UNIT-IV

INTRODUCTION OF PLD'S

In previous chapter we discussed about different IC's like Binary adder, Look a head Carry generator, 4-bit comparator, Decoders, multiplexer's etc..

the above mention IC's are used to Perform digital operations & other functions. These IC's having their fixed functions so these IC's are known as "fixed function IC's".

To design digital circuits we are using fixed function IC's.

There are 2 more approches for the design of digital circuits.

1. use of application specific integrated circuits (ASIC's)
2. use of programmable logic device's (PLDs)

In ASIC a single IC is designed and manufactured to implement the entire circuits.

In PLD's are used to implement logic functions.

→ A programmable logic device is an IC that is user configurable and is capable of implementing logic functions.

→ It allows the designer to customize it for any specific application. i.e it is programmed by the user to perform a function required for his application.

→ A PLD contains a large no. of gates, flip-flops, and registers that are interconnected on the chip.

→ PLD's can be reprogrammed in a few seconds and hence give more flexibility to experiment with designs. the reprogramming feature of PLD's also makes it possible to accept changes/modifications in the previously designed circuits

Advantages of PLD's over fixed function IC's:-

1. Low development cost
2. Less space requirement
3. Less power requirement
4. High reliability
5. Easy circuit testing
6. Easy design modification
7. High design security
8. Less design time
9. High switching speed.

PLD's have many of the advantages of ASIC as given below:-

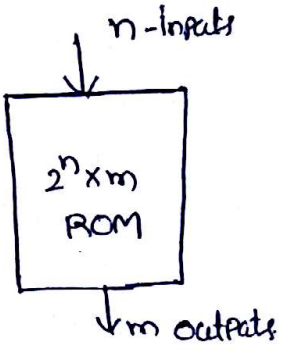
1. Higher densities.
2. Lower quantity production costs
3. Design security
4. Reduced power requirement
5. Reduced space requirement.

According to architecture, complexity and flexibility in programming PLD's are classified as

1. PROMS:- Programmable Read only memories.
2. PLA's:- Programmable Logic Array's.
3. PAL:- Programmable Array Logic.
4. FPGA:- Field Programmable Gate Array's
5. CPLD's:- Complex Programmable Logic Devices.

Programmable Read only Memory (PROM):-

PROM is a device that includes both decoder and the OR gates within a single IC package.



⇒ It consists of 'n' i/p lines and 'm' o/p lines. Each bit combination of the i/p variables is called an "address".

→ Each bit combination that comes out of the o/p lines is called a "word".

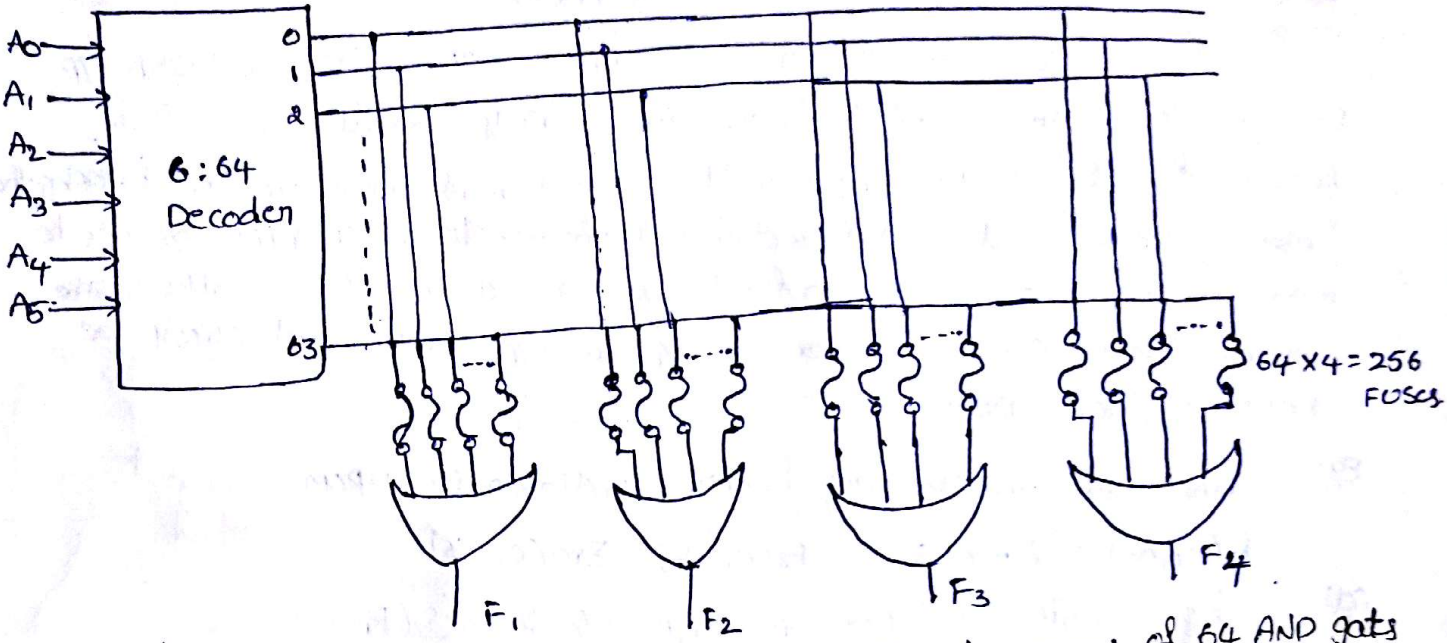
The No. of bits per word is equal to the number of o/p lines.

⇒ Let us consider 64x4 PROM. The PROM consists of 64 words of 4 bits each. This means that there are 4 o/p lines and particular word from 64 words presently available on the o/p lines is determined from the 6 i/p lines. There are only 6 i/p's in 64x4 PROM. because $2^6 = 64$. and with 6 variables

For each address i/p, there is a unique selected word, Thus, if the i/p address is 000000, word no. is '0' is selected and applied to the o/p lines. If the i/p address is 111111, word no. 63 is selected and applied to the o/p lines.

The below fig shows the internal logic construction of 64x4 PROM.

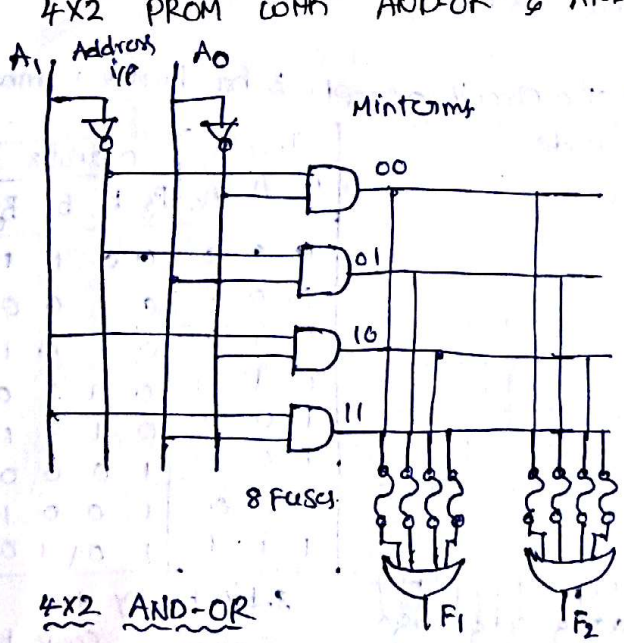
G.NAVEEN
EEE, NEC



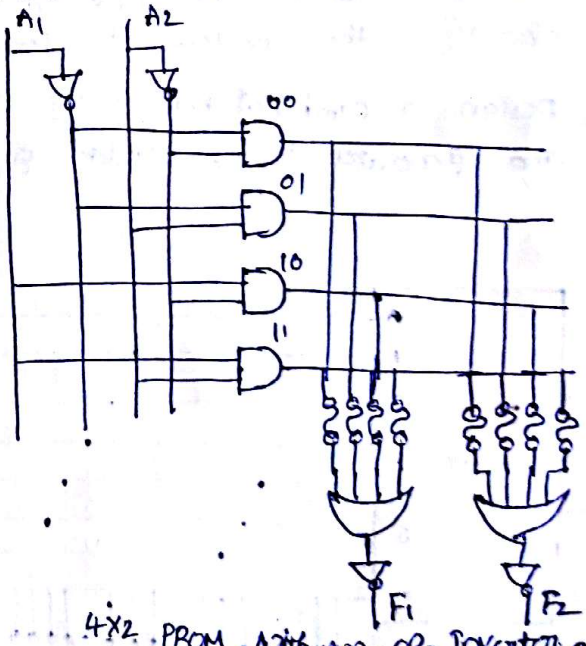
The 6 I/p variables are decoded in 64 lines by means of 64 AND gates and 6 inverters. Each o/p of the decoder represents one of the minterms of a function of 6 variables.

The 64 o/p's of the decoder are connected through fuses to each OR gate. only 4 of these fuses are shown in the diagram, but actually each OR gate has 64 i/p's and each goes through a fuse that can be blown as desired.

The PROM is a two-level implementation in sum of minterms form Let us see AND-OR & AND-OR-INVERTER implementation of PROM. fig shows the 4X2 PROM with AND-OR & AND-OR-INVERTER implementations.



4X2 AND-OR



4X2 PROM with AND-OR-Inverter gates.

Combination Logic Implementation using PROM:-

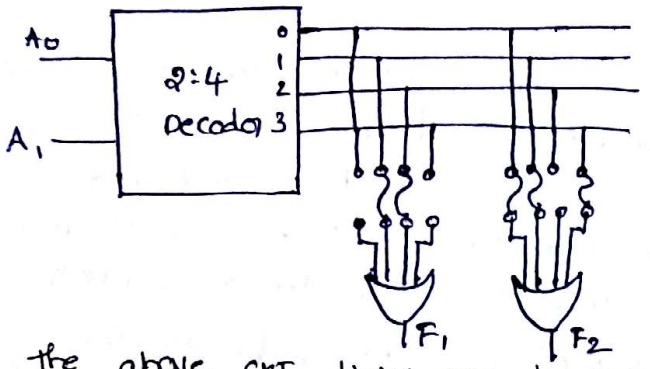
From Logic diagram of the PROM, we can realize that each o/p provides the sum of all the minterms of 'n' i/p variables. w.k.T any Boolean function can be expressed in sum of minterms form. By breaking the links of those minterms not included in the function, each PROM o/p can be made to represent the Boolean function of one of the o/p variables in the combinational circuit. For an n-i/p, m-o/p combinational circuit, we need a $2^n \times m$ PROM.

Eg:- Implement the following Boolean functions using PROM

$F_1(A_1, A_0) = \Sigma m(1,2)$ $\quad F_2(A_1, A_0) = \Sigma m(0,1,3)$

Sol:- This circuit has two i/p's (A_1, A_0) & 2 o/p's (F_1, F_2)

∴ Boolean function can be implemented by 4×2 PROM.



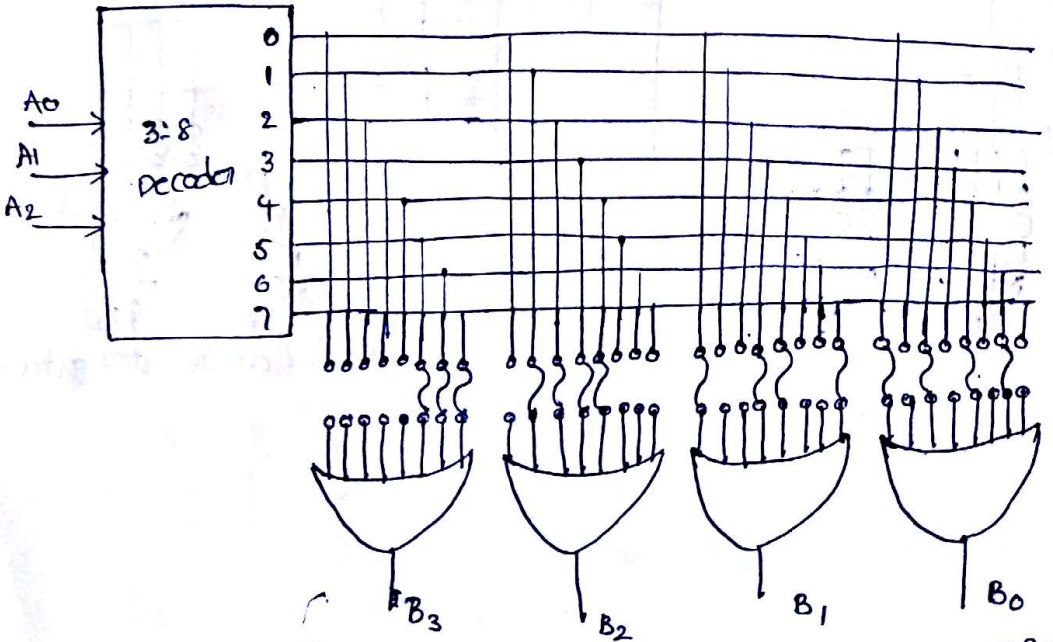
In the above CKT links are broken of those minterms which are not included in the Boolean function.

Eg:- Design a combinational using a PROM. the circuit accepts 3-bit binary number and generates its equivalent excess-3 code.

Inputs			Outputs			
A_2	A_1	A_0	B_3	B_2	B_1	B_0
0	0	0	0	0	1	1
0	0	1	0	1	0	0
0	1	0	0	1	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	1
1	0	1	1	0	0	0
1	1	0	1	0	0	1
1	1	1	1	0	1	0

3-bit binary to excess-3 convertor

Sol:-

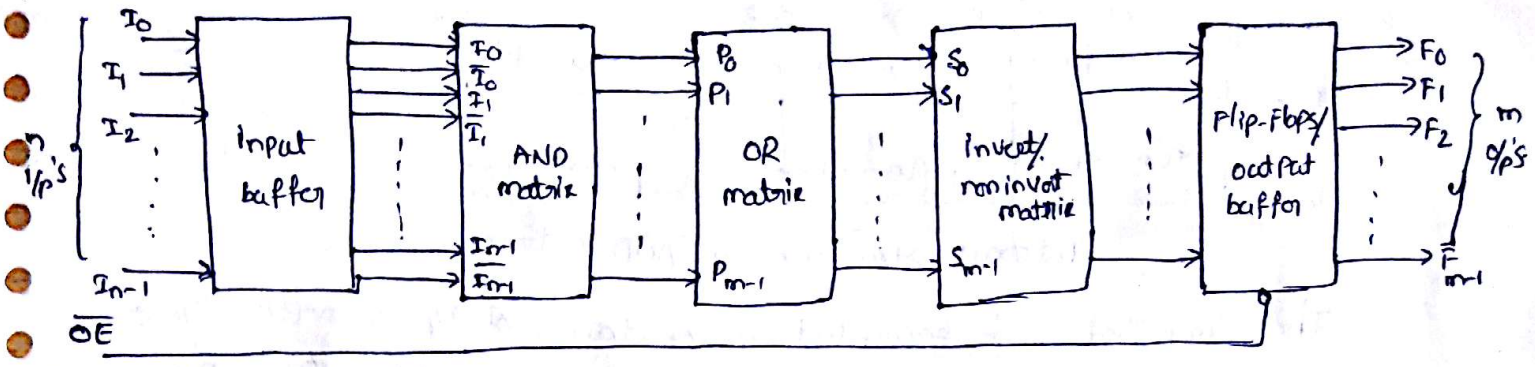


3-bit binary to excess-3 convertor using PROM

G.NAVEEN
EEE,NEC

Programmable Logic Array (PLA):—

The combinational CKT do not use all the minterms every time. occasionally they have don't care conditions. Don't care condition when implemented with a PROM becomes an address I/p that will never occur. The result is that not all the patterns available in the PROM are used, which may be considered a waste of available equipment.



It is economical to use PLA when don't care conditions is excessive.

PLA is similar to a PROM in concept; but it does not provide full decoding of the variables and does not generates all the minterms as in the PROM.

The PLA replaces decoder by group of AND gates, each of which can be programmed to generate a product term of the I/p variables

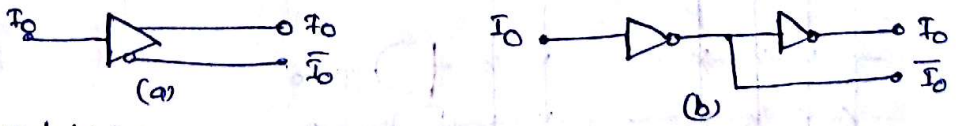
In PLA, both AND & OR gates have fuses at the I/p's, ∴ in PLA both AND & OR gates programmable.

Block diagram of PLA shows n I/p's, output buffer with m O/p's, m product terms, m sum terms, I/p & O/p buffers.

Input Buffer:—

I/p Buffer are provided in the PLA to limit loading of the sources that drive the I/p's. They also provide inverted and non-inverted form of I/p's at it's O/p's.

Two way's of representing I/p buffer for single I/p

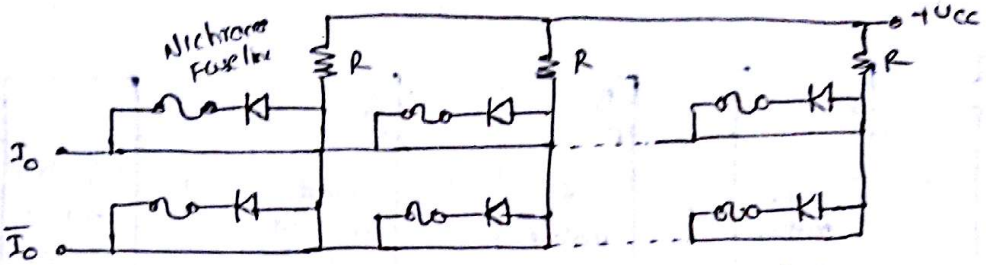


AND Matrix:—

It is used to form product terms. It has 'm' AND gates with '2n' I/p's and 'm' O/p's. one for each AND gate. Each AND gate has all the I/p variables in complemented and uncomplemented form.

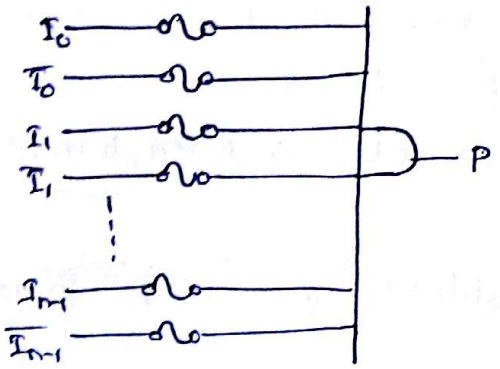
There is a 'nichrome' fuse link in series with each diode which can be burn out to disconnect particular I/p for that AND gate. Before programming all fuse links are intact and the product term for each AND gate is given by

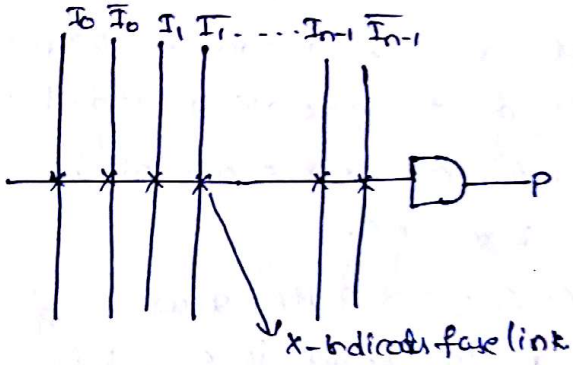
$$P = I_0 \cdot \bar{I}_0 \cdot I_1 \cdot \bar{I}_1 \cdots I_{n-1} \cdot \bar{I}_{n-1}$$



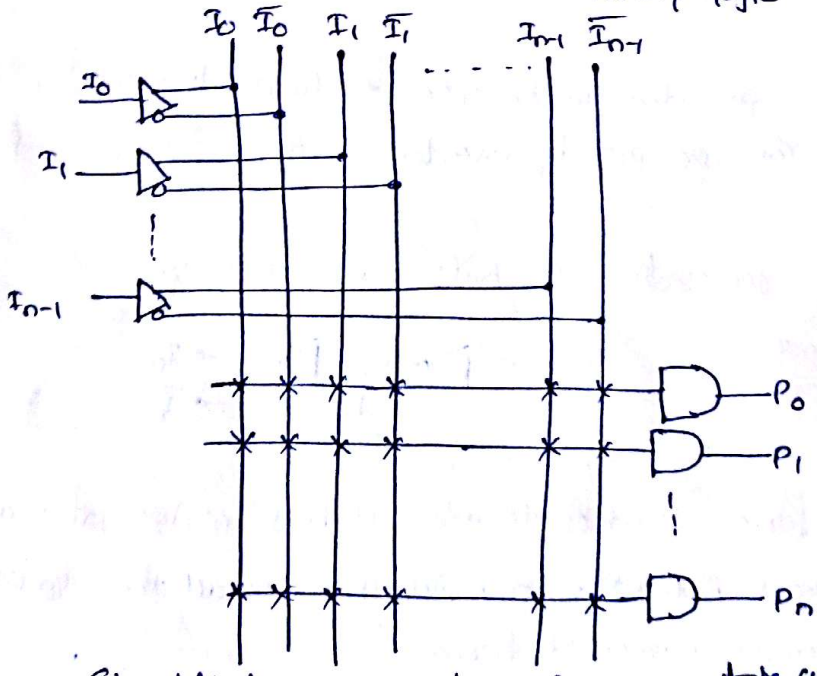
Internal structure of AND matrix

The simplified and equivalent representation of I/p connections for one AND gate. The array logic symbol is shown in below figures. Uses single horizontal line connected to the gate i/p & multiple vertical lines to indicate the individual i/p's. Each intersection b/w horizontal line and vertical line indicate the fuse connection.





Array logic



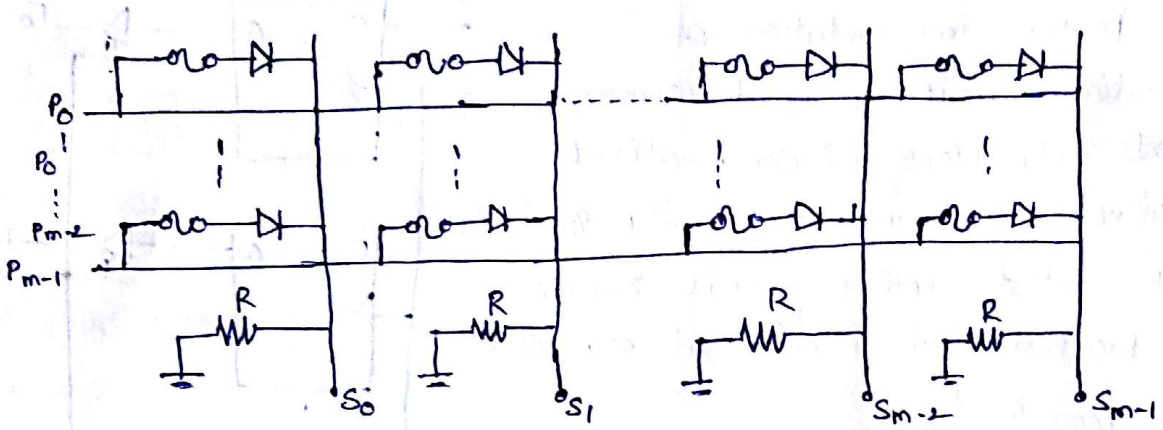
Simplified representation of AND matrix with I/p buffer

G. NAVEEN
EEE, NEC

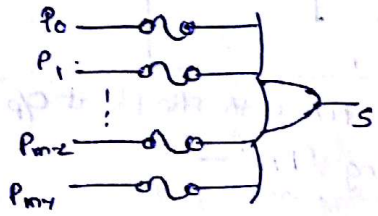
OR Matrix:-

The OR matrix is provided to produce the logical sum of product term outputs of the AND matrix. Fig shows the OR gates formed by diodes and resistors. Each OR gate has all the product terms as i/p variables. There is a nichrome fuse link in series with each diode which can be burnout to disconnect particular product term for that OR gate. Before programming, all fuse link in OR matrix are also intact and the sum term for each OR gate is given by

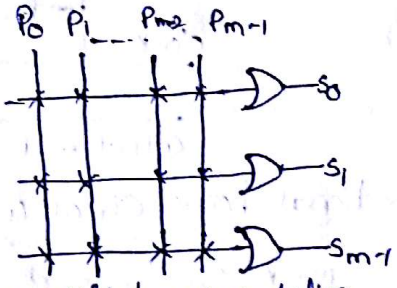
$$S = P_0 + P_1 + \dots + P_{m-2} + P_{m-1}$$



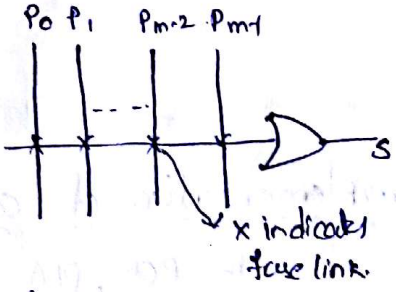
Below fig shows the simplified and equivalent representation of i/p connections for one OR gate.



(a) eq. of OR gate



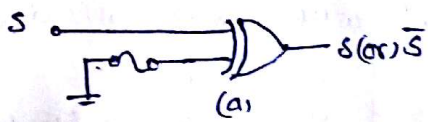
(c) simplified representation of OR matrix.

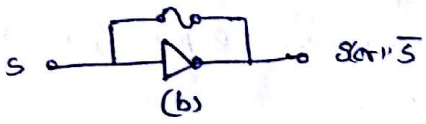


(b) Array logic.

Invert/Non-Invert Matrix:-

It provides output in the complement or uncomplemented form. The user can program the o/p in either complement or un-complement form as per design requirements. The typical circuits for invert/non-invert matrix is as shown in fig.

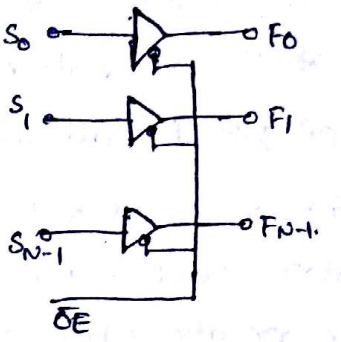




In both the cases if fuse is intact the o/p is in its uncomplemented form: otherwise o/p is in the complemented form.

Output Buffer:—

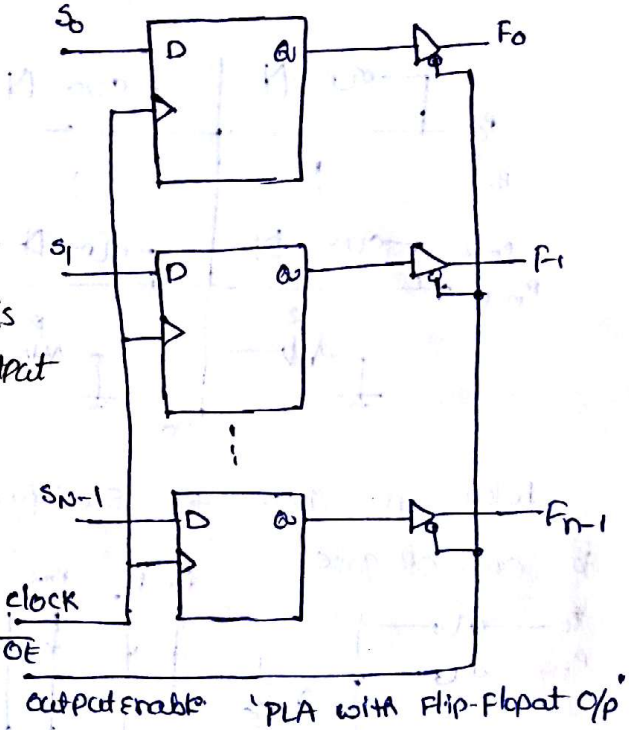
The deriving capacity of PLA is increased by providing buffers at the o/p. they are usually TTL compatible. Fig shows the 'tri-state' TTL compatible o/p buffer. the o/p buffer may provide 'totem-pole', 'open collector' (or) 'tri-state o/p'



Output Through Flip-Flops:—

For the implementation of sequential circuits we need memory elements, flip-flops and combinational circuitry for deriving the flip-flop i/p's.

To satisfy both the needs some PLAs are provided with flip-flop at each output as shown in fig.



output enable 'PLA with Flip-Flop at O/P'

Implementation of combinational Logic Circuit using PLA:—

Like ROM, PLA can be mask-programmable or field-programmable. with mask-programmable PLA, the user must submit a PLA program table to the manufacturer. This table is used by the vendor to produce a user-made PLA that has required internal paths b/w i/p's & outputs.

A second type of PLA available is called a "Field-programmable logic array" or FPLA. The FPLA can be programmed by the user by means of certain recommended procedures. FPLA's can be programmed with commercially available programmer units.

As mentioned earlier user has to submit PLA program table to the manufacturers to get the user made PLA.

G. NAVEEN
EEE, NEC

Determination of PLA program table :-

Eg:- A combinational circuit is defined by the functions

$F_1 = \Sigma m(3,5,7)$ $F_2 = \Sigma m(4,5,7)$

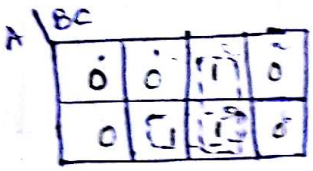
Implement the circuit with PLA having 3 i/p's, 3 product terms & 2 outputs

Sol: Truth table for given functions.

A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	1
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1

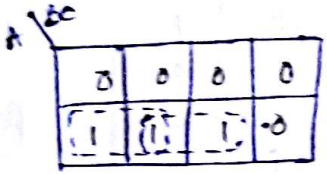
K-map Simplification:-

For F_1:-



$F_1 = AC + BC$

For F_2:-



$F_2 = A\bar{B} + AC$

	Product term	Inputs			Outputs	
		A	B	C	F_1	F_2
AC	1	1	-	1	1	1
BC	2	-	1	1	1	-
$A\bar{B}$	3	1	0	-	-	1
					T	T	T/C

There are 3 distinct product terms: AC, BC & $A\bar{B}$ and two Sum terms

The PLA program table consists of 3 columns specifying product terms, i/p & o/p's

→ The first column gives the list of product terms numerically

→ 2nd " specifies the required paths b/w i/p & AND gates.

→ 3rd " " " " AND gates & OR gates

Under each o/p variable, we write a 'T' (for true), the o/p inverter is to be by passed, and 'C' (for complement). if the function is to be complemented with the o/p inverter

The product terms listed on the left of first column are not the part of PLA program table they are included for reference only.

Eg:- Design a BCD to Excess-3 code converter and implement using suitable PLA.

Sol:-

Decimal	BCD code				Excess-3 code			
	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

K-Map Simplification:-

For E_3

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

$E_3 = B_3 + B_2B_0 + B_2B_1$

For E_2

B_3B_2 \ B_1B_0	00	01	11	10
00	0	1	1	1
01	1	0	0	0
11	X	X	X	X
10	0	1	X	X

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

For E_1

B_3B_2 \ B_1B_0	00	01	11	10
00	1	0	1	0
01	1	0	1	0
11	X	X	X	X
10	1	0	X	X

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

For E_0

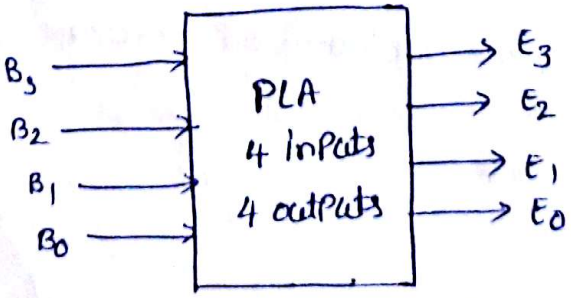
B_3B_2 \ B_1B_0	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	X	X	X	X
10	1	0	X	X

$E_0 = \bar{B}_1\bar{B}_0 + B_1\bar{B}_0$

Product terms		Inputs				outputs			
		B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
B_3	1	1	–	–	–	1	–	–	–
B_2B_0	2	–	1	–	1	1	–	–	–
B_2B_1	3	–	1	1	–	1	–	–	–
$B_2\bar{B}_1\bar{B}_0$	4	–	1	0	0	–	1	–	–
$\bar{B}_2B_0$	5	–	0	–	1	–	1	–	–
$\bar{B}_2\bar{B}_1$	6	–	0	1	–	–	1	–	–
$\bar{B}_1\bar{B}_0$	7	–	–	0	0	–	–	1	1
B_1B_0	8	–	–	1	1	–	–	1	–
$B_1\bar{B}_0$	9	–	–	1	0	–	–	–	1
						T	T	T	T	T/C

← PLA Program table

Logic diagram:-



BCD to Excess-3 code converter using PLA.

G. NAVEEN
EEE, NEC

Programmable Array Logic :- (PAL)

PLA is a programmable AND array & programmable OR array. PAL is a PLD with a fixed OR array & a programmable AND array. Because only AND gates are programmable, the PAL is easier to program but not flexible as PLA.

Fig shows 4 i/p & 4 o/p array logic for typical PAL.

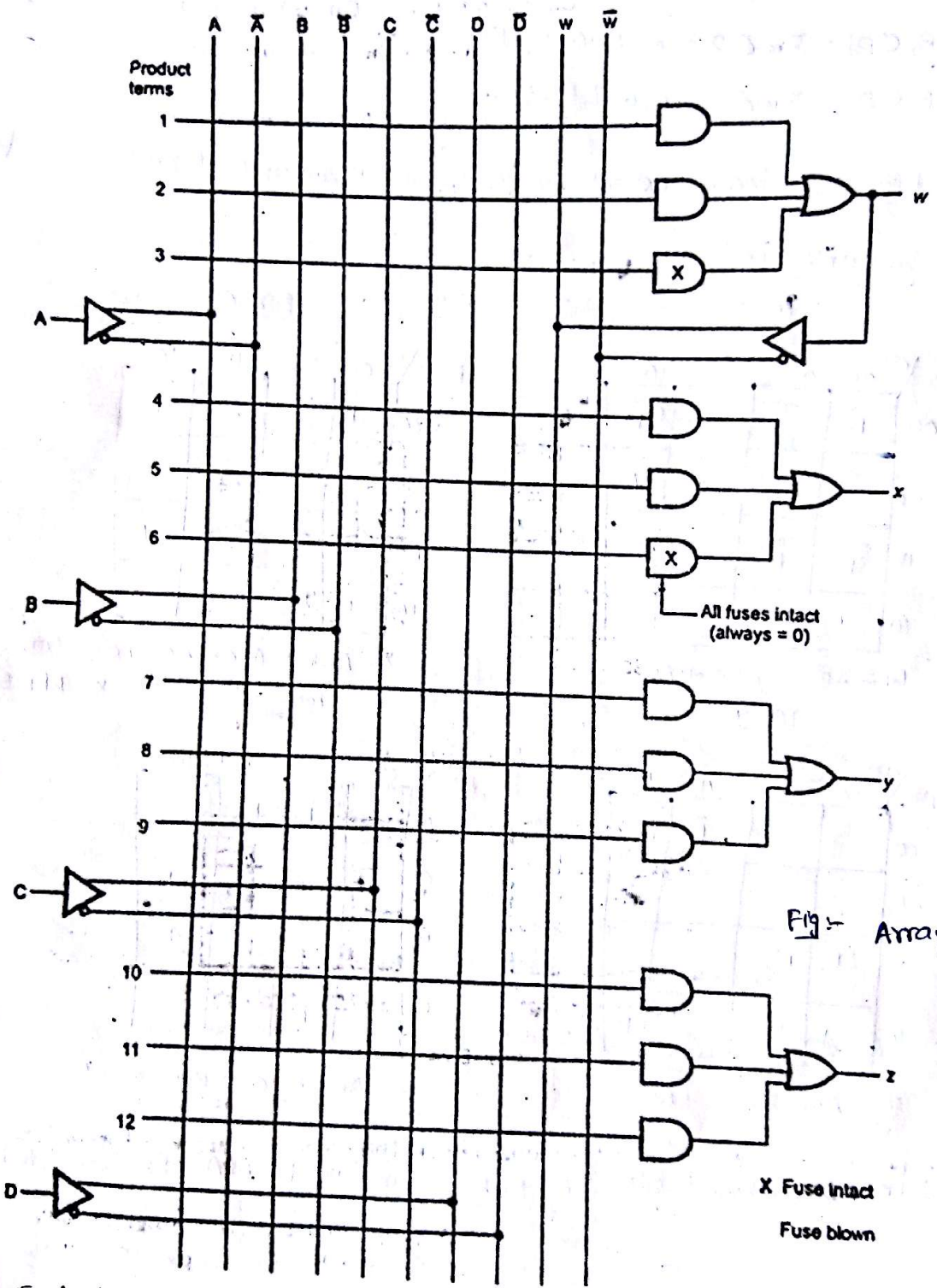


Fig :- Array logic for typical PAL

Each i/p has i/p buffer and an inverter gate. There are four sections, each section has 3 programmable AND gates & one fixed OR gate. The o/p of section '1' is connected to a buffer-inverter gate and then fedback into the i/p's of the AND gates, and through fuses.

This allows the logic designer to feed an o/p function back as an i/p variable to create a new function. Such PAL's are referred to as "programmable I/O PALs.

Implementation of combinational Logic circuit using PAL:-

Eg:- Implement the following Boolean functions using PAL

$w(A,B,C,D) = \Sigma m(0,2,6,7,8,9,12,13)$

$x(A,B,C,D) = \Sigma m(0,2,6,7,8,9,12,13,14)$

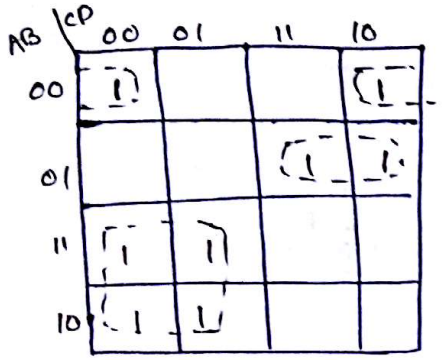
$y(A,B,C,D) = \Sigma m(2,3,8,9,10,12,13)$

$z(A,B,C,D) = \Sigma m(1,3,4,6,9,12,14)$

Sol:- Simplify the four functions for minimum number of terms.

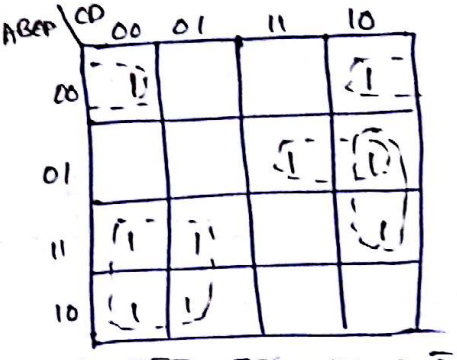
K-Map Simplification:-

For w



$W = \bar{A}\bar{B}\bar{D} + \bar{A}BC + A\bar{C}$

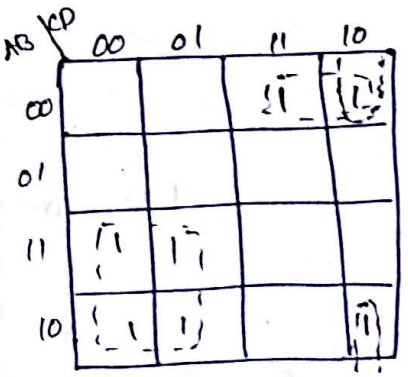
For x



$X = \bar{A}\bar{B}\bar{D} + \bar{A}BC + A\bar{C} + BC\bar{D}$ (or)

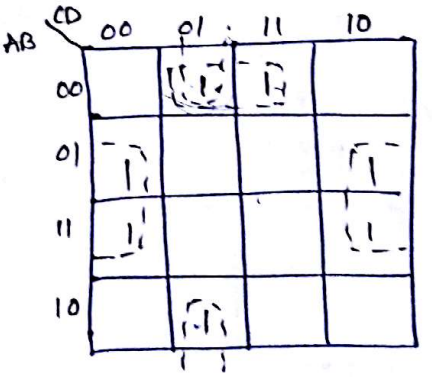
$X = W + BC\bar{D}$

For y



$Y = \bar{A}BC + \bar{B}C\bar{D} + A\bar{C}$

For z

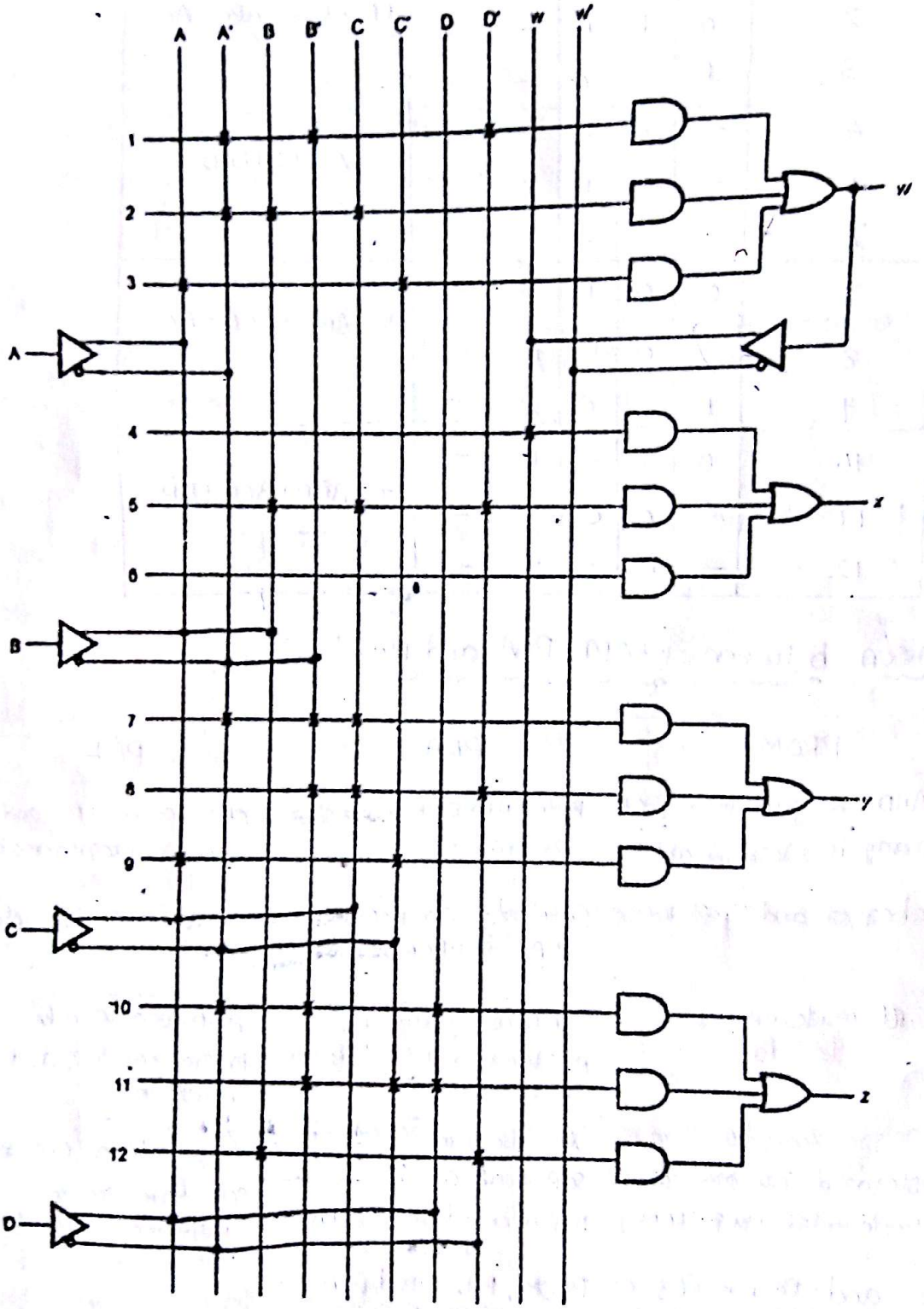


$Z = \bar{A}\bar{B}D + \bar{B}\bar{C}D + B\bar{D}$

The program table for PAL is similar to PLA Program table.

G. NAVEEN
EEE, NEC

Logic diagram:-



Program table is as shown in below table.

Product term	AND Inputs					Outputs
	A	B	C	D	W	
1	0	0	-	0	-	$W = \bar{A}\bar{B}\bar{D} + \bar{A}BC + A\bar{C}$
2	0	1	1	-	-	
3	1	-	0	-	-	
4	-	-	-	-	1	$X = W + BC\bar{D}$
5	-	1	1	0	-	
6	-	-	-	-	-	
7	0	0	1	-	-	$Y = \bar{A}\bar{B}C + \bar{B}C\bar{D} + A\bar{C}$
8	-	0	1	0	-	
9	1	-	0	-	-	
10	0	0	-	1	-	$Z = \bar{A}\bar{B}D + B\bar{C}D + B\bar{D}$
11	-	0	0	1	-	
12	-	1	-	0	-	

Comparison between PROM, PLA and PAL:—

S.No	PROM	PLA	PAL
1.	AND array is fixed & OR array is Programmable	Both AND & OR array's are Programmable	OR array is fixed and AND array is programmable.
2.	cheaper and Simple to use	Costliest & complex than PAL & PROM's	cheaper and Simpler
3.	All minterms are decoded	AND array can be programmed to get desired minterms.	AND array can be programmed to get desired minterms.
4.	Only Boolean functions in standard SOP form can be implemented using PROM	Any Boolean function's in SOP form can be implemented using PLA	Any Boolean function in SOP form can be implemented using PLA.

Merits and De merits of PROM, PAL and PLA:—

PROM:—

Merits:—

1. PROM is programmed by the user's
2. PROM's contents can be altered off-line
3. The reading speed is extremely high in PROM's

G.NAVEEN
EEE, NEC

Demerits:-

1. The maximum no. of I/p variables that can be taken by a PROM is only 8.
2. A PROM is not reprogrammable. Hence any mistake while programming cannot be corrected.

PAL:-

Merits:-

1. PAL has Low and fixed Propagation delay's.
2. It is simple to program and inexpensive.
3. PAL's eliminate the fuses in OR array.

Demerits:-

1. PAL's can be used to design simple state machines & combinational circuits
2. It is less flexible to program.

PLA:-

Merits:-

1. If two (or) more o/p functions have a common product term that can be shared by different OR gates, the PLA area can be reduced
2. These offer enhanced flexibility in the design of complex systems.

Demerits:-

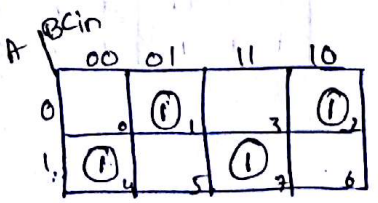
The Propagation delay of the gate will cause the inverted i/p to change for a short time after the application of non-inverted signal.

Implement the Full adder with PLA:-

The Truth table of full-adder is, & Drawing the k-map's for the Sum & Carry-out terms & minimizing them.

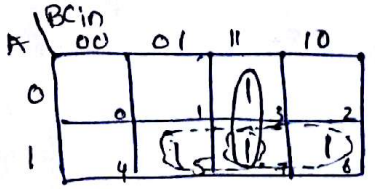
Inputs			Outputs		
A	B	C_{in}	S	C_o	
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	1	0	2
0	1	1	0	1	3
1	0	0	1	0	4
1	0	1	0	1	5
1	1	0	0	1	6
1	1	1	1	1	8

K-map for Sum:-

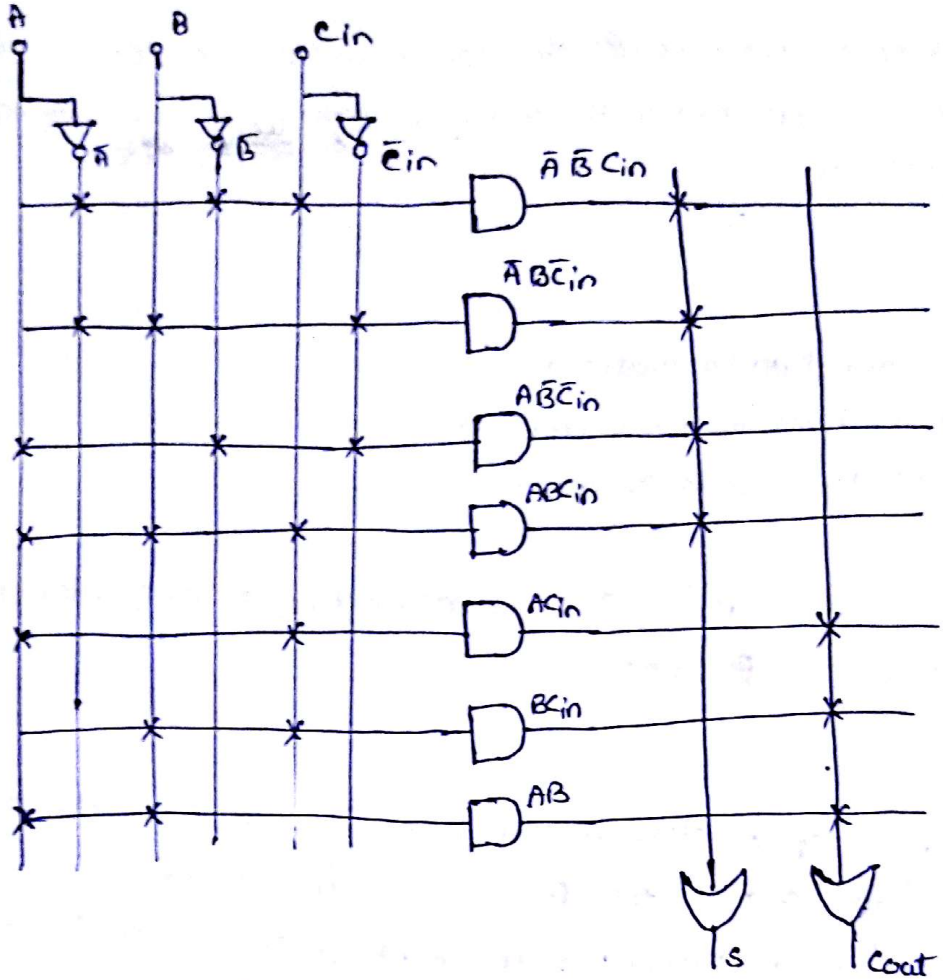


$S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$

K-map for carry-out:-



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



Implementation of 3-Bit Binary-to-Gray conversion:- using PLA:-

The conversion of 3-Bit Binary to Gray is shown in truth table.

Binary			Gray		
B_3	B_2	B_1	G_3	G_2	G_1
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

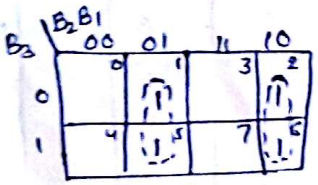
from the truth table the o/ps are.

$G_3 = \Sigma m\,(4,5,6,7)$

$G_2 = \Sigma m\,(2,3,4,5)$

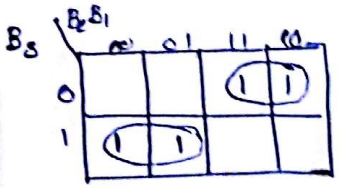
$G_1 = \Sigma m\,(1,2,5,6)$

K-map for G_1:-



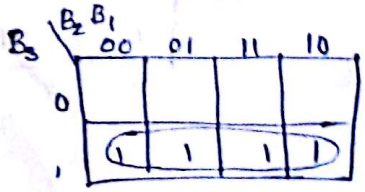
$G_1 = B_2\bar{B}_1 + \bar{B}_2B_1$

K-map for G_2



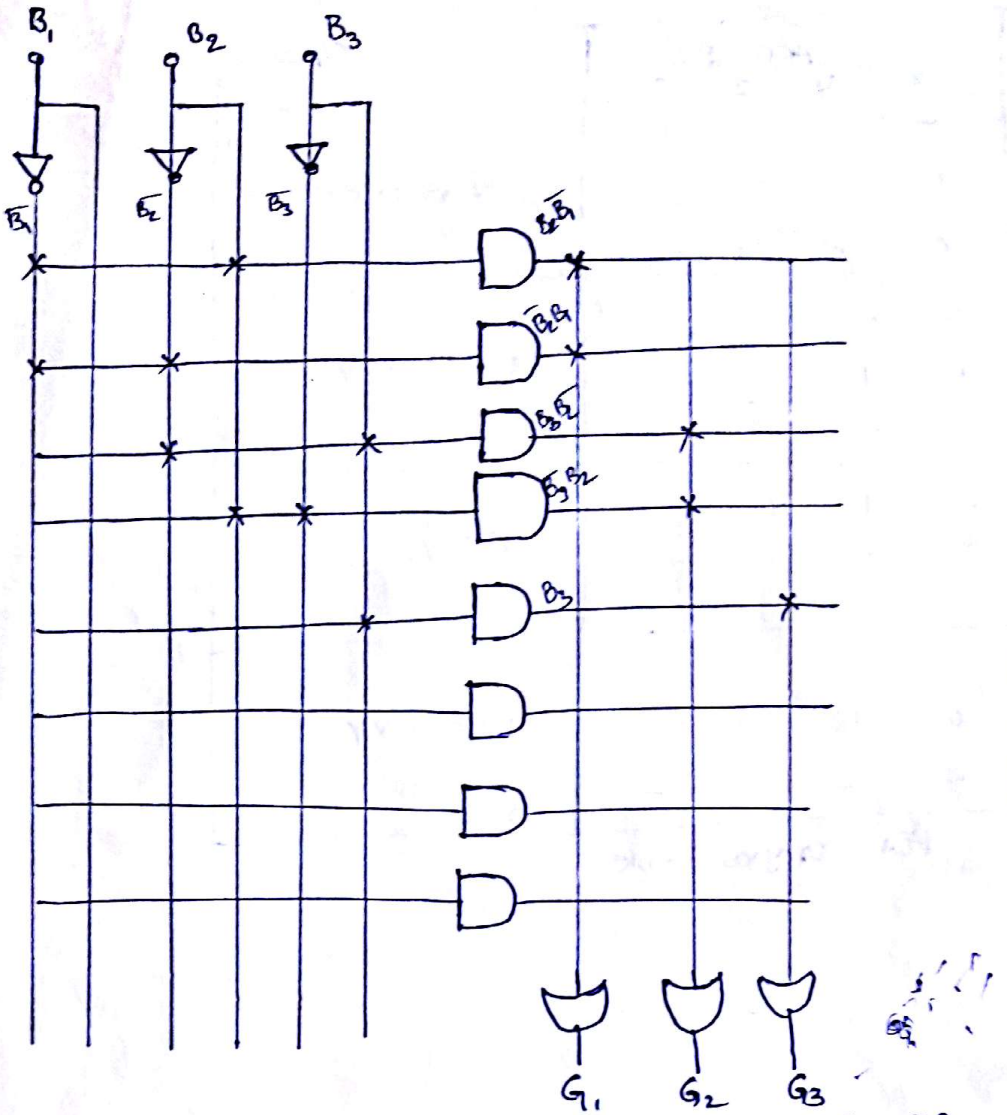
$G_2 = B_3\bar{B}_2 + \bar{B}_3B_2$

K-map for G_3:-



$G_3 = B_3$

G.NAVEEN
REE, NEC



3-bit binary to Gray code Convertor using PLA

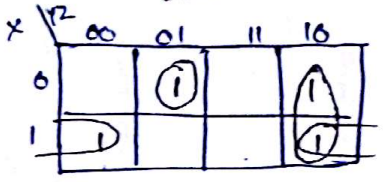
Eg:- Tabulate the PLA Programmable table for the 4-Boolean functions listed below.

$A(x,y,z) = \Sigma m(1,2,4,6)$ $B(x,y,z) = \Sigma m(0,1,6,7)$

$C(x,y,z) = \Sigma m(2,6)$ $D(x,y,z) = \Sigma m(1,2,3,5,7)$

Sol:-

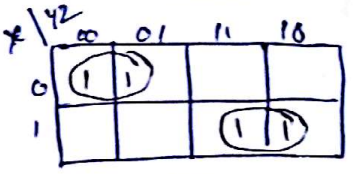
for A



$A = x\bar{z} + \bar{x}\bar{y}z + y\bar{z}$

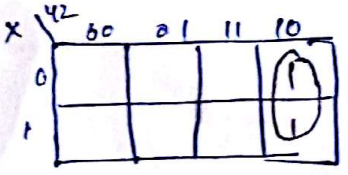
(or) $A = x\bar{z} + \bar{x}\bar{y}z + C$

for B



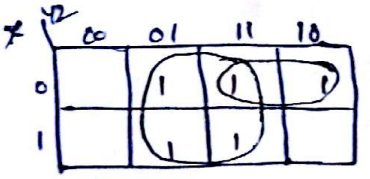
$B = \bar{x}\bar{y} + xy$

for C



$C = y\bar{z}$

for D



$D = z + \bar{x}y$

Product terms	AND I/P's				O/P's.
	x	y	z	c	
1	–	–	–	1	$A = C + 1\bar{x}\bar{z} + \bar{x}\bar{y}z$
2	1	–	0	–	
3	0	0	1	–	
4	0	0	–	–	$B = \bar{x}\bar{y} + xy$
5	1	1	–	–	
6	–	–	–	–	
7	–	1	0	–	$C = y\bar{z}$
8	–	–	–	–	
9	–	–	–	–	
10	–	–	1	–	$D = z + \bar{x}y$
11	0	1	–	–	
12	–	–	–	–	

PLA Programtable

G. NAVEEN
EEE, NEC