect the o/p of the circuit

(3) Requires less no: of states for implementing same function.

MEALY MODEL :

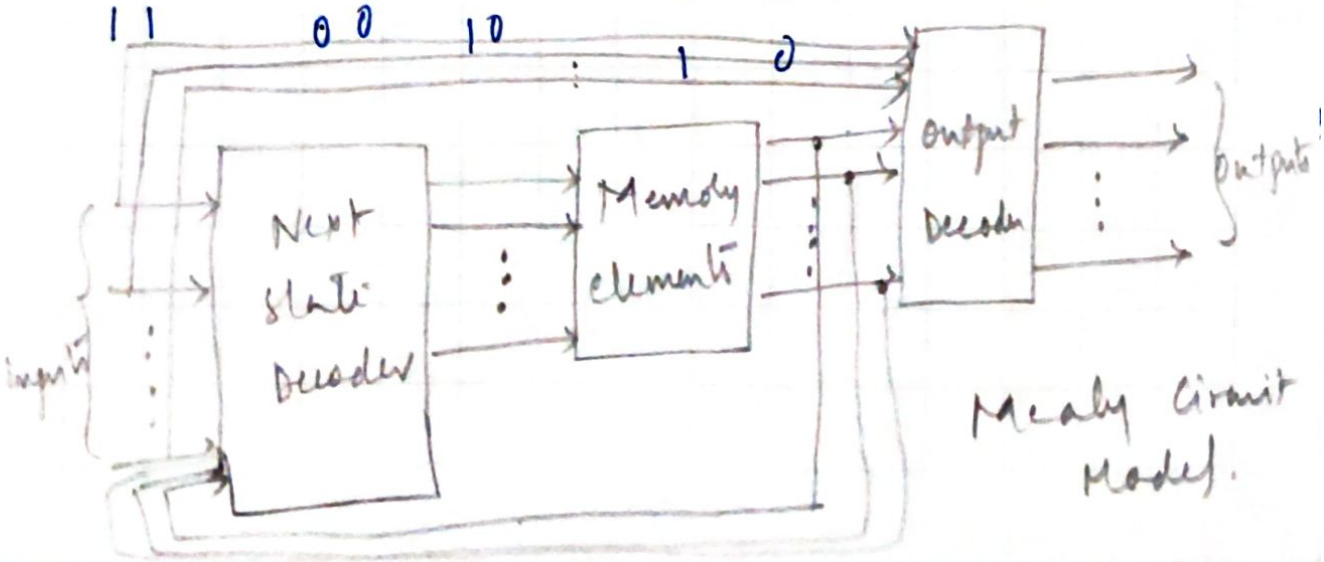
o/p of sequential circuit depends on both present state of FFs & on inputs such sequential circuit is refered as Mealy circuit or Mealy machine.

State diagram



State table

PS	NS		o/p	
	x=0	x=1	x=0	x=1
y_1 y_2	Y_1 Y_2	Y_1 Y_2	Z_0	Z_1
0 0	0 0	0 1	0	0
0 1	0 0	1 1	1	0
1 0	0 0	1 0	1	0
1 1	0 0	1 0	1	0

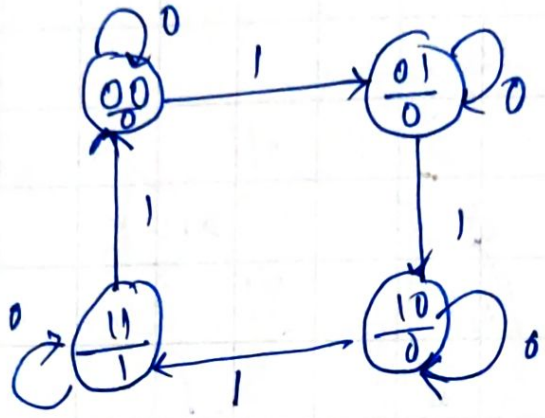


Mealy Circuit Model.

Moore Model:

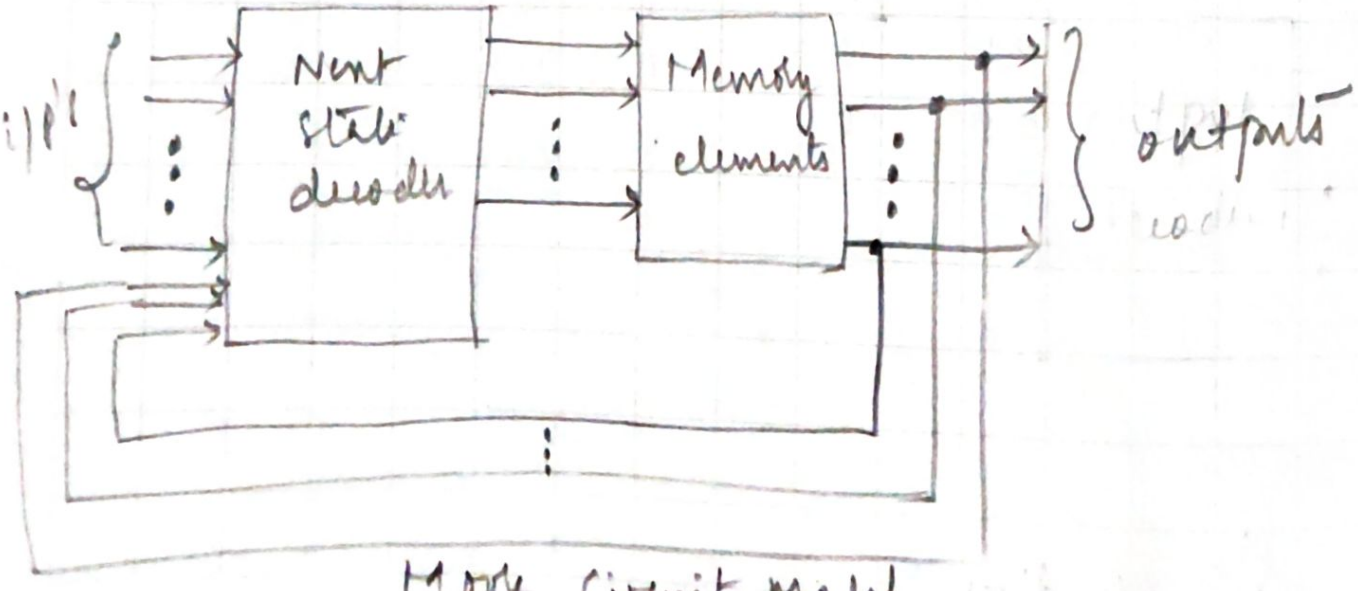
When o/p of sequential circuit depend only on present state of flip-flop, the sequential circuit is referred as the moore circuit or moore machine.

State diagram



State table

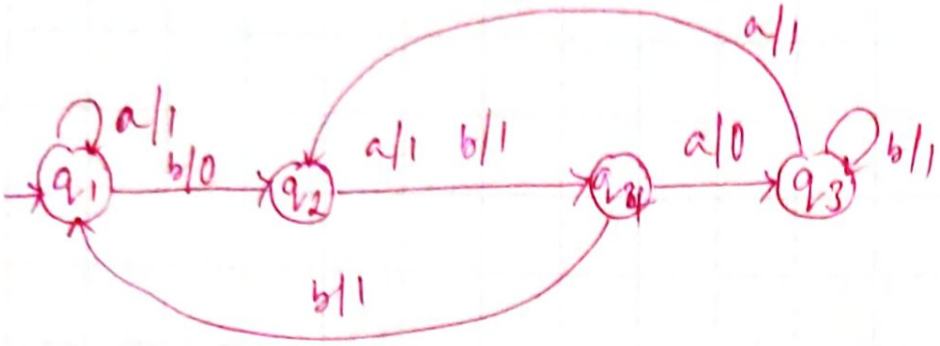
PS	NS		o/p
	x=0	x=1	
y_1 y_2	Y_1Y_2	Y_1Y_2	z
0 0	0 0	0 1	0
0 1	0 1	1 0	0
1 0	1 0	1 1	0
1 1	1 1	0 0	1



Moore Circuit Model

Mealy to Moore Conversion :-

Convert the following mealy m/c to moore m/c



Transition table for Mealy m/c

Step 1:

Present state	Next state			
	a		b	
	state	o/p	state	o/p
q_1	q_1	1	q_2	0
q_2	q_4	1	q_4	1
q_3	q_2	1	q_3	1
q_4	q_3	0	q_1	1

$q_2 \begin{cases} q_{20} - 0 \\ q_{21} - 1 \end{cases}$

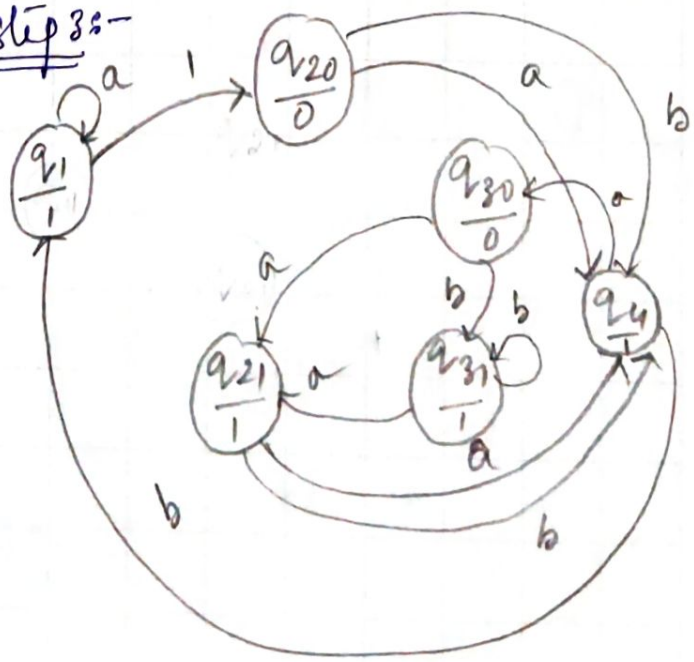
$q_3 \begin{cases} q_{30} - 0 \\ q_{31} - 1 \end{cases}$

Step 2:

Moore m/c

ps	a	b	o/p
q_1	q_1	q_{20}	1
q_{20}	q_4	q_4	0
q_{21}	q_4	q_4	1
q_{30}	q_{21}	q_{31}	0
q_{31}	q_{20}	q_{31}	1
q_4	q_{30}	q_1	1

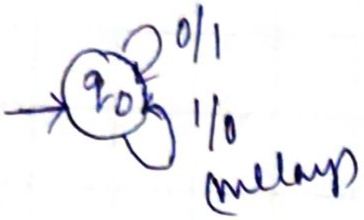
Step 3:-



Moore

Mealy to Moore Conversion

Eg: Draw the mealy m/c for doing 1's complement and convert it to its equivalent moore m/c.



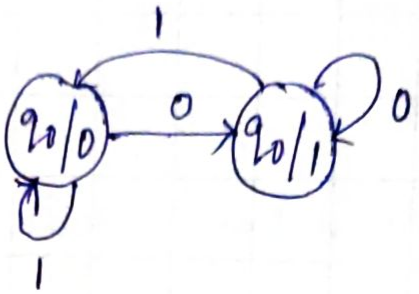
(mealy)

$Q = \{q_0\}$

$\Sigma/p = \{0,1\}$ & $\Delta/t = \{0,1\}$

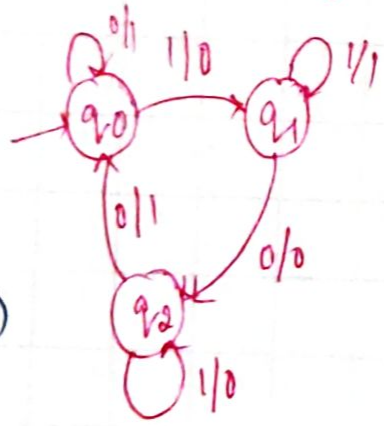
$q_0 \xrightarrow{1} q_0$

$q_0 \xrightarrow{0} q_0$

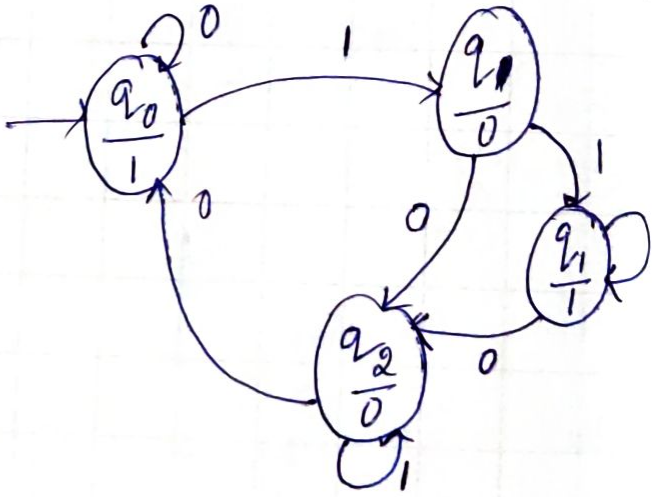


(moore)

Convert the given mealy m/c to its equivalent moore m/c



(mealy)



(moore machine)

q_0

$q_0 \xrightarrow{0} q_0$

$q_2 \xrightarrow{1} q_0$ ✓

q_1

$q_0 \xrightarrow{0} q_1$

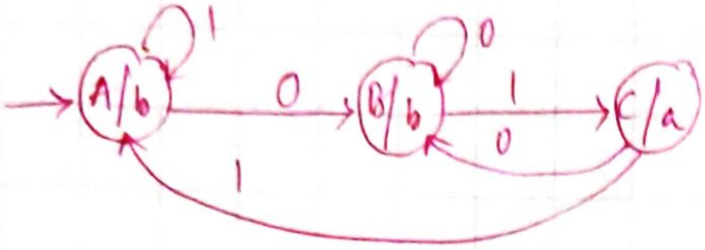
$q_1 \xrightarrow{1} q_1$

q_2

$q_1 \xrightarrow{0} q_2$

$q_2 \xrightarrow{0} q_0$ ✓

Moore to Mealy Conversion:



Moore

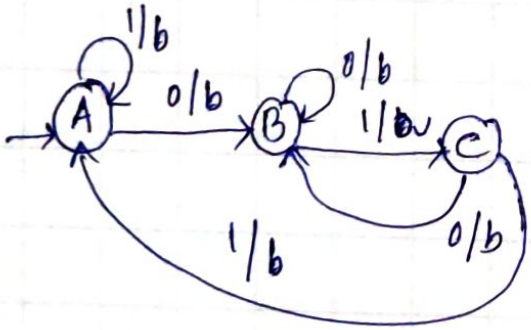
Step 1:-
Transition table

PS	NS		o/p
	x=0	x=1	
A	B	A	b
B	B	C	b
C	B	A	a

Step 2:- Mealy transition table

PS	NS			
	x=0	o/p	x=1	o/p
A	B	b	A	b
B	B	b	C	a
C	B	b	A	b

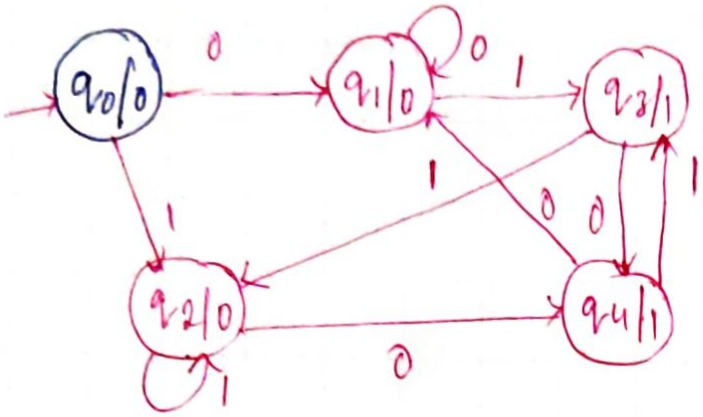
Step 3:



Mealy state diagram

convert Moore to Mealy

~~Finite State Machine~~



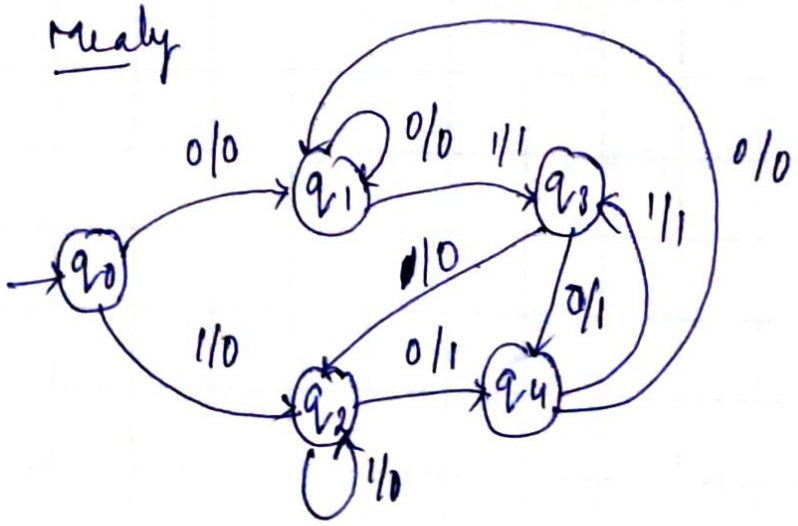
(1)

State	0	1	O/P
q_0	q_1	q_2	0
q_1	q_1	q_3	0
q_2	q_4	q_2	0
q_3	q_4	q_2	1
q_4	q_1	q_3	1

(2)

State	NS x=0	O/P	x=1	O/P
q_0	q_1	0	q_2	0
q_1	q_1	0	q_3	1
q_2	q_4	1	q_2	0
q_3	q_4	1	q_2	0
q_4	q_1	0	q_3	1

(3) Mealy



Minimization of completely specified sequential machines Using partition Technique

Steps

PS	NS, Z	
	X=0	X=1
A	C, 0	F, 0
B	D, 1	F, 0
C	E, 0	B, 0
D	B, 1	E, 0
E	D, 0	B, 0
F	D, 1	B, 0

0, 0
1, 0
0, 0
1, 0
0, 0
1, 0

Step 1: partition the states into subsets such that all states in the same subset are equivalent.

$P_1 = (A, C, E)(B, D, F)$

Step 2: (2-equivalent)

0-Successor of (A, C, E) → (C, E, D) different blocks

0-Successor of (B, D, F) → (D, B, D) same

1-Successor of (A, C, E) → (F, B, B) same

1-Successor of (B, D, F) → (F, E, B) different

$P_2 = (A, C)(E)(B, F)(D)$

Step 3: (3-equivalent)

$$P_2 = (A, C)(E)(B, F)(D)$$

Step 3: 0-Successor of (A,C) → (C, E) different blo
1- Successor of (A,C) → (F, B) same
0- Successor of (B, F) → (D, D) same
1- Successor of (B, F) → (F, B) same.

$$P_3 = (A)(C)(E)(B, F)(D)$$

Further, partitioning of states is not possib
0 & 1 successors of (B, F) are same block in P

Minimized state table

PS	NS, Z	
	X=0	X=1
A	C, 0	B, 0
B	D, 1	B, 0
C	E, 0	B, 1
D	B, 1	E, 0
E	B, 0	B, 0

Q: find the equivalence partition and a corresponding reduced machine in standard form

PS	NS, Z	
	x=0	x=1
A	B, 1	H, 1
B	F, 1	D, 1
C	D, 0	E, 1
D	C, 0	F, 1
E	D, 1	C, 1
F	C, 1	C, 1
G	C, 1	D, 1
H	C, 0	A, 1

① $P_1 = (A, B, E, F, G)(C, D, H)$

② 0- Successor of A, B, E, F, G → (B, F, D, C, C) different
1- Successor of A, B, E, F, G → (H, D, C, C, D) ~~different~~ same
0- Successor of C, D, H → (D, C, C) same
1- Successor of C, D, H → (E, F, A) ~~different~~ same

$P_2 = (A, B)(E, F, G)(C, D, H)$

③ ~~One Successor~~ A, B 0- Successor → (B, F) different
1- Successor → (H, D) same

E, F, G 0 → (D, C, C) same
1 → (C, C, D) same

C, D, H 0 → (D, C, C) same
1 → (E, F, A) different

$P_3 = (A)(B)(E,F,G)(C,D)(H)$

Reduced state table

PS	NS, Z	
	X=0	X=1
A	B, 1	H, 1
B	E, 1	C, 1
C	C, 0	E, 1
E	C, 1	C, 1
H	C, 0	A, 1

Distinguishability of states A & B:-

X=0

A → B, 1 ; B → F, 1 — o/p's same

B → F, 1 ; F → C, 1 — o/p's same

F → C, 1 ; C → D, 0 — different

X=1

A → H, 1 ; B → D, 1 — same

H → A, 1 ; D → F, 1 — same

A → H, 1 ; F → C, 1 — same

H → A, 1 ; C → E, 1 — same

Minimum length that distinguishes is 3.

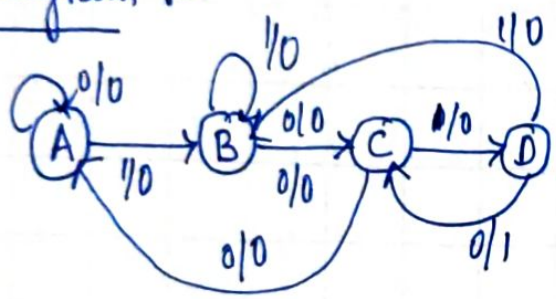
Sequence Detector

Mealy Type Model:

A sequence Detector is a sequential machine which produces an output 1 every time the desired sequence is detected and o/p 0 at all other times.

Sequence 1010 Overlapping is permitted.

State diagram :-



State table

PS	NS, Z	
	$x_2 0$	$x_2 1$
A	$A, 0$	$B, 0$
B	$C, 0$	$B, 0$
C	$A, 0$	$D, 0$
D	$C, 1$	$B, 0$

The machine is already in reduced standard form state table. So no need to do for

State Assignment and transition and output table

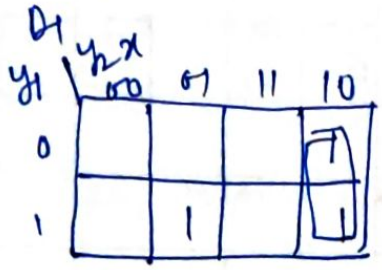
Transition o/p table

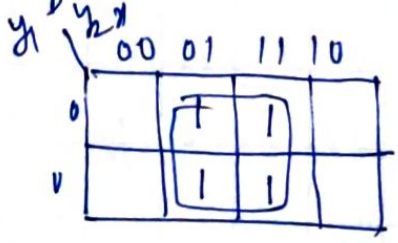
PS $(y_1 y_2)$	NS $(Y_1 Y_2)$		o/p (z)	
	$x_2 0$	$x_2 1$	$x_2 0$	$x_2 1$
A → 00	00	01	0	0
B → 01	10	01	0	0
C → 10	00	11	0	0
D → 11	10	01	1	0

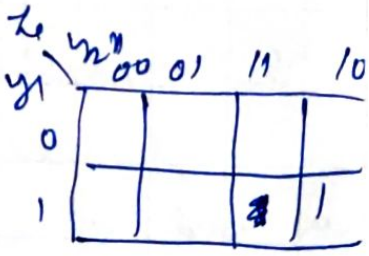
D-ffs are choosen as memory elements.

Excitation table

PS		I/P	NS		I/P to FFs		O/P
y_1	y_2	x	Y_1	Y_2	D_1	D_2	Z
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	1	0	1	0	0
0	1	1	0	1	0	1	0
1	0	0	0	0	0	0	0
1	0	1	1	1	1	1	0
1	1	0	1	0	1	0	1
1	1	1	0	1	0	1	0

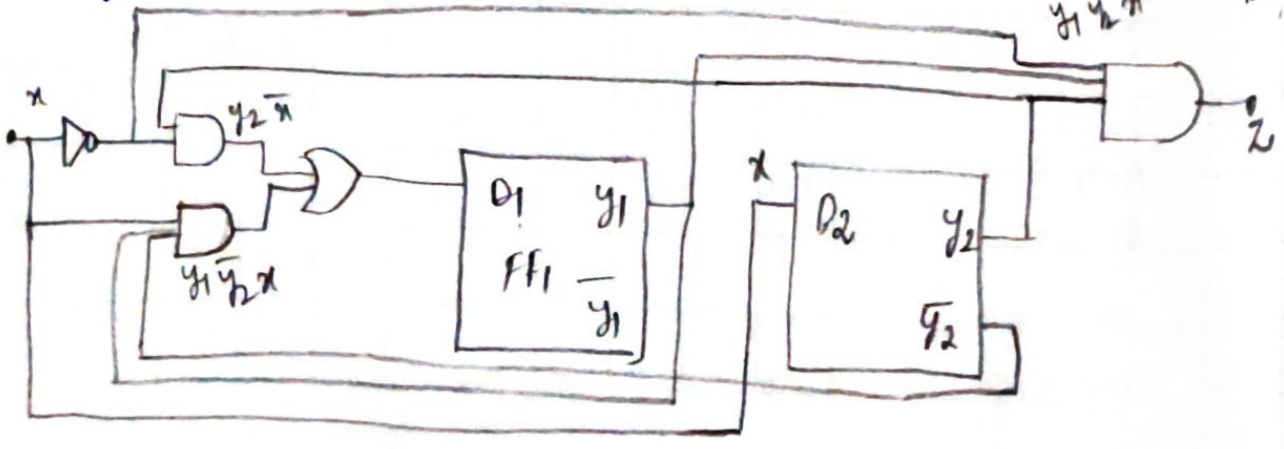






$D_1 = y_1\bar{y}_2x + y_2\bar{x}$ $\quad D_2 = x$ $\quad Z = y_1y_2\bar{x}$.

Implementation Logic diagram of sequence (1010) detector using

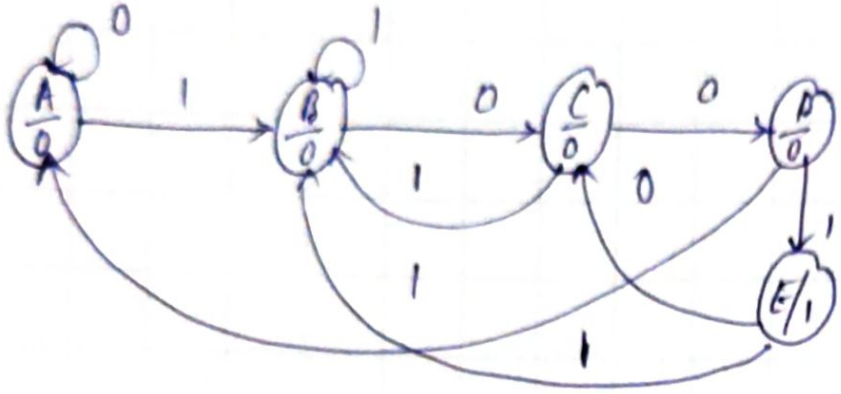


Mode type Model :-

Same procedure as Mealy model.

State diagram

1001



State table

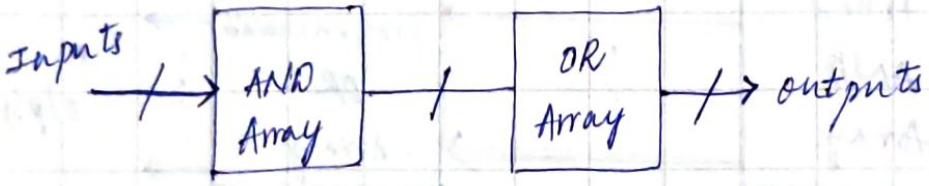
ps	NS		zt
	x=0	x=1	
A	A	B	0
B	C	B	0
C	D	B	0
D	A	E	0
E	C	B	1

Programmable Logic Devices

An IC that contains large number of gates, flip-flops, etc. that can be configured by the user to perform different functions is called Programmable Logic Device (PLD).

The internal logic gates and/or connections of PLDs can be changed/configured by a Programming Process. (*Here, Programming refers to Hardware not Software).

PLDs are typically built with an array of AND gates (AND-array) and an array of OR gates (OR-array).



There are three fundamental types of standard PLDs: PROM, PAL and PLA.

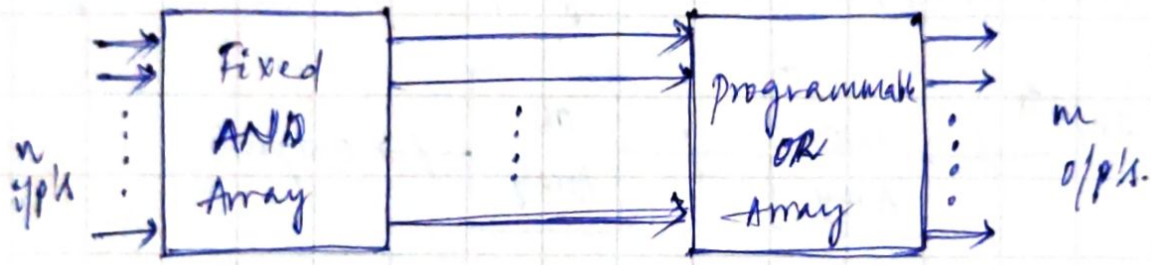
- Programmable Read only Memory (PROM)
- Programmable Array Logic (PAL)
- Programmable Logic Array (PLA)

The Process of entering information into these devices is known as Programming. Basically, users can program these devices or IC's electrically in order to implement the boolean functions based on the requirement.

Programmable Read only Memory (PROM)

Read only Memory (ROM) is a memory device, which stores binary information Permanently. i.e; we cannot change stored information by any means later. If the ROM has programmable feature, then it is called PROM (Programmable Read only Memory). The user has flexibility to program the binary information electrically once by using PROM programmer.

* PROM is a PLD that has fixed AND Array & Programmable OR Array



Block Diagram

Here, the i/p's of AND gates are not of programmable type. We have to generate 2^n product terms by using 2^n AND gates. We can implement these product terms by using $n \times 2^n$ decoder. Decoder generates '2^n' minterms.

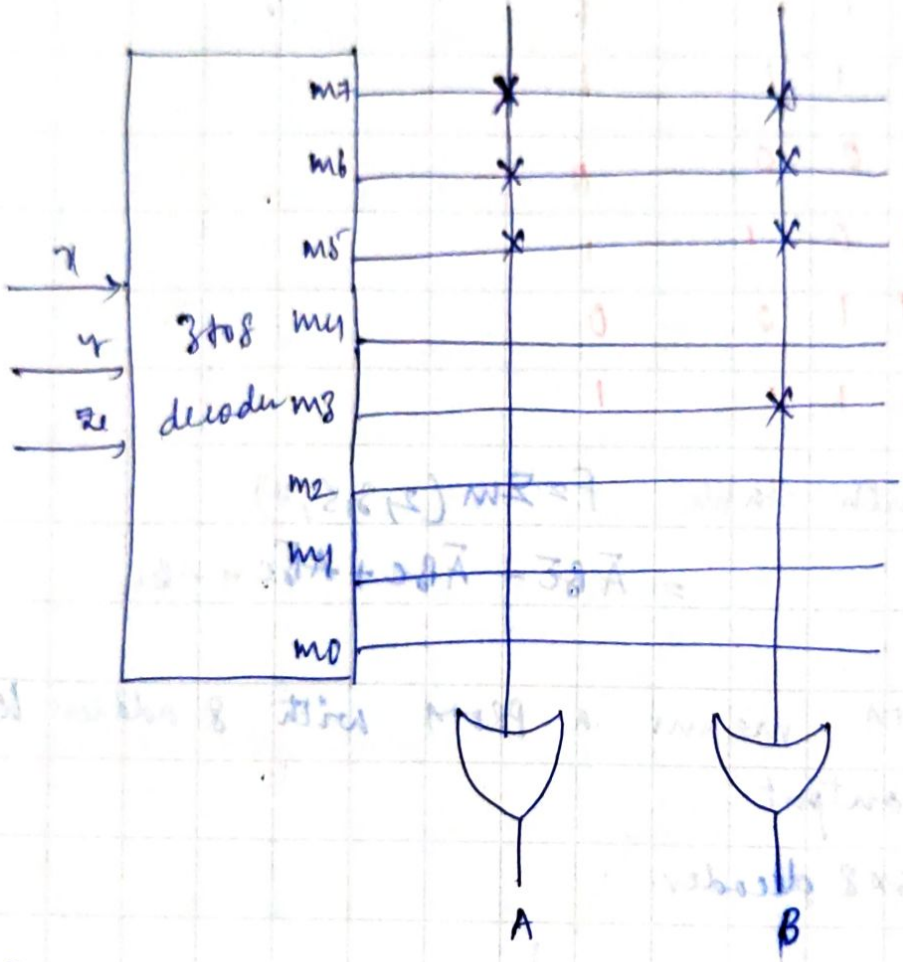
Here, the inputs of OR gates are programmable. We can program any no: of required product terms, since all the outputs of AND gates are applied as inputs to each OR gate. Therefore the outputs of PROM will be in the form of sum of minterms.

Example :-

Implement the following Boolean function using PROM. $A(x, y, z) = \Sigma m(5, 6, 7)$

$B(x, y, z) = \Sigma m(3, 5, 6, 7)$

Each function is have three variables (x, y, z). So, we require 3 to 8 decoder and two Programmable OR gates



3 to 8 decoder generates eight minterms. The two Programmable OR gates have the access of all these minterms. But only the required minterms are programmed in order to produce the respective Boolean function by OR gate. The symbol 'X' is used for programmable connections.

Show how an 8×1 PROM can be programmed to implement the logic function whose truth table is as shown.

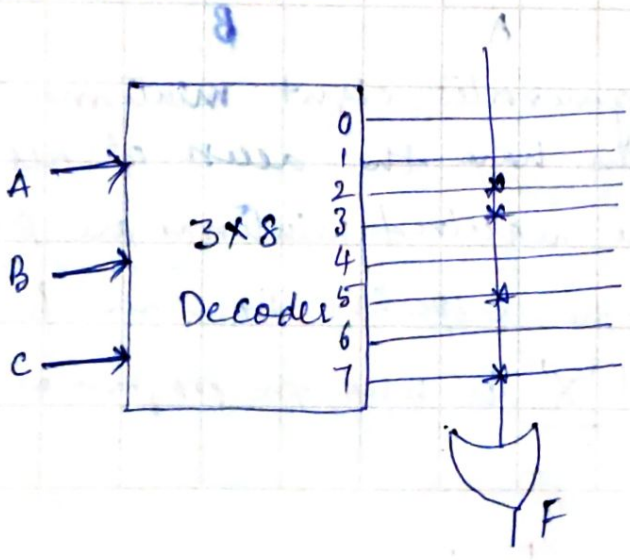
Inputs			Output
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

From truth table $F = \Sigma m(2,3,5,7)$

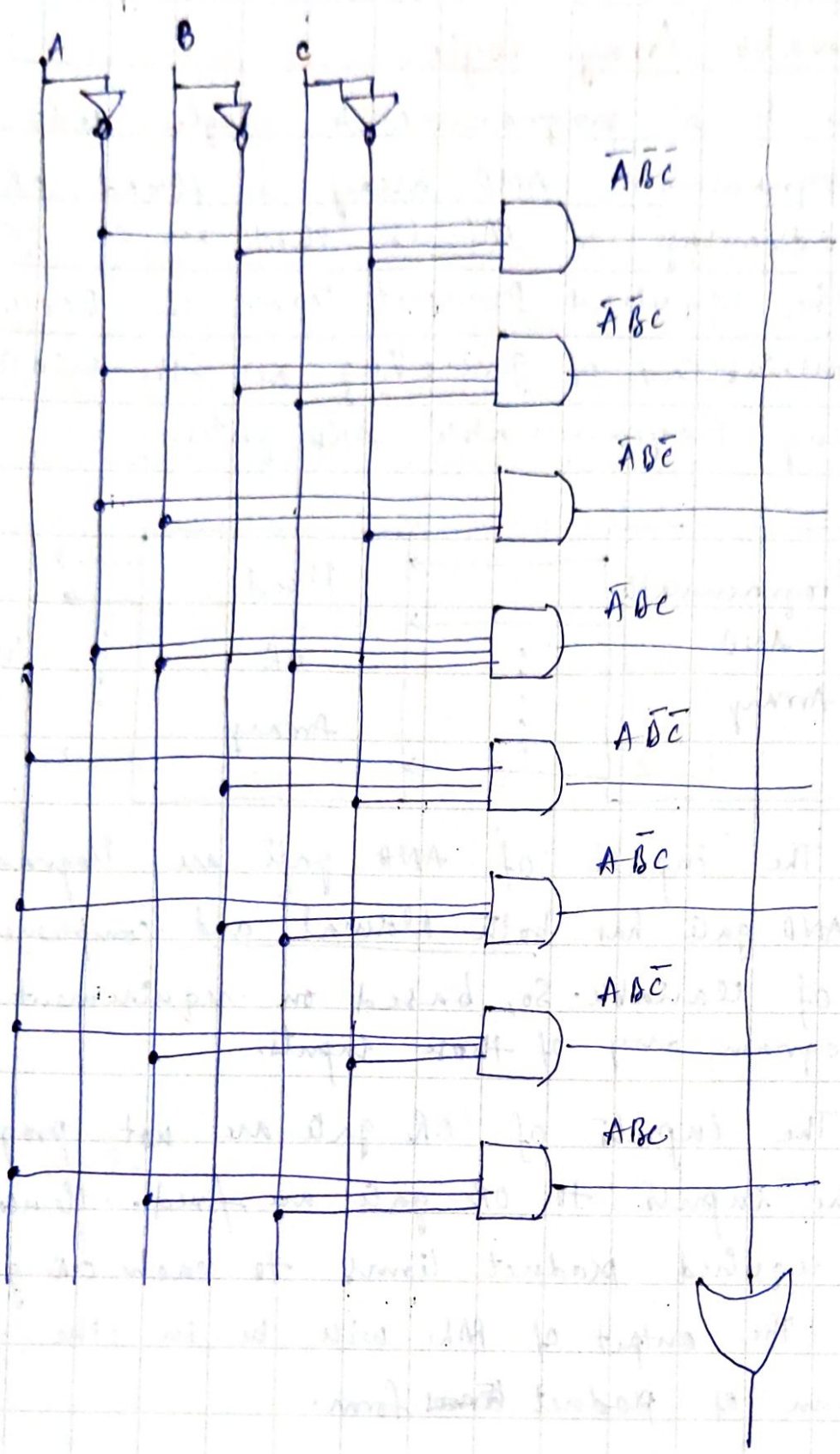
$= \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + ABC$

8×1 PROM means a PROM with 8 address lines and one output.

So 3×8 decoder.



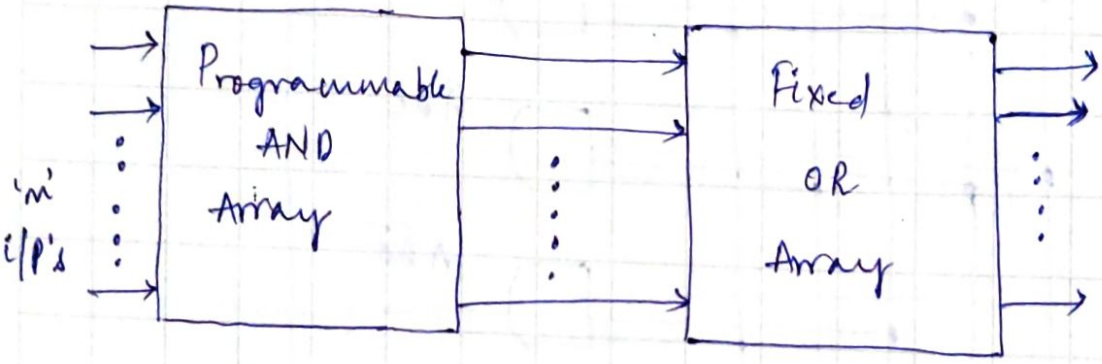
Block diagram



PROM implementation

Programmable Array Logic :-

PAL is a programmable logic device that has programmable AND array & fixed OR Array. The advantage of PAL is that we can generate only the required product terms of Boolean functions instead of generating all the minterms by using Programmable AND gates.



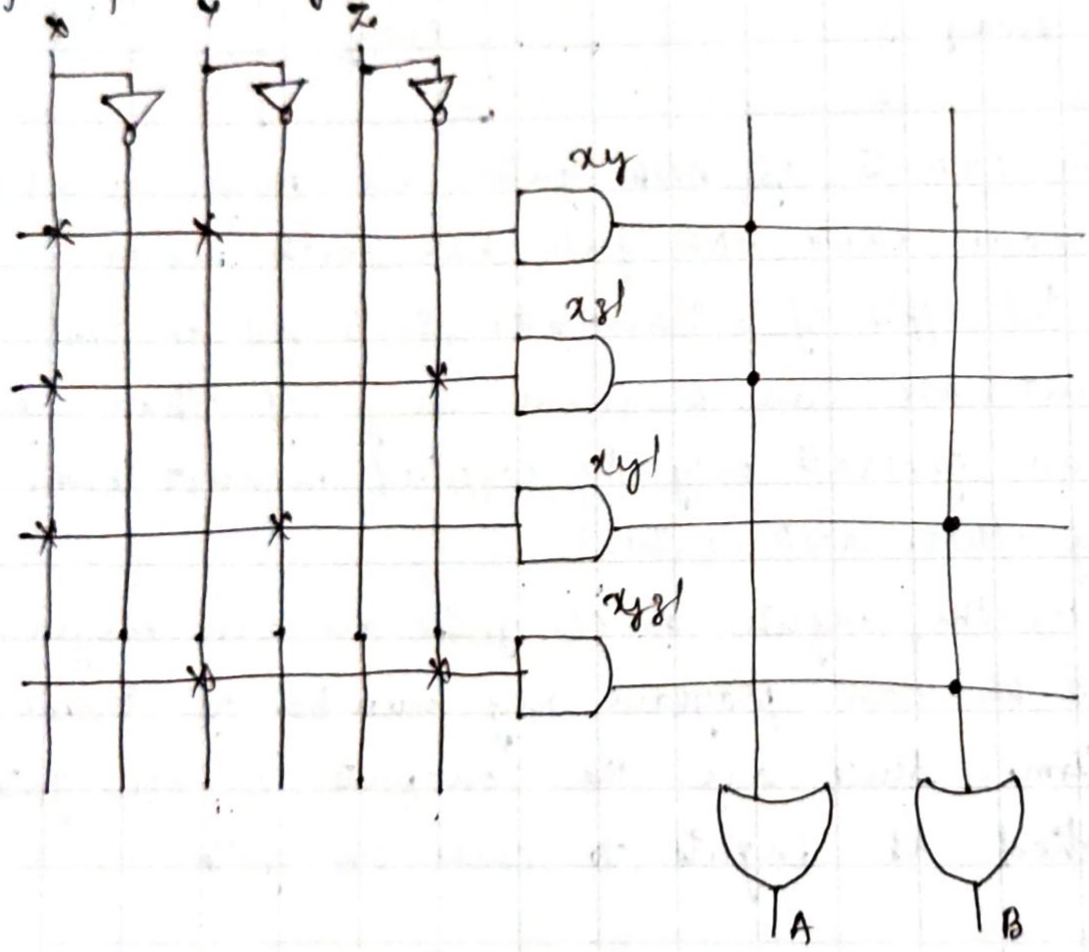
The inputs of AND gate are Programmable. Each AND gate has both Normal and complement inputs of variable. So, based on requirement we can program any of those inputs.

The inputs of OR gate are not programmable So, the inputs to OR gate are fixed. Hence, apply those required product terms to each OR gate as inputs. The output of PAL will be in the form of Sum of product form.

Let us implement the following Boolean function using PALs:

$A = xy + xz'$ $B = xy' + yz'$

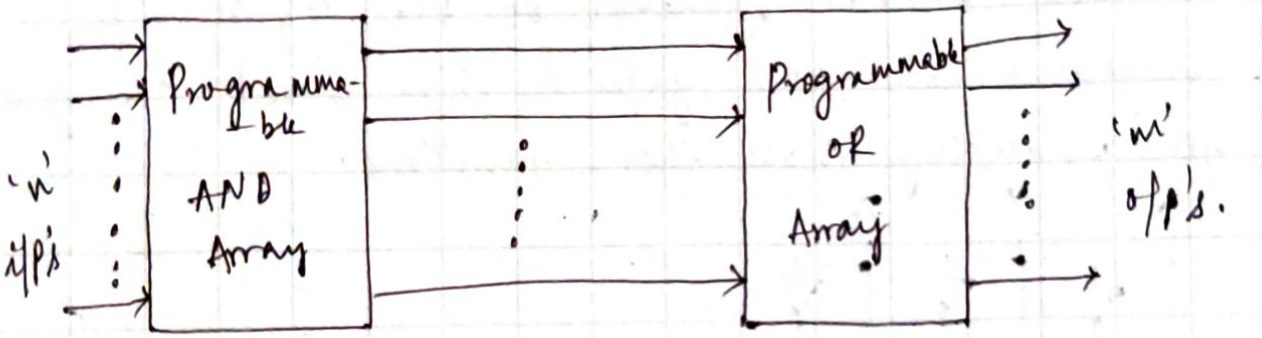
The given two functions are in the sum of product form. There are two product terms present in each Boolean function. So, we require four programmable AND gates & two fixed OR gates for producing those two functions.



The Programmable AND gates have the access of both normal and complemented inputs of variables.

Programmable logic Array (PLA):-

PLA is a Programmable logic device that has both programmable AND array & Programmable OR array. Hence, it is the most flexible PLD. The block diagram of PLA is



Here, the inputs of AND gates are programmable. That means each AND gate has both normal & complemented i/p's of variables. So, based on the requirement, we can program any of those inputs. So, we can generate only the required product terms by using these AND gates.

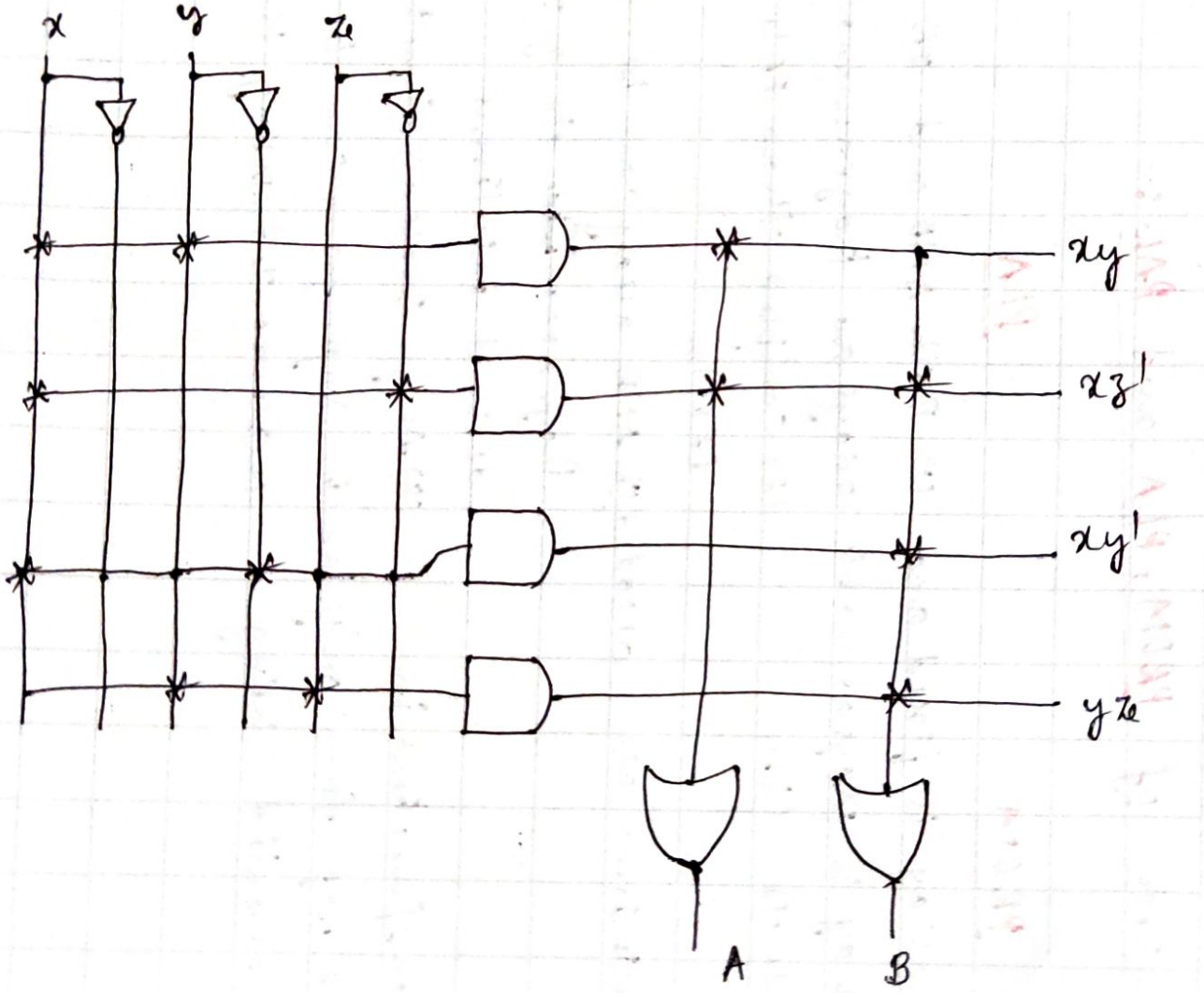
Here, the inputs of OR gates are also programmable. So we can program any number of required product terms, since all the outputs of AND gates are applied as inputs to each OR gate.

Let us implement the following Boolean functions using PLA.

$A = xy + xz'$ $B = xy' + yz + xz'$

The given two functions are in SOP form. The product term xz' is common in each function.

We require 4 programmable AND gates
2 programmable OR gates



Implement the following Boolean function using PAL with four inputs and 3-wide AND-OR structure. Also write PAL programming table

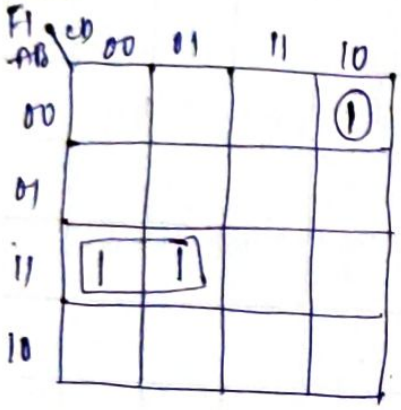
$F_1(A,B,C,D) = \Sigma m(2,12,13)$

$F_2(A,B,C,D) = \Sigma m(7,8,9,10,11,12,13,14,15)$

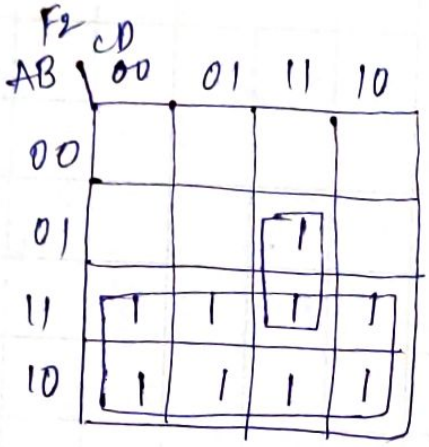
$F_3(A,B,C,D) = \Sigma m(0,2,3,4,5,6,7,8,10,11,15)$

$F_4(A,B,C,D) = \Sigma m(1,2,8,12,13)$

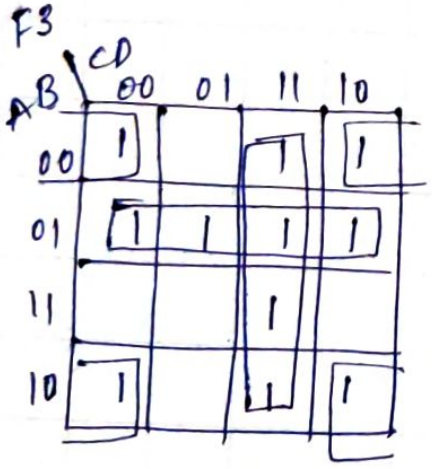
Sol:



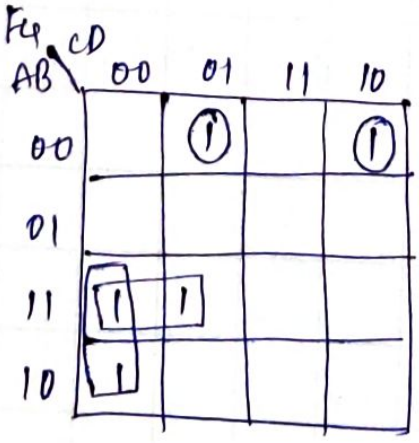
$F_1 = AB\bar{C} + \bar{A}\bar{B}C\bar{D}$



$F_2 = A + BCD$



$F_3 = \bar{B}\bar{D} + \bar{A}B + CD$



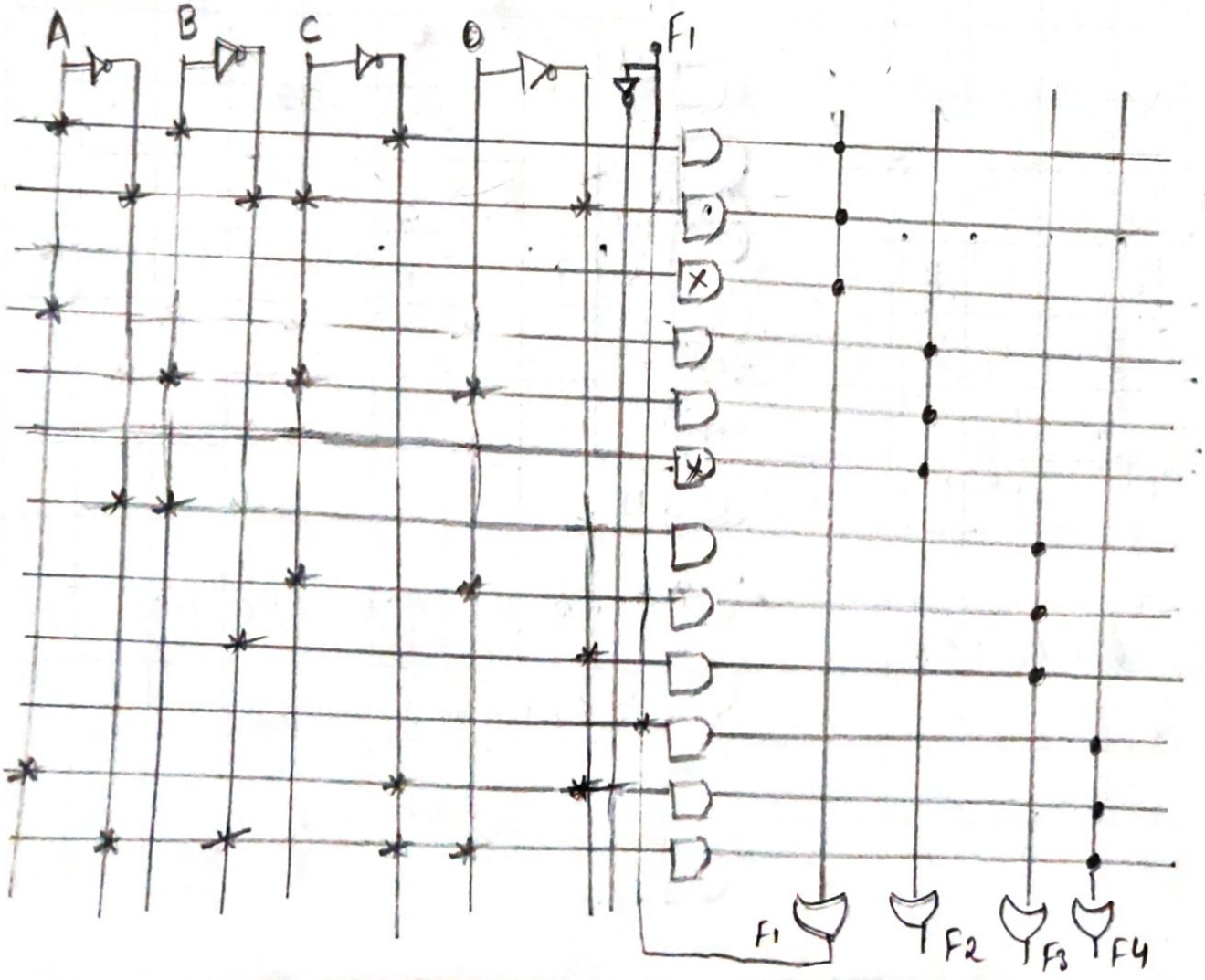
$F_4 = \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}C\bar{D} + AB\bar{C} + A\bar{C}\bar{D}$

$= F_1 + \bar{A}\bar{B}\bar{C}D + A\bar{C}\bar{D}$

Programming table :

product term	AND inputs A	B	C	D	Outputs
1	1	1	0	–	$F_1 = AB\bar{C} + \bar{A}\bar{B}C\bar{D}$
2	0	0	1	0	
3	–	–	–	–	
4	1	–	–	–	$F_2 = A + BCD$
5	–	1	1	1	
6	–	–	–	–	
7	0	1	–	–	$F_3 = \bar{A}B + CD + \bar{B}\bar{D}$
8	–	–	1	1	
9	–	0	–	0	
10	–	–	–	1	$F_4 = F_1 + A\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D$
11	1	–	0	0	
12	0	0	0	1	

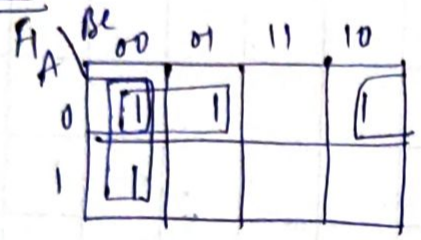
Realization



Implement the following two Boolean function with a PLA :. $F_1(A,B,C) = \Sigma m(0,1,2,4)$

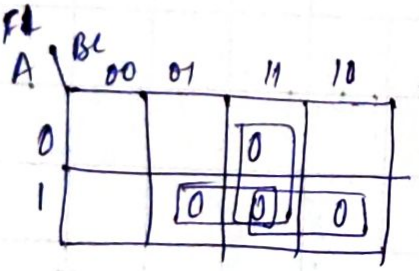
$F_2(A,B,C) = \Sigma m(0,5,6,7)$

Sol:

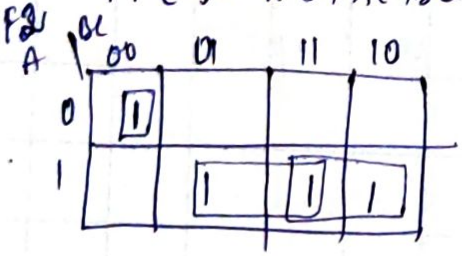


$F_1(T) = \bar{A}\bar{C} + \bar{B}\bar{C} + \bar{A}\bar{B}$

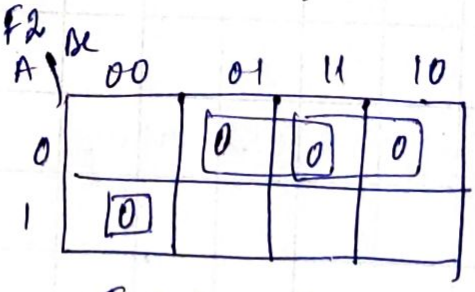
* for finding minimal in true form, consider the 1s on the map.
* for finding minimal in complement form, consider the 0s on the map.



$F_1(C) = AB + AC + BC$



$F_2(T) = \bar{A}\bar{B}\bar{C} + AC + AB$

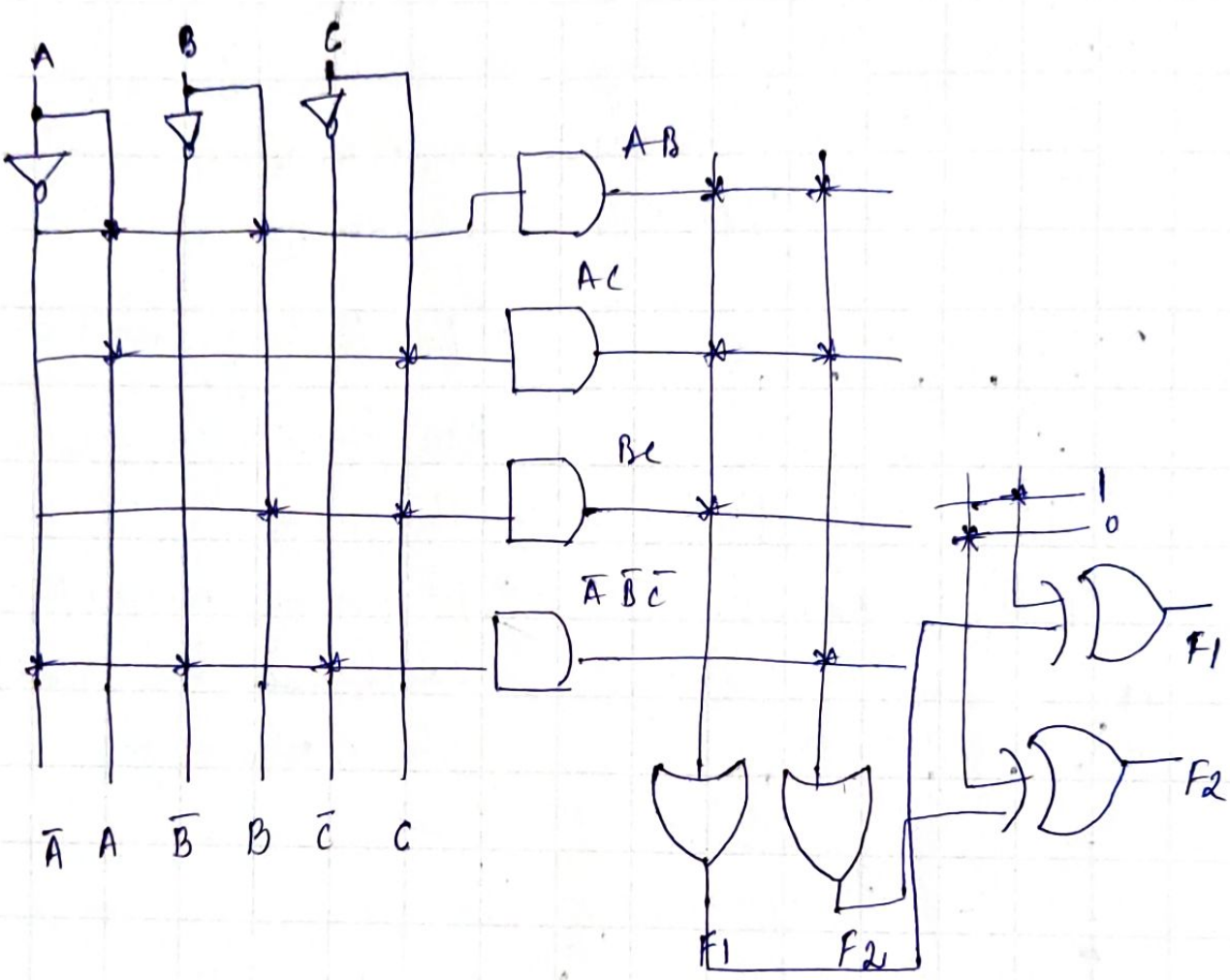


$F_2(C) = A\bar{B}\bar{C} + \bar{A}B + \bar{A}C$

[AB, AC are present in $F_1(C)$ & $F_2(T)$]

PLA Programming table

product item		Inputs A B C	outputs (C) F_1	outputs (T) F_2
1	AB	1 1 –	1	1
2	AC	1 – 1	1	1
3	BC	– 1 1	1	–
4	$\bar{A}\bar{B}\bar{C}$	0 0 0	–	1



programming and fuse map.