8/7/19

DESIGN CONCEPTS

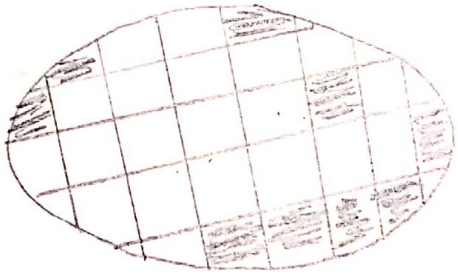
Digital Hardware :-

Logic circuits are used to build computer hardware and many other types of products. All such products are Broadly classified as Digital hardware.

Integrated Circuit :-

-> Integrated Circuit placing a no. of transistors (an entire circuit) on a single chip.

-> Integrated circuit chips are manufactured on a silicon wafer shown in figure. The wafer is cut to produce the individual chips.



-> There exist a wide variety of chips that impliment various functions that are useful in the design of digital hardware. the chips range from very simple chips with low functionality to extremely complex chips.

Ex:- A digital hardware product may require a Micro processor to perform some Arithmatic operations, memory chips to provide storage capability and Interface chips that allow easy connection to input and output devices.

-> For Implimenting logic circuits three main types of chips are used.

1) standard
2) programmable logic devices (PLD)
3) Custom

1. standard chips :-

→ standard chips confirm to an agreed upon standard interms of functionality (fixed functionality) and physical configuration.

→ Each standard chips contains less than 100 transistors and perform an single function.

2. programmable Logic devices (PLD) :-

→ In contrast to standard chip that have fixed functionality. it is possible to contract chip.

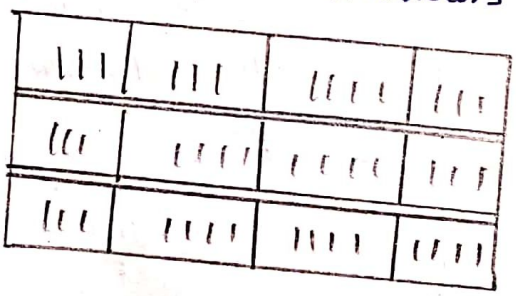
→ These chips include a collection of programmable switches that allow the internal circuitary in the chip to be configured in many different ways.

Advantage :-

1) most types of PLD's can be program multiple times.

2) PLD's are available in wide range of sizes.

ex:- Field programmable gate Array (FPGA) that contains more than 100 million transistors.



Disadvantage :-

The programmable switches consumes valuable chip area and limit the speed of operation.

3. Custom Designed chips :-

The logic circuiting is designed first and then an appropiate technology is chosen to implement the chip. finally the chip is manufactured by a company. This approach is known as Custom or semi - Custom Designed and such chips are called Custom or semi - Custom chips. And some times called Application Specific integrated chips (ASIC).

Advantage :-

-> Designed can be optimism for a specific task.

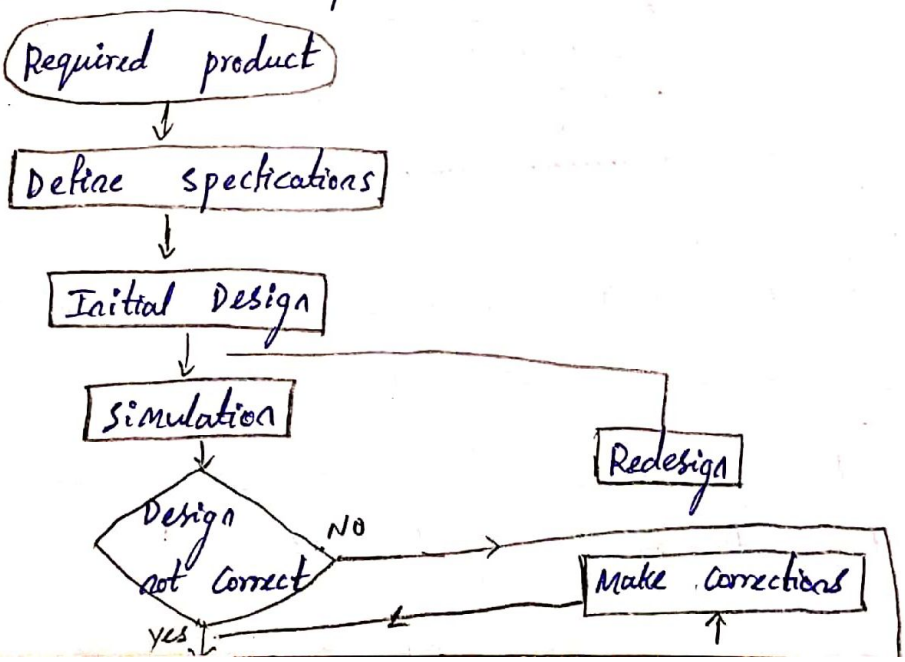
DisAdvantage :-

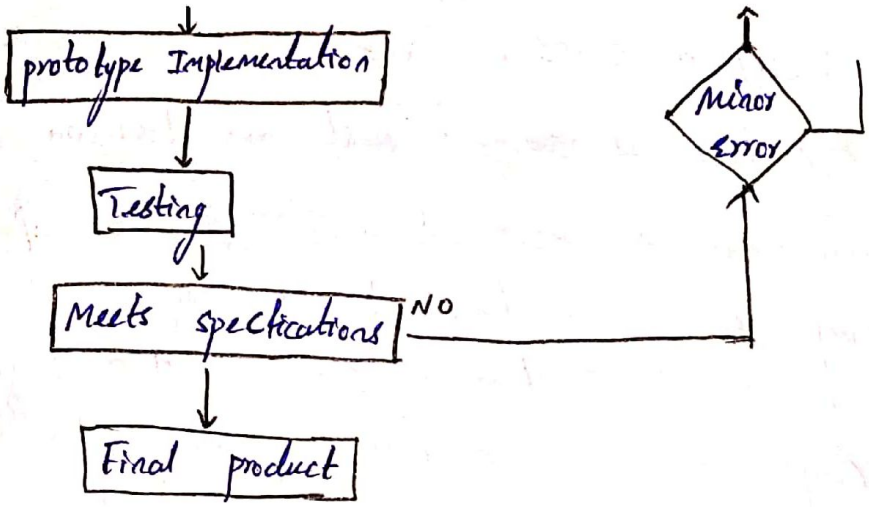
-> Takes considerable amount of time (months).

10|7 Design process :-

The flow chart epics a tipical development process. The process begins with delination of product specifications. the essentials feature of the product are identified and an acceptable Method of Evaluating the implemented features is done and the final product is established.

Fig Development process

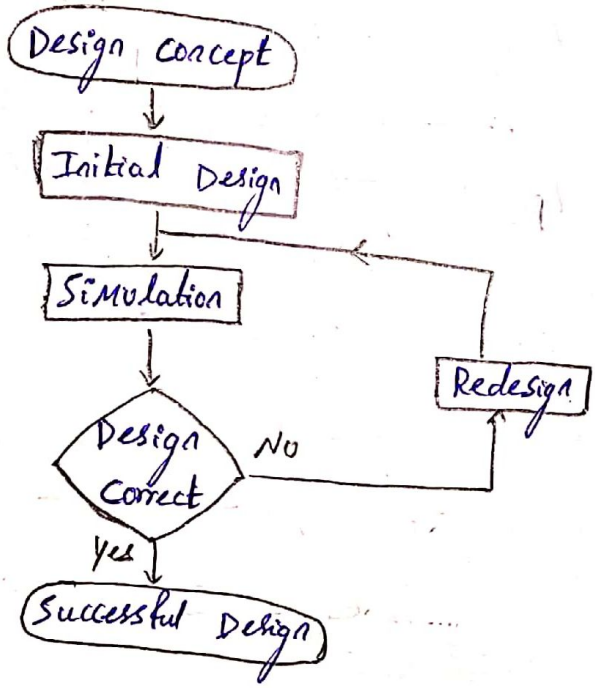




Design of Digital Hardware :-

1. Basic Design Loop :-

Any design process comprises basic sequence of task that are performed in various situations. this sequence is shown in figure.



→ The first step is to generate and initial design. this step requires a lot of manual effort.

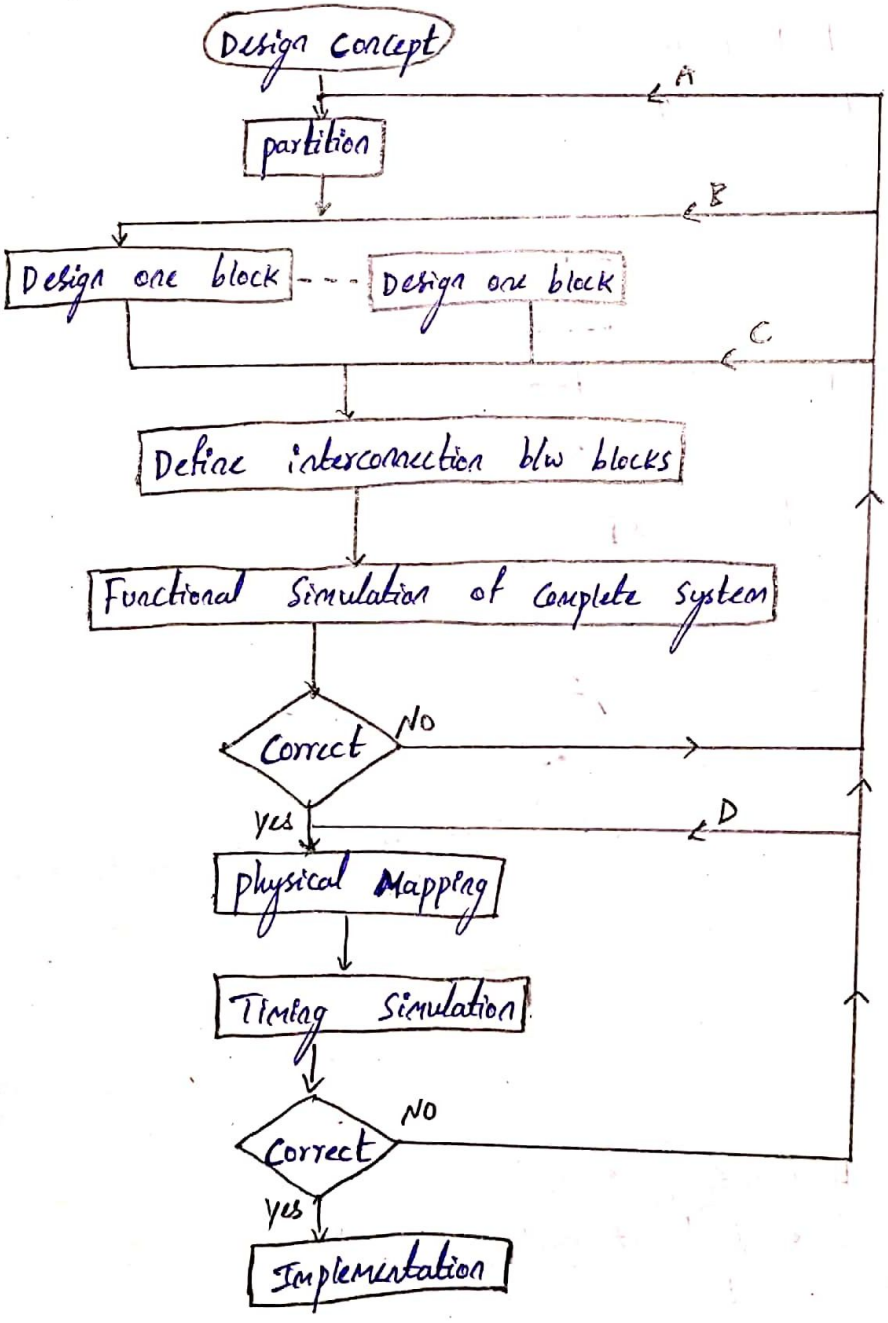
→ The next step is simulation of the design. There exist CAD tools to assists in this step.

→ If this simulation reveals some errors then the design must be change to overcome the problems.

→ The Redesigned version is again simulated to determine whether the errors have disappeared. This loop is Repeated until the simulation indicates a successful design.

2. Design of digital Hardware unit :-

Fig :- Design flow for logic Circuit



→ partitioning the circuit into smaller block & then designing each block separately.

→ Breaking down a large task into more manageable smaller parts is known as divide and conquarer approach.

12/7/19

Introduction to Logic Circuits :-

Variables & Function :-

Consider a Flash light

L=0 when light is off

L=1 when light is on

x is the variable represent switch

Switch open x=0

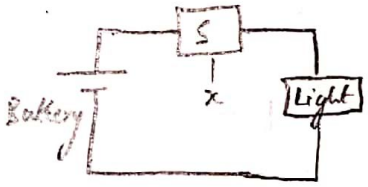
switch closed x=1

x is an i/p variables

When x=0 L=0

" x=1 L=1

$L(x) = x$

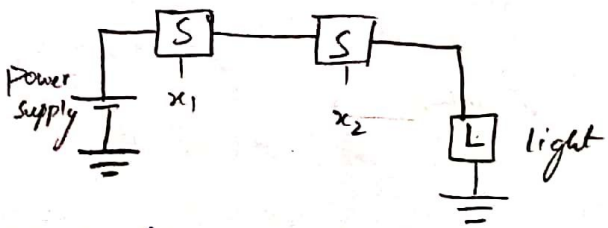


Consider 2 switches

x_1, x_2 - control i/ps

Switches can be connected in series & parallel

a) Logical AND function (series connection)



$L(x_1, x_2) = x_1 \cdot x_2$

where L=1 if $x_1=1$ and $x_2=1$

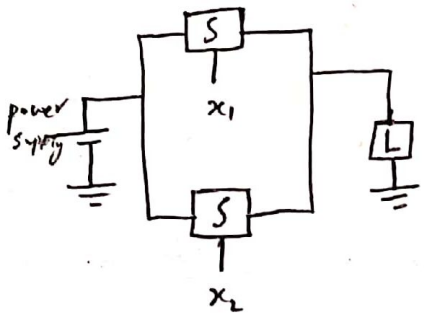
L=0 otherwise

$x_1 \cdot x_2$

x_1	x_2	L
0	0	0
0	1	0
1	0	0
1	1	1

∴ symbol is AND operator

b) The logical OR function :-



$x_1 + x_2$

x_1	x_2	L
0	0	0
0	1	1
1	0	1
1	1	1

$L(x_1, x_2) = x_1 + x_2$

$L = 1$ if $x_1 = 1$ or $x_2 = 1$

or if $x_1 = x_2 = 1$

$L = 0$ if $x_1 = x_2 = 0$

'+' symbol is OR operator

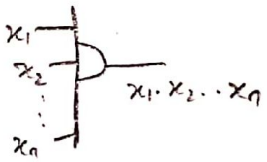
Logic Gates & Networks

→ Each logic operation can be implemented electronically with transistors Result in a circuit element called a logic gate.

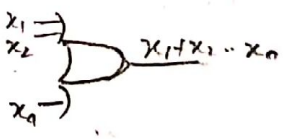
→ A logic gate has one or more inputs and one output that is a function of its input

Graphical symbols for AND, OR, NOT gates

a) AND gates

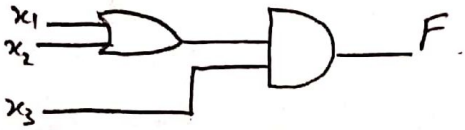


b) OR gate



c) NOT gate

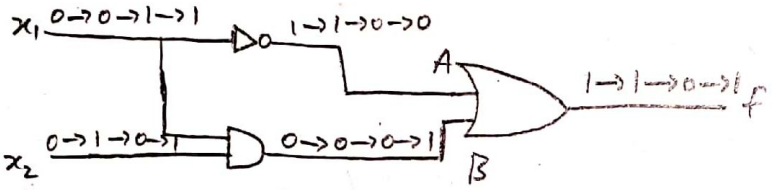
$F = (x_1 + x_2) \cdot x_3$



→ A larger circuit is implimented by a network of gates. Example :- the logic function $F = (x_1 + x_2) \cdot x_3$ can be implimented by the network shown in above figure.

Analysis of a logic network :-

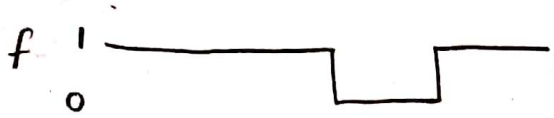
a) Network that implements $f = \bar{x}_1 + x_1 \cdot x_2$



b) Truth Tables for f

x_1	x_2	$f(x_1, x_2)$
0	0	1
0	1	1
1	0	0
1	1	1

c) Timing diagram



15/7/19

Boolean Algebra :-

-> In 1854, George Boolean developed one algebraic system now called Boolean Algebra.

-> Boolean Algebra is an algebraic structure defined by a set of elements 'B' together with 2 Binary operators (+ and .).

-> Boolean Algebra is based on set of Rules that are derived from a small no. of basic asumption. these assumptions are called Axioms. (Accepted to be true)

Following axioms are true

$0.0 = 0$

$1+1 = 1$

$1.1 = 1$

$0+0 = 0$

$0.0 = 1.0 = 0$

$1+0 = 0+1 = 1$

If $x=0$ then $\bar{x}=1$, If $x=1$ then $\bar{x}=0$

Single Variable theorems

$1.0 = 0$

$x.0 = 0$

$x+1 = 1$

$x+0 = x$

$x.x = x$

$x.\bar{x} = 0$

$x+\bar{x} = 1$

$\bar{\bar{x}} = x$

~~Two~~ variable Theorem :-

Duality principle :-

Duality principle is an important property of Boolean Algebra. It states that every algebraic expressions deducible from the postulates of Boolean Algebra remains valid if the operators and Identity elements are interchanged.

step :-1 Interchange OR and AND operator

step :-2 Replace all 1's by 0's, 0's by 1's

Ex :- $x+1 = 1$

$x.0 = 0$

Two Variable Theorem :-

1) $x.y = y.x$
$x+y = y+x$ } Commutative

2) $x+(y+z) = (x+y)+z$
$x.(y.z) = (x.y).z$ } Associative

3) $x.(y+z) = x.y + x.z$
$x+(y.z) = (x+y)(x+z)$ } Distributive

4) $x+xy = x$
$x(x+y) = x$ } Absorption

5) $(x+y)' = x'.y'$
$(xy)' = x'+y'$ } De morgan's

6) $x.y + x.\bar{y} = x$
$(x+y)(x+\bar{y}) = x$ } Combining

Simplify the Boolean expression to a minimum no. of literals

a) $x \cdot y + xy'$

$x(y+y')$

$x \cdot 1$

$= x$

b) $(x+y)(x+y')$

$x + (y \cdot y')$

$x + 0$

$= x$

c) $xyz + x'y + xz'y$

$xy(z+z') + x'y$

$xy + x'y$

$y(x+x')$

$\Rightarrow y \cdot 1 = y$

d) $(A+B)'(A'+B')'$

$A' \cdot B' \cdot ((AB')')'$

$(A' \cdot B') \cdot (AB)$

~~$(AA'B)' \cdot (A+B)$~~ $(A'A) \cdot (B'B)$ $\therefore x' \cdot x = 0$

~~$\neq 0$~~ $0 \cdot 0$

$= 0$

e) $xyz' + x'yz + xyz + x'yz'$

$xy(z'+z) + x'y(z+z')$

$xy \cdot 1 + x'y(1)$

$y(x+x')$

$y \cdot 1$

$= y$

Canonical Form & Standard Form :-

The standard form of minterms and maxterms demonstrate two important properties of Boolean Algebra.

1) Any Boolean function can be expressed as Sum of Minterms (by 'Sum' means ORing of terms)

2) Any Boolean function can be expressed as Sum of Maxterms (by 'product' means ANDing of terms)

→ Thus, expressing the Boolean function as a Sum of minterms or product of maxterms is said to be in canonical form.

14/7/19

Minterm :-

A product term in which all the variables appear exactly once either complimented or uncomplimented form is called minterm. There are 2^n distinct minterm for n variables. its symbol is m_j.

→ for two variables 'x' & 'y' there are four minterms

x y $2^2 = 4$ unit

x	y		
0	0	$\bar{x}\bar{y}$	m_0
0	1	$\bar{x}y$	m_1
1	0	$x\bar{y}$	m_2
1	1	xy	m_3

→ for 3 variables

3 X, Y, Z ⟶ 8 minterms

X	Y	Z	product term	Symbol (Small m)	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7
0	0	0	$\bar{X}\bar{Y}\bar{Z}$	m_0	1	0	0	0	0	0	0	0
0	0	1	$\bar{X}\bar{Y}Z$	m_1	0	1	0	0	0	0	0	0
0	1	0	$\bar{X}Y\bar{Z}$	m_2	0	0	1	0	0	0	0	0
0	1	1	$\bar{X}YZ$	m_3	0	0	0	1	0	0	0	0
1	0	0	$X\bar{Y}\bar{Z}$	m_4	0	0	0	0	1	0	0	0
1	0	1	$X\bar{Y}Z$	m_5	0	0	0	0	0	1	0	0
1	1	0	$XY\bar{Z}$	m_6	0	0	0	0	0	0	1	0
1	1	1	XYZ	m_7	0	0	0	0	0	0	0	1

Maxterm :-

A Sum term in which all the variables appear exactly once either complimented or uncomplimented form is called Maxterm. There are 2^n distinct Maxterm for n variables its Symbol is M_j.

→ for two variables 'x' & 'y' there are four maxterms

$X\ Y\ 2^2 = 4$ units

X	Y	
0	0	$- X + Y - M_0$
0	1	$- X + \bar{Y} - M_1$
1	0	$- \bar{X} + Y - M_2$
1	1	$- \bar{X} + \bar{Y} - M_3$

→ For 3 variables

3 xyz variable → 8 Maxterm

X	y	Z	Sum term	Symbol	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7
0	0	0	$x+y+z$	M_0	0	1	1	1	1	1	1	1
0	0	1	$x+y+\bar{z}$	M_1	1	0	1	1	1	1	1	1
0	1	0	$x+\bar{y}+z$	M_2	1	1	0	1	1	1	1	1
0	1	1	$x+\bar{y}+\bar{z}$	M_3	1	1	1	0	1	1	1	1
1	0	0	$\bar{x}+y+z$	M_4	1	1	1	1	0	1	1	1
1	0	1	$\bar{x}+y+\bar{z}$	M_5	1	1	1	1	1	0	1	1
1	1	0	$\bar{x}+\bar{y}+z$	M_6	1	1	1	1	1	1	0	1
1	1	1	$\bar{x}+\bar{y}+\bar{z}$	M_7	1	1	1	1	1	1	1	0

$$M_j = \bar{m_j}$$

Verification :-

$m_1 = \bar{x}\bar{y}z$

$M_1 = \bar{m_1} = \overline{\bar{x}\bar{y}z}$

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

$M_1 = \bar{m_1}$

ii) $m_5 = x\bar{y}z$

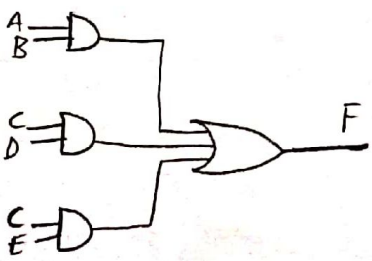
$M_5 = \bar{m_5} = \overline{x\bar{y}z}$

$= \bar{x}+y+\bar{z} = M_5$

$\therefore M_5 = \bar{m_5}$

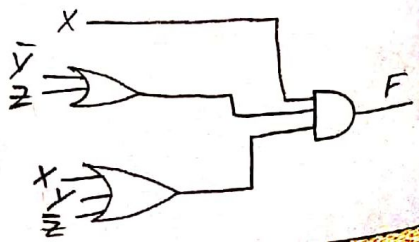
Sum of product (SOP)

ex :- $F = AB+CD+CE$



product of sum (POS)

ex :- $F = x(\bar{y}+z).(x+y+\bar{z})$



→ Sop are sum of minterms

→ pos are product of maxterms

Q) Express the Boolean function $F = A + \bar{B}C$ as sum of minterms or sum of product terms

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

missing Variables

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

$= A(BC + BC' + B'C + B'C') + AB'C + A'B'C$

$= ABC + ABC' + AB'C + AB'C' + AB'C + A'B'C$

Removing the terms which appear more than once

$\Rightarrow ABC + ABC' + AB'C + AB'C' + A'B'C$

$111 \quad 110 \quad 101 \quad 100 \quad 001$

$F = m_7 + m_6 + m_5 + m_4 + m_1$

$F(A,B,C) = \sum m(1,4,5,6,7)$

$F = A + B'C$

A	B	C	F	
0	0	0	0	$-m_0$
0	0	1	1	$-m_1$
0	1	0	0	$-m_2$
0	1	1	0	$-m_3$
1	0	0	1	$-m_4$
1	0	1	1	$-m_5$
1	1	0	ϕ	$-m_6$
1	1	1	ϕ	$1-m_7$

$\sum m(1,4,5,6,7)$

Hence verified //

$A + B'C$

$0+0'.0 \quad 0+0'.1$

$0+1.0 \quad 0+1.1$

$0+0=0 \quad 0+1=1$

$0+1'.0 \quad 0+1'.\phi$

$0+0.0 \quad 0+0.1$

$0+0=0 \quad 0+0=0$

$1+0'.0 \quad 1+0'.1$

$1+1.0 \quad 1+1.1$

$1+0=1 \quad 1+1=1$

$1+1'.0 \quad 1+1'.1$

$1+0.0 \quad 1+0.1$

$1+0=\phi \quad 1+0=\phi$

ii) $F(x,y,z) = x' + x(x+y')(y+z')$

Sol $= x' + x(xy + xz' + yy' + y'z')$

$x \cdot x = x$ $\quad y \cdot y' = 0$

$= x' + xy + xz' + xy'z'$

Missing variables

$\Rightarrow x'(y+y')(z+z') + xy(z+z') + xz'(y+y') + xy'z'$

$\Rightarrow (x'y + x'y')(z+z') + xyz + xyz' + xyz' + xy'z' + xy'z'$

$\Rightarrow x'yz + x'yz' + x'y'z + x'y'z' + xyz + xyz' + xyz' + xy'z' + xy'z'$

$\Rightarrow$ ~~$x'yz + x'yz' + x'y'z + x'y'z' + xyz + xyz' + xy'z' + xy'z'$~~

Removing the terms which appears more than once

$\Rightarrow x'yz + x'yz' + x'y'z + x'y'z' + xyz + xyz' + xy'z'$

$011 \quad 010 \quad 001 \quad 000 \quad 111 \quad 110 \quad 100$

$F = m_3 + m_2 + m_1 + m_0 + m_7 + m_6 + m_4$

$F(x,y,z) = \Sigma m(0,1,2,3,4,6,7)$ //

2) Express the Boolean function $F = xy + x'z$ as product of maxterms or product of sum

i) $F = xy + x'z$

Sol $= (xy + x')(xy + z)$ from distributive law $(x+yz) = (x+y)(x+z)$

Again distributive law

$\Rightarrow (x+x')(y+x')(x+z)(y+z)$ $\quad \because x + x' = 1$

$\Rightarrow 1 \cdot (y+x')(x+z)(y+z)$

$\Rightarrow (x'+y)(x+z)(y+z)$

missing variables

$(x'+y+zz')(x+z+yy')(y+z+xx')$

$\Rightarrow (x'+y+z)(x'+y+z')(x+z+y)(x+z+y')(y+z+x)(y+z+x')$

$F = (x'+y+z)(x'+y+z')(x+y+z)(x+y'+z)(x+y+z)(x'+y+z)$

Removing the terms which appears more than once

$F = (x'+y+z)(x'+y+z')(x+y+z)(x+y'+z)$

$F = (x+y+z)(x+y'+z)(x'+y+z)(x'+y+z')$

$0\ 0\ 0 \quad 0\ 1\ 0 \quad 1\ 0\ 0 \quad 1\ 0\ 1$

$M_0 \quad M_2 \quad M_4 \quad M_5$

$F(x,y,z) = \pi M(0,2,4,5)$

$F = xy + x'z$

x	y	z	F	
0	0	0	0	M_0
0	0	1	1	M_1
0	1	0	0	M_2
0	1	1	1	M_3
1	0	0	0	M_4
1	0	1	0	M_5
1	1	0	1	M_6
1	1	1	1	M_7

$\pi M(0,2,4,5)$

0·0+0·0 0+0+0·1 0·1+0·0
0·0+1·0 0·0+1·1 0·1+1·0
0+0=0 0+1=1 0+0=0

0·1+0·1 1·0+1·0 1·0+1·1
0+1·1 0+0·0 1·0+0·1
0+1=1 0+0=0 0+0=0

1·1+1·0 1·1+1·1
1+0·0 1+0·1
1+0=0 1+0=1

Hence verified.

19/7/19 Optimized implementation of logic functions using K-map

Map method :-

→ It is used to simplify B.F. 4 variables

→ It is also known as Karnaugh-map, K-map or map

→ It simplified expressions of sop and pos.

Two variables map :-

$2^2 = 4$ minterms

m_0	m_1
m_2	m_3

X \ Y	0	1
0	0	1
1	2	3

→ The map method provides a straight forward procedure for simplifying Boolean function up to 4 variables.

→ The map is also known as karnaugh map or K-map.

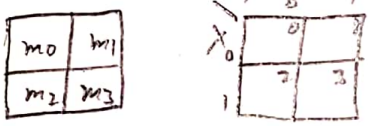
→ The map is a diagram made up of squares with each square representing one minterm of the function.

→ K-map is a modified form of truth table in which arrangement of the combination is very much convinent.

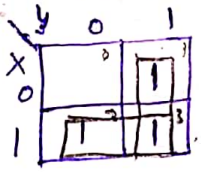
→ The simplified expressions reduce by the map are always in sum of product (SOP) or product of sum (POS)

Two variable map :-

There are four minterms for 2 binary variables ($2^2 = 4$ minterms)



Ex :- 1) Simplify $F(x,y) = \sum m(1,2,3)$

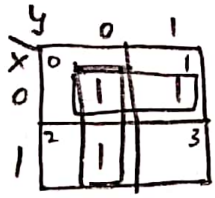


$F = y + x$

$F = x + y$

x	y
0	1
1	1

2) $F(x,y) = \sum m(0,1,2)$



$F = \bar{x} + \bar{y}$

x	y
0	0
1	0

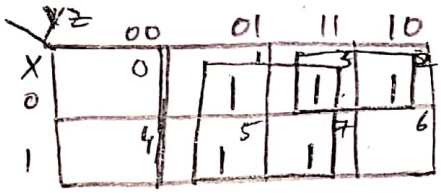
Three - variable map :-

There are sight minterms for 3 binary variables ($2^3 = 8$ minterms)

Three variable map consist of 8 squares

X \ YZ	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

Ex:-1) Simplify $F(x,y,z) = \sum m\,(1,2,3,5,7)$ using k-map

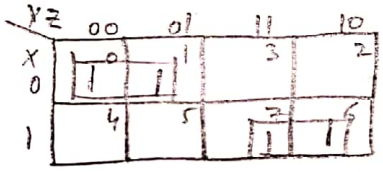


X	Y	Z
0	1	1
0	1	0

$F = \bar{X}y + Z$

X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	1

2) $F(x,y,z) = \sum m\,(0,1,6,7)$

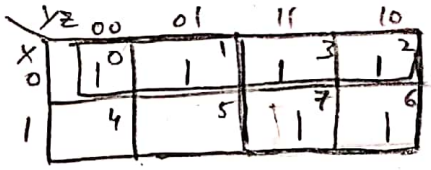


X	Y	Z
1	1	1
1	1	0

X	Y	Z
0	0	0
0	0	1

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

3) $F(x,y,z) = \sum m\,(0,1,2,3,6,7)$

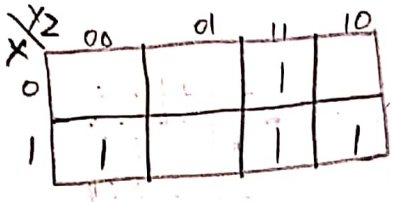


X	Y	Z
0	0	0
0	0	1
0	1	1
0	1	0

X	Y	Z
1	1	1
1	1	0
0	1	1
0	1	0

$F = \bar{X} + y$

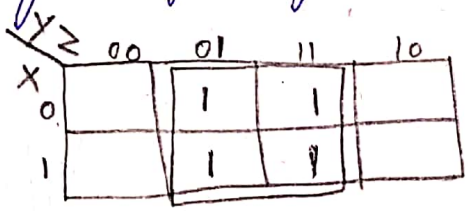
1) Represent $F(x,y,z) = yz + x\bar{z}$ in K-map



x	y z
0	1 1
1	1 1

x	y	z̄
1	0	0
1	1	0

2) Simplify $\bar{x}\bar{y}z + yz + xz$



$F = z$

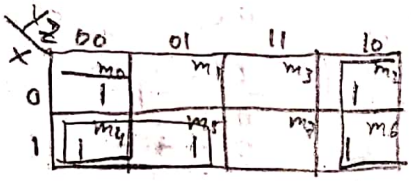
x̄ ȳ z
0 0 1
0 0 1

x	y z
0	1 1
1	1 1

x	y	z
0	1	1
1	1	1

x	y	z
0	0	1
0	1	1
1	0	1
1	1	1

3) Simplify $\Sigma m(0,2,4,5,6)$ using K-map



x	y	z
1	0	1
1	0	0

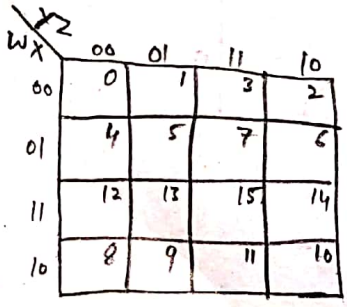
$x\bar{y} + \bar{z}$

x	y	z
0	0	0
0	1	0
1	0	0
1	1	0

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

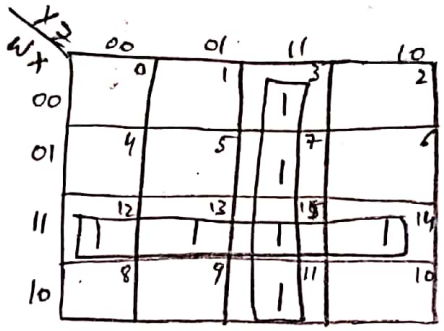
4 - Variable map :-

$2^4 = 16$ minterms



1) Simplify $F(A,B,C,D) = \Sigma m(3,7,11,12,13,14,15)$

(a)



x	y	z	w
0	0	1	1
0	1	1	1
1	1	1	1
1	0	1	1

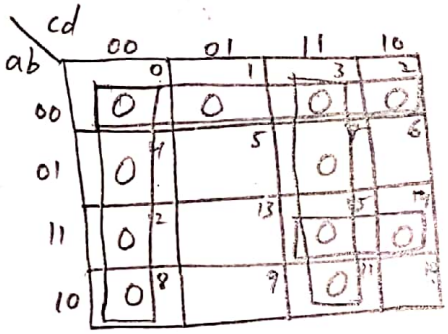
wz

x	y	z	w
1	1	1	1
1	1	0	1
1	1	1	1
1	1	1	0

xy

$F = xy + wz$

2) Simplify $F(a,b,c,d) = \pi M(0,1,2,3,4,7,8,11,12,14,15)$



a	b	c	d
0	0	0	0
0	0	0	1
0	0	1	1
0	0	1	0

$a'b'$

a	b	c	d
0	0	0	0
0	1	0	0
1	1	0	0
1	0	0	0

$c'd'$

a	b	c	d
0	0	1	1
0	1	Φ	Φ
1	1	1	1
1	0	1	1

cd

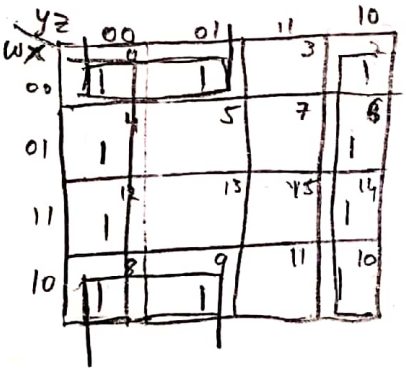
a	b	c	d
1	1	1	1
1	1	1	0

abc

$F = a'b' + cd + c'd' + abc$

$F = (a+b)\cdot(c'+d')\cdot(c+d)\cdot(a'+b'+c')$

3) $f(w,x,y,z) = \Sigma m(0,1,2,4,6,8,9,10,12,14)$



w	x	y	z
0	0	0	0
0	1	0	0
1	1	0	0
1	0	0	0
0	0	1	0
0	1	1	0
1	1	1	0
1	0	1	0

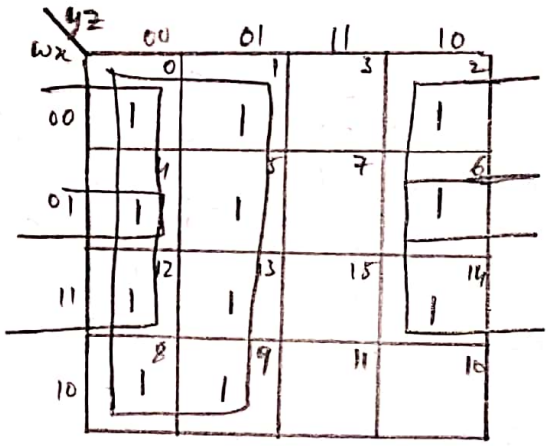
$\bar{z}$

w	x	y	z
0	0	0	0
0	0	0	1
1	0	0	0
1	0	0	1

$\bar{x}\bar{y}$

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

4) ** S^2 $f(w,x,y,z) = \Sigma m(0,1,2,4,5,6,8,9,12,13,14)$



w	x	y	z
0	0	0	0
0	0	0	1
0	1	0	1
1	1	0	0
1	1	0	0
1	0	0	0
1	0	0	1

$\bar{y}$

w	x	y	z
0	0	0	0
0	1	0	0
0	0	1	0
0	1	1	0

$\bar{w}\bar{z}$

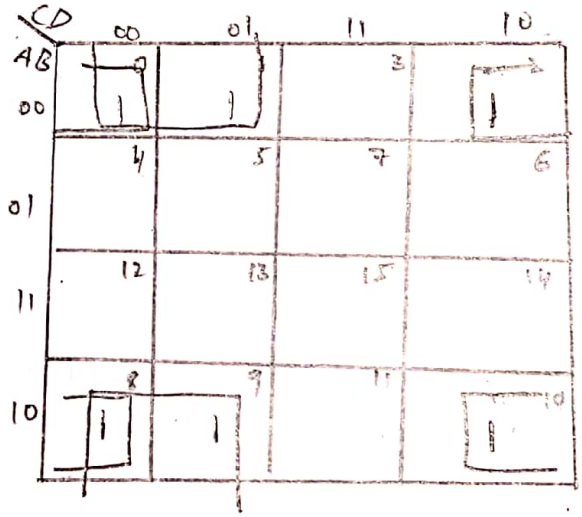
w	x	y	z
0	1	0	0
0	1	0	0
0	1	1	0
1	1	1	0

$x\bar{z}$

$F = x\bar{z} + \bar{y} + \bar{w}\bar{z}$

5) Simplify $F = \bar{A}\bar{B}\bar{C} + \bar{B}C\bar{D} + A\bar{B}\bar{C} + A\bar{B}C\bar{D}$

Sol



A	B	C	D
0	0	0	0
0	0	0	1
1	0	0	1
1	0	0	0

$\bar{B}\bar{C}$

A	B	C	D
0	0	0	0
0	0	1	0
1	0	1	0

$\bar{B}\bar{D}$

A	B	C	D
1	0	0	0
1	0	0	1

$A\bar{B}\bar{C}$

A	B	C	D
1	0	1	0

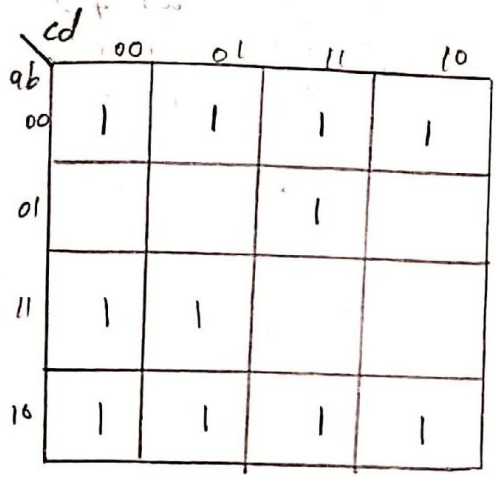
$A\bar{B}C\bar{D}$

$F = \bar{B}\bar{C} + \bar{B}\bar{D}$

26/7/19

1) Represent the B.F in K-map $\bar{b} + a\bar{c} + \bar{a}cd$

Sol



$a\bar{c}$

a	b	$\bar{c}$	d
1	0	0	0
1	0	0	1
1	1	0	0
1	1	0	1

$\bar{b}$

a	b	c	d
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1

a	b	c	d
0	0	1	1
0	1	1	1

Don't Care Conditions :-

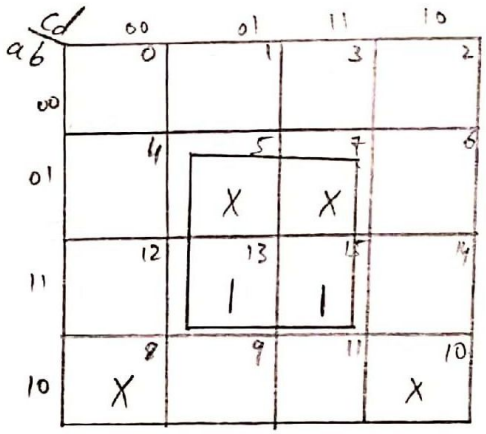
Functions that have unspecified outputs for some input combinations are called incompletely specified functions.

The unspecified minterm of a function is called Don't Care condition.

To Distinguish Don't Care condition from 1's & 0's and 'X' is used.

ex :- 1 simplify $F(a,b,c,d) = \Sigma m(13,15) + \Sigma d(5,7,8,10)$

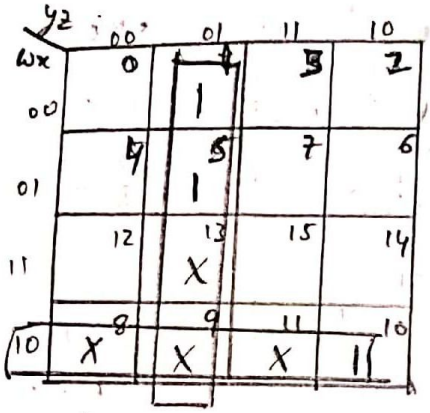
Sol



a	b	c	d
0	1	0	1
0	1	1	1
1	1	0	1
1	1	1	1

$F = bd$

4) 2) Simplify $F(w,x,y,z) = \Sigma(1,5,10) + d(8,9,11,13)$

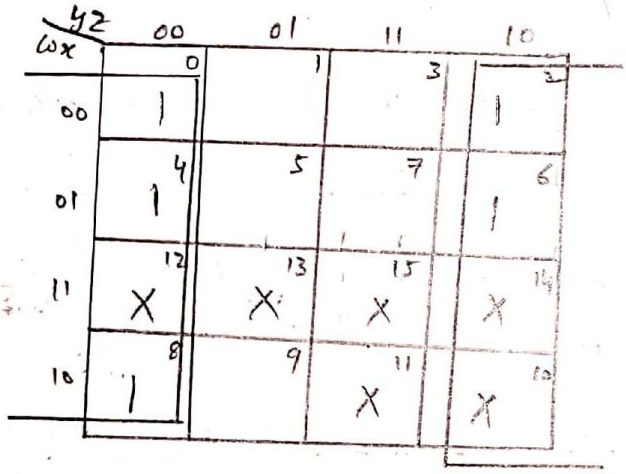


wx	yz
00	01
01	01
11	01
10	01

wx	yz
10	00
10	01
10	11
10	10

$F = \bar{y}z + w\bar{x}$

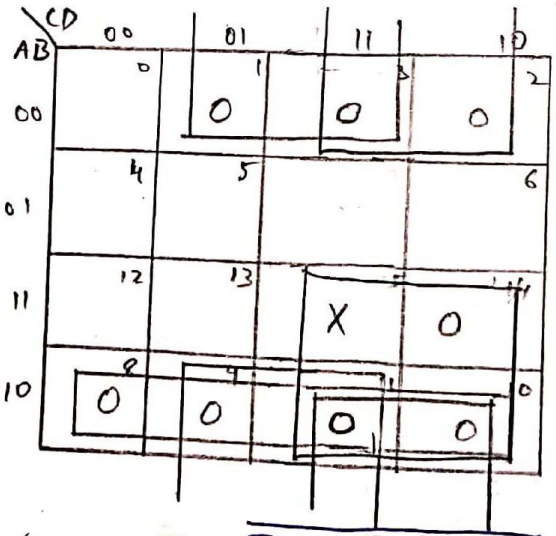
3). Simplify $F(w,x,y,z) = \Sigma(0,2,4,6,8) + \Sigma d(10,11,12,13,14,15)$



w	x	y	z
0	0	0	0
0	1	0	0
1	1	0	0
1	0	0	0
0	0	1	0
0	1	1	0
1	1	1	0
1	0	1	0

$F = \bar{z}$

4) $F(A,B,C,D) = \Pi(1,2,3,8,9,10,11,14) \cdot d(7,15)$



A	B	C	D
0	0	0	1
0	0	1	1
1	0	0	1
1	0	1	1

$\bar{B}D$

A	B	C	D
0	0	1	1
0	0	1	0
1	0	1	1
1	0	1	0

$\bar{B}C$

A	B	C	D
1	0	0	0
1	0	0	1
1	0	1	1
1	0	1	0

$A\bar{B}$

A	B	C	D
1	1	1	1
1	1	1	0
1	0	1	1
1	0	1	0

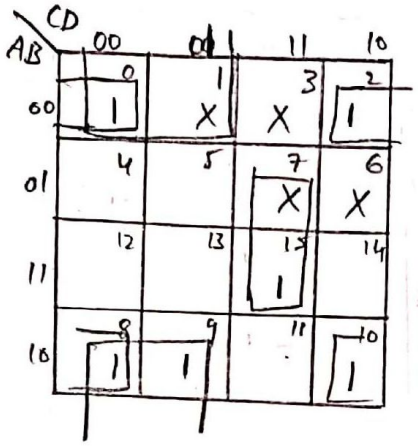
AC

$F = \bar{B}D + \bar{B}C + A\bar{B} + AC$

$F = \bar{A}\bar{B} + \bar{A}C + \bar{B}\bar{D} + \bar{B}\bar{C}$ $(\bar{A}+B) \cdot (\bar{A}+\bar{C}) \cdot (B+\bar{D}) \cdot (B+\bar{C})$

5) $F(A,B,C,D) = \Sigma m(0,2,8,9,10,15) + \Sigma d(1,3,6,7)$

Sol

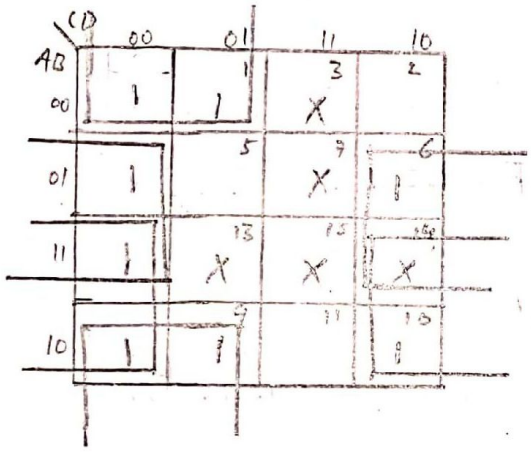


A B C D	A B C D
0 0 0	0 0 0 0
0 0 1 0	0 0 0 1
1 0 0 0	1 0 0 0
1 0 1 0	1 0 0 1
$B\bar{D}$	$B\bar{C}$

A B C D
0 1 1 1
1 1 1 1
BCD

$$F = B\bar{C} + \bar{B}\bar{D} + BCD$$

6) $F(A,B,C,D) = \Sigma m(0,1,4,6,8,9,10,12) + \Sigma d(3,7,13,14,15)$



A B C D
0 0 0 0
0 0 0 1
1 0 0 0
1 0 0 0
$B\bar{C}$

A B C D	A B C D
0 1 0 0	1 1 1 0
1 1 0 0	1 1 0 0
0 1 1 0	1 0 0 0
1 1 1 0	1 0 1 0
$B\bar{D}$	$A\bar{D}$

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

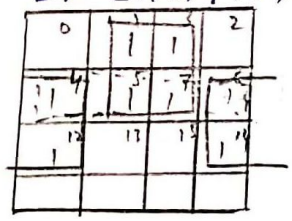
31/7/19 Prime Implicants :-

→ It is the product term obtained by combining maximum possible no. of adjacent squares in the map.

Essential prime implicant :-

→ If a minterm in a square is covered by only one prime implicant than that prime implicant is called essential prime implicant.

ex:- $F(A,B,C,D) = \Sigma m(1,3,4,5,6,7,12,14)$



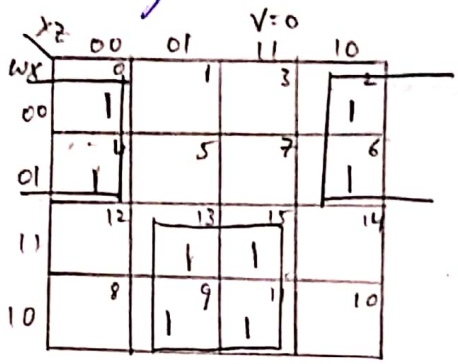
Here A'D, BD', A'B are p.I
But A'B is non essential p.I
A'D and BD' are essential p.I
$F = A'D + BD'$

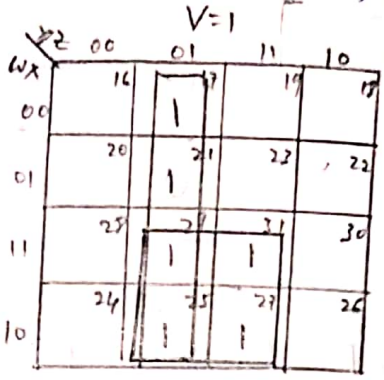
5 Variable Map :-

There are 32 minterms for 5 variables K-mapping.

Ex:- $f(V, W, X, Y, Z) = \Sigma (0, 2, 4, 6, 9, 11, 13, 15, 17, 21, 25, 27, 29, 31)$

Formal :-





V :

V	W	X	Y	Z
0	0	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	1	1	0

$\bar{V}\bar{W}\bar{Z}$

V	W	X	Y	Z
0	1	1	0	1
0	1	1	1	1
0	1	0	0	1
0	1	0	1	1

WZ

V	W	X	Y	Z
1	0	0	0	1
1	0	1	0	1
1	1	1	0	1
1	1	0	0	1

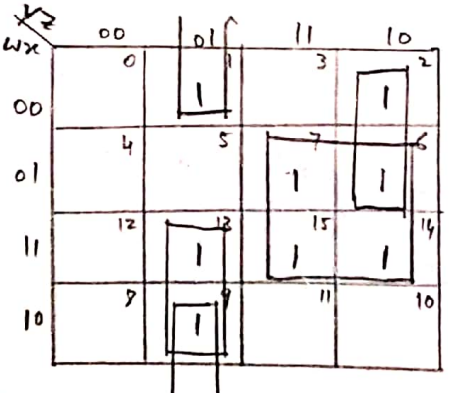
$V\bar{Y}Z$

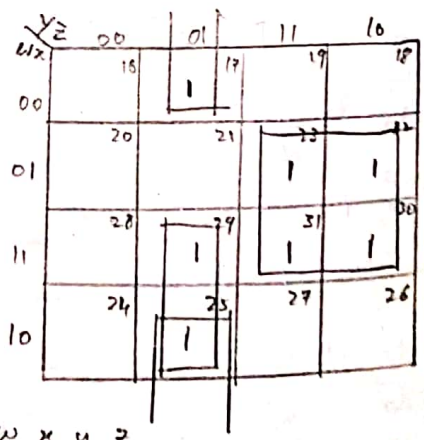
$$F = WZ + \bar{V}\bar{W}\bar{Z} + V\bar{Y}Z$$

Note :- if same pair exist in both the boxes then V term is not included.

1) $f(V, W, X, Y, Z) = \Sigma (1, 2, 6, 7, 9, 13, 14, 15, 17, 22, 23, 25, 29, 30, 31)$

Sol





W	X	Y	Z
0	1	1	1
0	1	1	0
1	1	1	1
1	1	1	0

WX

W	X	Y	Z
1	1	0	1
1	0	0	1

$WZ\bar{Y}$

W	X	Y	Z
0	0	0	1
1	0	0	1

$\bar{X}\bar{Y}$

V	W	X	Y	Z
0	0	0	1	0
0	0	1	1	0

$\bar{W}Y\bar{Z}$

$$F = XY + WZ\bar{Y} + \bar{X}\bar{Y} + \bar{V}\bar{W}Y\bar{Z}$$

Quine Mc Clusky Tabular method :-

Q) $F(A,B,C,D) = \Sigma(0,1,3,7,8,9,11,15)$

Step 1 :- Represent all given minterms with their binary equivalents

Minterms	A	B	C	D
0	0	0	0	0
1	0	0	0	1
3	0	0	1	1
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
11	1	0	1	1
15	1	1	1	1

0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1
1 0 0 0
1 0 0 1

Step 2 :- Arrange the minterms in the form of groups according to the no. of one's

Minterms	A	B	C	D	
0	0	0	0	0	→ 0 no. of 1's
1	0	0	0	1	→ 1 no of 1's
8	1	0	0	0	
3	0	0	1	1	→ 2 no of 1's
9	1	0	0	1	
7	0	1	1	1	→ 3 no of 1's
11	1	0	1	1	
15	1	1	1	1	→ 4 no of 1's

step 3 :- Compare minterms in one group with every minterm in next highest index group place a '—' where the variable is unmatched.

Note :- only a single difference should come.

minterm	A	B	C	D
0,1	0	0	0	-
0,8	-	0	0	0
1,3	0	0	-	1
1,9	-	0	0	1
8,9	1	0	0	-
3,7	0	-	1	1
3,11	-	0	1	1
9,11	1	0	-	1
7,15	-	1	1	1
11,15	1	-	1	1

2/8/19 step 4 :- Apply same process for resultant column and continue this until no further elimination of literals.

Numbers	A	B	C	D	
0,1,8,9	0	0	0	-	
0,8,1,9	-	0	0	-	0,1,8,9 —— -00-
1,3,9,11	-	0	-	1	
1,9,3,11	-	0	-	1	0,8,1,9 —— -00-
3,7,11,15	-	-	1	1	
3,11,7,15	-	-	1	1	

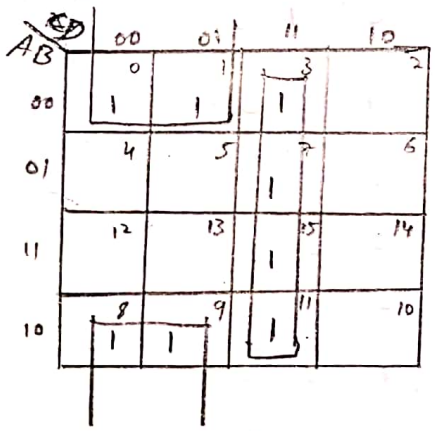
Numbers	A	B	C	D
0,1,8,9	0	0	0	-
1,3,9,11	-	0	-	1
3,7,11,15	-	-	1	1

Step 5 :- List all the prime implicants in the prime implicant chart

prime implicant	Literal representation	minterms Σm							
		m_0	m_1	m_3	m_7	m_8	m_9	m_{11}	m_{15}
3,7,11,15	CD			X	(X)			X	(X)
1,3,9,11	$\bar{B}D$		X	X			X	X	
0,1,8,9	$\bar{B}\bar{C}$	(X)	X		(X)		X		

$$F = CD + \bar{B}\bar{C}$$

K- mapping :-



A	B	C	D
0	0	0	0
0	0	0	1
1	0	0	0
1	0	0	1

$\bar{B}\bar{C}$

A	B	C	D
0	0	1	1
0	1	1	1
1	1	1	1
1	0	1	1

CD

$$F = \bar{B}\bar{C} + CD$$

∴ Hence Verified ,,

1) Simplify using tabulation method

$F(A,B,C,D) = \Sigma(0,2,4,6,7,9) + \Sigma d(10,11)$

Sol Step 1 :-

minterms	A	B	C	D
0	0	0	0	0
2	0	0	1	0
4	0	1	0	0
6	0	1	1	0
7	0	1	1	1
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1

Step 2 :-

minterms	A	B	C	D
0	0	0	0	0
2	0	0	1	0
4	0	1	0	0
6	0	1	1	0
9	1	0	0	1
10	1	0	1	0
7	0	1	1	1
11	1	0	1	1

Step 3 :-

minterms	A	B	C	D
0,2	0	0	-	0
0,4	0	-	0	0
2,6	0	-	1	0
2,10	-	0	1	0
4,6	0	1	-	0

		A	B	C	D
R	6,7	0	1	1	-
Q	9,11	1	0	-	1
	10,11	1	0	1	-

Step 4 :-

Numbers	A	B	C	D
0,2,4,6	0	-	-	0
0,4,2,6	0	-	-	0

	Number	A	B	C	D
P	0,2,4,6	0	-	-	0

Step 5 :-

	prime implicant	literal representation
P	0,2,4,6	$\bar{A}\bar{D}$
Q	9,11	$A\bar{B}D$
R	6,7	$\bar{A}BC$

minterm Σm

m_0	m_2	m_4	m_6	m_7	m_9
⊗	⊗	⊗	x		
				⊗	⊗
			x	⊗	

$$F = \bar{A}\bar{D} + A\bar{B}D + \bar{A}BC$$

Verification :-

CD / AB	00	01	11	10
00	1 (0)	1	3	1 (2)
01	1 (4)	5	1 (7)	1 (6)
11	12	13	15	14
10	8	1 (9)	X (11)	X (10)

A	B	C	D
0	1	1	1
0	1	1	0

$\bar{A}BC$

A	B	C	D
1	0	0	1
1	0	1	1

$A\bar{B}D$

A	B	C	D
0	0	0	0
0	1	0	0
0	0	1	0
0	1	1	0

$\bar{A}\bar{D}$

$$F = \bar{A}BC + A\bar{B}D + \bar{A}\bar{D}$$

Hence verified

2) $F(A,B,C,D) = \Sigma m(4,8,10,11,12,15) + \Sigma d(9,14)$

Sol) Step 1 :-

Minterms	A	B	C	D
4	0	1	0	0
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
14	1	1	1	0
15	1	1	1	1

step 2 :-

Minterms	A	B	C	D
4	0	1	0	0
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
12	1	1	0	0
11	1	0	1	1
14	1	1	1	0
15	1	1	1	1

step 3 :-

Minterms	A	B	C	D
4,12	-	1	0	0
8,9	1	0	0	-
8,10	1	0	-	0
8,12	1	-	0	0
9,11	1	0	-	1
10,11	1	0	1	-
10,14	1	-	1	0
12,14	1	1	-	0
11,15	1	-	1	1
14,15	1	1	1	-

step 4 :-

Minterms	A	B	C	D
8,9,10,11	1	0	-	-
8,10,9,11	1	0	-	-
8,10,12,14	1	-	-	0
8,12,10,14	1	-	-	0
10,11,14,15	1	-	1	-
10,14,11,15	1	-	1	-

Minterms	A	B	C	D
8,9,10,11	1	0	-	-
8,10,12,14	1	-	-	0
10,11,14,15	1	-	1	-
4,12	-	1	0	0

step 5 :-

prime implicant	Literal representation	Minterm ΣM M_4	M_8	M_{10}	M_{11}	M_{12}	M_{15}
8,9,10,11	$A\bar{B}$		x	x	x		
8,10,12,14	$A\bar{D}$		x	x		x	
10,11,14,15	AC			x	x		⊛
4,12	$B\bar{C}\bar{D}$	⊛				x	

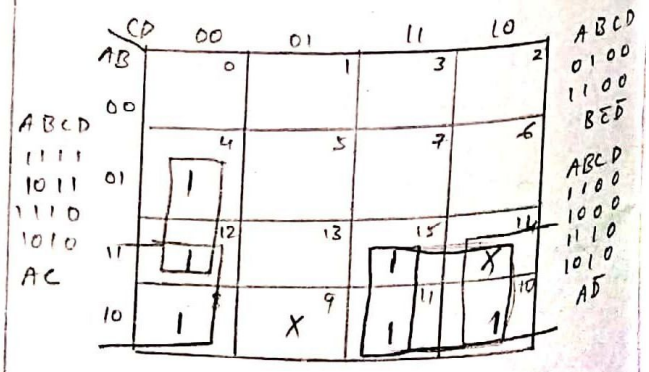
Essential prime Implicant - $AC + B\bar{C}\bar{D}$

Non Essential p.I - $A\bar{D} + A\bar{B}$

Since 8 is missing in pI

Answer = $AC + B\bar{C}\bar{D} + A\bar{B}$ or $A\bar{D}$

verification :-



$F = AC + B\bar{C}\bar{D} + A\bar{D}$

5/9/19 UNIT-2

Number Representation :

Addition & subtraction of signed & unsigned numbers :-

→ positive Integers (including zero) can be represented as un signed numbers.

→ Both signed and unsigned Binary numbers consist of string of Bits when represented in a computer. the left most Bit represents the sign and rest of Bit represent the number.

ex:- 01001 can be represented as 9 (unsigned binary) or +9 (signed binary) because the left most Bit is zero.

→ 11001 represents -9 when considered as a signed number because '1' in left most position designates a '-ve'.

→ Consider 9 represented in binary with 8 bits +9 (00001001) all though there is only one way to represent +9, there are 3 different ways to represent -9 with 8 bits.

3 different ways :-

1) Signed - magnitude representation
   00001001
   Changing sign 0 to 1

2) Signed - 1's - complement representation
   11110110

3) Signed - 2's complement representation
   2's Complement of 9 => 00001001
   Take 1's Complement => 11110110
   Add 1 => +1

   11110111

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1
1	1	1	1

Signed Binary Numbers :-

Decimal	Signed 2's complement	Signed 1's complement	Signed Magnitude
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
-1	1111	1110	1001
-2	1110	1101	1010
-3	1101	1100	1011

3's comp = 0011
2's " = 1100 +1
1101
3 = 0110
-3 = 1011

1 => 0001
-1 => 1001
2 => 0010
-2 => 1010
2's Comple = 0010
1's " = 1101 +1
1110

Addition :-

1) +6 → 00000110
+13 → 00001101
+19 00010011

verify: 2|19 - 1
2|9 - 1
2|4 - 0
2|2 - 0
1
10011

2) +5 → 00000101
+15 → 00001111
+20 00010100

2|5 - 1
2|2 - 1
2|1 - 1
1

2|20 - 0
2|10 - 0
2|5 - 1
2|2 - 0
1

3) -6 → 11111010
+13 → 00001101
+7 100000111

2's Comple of 6
00000110
1's Comp 11111001
+1
11111010
→ 7

4) +6 00000110
-13 11110011
-7 11111001

2's Com of 13 00001101
00000010
11111000
+1
11111001

2's Com of 13 00001101
11110010
+1
11110011

7/8/19

Substraction :-

Ex :- 1) (-6) - (-13)

-6 + 13

1

Take 2's complement of 6

```
  1 1 1 1
  1 1 1 1 1 0 1 0
+ 0 0 0 0 1 1 0 1
-------------------
1 0 0 0 0 0 1 1 1
```

discard carry

2) (-6) - (+13)

-6 - 13

1 1

Take 2's complement of both

```
  1 1 1   1
  1 1 1 1 1 0 1 0
  1 1 1 1 0 0 1 1
-------------------
1 1 1 1 0 1 1 0 1
```

discard carry

1' means -ve

Combined Circuit building block :

3 operations :-

$X \overset{(XOR)}{\oplus} Y = X\bar{Y} + \bar{X}Y$ =>

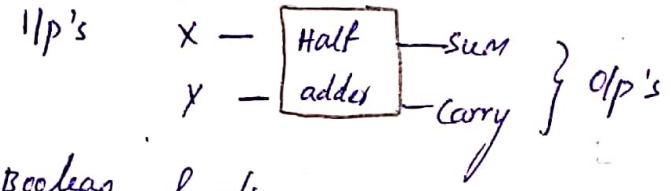
$X \overset{(XNOR)}{\odot} Y = XY + \bar{X}\bar{Y}$ =>

$X \oplus Y \oplus Z = \bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z} + XYZ$

Half Adder :-

Addition 2 bits :-

I/p		o/p	
x	y	C	S
0	0	0	0
0	1	0	1 ←
1	0	0	1 ←
1	1	1	0



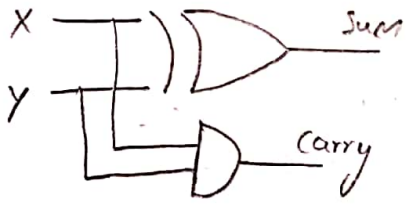
Boolean function :-

$S = x\bar{y} + \bar{x}y$

$S = x \oplus y$

$C = x \cdot y$

Logic diagram



Full Adder :-

Addition of 3 bits

X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0

X	Y	Z	C	S
1	1	1	1	1

Boolean expression

$$S = \bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z} + xyz$$

$$S = x \oplus y \oplus z$$

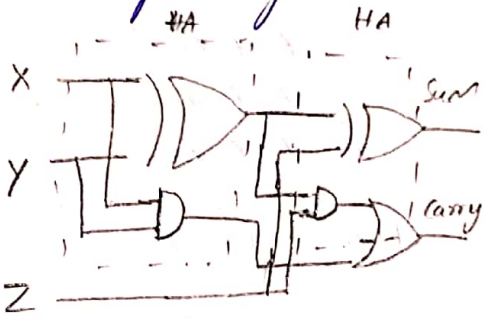
$$C = \bar{x}yz + x\bar{y}z + xy\bar{z} + xyz$$

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

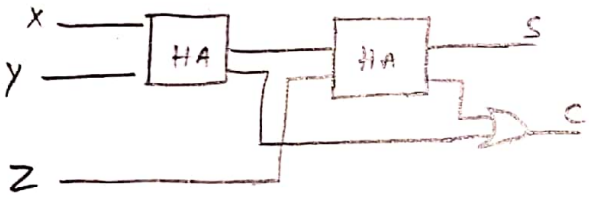
$$= z(x \oplus y) + xy(1)$$

$$C = z(x \oplus y) + xy$$

Logic diagram



(or)



Decorders ..

n I/ps — 2^n O/ps

A D

1) Draw the functional diagram for 3 to 8 line decorder

A_2	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

9/8/19 Boolean expression:

$D_0 = \bar{A}_2 \bar{A}_1 \bar{A}_0$

$D_1 = \bar{A}_2 \bar{A}_1 A_0$

$D_2 = \bar{A}_2 A_1 \bar{A}_0$

$D_3 = \bar{A}_2 A_1 A_0$

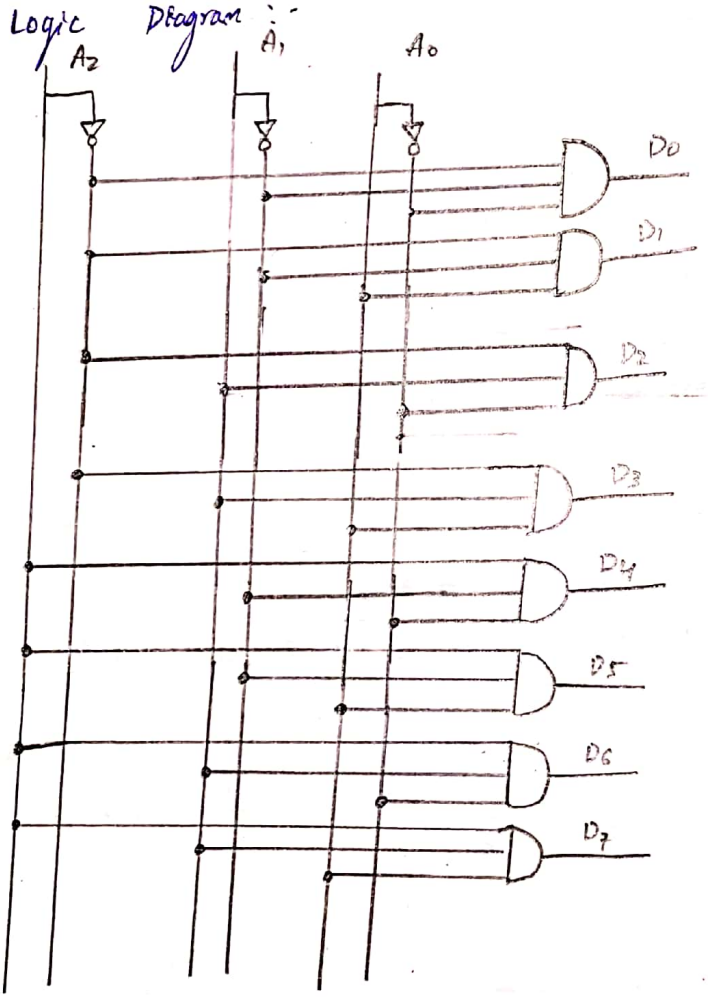
$D_4 = A_2 \bar{A}_1 \bar{A}_0$

$D_5 = A_2 \bar{A}_1 A_0$

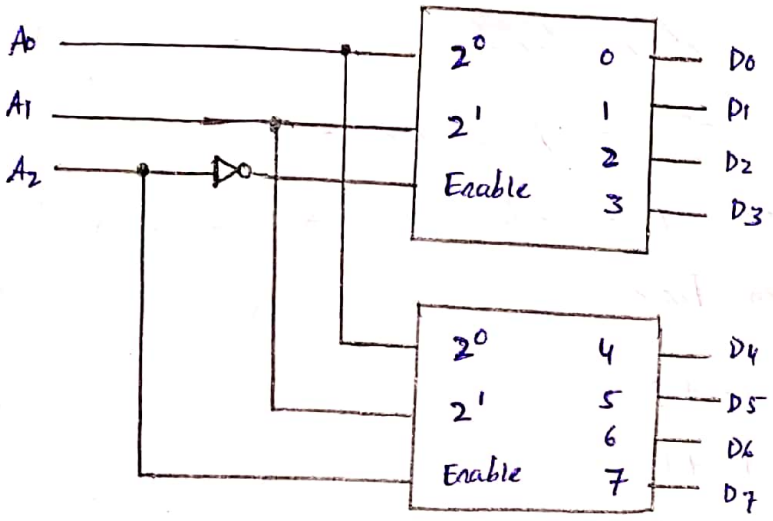
$D_6 = A_2 A_1 \bar{A}_0$

$D_7 = A_2 A_1 A_0$

Logic Diagram:



4) Draw 3×8 decoder using 2×4 decorder only



When $A_2 = 0$ the upper decoder enabled and the lower decorder is disabled. The outputs of the lower decorder becomes inactive and equal to zero. The upper decoder generates the minterms D_0, D_1, D_2, D_3 using the values A_0 & A_1.

→ When $A_2 = 1$ the enabled conditions are reversed. minterms D_4, D_5, D_6, D_7 are generated.

Encorder :-

The reverse operations of decorder

ex :- octal to Binary (8 i/ps) & (3 i/ps)

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_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

→ An encoder is the digital function that performs the inverse operation of a decoder.

→ Encoder has 2^n i/p lines and n o/p lines.

→ It is assumed that only one i/p has a value of 1 at any given time.

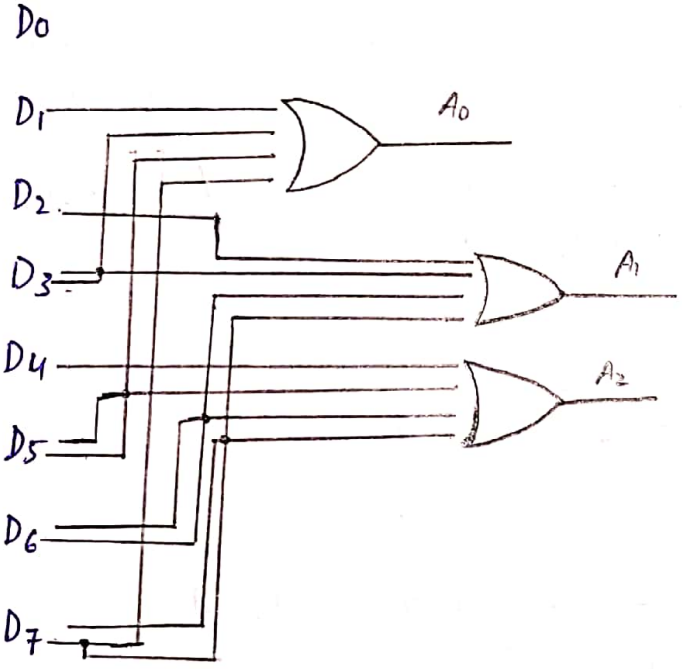
Boolean exp :

$$A_0 = D_1 + D_3 + D_5 + D_7$$

$$A_1 = D_2 + D_3 + D_6 + D_7$$

$$A_2 = D_4 + D_5 + D_6 + D_7$$

Logic diagram :-



Difference b/w Encoder and decoder :-

parameters	Encoder	Decoder
Input lines	2^n	n
output lines	n	2^n
operation	simple	complex
Applications	small, video encoders	Microprocessors, Memory chips

priority Encoder :-

A priority Encoder is a combinational circuit that implements a priority. the operation of priority encoder is such that if 2 or more i/p's are equal to 1 at the same time, than the i/p having the highest priority take precedence

4 i/p priority encoder

Inputs				outputs		
D_3	D_2	D_1	D_0	A_1	A_0	V
0	0	0	0	x	x	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

D_3 - highest priority
o/p of $A_1 A_0 = 11$

1) i/p D_3 has highest priority so regardless of the values of other i/p's when D_3 i/p is 1 the o/p for A_1, A_0 is 1,1 from this obtain the last row of the table.

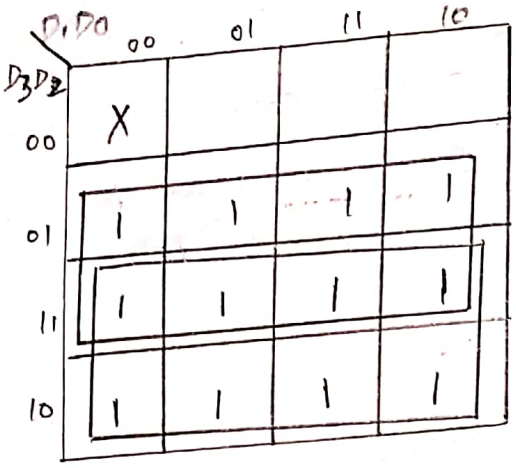
2) D_2 has the next priority level. the o/p is 1,0 if D_2 is 1. provided that $D_3 = 0$ regardless of the values of the lower priority i/p from this we obtained the 4th row of the table.

3) the valid o/p designated by V is said to ~~be~~ 1 only when 1 or more of the i/p's are equal to 1. if all the i/p's are zero than V=0 and the 2 o/p's A_1 & A_0 are not used (X) and are specified as don't care conditions in the output part of the table.

1618119

K-Map :-

A_1 =



$A_1 = D_2 + D_3$

A_1 = 4th row 2^2

0	1	X	X
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1

D_3	D_2	D_1	D_0
0	1	0	0
0	1	0	1
0	1	1	1
0	1	1	0
1	1	0	0
1	1	0	1
1	1	1	1
1	1	1	0

D_2

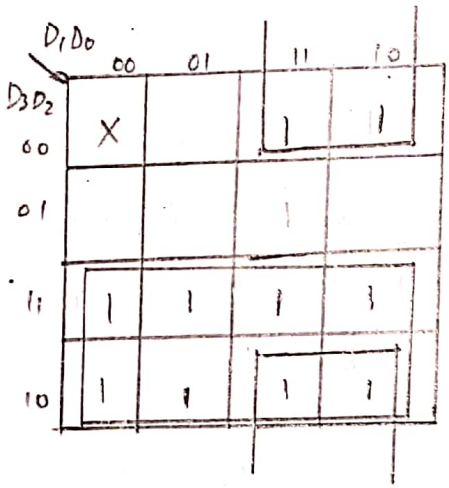
A_1 = 5th row

D_3	D_2	D_1	D_0	2^1
1	X	X	X	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

D_3	D_2	D_1	D_0
1	1	0	0
1	1	0	1
1	1	1	1
1	0	1	1
1	0	0	0
1	0	0	1
1	0	1	1
1	0	1	0

D_3

A_0 = A_0 = 3th row $2^1 = 2$ A_0 = 5th row $2^3 = 8$



$A_0 = D_1\bar{D}_2 + D_3$

0	0	1	X
0	0	1	0
0	0	1	1

D_3	D_2	D_1	D_0
1	1	0	0
1	1	0	1
1	1	1	1
1	1	1	0
1	0	0	0
1	0	0	1
1	0	1	1
1	0	1	0

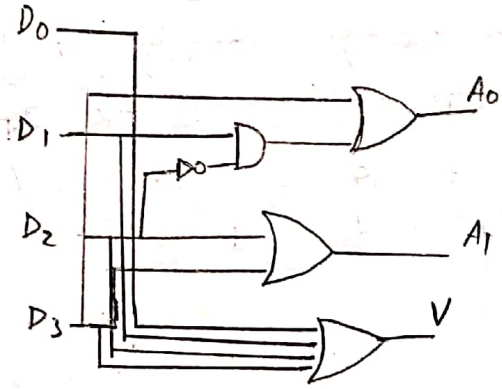
D_3

D_3	D_2	D_1	D_0
0	0	1	1
0	0	1	0
1	0	1	1
1	0	1	0

$\bar{D}_2 D_1$

$V = D_0 + D_1 + D_2 + D_3$

Logic Diagram :-



Multiplexer (MUX) :-

A multiplexer is the combinational circuit that selects binary information from 1 of many input line and directs the information to a single output line. The selection of a particular input line is controlled by a set of input variables called selection inputs.

There are 2^n input lines and n selection inputs whose input combination determines which input is selected.

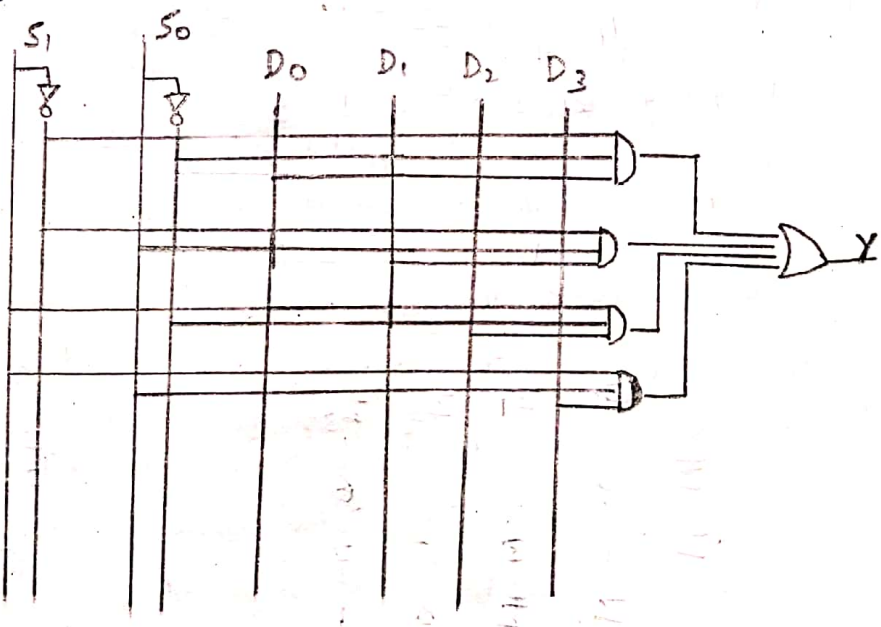
ex:- 4 to 1 Line multiplexer (MUX)

the o/p's of AND gates are applied to a single OR gate to provide 1 line o/p.

Function Table :-

S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

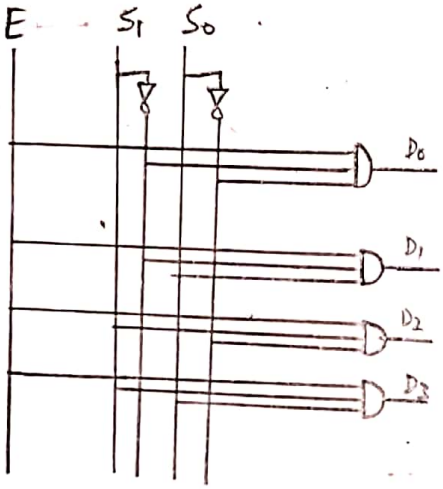
Selection i/p's :-



De Multiplexer :-

A DeMultiplexer is a digital function that performs the inverse operation of multiplexer i.e, A demultiplexer receives information from a single line and transmits it to one of 2^n o/p lines.

Ex:- 1:4 line Dimultiplexer



19/8/19

Q) Implement the function $F(X,Y,Z) = \Sigma m(1,2,6,7)$ using 4 to 1 line multiplexer

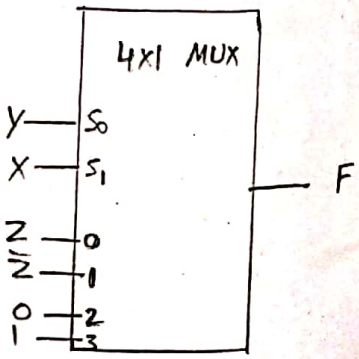
Sol 4 to 1 MUX

4 'i/p' => 2^2 select i/p's (X,Y)

Remaining inputs => 3-2 = 1 (Z)

(3 — no. of variables given in ques; 2 — select i/p's)

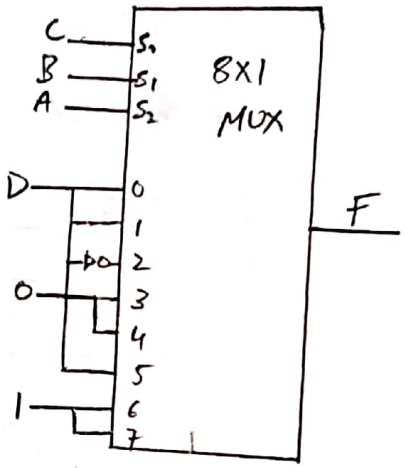
	Z_0	Z_1	Z_2	Z_3
$\bar{Z}$	0	(2)	4	(6)
Z	(1)	3	5	(7)
	Z	$\bar{Z}$	0	1



Q) Implement the function $F(A,B,C,D) = \Sigma m(1,3,4,11,12,13,14,15)$ using 8 to 1 MUX.

Sol) $8 \rightarrow 2^{(3)}$ sel i/p's (A,B,C)

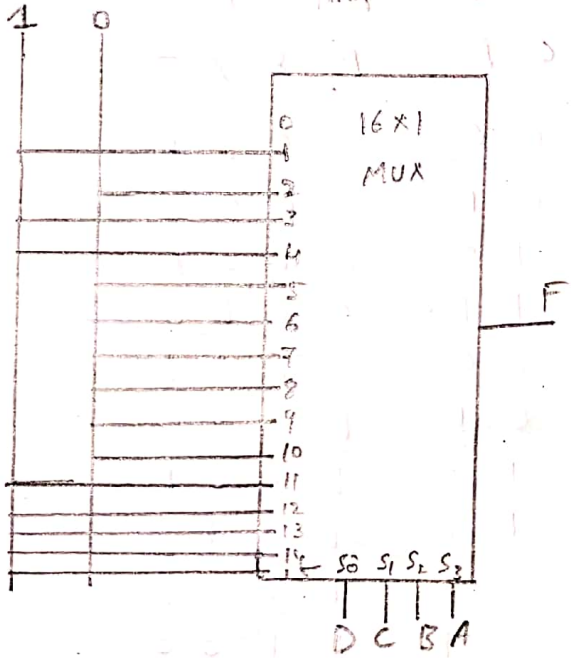
	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{D}$	0	2	(4)	6	8	10	(12)	(14)
D	(1)	(3)	5	7	9	(11)	(13)	(15)
	D	D	$\bar{D}$	0	0	D	1	1



Q) Implement the function $F(A,B,C,D) = \Sigma(1,3,4,11,12,13,14,15)$ using 16 to 1 MUX.

Sol) 16 to 1 MUX

16 i/p's $\Rightarrow 2^{(4)}$ - select i/ps



Q) Implement the function $F(w,x,y,z) = \Sigma m(0,3,4,7,9,13,15)$ using 4:1 MUX

Sol) 4 to 1 MUX

4 i/p's $= 2^2$ - select i/p's (w,x)

Remaining i/p's $\Rightarrow 4-2=2$ (y,z)

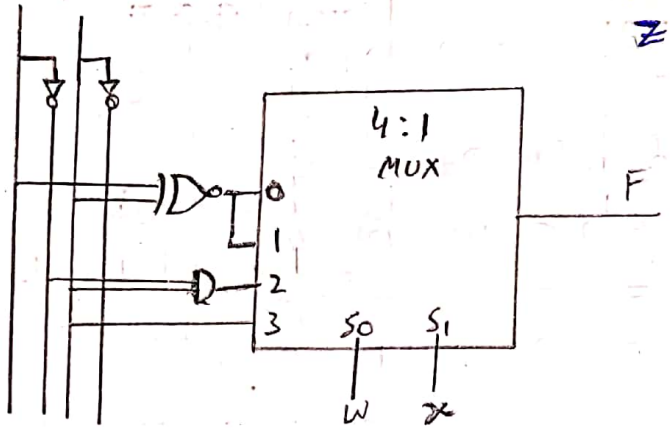
	Z_0	Z_1	Z_2	Z_3
$\bar{y}\bar{z}$	(0)	(4)	8	12
$\bar{y}z$	1	5	(9)	(13)
$y\bar{z}$	2	6	10	14
yz	(3)	(7)	11	(15)

$\bar{y}\bar{z}+yz$ $\bar{y}\bar{z}+yz$ $\bar{y}z$ $\bar{y}z+yz$

$y \odot z$ $y \odot z$ $\bar{y}z$ $z(y+\bar{y})$

$z(1)$

z



#S3

Code converters :-

BCD to Excess-3 (XS3)

Truth table :-

Decimal digit	Input (BCD) A	B	C	D	output (Excess-3) W	X	Y	Z	
0	0	0	0	0	0	0	1	1	-3
1	0	0	0	1	0	1	0	0	-4
2	0	0	1	0	0	1	0	1	-5
3	0	0	1	1	0	1	1	0	-6
4	0	1	0	0	0	1	1	1	-7
5	0	1	0	1	1	0	0	0	-8
6	0	1	1	0	1	0	0	1	-9
7	0	1	1	1	1	0	1	0	-10
8	1	0	0	0	1	0	1	1	-11
9	1	0	0	1	1	1	0	0	-12

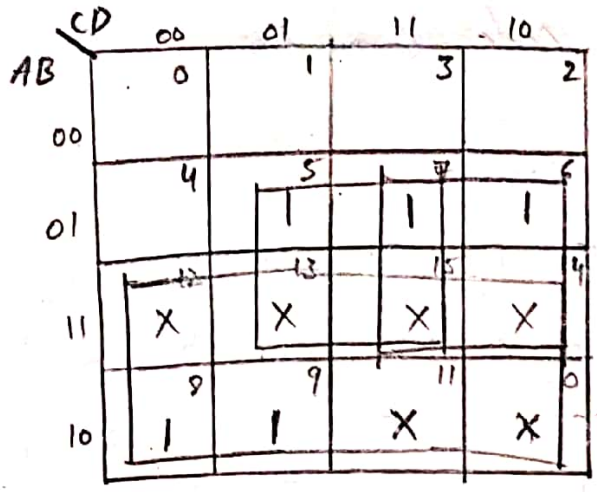
From the table 10, 11, 12, 13, 14, 15 are dont care

$W = \Sigma m\,(5,6,7,8,9) + \Sigma d\,(10,11,12,13,14,15)$

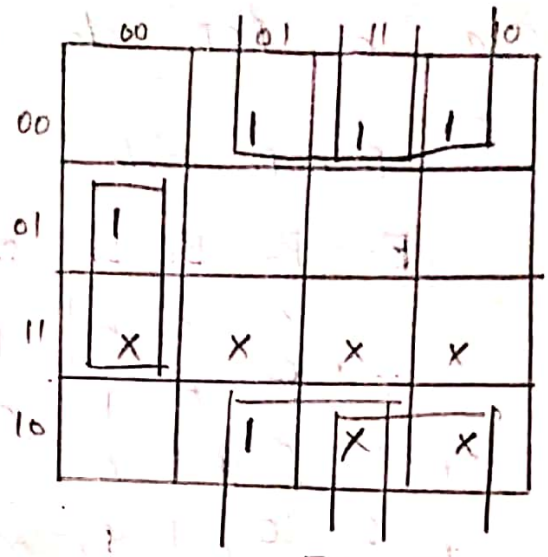
$X = \Sigma m\,(1,2,3,4,9) + \Sigma d\,(10,11,12,13,14,15)$

$Y = \Sigma m\,(0,3,4,7,8) + \Sigma d\,(10,11,12,13,14,15)$

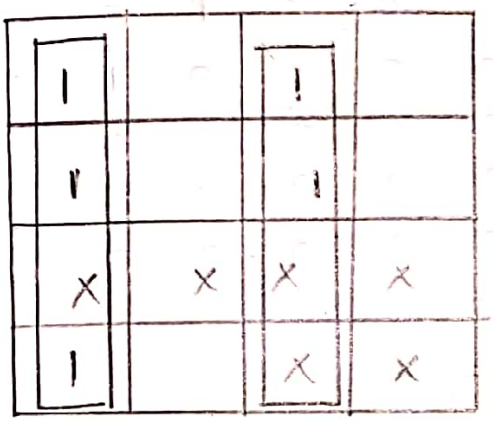
$Z = \Sigma m\,(0,2,4,6,8) + \Sigma d\,(10,11,12,13,14,15)$



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

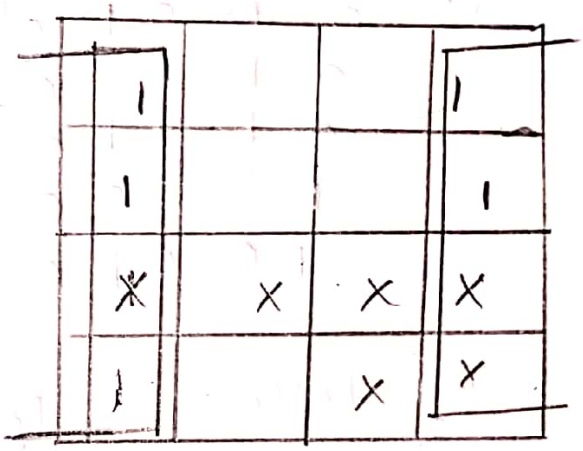


$X = \bar{B}C + \bar{B}D + B\bar{C}\bar{D}$

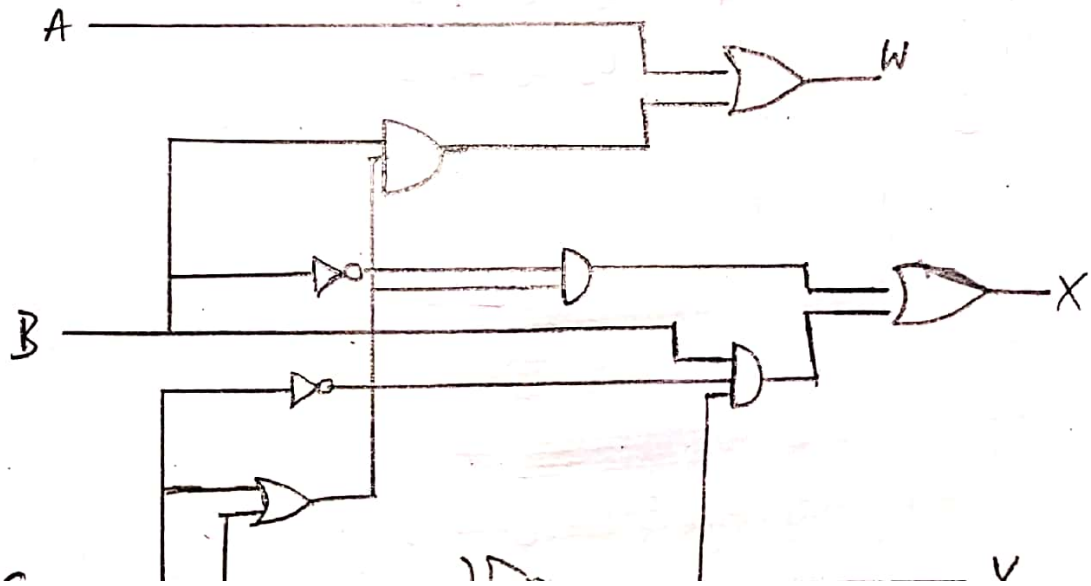


$Y = \bar{C}\bar{D} + CD$

$= \overline{C \oplus D}$



$Z = \bar{D}$



Ans 2

Q) Design a BCD to decimal code Converter

Sol

BCD code				Decimal									
B_3	B_2	B_1	B_0	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9
0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1

$D_0 = \overline{B_3}\,\overline{B_2}\,\overline{B_1}\,\overline{B_0}$

$D_1 = \overline{B_3}\,\overline{B_2}\,\overline{B_1}\,B_0$

$D_2 = \overline{B_3}\,\overline{B_2}\,B_1\,\overline{B_0}$

$D_3 = \overline{B_3}\,\overline{B_2}\,B_1\,B_0$

$D_4 = \overline{B_3}\,B_2\,\overline{B_1}\,\overline{B_0}$

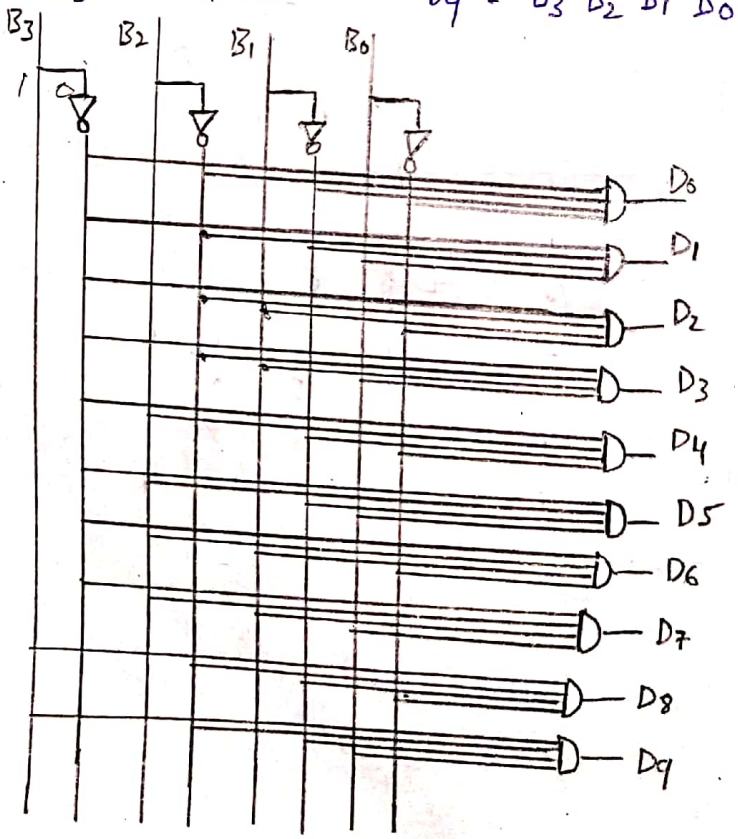
$D_5 = \overline{B_3}\,B_2\,\overline{B_1}\,B_0$

$D_6 = \overline{B_3}\,B_2\,B_1\,\overline{B_0}$

$D_7 = \overline{B_3}\,B_2\,B_1\,B_0$

$D_8 = B_3\,\overline{B_2}\,\overline{B_1}\,\overline{B_0}$

$D_9 = B_3\,\overline{B_2}\,\overline{B_1}\,B_0$

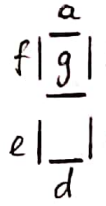


Q) Design a BCD to Seven segment Decorder

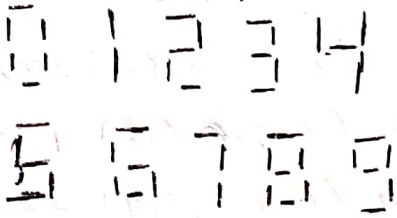
Sol

Fig:- Seven Segment Display

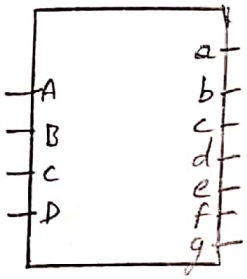
a) Segment designation



b) Numeric designation for display



code converter



Truth table :-

	BCD				Seven Segment Decorder						
	A	B	C	D	a	b	c	d	e	f	g
0-	0	0	0	0	1	1	1	1	1	1	0
1-	0	0	0	1	0	1	1	0	0	0	0
2-	0	0	1	0	1	1	0	1	1	0	1
3-	0	0	1	1	1	1	1	1	0	0	1
4-	0	1	0	0	0	1	1	0	0	1	1
5-	0	1	0	1	1	0	1	1	0	1	1
6-	0	1	1	0	1	0	1	1	1	1	1
7-	0	1	1	1	1	1	1	0	0	0	0
8-	1	0	0	0	1	1	1	1	1	1	1
9-	1	0	0	1	1	1	1	1	0	1	1
All other i/p's					10, 11, 12, 13, 14, 15						

from above table
10, 11, 12, 13, 14, 15 are assigned as 0's (in d's)

$a = \Sigma m(0,2,3,5,6,7,8,9) + \Sigma d(1,4,)$

$b = \Sigma m(0,1,2,3,4,7,8,9) + \Sigma d(5,6)$

$c = \Sigma m(0,1,3,4,5,6,7,8,9) + \Sigma d(2)$

$d = \Sigma m(0,2,3,6,8,9) + \Sigma d(1,4,5,7)$

$e = \Sigma m(0,2,6,8) + \Sigma d(1,3,4,5,7,9)$

$f = \Sigma m(0,4,5,6,9) + \Sigma d(1,2,3,7)$

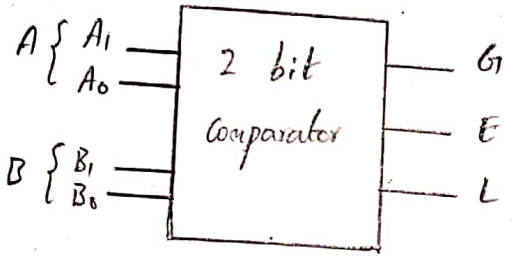
$g = \Sigma m(2,3,4,5,6,8,9) + \Sigma d(0,1,7)$

6/9/19

Arithmetic Comparision Circuits :-

2- bit Comparator :

2- bit Comparator Compares 2 binary numbers each of 2- bits and reduces their relation such as $=, >, <$ the other. The figure below shows the block diagram of a 2-bit Comparator which has 4 i/p's and 3 o/p's



The first no. A is designated as $A = A_1A_0$

The 2nd no. B is designated as $B = B_1B_0$

This Comparator produces 3 o/p's as G (G=1 if A > B)
E (E=1 if A = B)
L (L=1 if A < B)

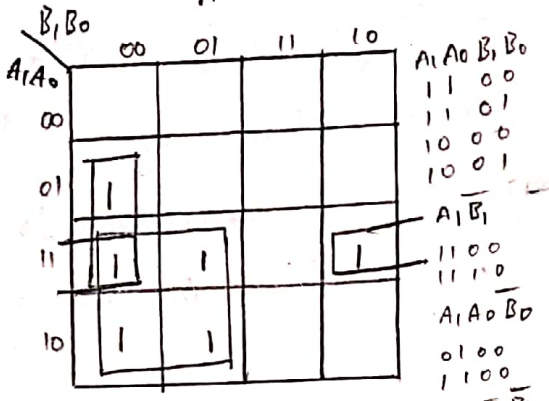
Truth table :-

I/p's				o/p's		
				G	E	L
A_1	A_0	B_1	B_0	$A > B$	$A = B$	$A < B$
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

00 - 1
01 - 1
10 - 2
11 - 3

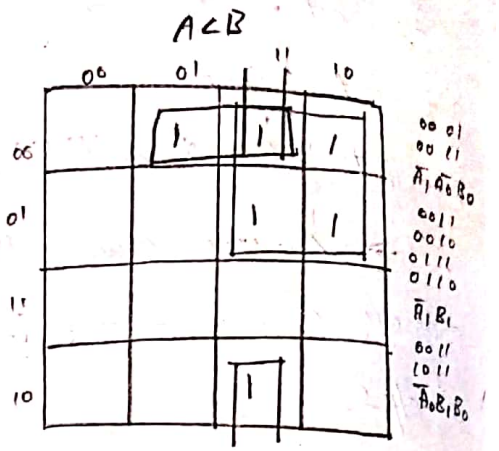
K-map :-

A > B



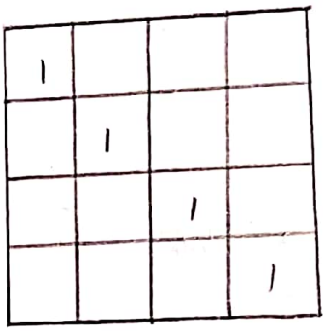
$G = A_1\bar{B}_1 + A_1A_0\bar{B}_0 + A_0\bar{B}_1\bar{B}_0$

A < B



$L = \bar{A}_1B_1 + \bar{A}_1\bar{A}_0B_0 + \bar{A}_0B_1B_0$

A = B



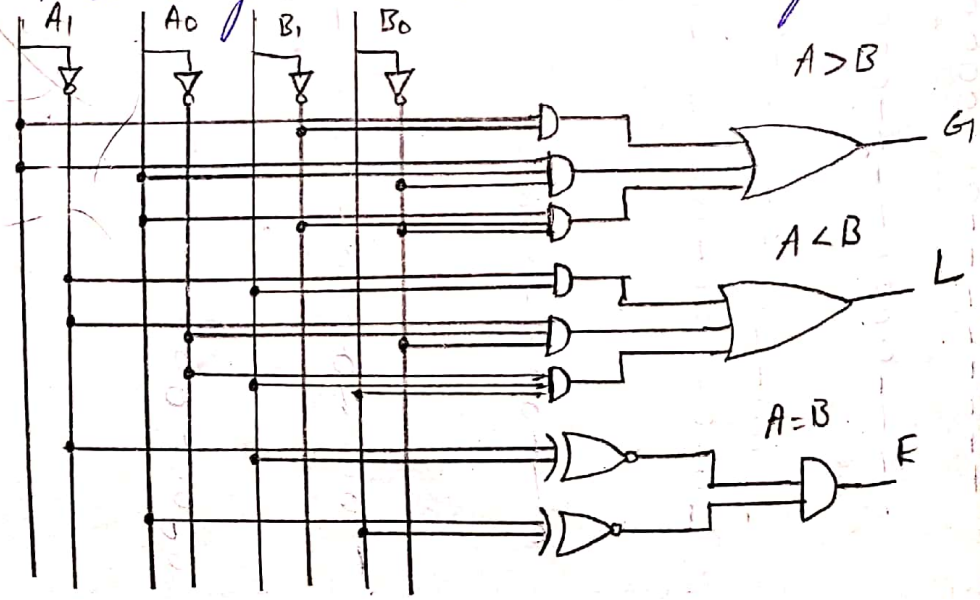
$E = \bar{A}_1\bar{A}_0\bar{B}_1\bar{B}_0 + \bar{A}_1A_0\bar{B}_1B_0 + A_1A_0B_1B_0 + A_1\bar{A}_0B_1\bar{B}_0$

$E = \bar{A}_1\bar{B}_1(\bar{A}_0\bar{B}_0 + A_0B_0) + A_1B_1(A_0B_0 + \bar{A}_0\bar{B}_0)$

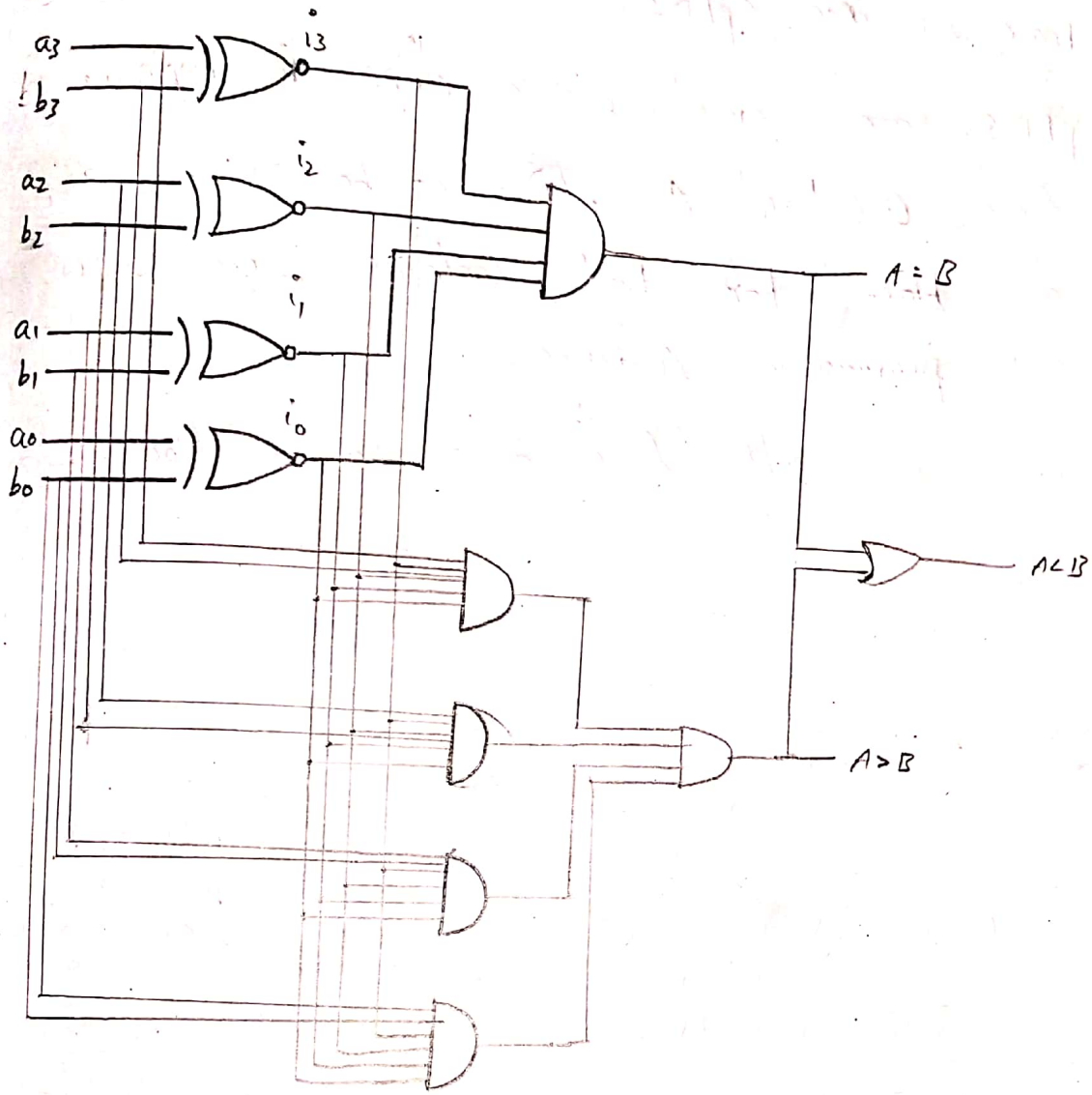
$E = (\bar{A}_1\bar{B}_1 + A_1B_1)(\bar{A}_0\bar{B}_0 + A_0B_0)$

$E = (A_1 \odot B_1)(A_0 \odot B_0)$

By using above Boolean expression for each o/p the logic diagram can be implemented by using 4 not gates, 7 AND gates, 2 OR gates and 2 Exclusive NOR gates.



4 - bit Comparator :-



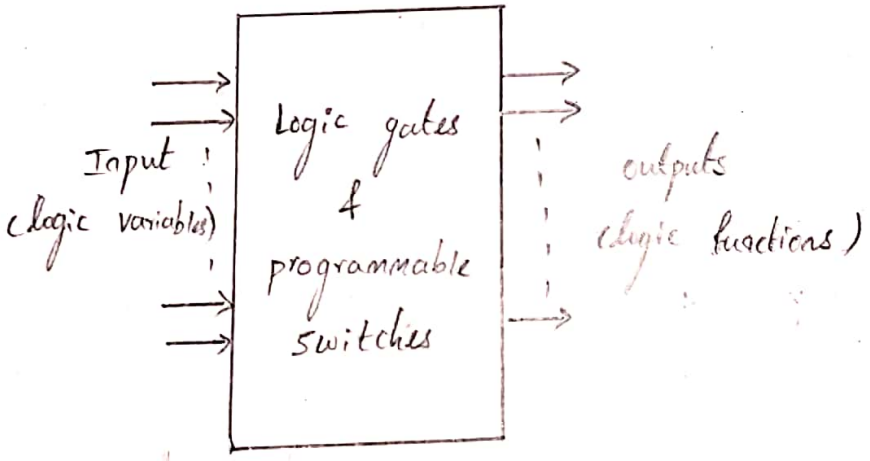
9/9/19

UNIT - III

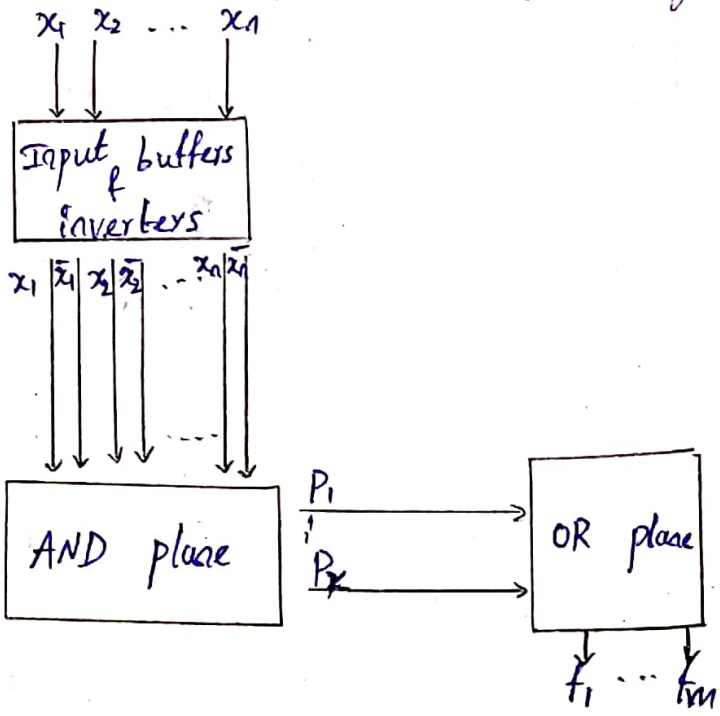
Design of Combinational Circuits using programmable Logic Devices (PLD's)

PLD's are general purpose chips for implementing logic circuitary. A PLD can be viewed as a black box that contains logic gates and programmable switches.

fig :- programmable logic device as a black box:.



General structure of programmable Logic Array (PLA):



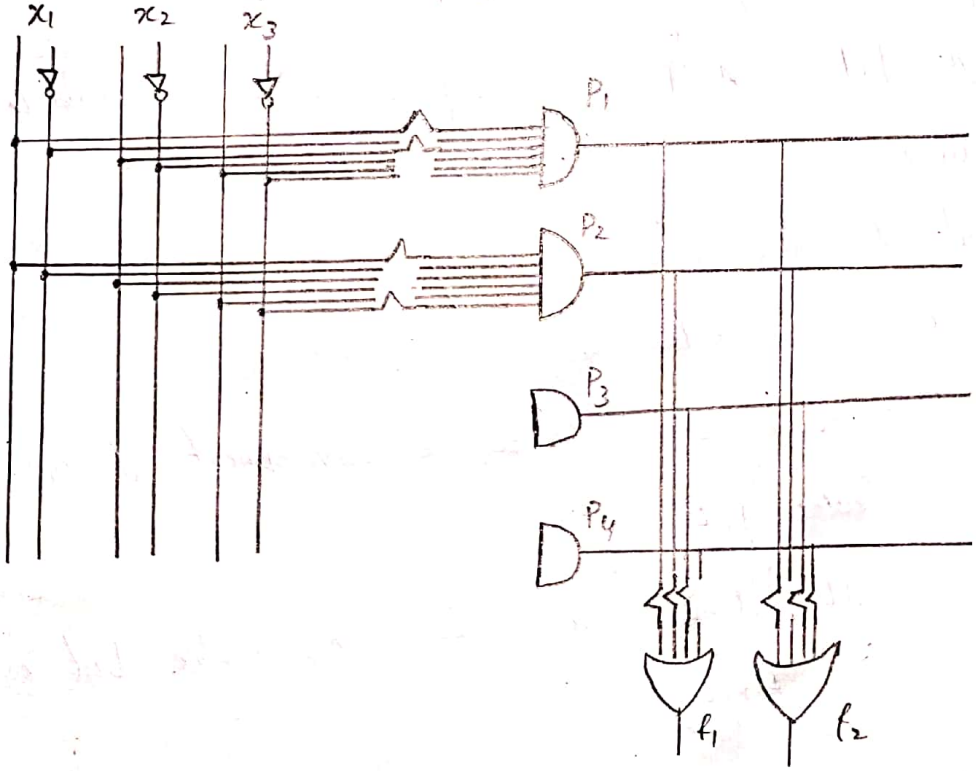
→ Several types of PLD's are available. The 1st developed was PLA. The general structure of PLA is shown in figure. the PLA i/p's $x_1, x_2 \cdots x_n$ pass through a set of buffers (which provide both true and compliment of each i/p) into a circuit block called AND plane (or) AND Array.

→ The AND array produces a set of product terms $P_1, P_2 \ldots P_k$ the product terms serves as i/p's to OR plane. which produces the o/p $f_1, f_2 \ldots f_m$.

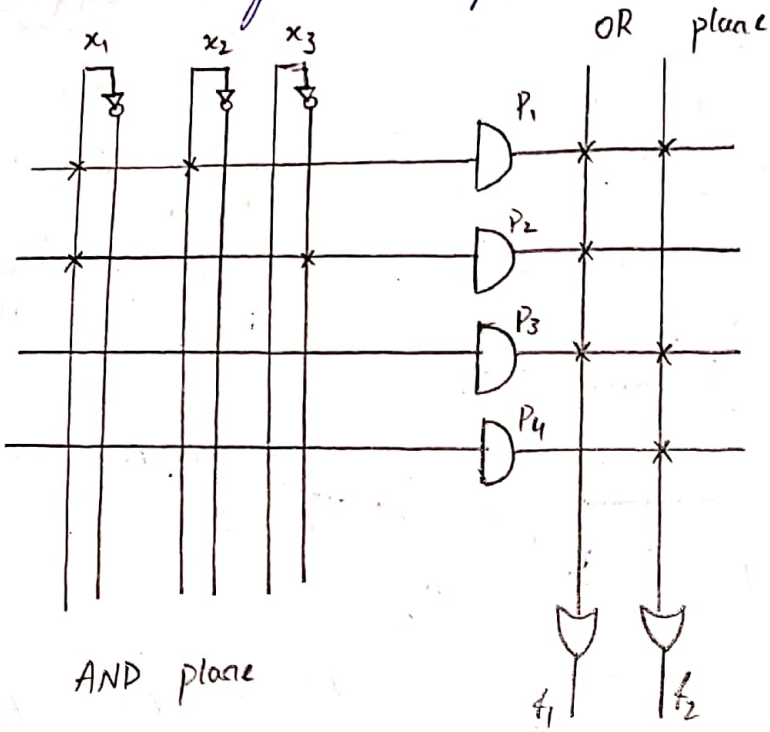
Q) Draw the gate level diagram of PLA for the function

$$f_1 = \underset{P_1}{x_1 x_2} + \underset{P_2}{x_1 \bar{x}_3} + \underset{P_3}{\bar{x}_1 \bar{x}_2 x_3}$$

$$f_2 = x_1 x_2 + \bar{x}_1 \bar{x}_2 x_3 + x_1 x_3$$



Schematic Diagram for PLA :-



11/9/19

programmable Array logic (PAL) :-

In PLA both AND, OR planes are programmable

draw backs :-

1) Hard to fabricate correctly
2) These reduce the speed performance

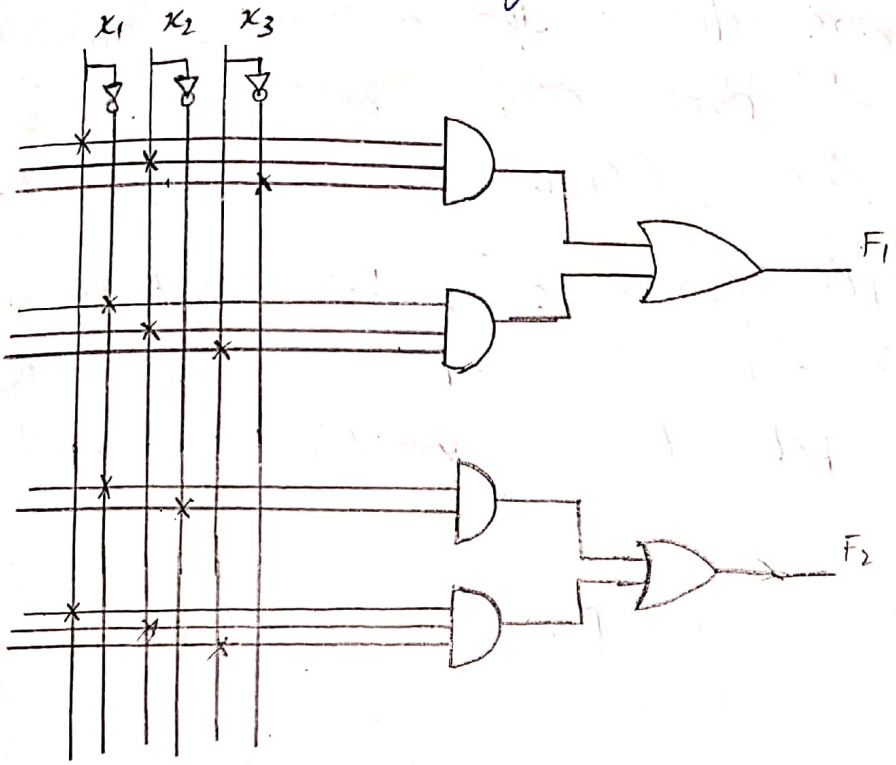
→ These drawbacks led to be development of similarly device called 'PAL'.

→ In PAL 'AND' plane is programmable but 'OR' plane is fixed.

Q1 Realize 2 logic functions

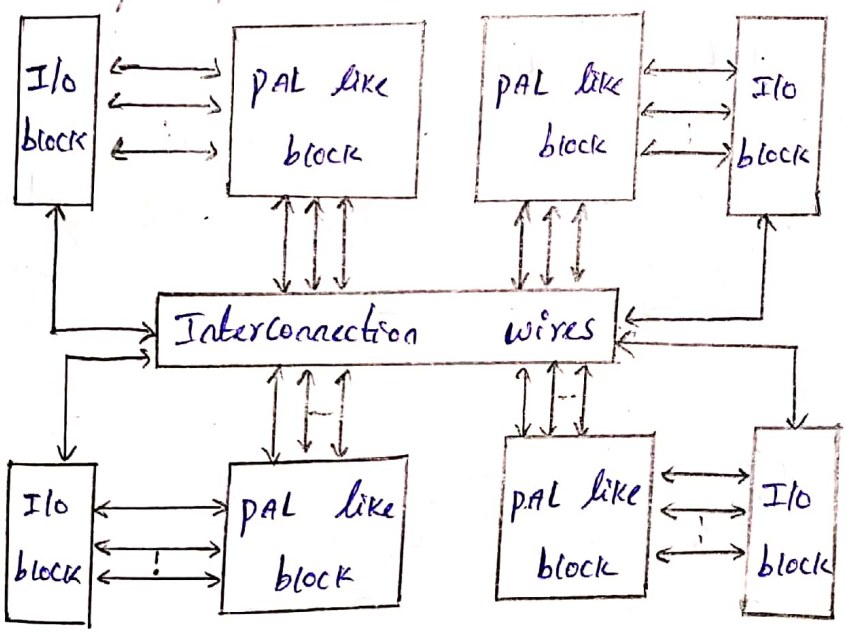
$f_1 = x_1 x_2 \bar{x}_3 + \bar{x}_1 x_2 x_3$

$f_2 = \bar{x}_1 \bar{x}_2 + x_1 x_2 x_3$ using PAL



Complex programmable Logic Devices (CPLD's)

multiple circuit blocks in single chip



→ PLA's & PAL's chips are limited modest sizes supporting a combine no. of I/p's + o/p's of not more than 32. for implementation of circuits that require more i/p's & o/p's a more sutlesticated types of chips called complex programmable logic devices (CPLD) can be used.

→ A CPLD comprises of multiple circuit blocks on a single chip. Each circuit block is similar to PLA or PAL. the structure is shown in above figure.

→ It includes 4 PAL like blocks that are connecte to a set of interconnection wire each PAL like block is also connected to a sub circuit called I/O block which is attached to a no. of chips i/p & o/p pins

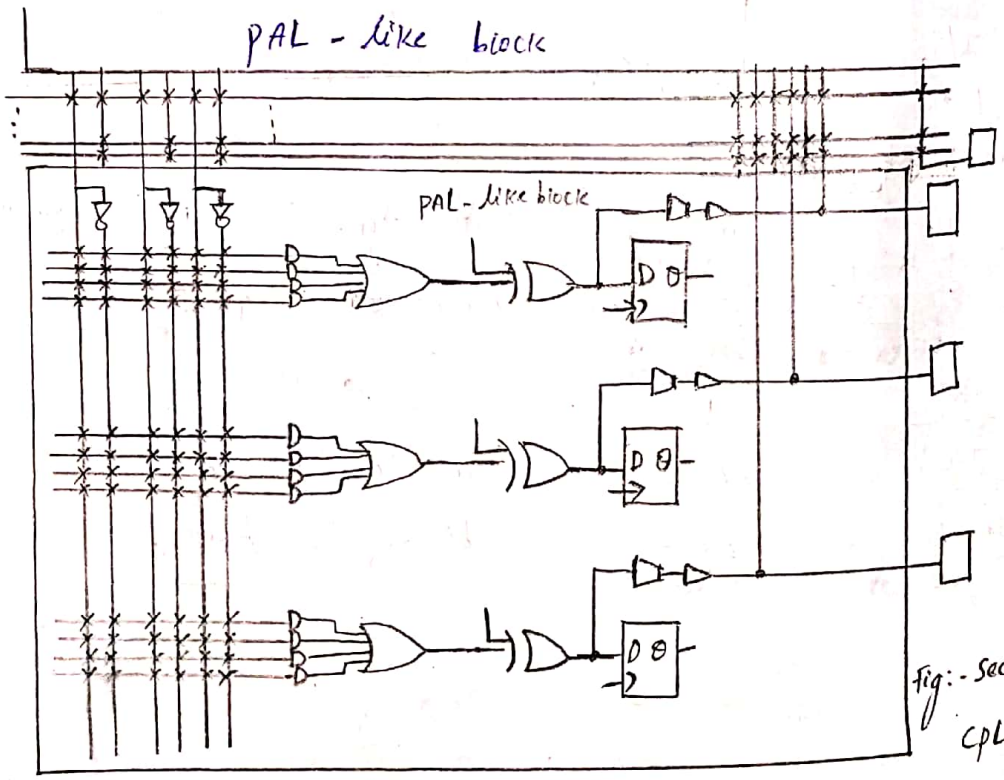


fig:- Section of CPLD

13/9/19

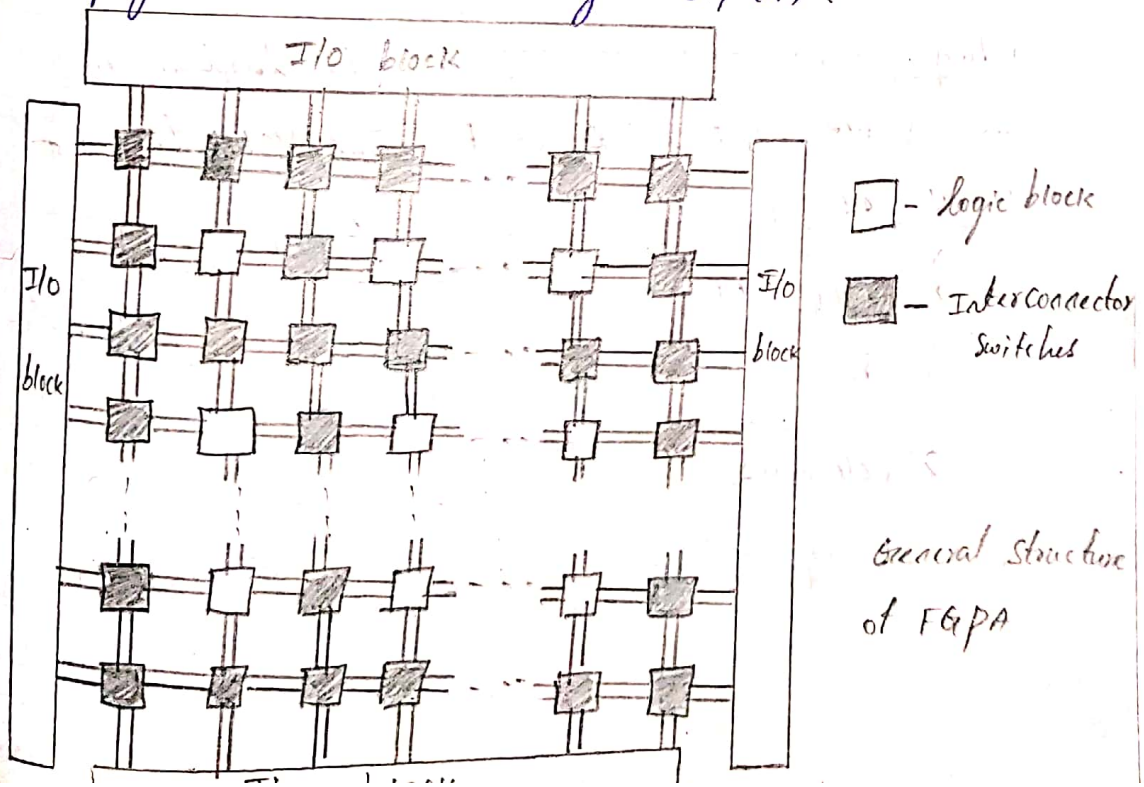
The PAL like block includes 3 macro cells each consisting of 4 i/p OR gate. The OR gate o/p is connected to exclusive OR. 1 i/p to the exclusive OR gate can be connected to 1 or 0. If connected to '1' than XOR gate compliments OR gate o/p. if connected to '0' then XOR gate has no effect.

-) The macro cell also includes a flip-flop, a multiplexer and tri-state buffer. The flip-flop is used to store the o/p value produced by OR gate.

#5 Applications of CPLD:-

1) Used for simple Applications like address, decoding.
2) Used in digital design to perform the functions of Boot loader.
3) Loading configuration data.
4) CPLD are ideal for high performance control applications

Field programmable Gate Arrays (FPGA):-



General Structure of FGPA

A filled programmable gate array is a programmable logic device that supports implementation of relatively large logic circuit. FPGA do not contain AND, OR plane.

→ FPGA contains 3 main types of resources

1) logic block
2) I/O block
3) Interconnection wires and switches

→ The logic blocks are arranged in a 2-D array I/o blocks are used for connecting to the pins of the package. The interconnection wires are organized as horizontal and vertical routing channel between rows and column of logic blocks.

~~Each~~ The most commonly used logic block is lookup table (LUT) which contains storage cells that are used to implement a small logic function. Each cell is capable of holding a single logic value either 0 or 1. The stored value is produced as the o/p of storage cell.

2 i/p, 3 i/p - LUT

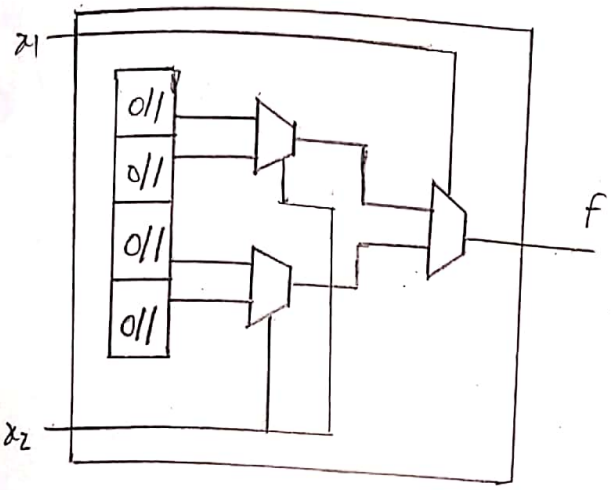
Ex:- 2 input LUT

$f_1 = \bar{x}_1 \bar{x}_2 + x_1 x_2$

$2^2 = 4$ rows

4 storage cells

a) circuit for a 2-i/p LUT

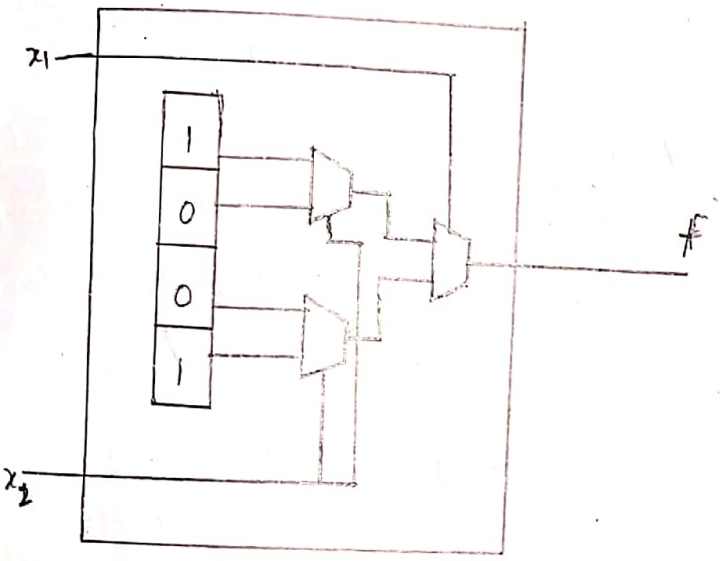


b) $f_1 = \bar{x}_1 \bar{x}_2 + x_1 x_2$

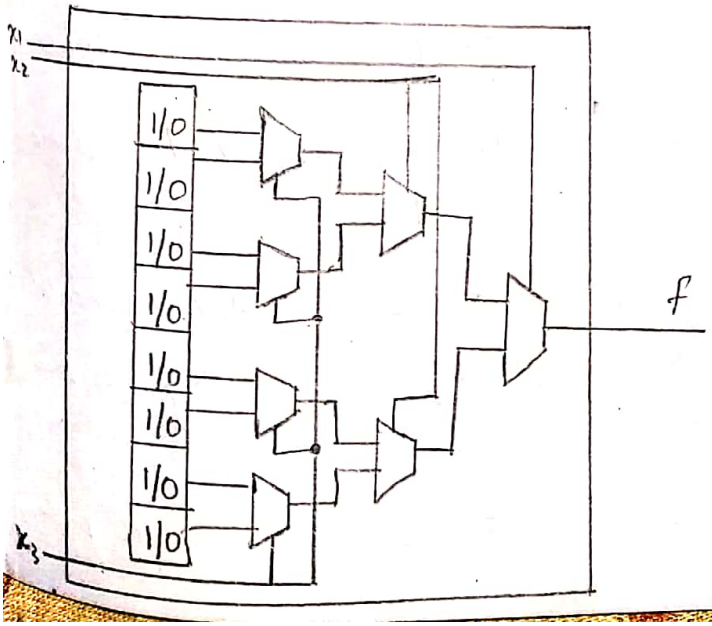
Truth table :-

x_1	x_2	f_1
0	0	1
0	1	0
1	0	0
1	1	1

c) Storage cell contents in LUT



ex:- 3 input LUT



$f = \bar{x}_1 \bar{x}_2 \bar{x}_3 + x_1 x_2 x_3$

Truth table :-

x_1	x_2	x_3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

16/9/19

Differences blw CPLD and FPGA

CPLD	FPGA
1) An integrated circuit that helps to implement digital system	1) An integrated ckt that is designed to be configured by a customer or Designer after manufacturing
2) provides min logic resources	2) provides a massive amount of logic resources and storage elements to create complex systems.
3) Cost effective	3) Expensive than CPLD
4) Made up of ~~lager~~ blocks	4) Made up of tiny logic blocks.
5) Uses eeprom (not volatile)	5) uses RAM (volatile)
6) Easier to predict delays.	6) Not easy to predict delays.
7) Consumes low power	7) Consumes more power

8) more secure

9) Suitable for small to medium scale applications.

10) It has PAL like architecture

11) It contains only a few blocks of logic (1000's)

8) less secure

9) Suitable for complex applications.

10) It has gate array like architecture

11) It contains width of tiny logic blocks (100000)

8.8. Verilog HDL :- (Hardware discription language)

It is a case sensitive language all key words are in lower case

ports :-

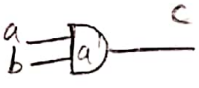
provides a mean for a module to communicate through i/p & o/p.

module :-

It is defined as an element or as a collection of lower level design blocks.

Verilog code for Basic gates :-

AND



Truth table :-

a	b	c
0	0	0
0	1	0
1	0	0
1	1	1

code :-

```
module and1 (a,b,c);
// ports declaration
input a,b;
output c;
and a1(c,a,b);
endmodule;
```

OR gate :-

TT :-

a	b	c
0	0	0
0	1	1
1	0	1
1	1	1

code :-

```
module or1 (a,b,c);
// ports declaration
input a,b;
output c;
or o1(c,a,b);
endmodule;
```

XOR :-

$x \oplus y = x\bar{y} + \bar{x}y$

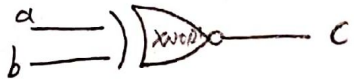
Truth Table :-

a	b	c
0	0	0
0	1	1
1	0	1
1	1	0

code :-

```
module xor1 (a,b,c);
// ports declaration
input a,b;
output c;
xor s1 (c,a,b);
end module;
```

XNOR :-



Truth table :-

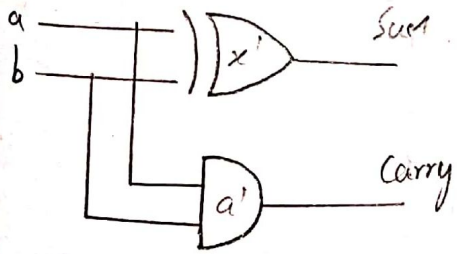
a	b	c
0	0	1
0	1	0
1	0	0
1	1	1

code :-

```
module xnor (a,b,c);
// ports declaration
 input a,b;
 output c;
 xNor s1 (c,a,b);
 end module;
```

Verilog code for adders :-

Half Adder :-



Sum = $a \oplus b$

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

Carry = $a.b$

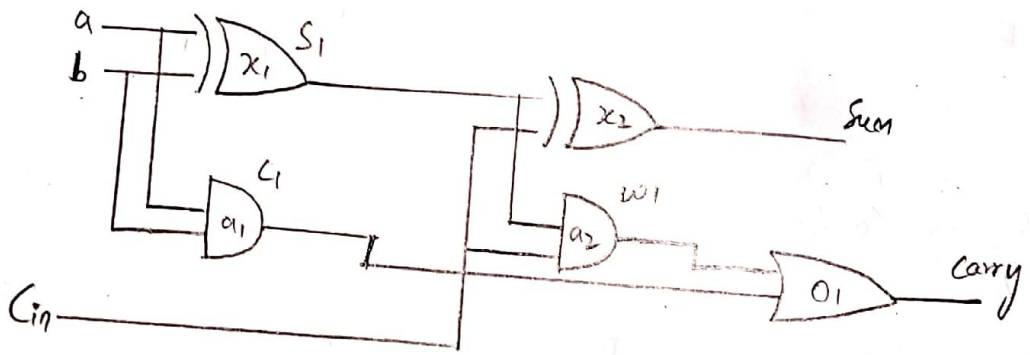
Truth Table :-

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

Code :-

```
module halfadder (a,b, sum, carry)
input (a,b);
output sum, carry;
xor x1 (sum,a,b);
and a1 (carry,a,b);
end module;
```

Full Adder :-



Truth Table :-

a	b	C_{in}	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

$Sum = x \oplus y \oplus C_{in}$

$carry = C_{in}(x \oplus y) + xy$

18/9/19

Code :-

```
module fulladder (a, b, cin, sum, carry)
input a, b, cin);
output sum, carry;
wire s1, c1, w1;
xor x1 (s1, a, b);
and a1 (c1, a, b);
xor x2 (sum, s1, cin);
and a2 (w1, s1, cin);
or o1 (carry, w1, c1);
endmodule;
```

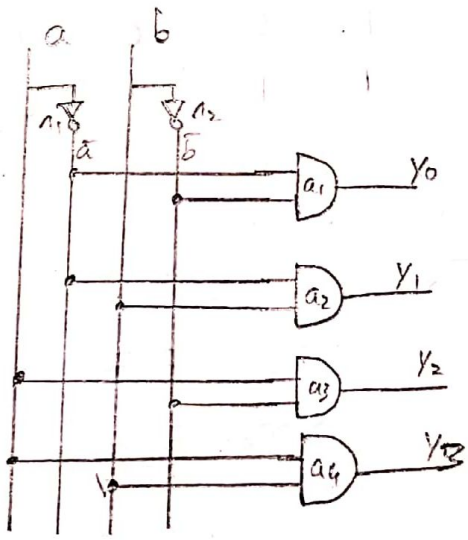
Verilog code for decorders :-

2:4 Decorder :-

Truth Table

a	b	Y_3	Y_2	Y_1	Y_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

circuit :-



Code :-

```
module 2:4 decoder (a, b, Y);
input a, b;
output [3:0]Y;
wire abar, b bar;
not n1 (abar, a);
not n2 (bbar, b);
and a1 (Y[0], abar, bbar);
```

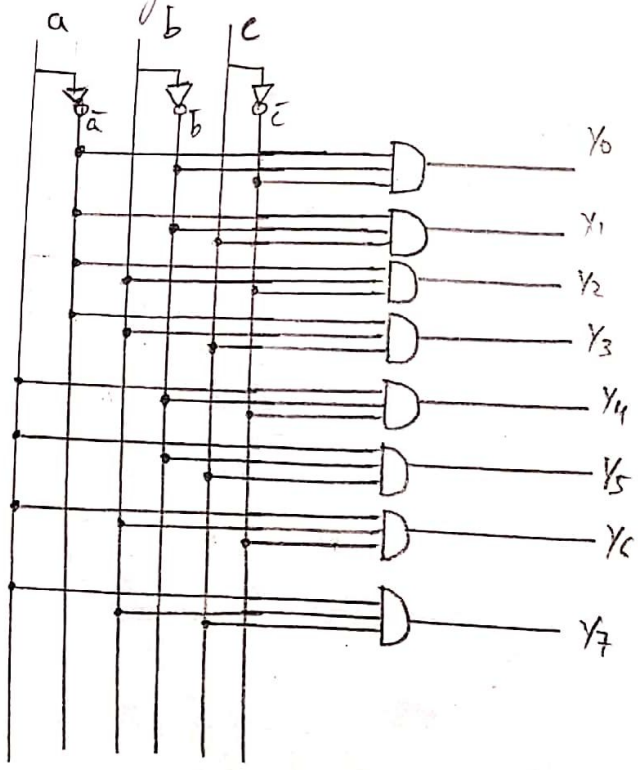
```
and   a2 [y[1], abar, b);
and   a3 [y[2], a, bbar);
and   a4 [y[3], a, b);
endmodule;
```

3:8 decoders :-

Truth Table

a	b	c	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	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

Circuit diagram :-



Code :-

```
module 3:8 decoder (a,b,c,y);
input a,b,c;
output [7:0]y;
wire abar, b bar, c bar;
not n1 (abar, a);
not n2 (b bar, b);
not n3 (c bar, c);
and a1 (y[0], a bar, b bar, c bar);
and a2 (y[1], a bar, b bar, c);
and a3 (y[2], a bar, b bar, c bar);
and a4 (y[3], a bar, b, c);
and a5 (y[4], a, b bar, c bar);
and a6 (y[5], a, b bar, c);
and a7 (y[6], a, b, c bar);
and a8 (y[7], a, b, c);
end module;
```

Universal Gates :-

i) NAND :- 

ii) NOR :- 

AND			NAND
a	b	f	f'
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

OR			NOR
a	b	f	f_1
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

20191119

Using NAND GATES :-

i, In Sop, you will get one NAND gate at the last, so no need to compliment the literal.

ii, In pos, you will get two NAND gates at the last, so compliment the literals.

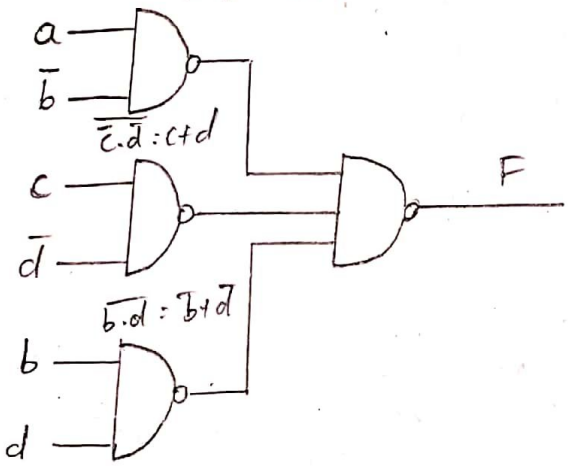
Using NOR GATES :-

i, In Sop, two NOR gates at the last, so compliment the literals.

ii, In pos, one OR gate at the last so no need to compliment the literals

Ex :- 1, $a\bar{b} + c\bar{d} + bd$ realize using NAND gates only

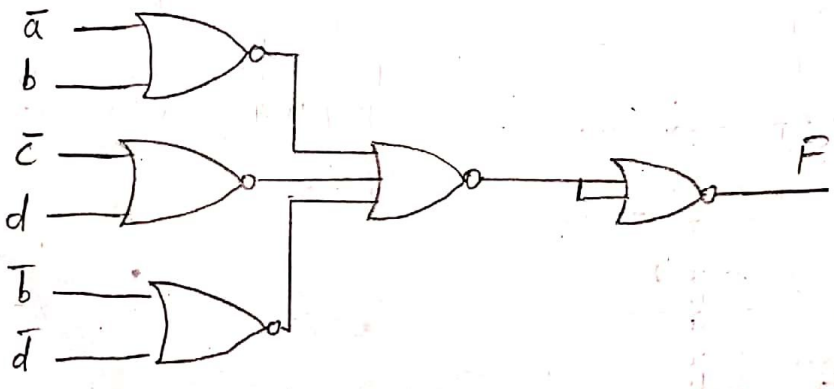
$\overline{a.\bar{b}} = \bar{a}+b$



$\overline{(\bar{a}+b).(c+d).(\bar{b}+\bar{d})}$

$a.\bar{b} + c\bar{d} + bd$

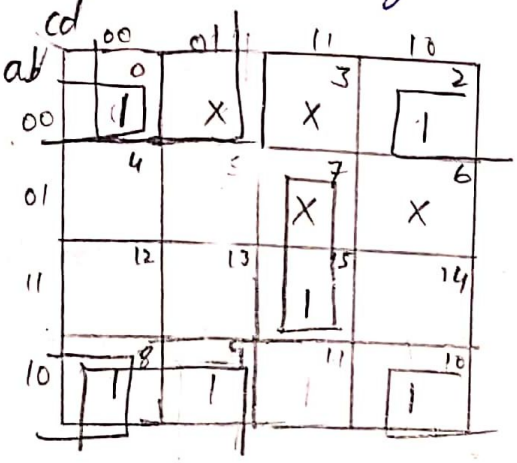
2, NOR gates only



1) Find the minimal Sop & pos form for the function by map method.

$F(a,b,c,d) = \Sigma m(0,2,8,9,10,15) + \Sigma d(1,3,6,7)$

realize the logic circuit using NAND gates only.



a	b	c	d
0	0	0	0
0	0	1	0
1	0	0	0
1	0	1	0

$\bar{b}\bar{d}$

a	b	c	d
0	0	0	0
0	0	0	1
1	0	0	0
1	0	0	1

$\bar{b}\bar{c}$

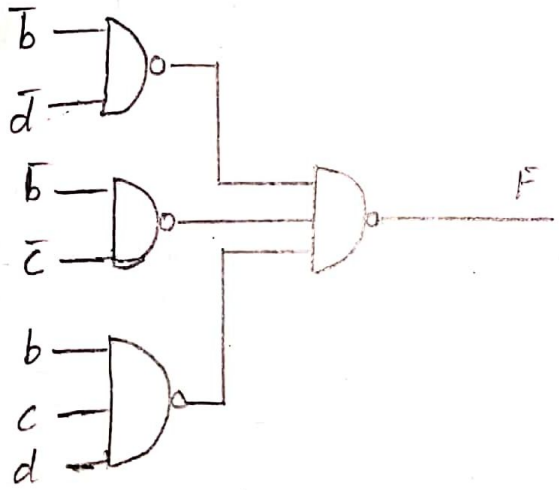
a	b	c	d
0	1	1	1
1	1	1	1

bcd

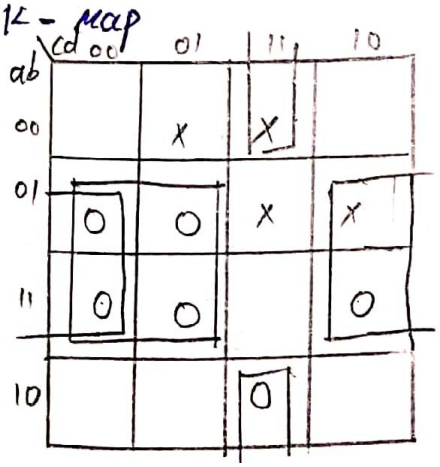
Sop = $F = \bar{b}\bar{d} + \bar{b}\bar{c} + bcd$

~~pos = $F = (b+d)\cdot(b+c)\cdot(\bar{b}+\bar{c}+\bar{d})$.~~

NAND :- Sop



pos :- K - map



a	b	c	d
0	1	0	0
1	1	0	0
0	1	0	1
1	1	0	1

$\bar{b}+c$

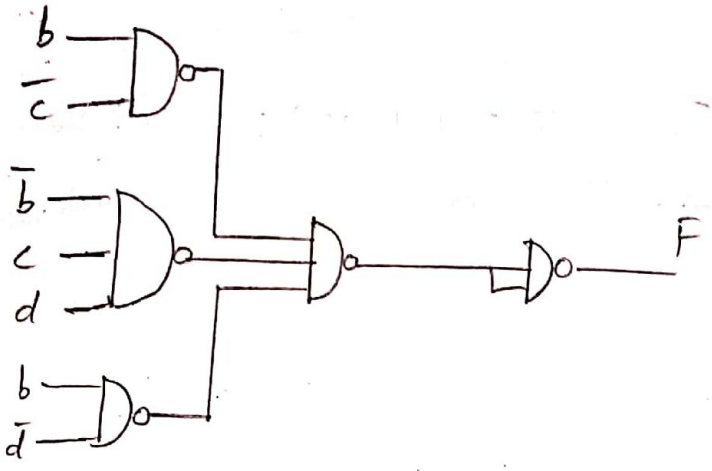
a	b	c	d
0	0	1	1
1	0	1	1

$b+\bar{c}+\bar{d}$

a	b	c	d
0	1	1	0
0	1	0	0
1	1	1	0
1	1	0	0

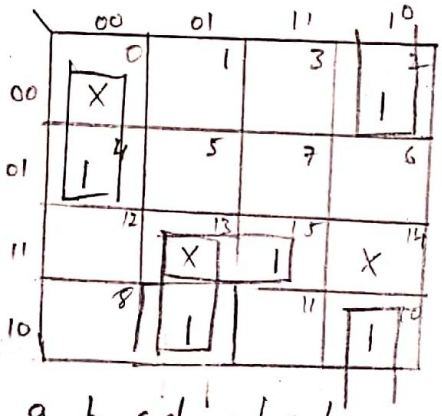
$\bar{b}+d$

$F = (\bar{b}+c)(b+\bar{c}+\bar{d})(\bar{b}+d)$



2) Find the minimal sop & pos for fuction by map method $F(a,b,c,d) = \Sigma m(2,4,9,10,15) + \Sigma d(0,13,14)$ realize the logic circuit using NOR gates only.

Sol) sop :-



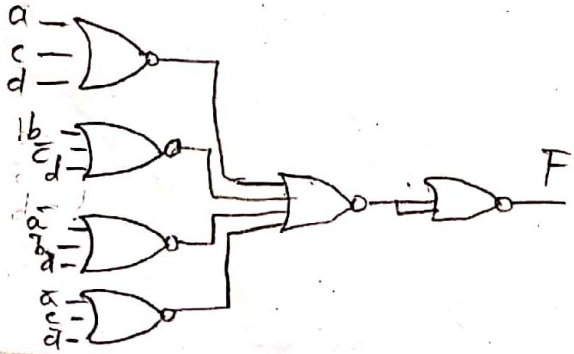
a b cd a b c d
0 0 0 0 0 0 1 0
0 1 0 0 1 0 1 0

$\bar{a}\bar{c}\bar{d}$ $\bar{b}c\bar{d}$

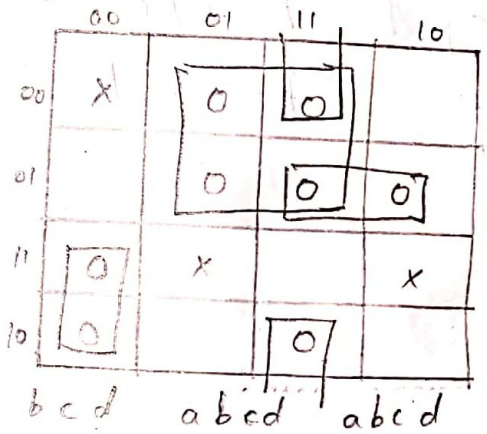
a b c d a b c d
1 1 0 1 1 1 0 1
1 1 1 1 1 0 0 1

abd $a\bar{c}d$

$F = \bar{a}\bar{c}\bar{d} + \bar{b}c\bar{d} + abd + a\bar{c}d$



pos :-



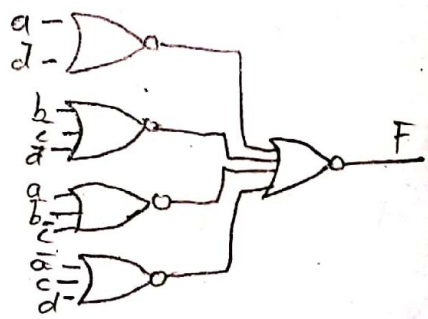
a b c d a b c d a b c d
0 0 0 1 0 0 1 1 0 1 1 1
0 0 1 1 1 0 1 1 0 1 1 0
0 1 0 1 $\bar{b}+\bar{c}+\bar{d}$ $a+\bar{b}+\bar{c}$
0 1 1 1

$a+\bar{d}$

a b c d
1 1 0 0
1 0 0 0

$\bar{a}+c+d$

$F = (a+\bar{d}).(\bar{b}+\bar{c}+\bar{d}).(a+\bar{b}+\bar{c})(\bar{a}+c+d)$



25/9/19

UNIT - IV

Flip flops :-

Difference b/w Latch and flip flops :-

1) L A Latch is a electronic sequential logic circuit used to store information in an ~~a~~ Asincronous arrangement.

F A flip flop is a electronic sequential logic circuit used to store information in an sincronous arrangement.

2) L one latch can stored 1 bit information.

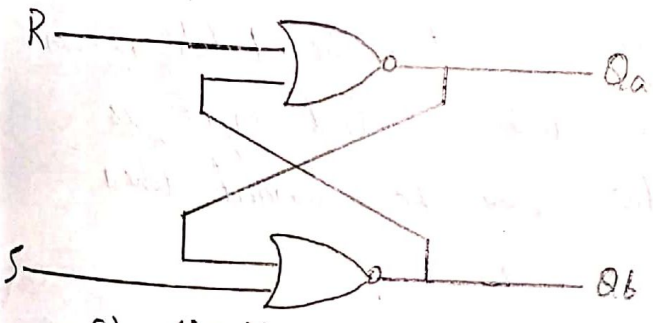
F It has 2 stable states.

3) L Latch has no clock input

F flip flop has clock input

Basic Latch :-

Fig :- A basic Latch (SR latch)

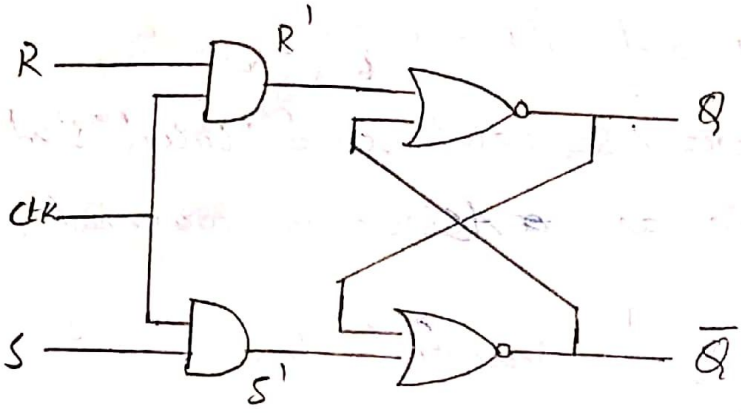


a) Circuit

S	R	Qa	Qb
0	0	0/1	1/0 (no change)
0	1	0	1
1	0	1	0
1	1	0	0

Basic Latch is a feedback connection of 2 NOR gates or 2 NAND gates which can store 1 bit of information. it can set to one using S input and reset to zero using R input.

Gated SR latch :-

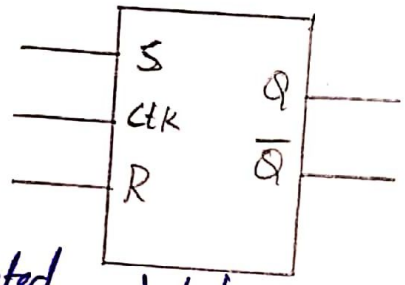


a) Circuit

Clk	S	R	Q(t+1) next state
0	x	x	Q(t) present state, no change
1	0	0	Q(t) no change
1	0	1	0
1	1	0	1
1	1	1	x

b) Truth Table

c) Graphical symbol



Gated Latch :-

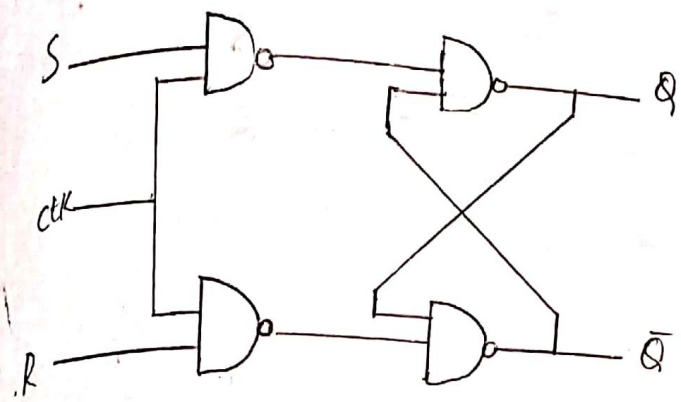
→ Gated latch is a basic latch that includes inputs & controlled input signal. The Latch remain in its existing state when control input is equal to zero. it state may be changed when the control signal is equal to 1.

→ There are 2 types of gated latches

1) Gated SR Latch :- uses the S & R input to set the latch to one or reset it to 0 respectively.

2) Gated D latch :- uses the D input to force the latch into a state that has the same logic value as D input.

Gated SR Latch with NAND gates :-

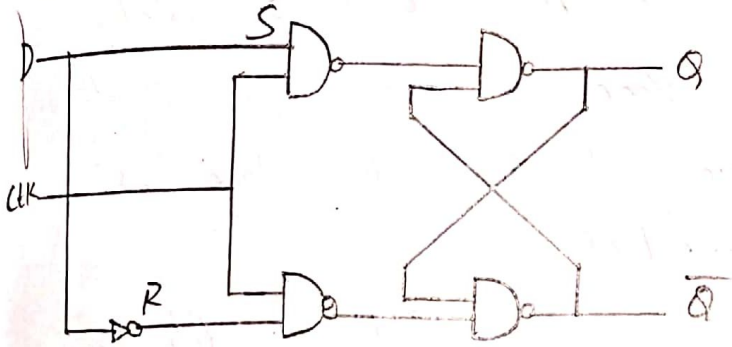


a) Circuit

b) Truth Table

Same of Gated SR Latch

Gated D Latch :-

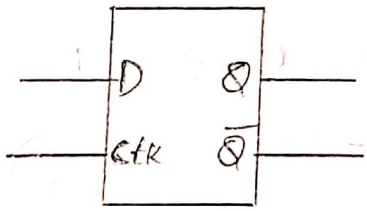


a) circuit

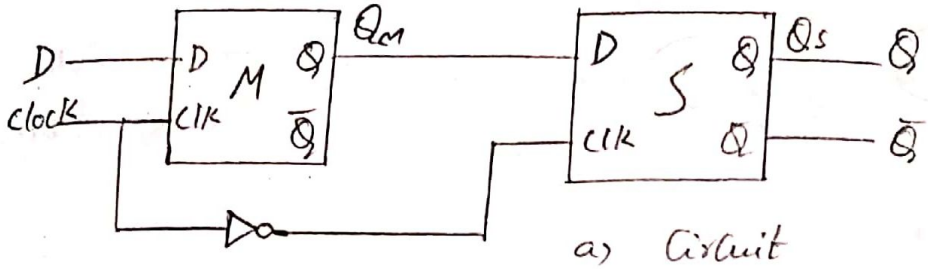
b) Truth Table

CLK	D	Q(t+1)
0	X	Q(t)
1	0	0
1	1	1

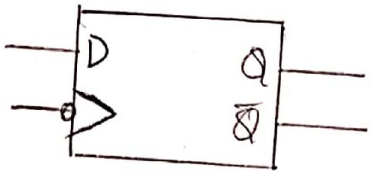
c) Graphical Symbol



Master Slave & Edge triggered D ff



a) Circuit



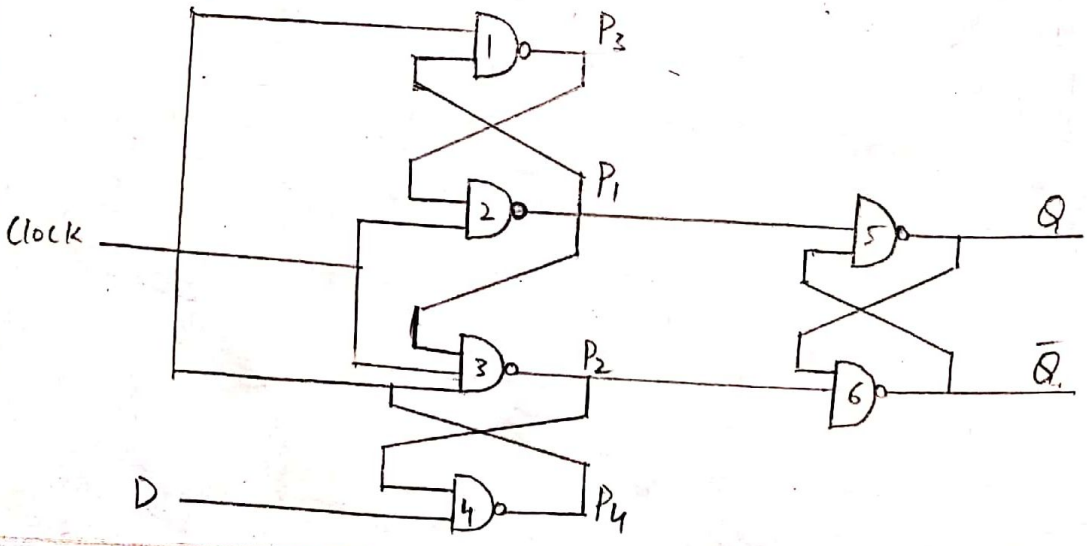
b) Graphical Symbol

→ Master Slave ff is build with 2 gated latchs. the Mater Stage is active during half of the clock cycle and the Slave Stage is active during the other half.

→ If the master stage is gated D latch then it behaves as Edge triggered flip flop

→ If the master stage is gater SR latch then the flip flop is level sensitive.

Edge triggered D ff



It has 6 NAND gates

when, clock = 0

$P_1 = P_2 = 1$

$P_3 = D$

$P_4 = \bar{D}$

when clock = 1

$P_1 = \bar{D}$

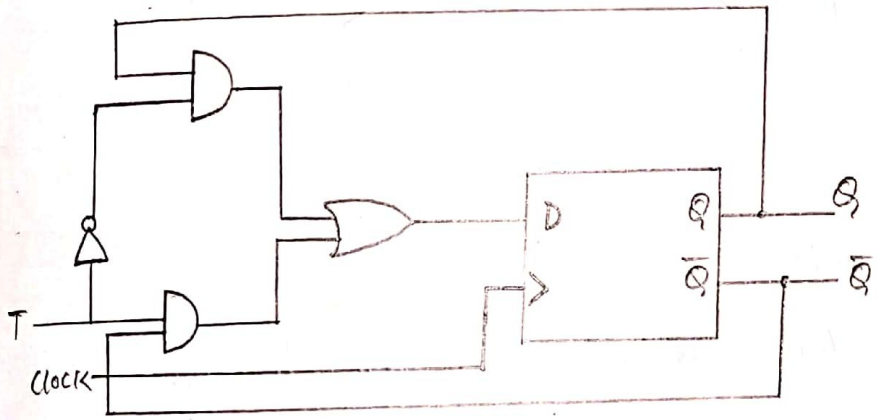
$P_2 = D$

which sets $Q = D$

$\bar{Q} = \bar{D}$

27/9/19

T-ff :-

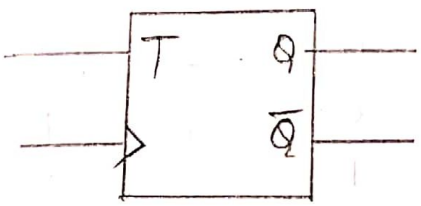


a) Circuit

b) Truth Table

T	Q(t+1)
0	Q(t)
1	$\bar{Q}$(t)

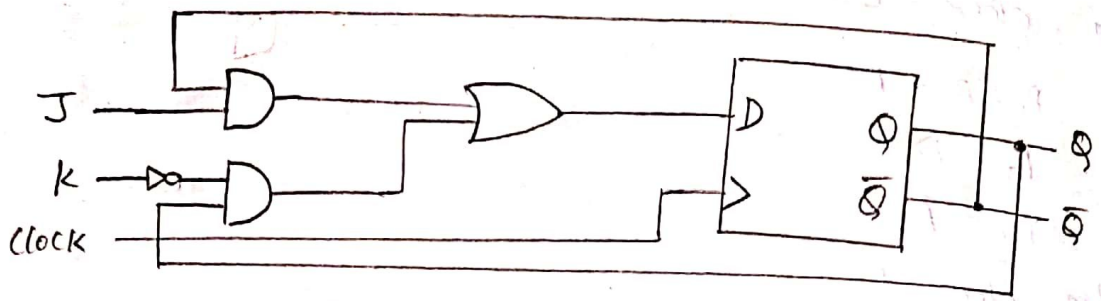
c) Graphical Symbol



→ The name T-ff derives from the behaviour of the circuit which Toggles its state when T=1

→ The Toggle features makes the T-ff a useful Element for building Counter circuit.

JK ff :-



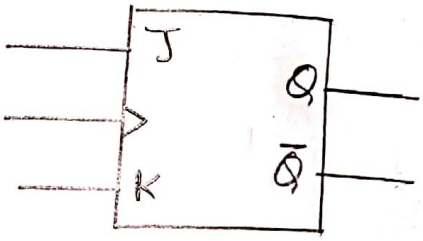
a) Circuit

$\therefore D = J\bar{Q} + \bar{K}Q$

b) Truth Table

J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\overline{Q(t)}$

c) Graphical Symbol



Excitation Tables :-

It list the required i/p's for a given change of state.

1) JK - ff

Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

2) SR - ff

Q(t)	Q(t+1)	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

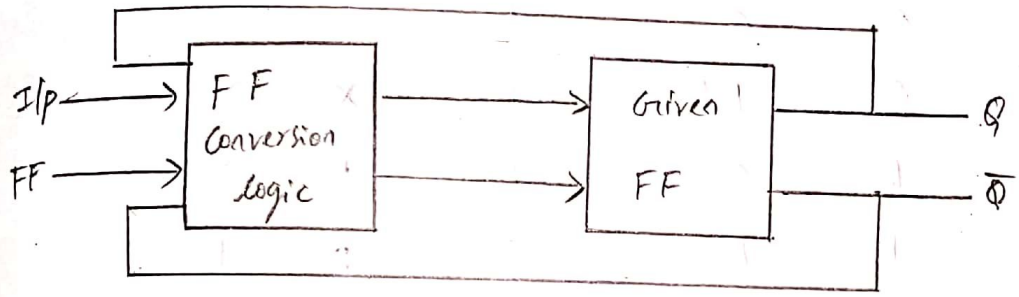
3) D-ff

Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

4) T-ff

Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

* S^2FF Conversion :-



1) SR to JK ff

FF to FF

Exc table Char table (or) Truth Table

Excitation table of SR

Q(n)	Q(n+1)	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

Char table of JK

J	K	Q(n+1)
0	0	Q(n)
0	1	0
1	0	1
1	1	$\overline{Q(n)}$

Combine 2 tables

I/p's J	I/p's K	present state Q(n)	Next state Q(n+1)	FF i/p's S	FF i/p's R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

K-map :-

S :-

J \ KQ(n)	00	01	11	10
0		X		
1	1	X		1

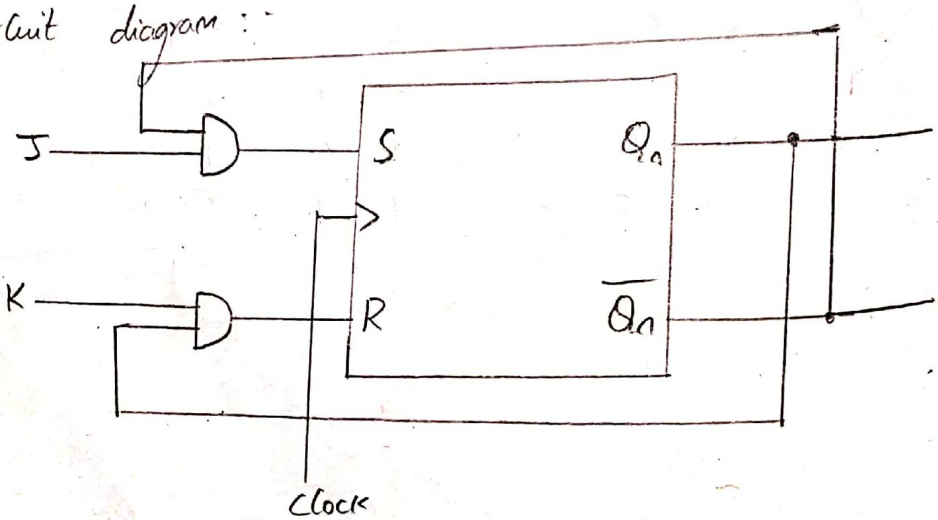
$S = J\overline{Q_n}$

R :-

J \ KQ(n)	00	01	11	10
0	X		1	X
1			1	

$R = KQ_n$

Circuit diagram :-



21 JK to D - ff

FF to FF

Exc table — Char table

Exc table of JK

J	K	Q(n+1)
0	1	Q(n)
0	1	0
1	0	1
1	1	$\overline{Q(n)}$

Char table of D

Clk	D	Q(n+1)
0	X	Q(n)
1	0	0
1	1	1

Combine 2 tables :-

I/p's	present state	Next state	FF i/p's	
D	Q(n)	Q(n+1)	J	K
0	0	0	0	X
0	1	0	X	X
1	0	1	1	X
1	1	1	X	0

K-map :- $\overline{J}$ =

D \ Q(n)	0	1
0	0	X
1	1	X

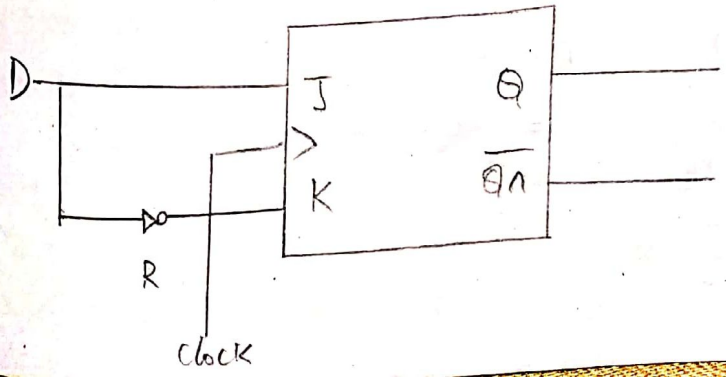
J = D

D \ K = 0	0	1
0	X	1
1	X	0

K = $\overline{D}$

10 D
00
01
11

circuit diagram :-



30/9/19

Characteristic Table :-

Characteristic table define next state as a function of i/p & present state

1) Jk ff

J	K	Q(t+1)	operation
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	set
1	1	$\overline{Q(t)}$	Complement

2) SR ff

S	R	Q(t+1)	operation
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	set
1	1	?	undefined

3) D ff

D	Q(t+1)	operation
0	0	Reset
1	1	set

4) T ff

T	Q(t+1)	operation
0	Q(t)	No change
1	$\overline{Q(t)}$	Complement

3) SR to T

Excitation character

Exc table of SR

Q(n)	Q(n+1)	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

char table of T

T	Q(t+1)
0	Q(t)
1	$\overline{Q(t)}$

Combine 2 tables :-

I/p's T	present Q(n)	Next Q(n+1)	FF i/p's S	R
0	0	0	0	x
0	1	1	x	0
1	0	1	1	0
1	1	0	0	1

K-map S =

T|Q(n)	0	1
0	0	x
1	1	0

$S = T\,\overline{Q(n)}$

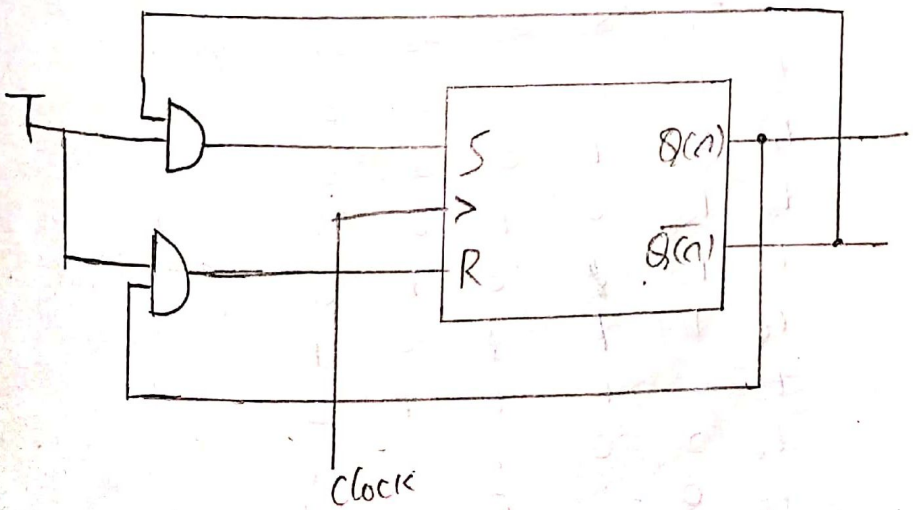
R =

T|Q(n)	0	1
0	x	0
1	0	1

$R = T\,Q(n)$

10 T Q(n) 11 T Q(n)

Circuit diagram :-



4/10/19

Registers :-

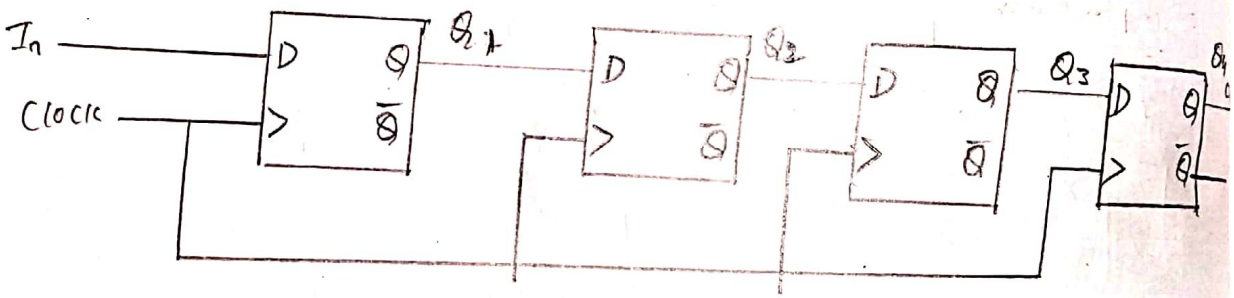
When a set of n-ff is used to store n-bits of information than these flip flops are called as Registers.

→ A common clock is used for each flip flop in a register.

Shift Register :-

A Register that provides the ability to shift it's contents is called a shift register.

Ex :- 4 bit shift register



a) Circuit

b) Sample sequence

8 conversation clock cycles 1, 0, 1, 1, 0, 0, 1 and 0

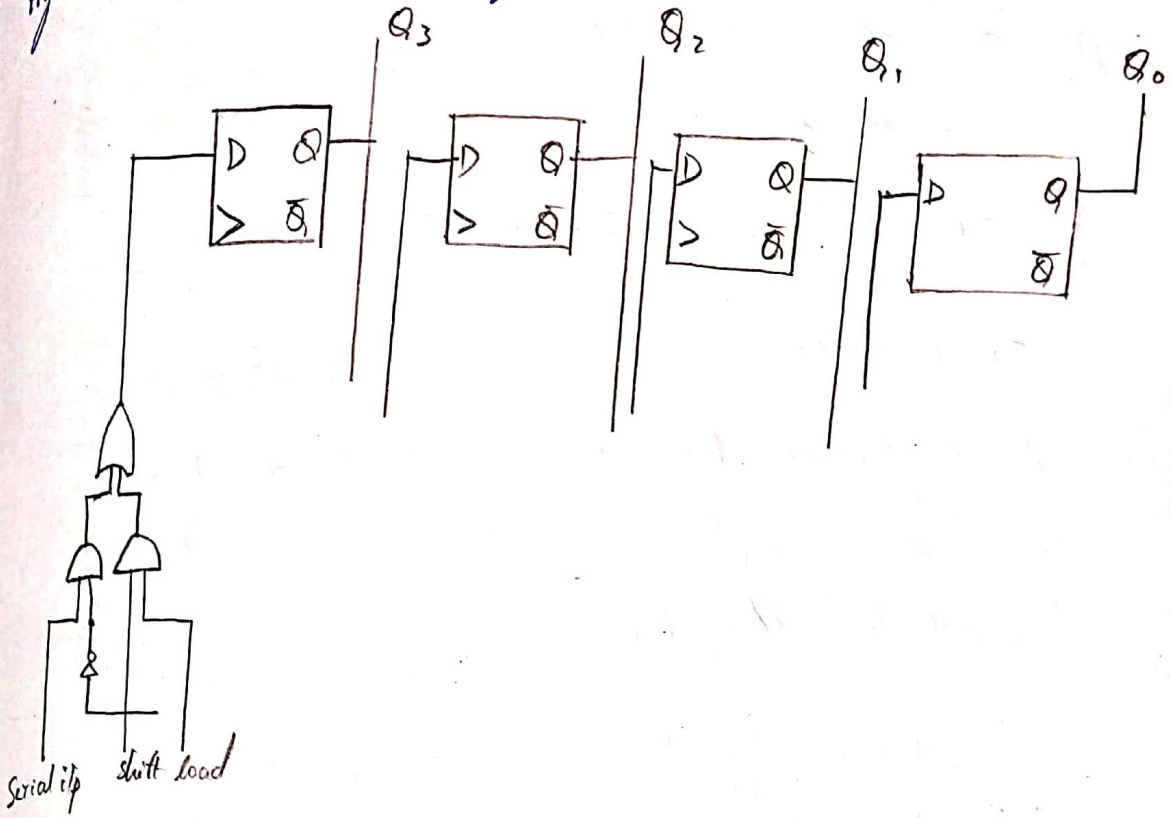
	I_n	Q_1	Q_2	Q_3	Q_4 = o/p
t_0	1	0	0	0	0
t_1	0	1	0	0	0
t_2	1	0	1	0	0
t_3	1	1	0	1	0
t_4	1	1	1	0	1
t_5	0	1	1	1	0
t_6	0	0	1	1	1
t_7	0	0	0	1	1

parallel - access shift Registers :-

all bits at once - 11 (parallel)

1 bit at a time - serial

Fig :- 4 - bit shift register with parallel access

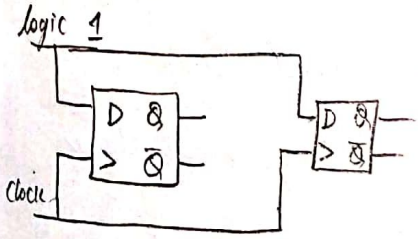


Counters :- (next page diff) (rough)

Synchronize

clock pulse

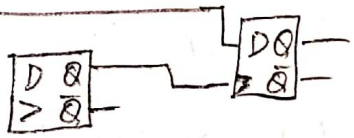
Simul



Asychronize

clock pulse p^t f4

logic 1



Counters :-

diff :-

A Counter is a register that goes through a pri determined sequence of states upon the application of clock pulse

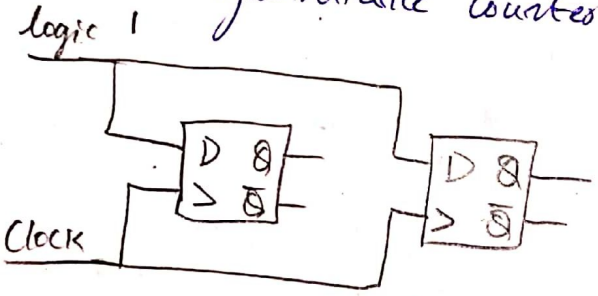
-1 2 types of Counter

1) Synchronance 2) A synchronance

Synchronance

1) A Synchronance the clock pulse are applied simultaneously to all the flip flops.

2) Circuit diagram of 2 bit synchronance counter



3) Speed low

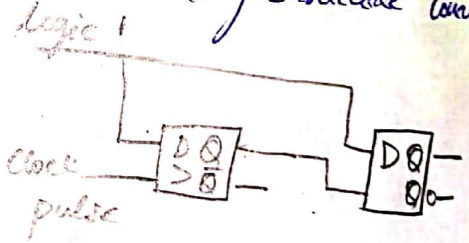
4) Simple hardware

5) higher & lower frequency

A ASynchronance

1) In a Asynchronance counters the clock pulse is applied to the 1st flipflop.

2) Circuit diagram of 2 bit A synchronance counter

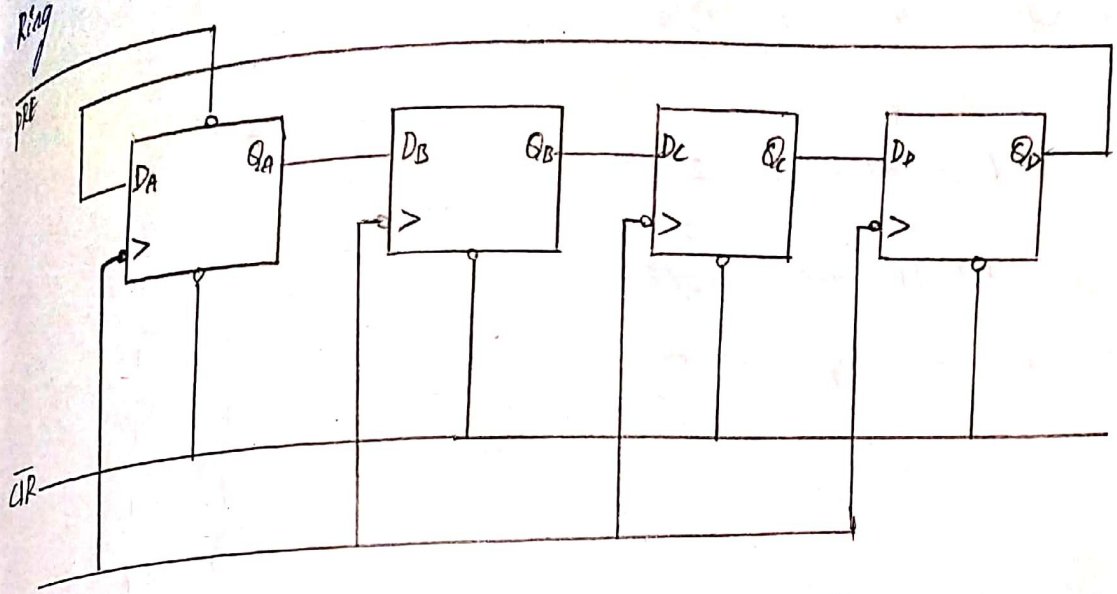


3) Speed high

4) Complex hardware

5) Lower frequency

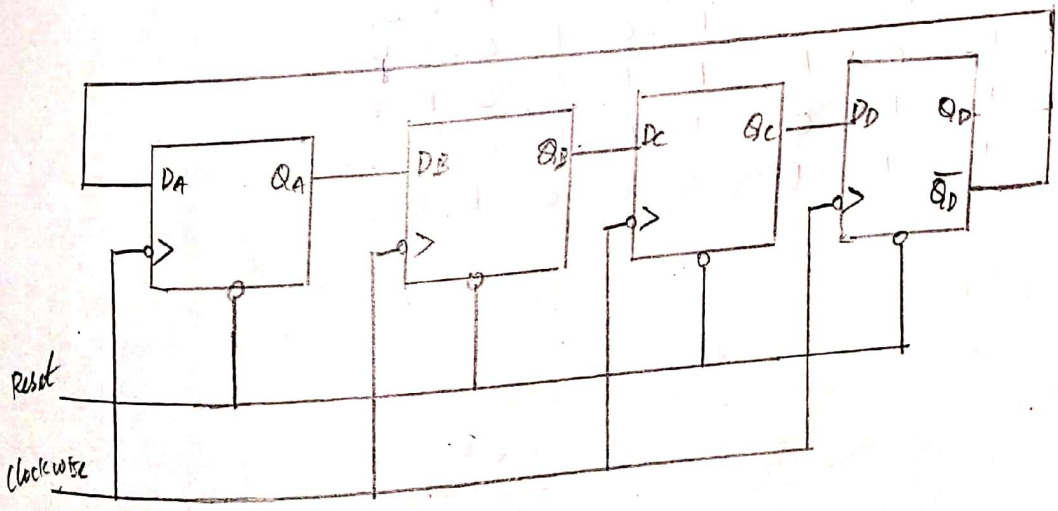
Ring Counters :-



Clock pulse	Q_A	Q_B	Q_C	Q_D
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	1	0	0	0

shift 0's

Johnson or Twisting Ring or Switch Tail Counter :-



clock pulse	Q_A	Q_B	Q_C	Q_D
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

shift '1' to right

the shift 0's

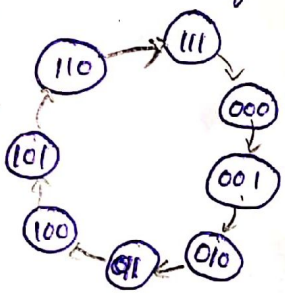
21/10/19

3-bit synchronous up counter using TFF

State table:

present state Q_C Q_B Q_A	N.S Q_C' Q_B' Q_A'	ex. table of TFF T_C T_B T_A
0 0 0	0 0 1	0 0 1
0 0 1	0 1 0	0 1 1
0 1 0	0 1 1	0 0 1
0 1 1	1 0 0	1 1 0
1 0 0	1 0 1	0 0 1
1 0 1	1 1 0	0 1 1
1 1 0	1 1 1	0 0 1
1 1 1	0 0 0	1 1 1

State diagram



Truth Table:

$T_A = 1$

T_B

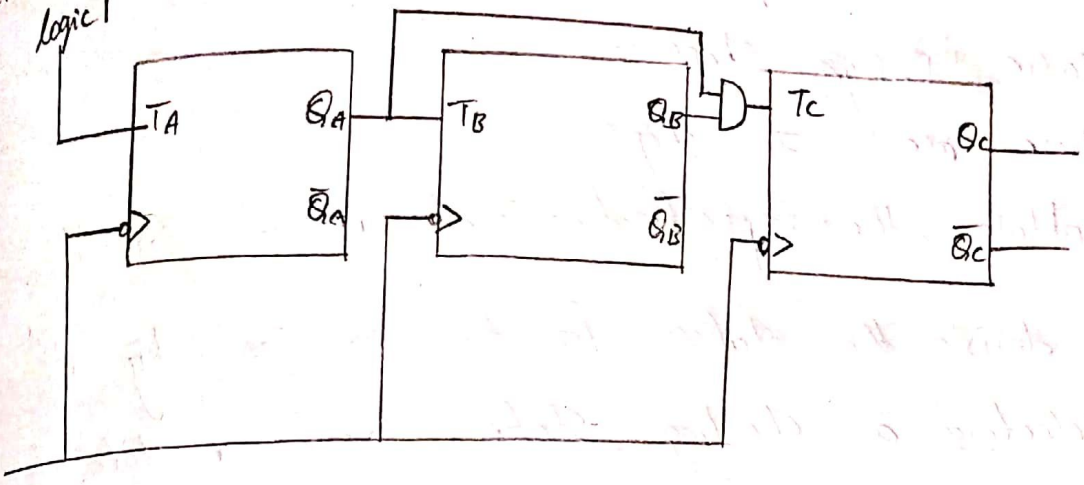
Q_C \ $Q_B Q_A$	00	01	11	10
0		1	1	
1		1	1	

T_C

Q_C \ $Q_B Q_A$	00	01	11	10
0			1	
1			1	

Circuit diagram :-

Logic 1



21/10/19 UNIT - 5

Synchronous Sequential Circuits

Basic Design Steps :-

There are 7 steps :-

1) obtain the specification of desired circuit

2) derive the states for the machine by first selecting a starting state. Create a state diagram.

3) Create a state table from the state diagram.

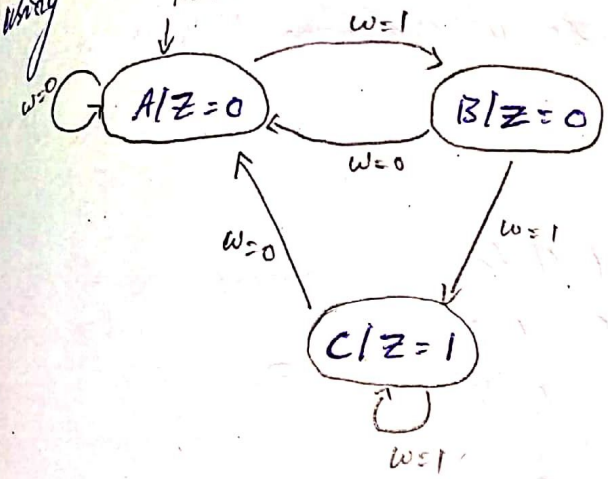
4) performed state minimization that is a procedure that minimizes the no. of states (if many states are given).

5) Decide on the no. of state variables needed to represent to all states and perform the State assignment. (There can be many state assignments).

6) Choose the type of flip flops to be used in the circuit.

7) Implement the circuit as indicated by logic expression.

#5 Q) An FSM (finite state machine) is defined by following state diagram Derive a circuit that realizes the FSM using Dff.



Sol step 1:- State table

present state	Next State		output
	w=0	w=1	z
A	A	B	0
B	A	C	0
C	A	C	1

Each state variable may be implemented in the form of flip flop (ff). Since, there are 3 states to be realize it is sufficient to use 2 state variables y_1 & y_2.

$2^2 = 4$

3 (A, B, C)

$3 < 4$

1 is unused (don't - Cared)

State assignment :-

$y_2 y_1$: 00 (A)
01 (B)
10 (C)

As D is not there so 4th valuation $y_2 y_1 = 11$ is not needed

Fig. state assignment :-

	present state		Next state $w=0$		$w=1$		output
	y_2	y_1	y_2	y_1	y_2	y_1	z
A	0	0	0	0	0	1	0
B	0	1	0	0	1	0	0
C	1	0	0	0	1	0	1
	1	1	d	d	d	d	d

K-map :-

Y_1 :

w \ $y_2 y_1$	00	01	11	10
0			d	
1	1		d	

$Y_1 = w\bar{y}_1\bar{y}_2$

Y_1 is 1 at 1st line where $w=1, y_2 y_1 = 00$

Y_2 :

w \ $y_2 y_1$	00	01	11	10
0			d	
1		1	d	1

$Y_2 = wy_1 + wy_2$
$Y_2 = w(y_1 + y_2)$

z :

y_2 \ y_1	0	1
0		
1	1	d

$z = y_2$

23/10/19

Q) Explain State assignment problem with ex.

	y_2	y_1
A =	0	0
B =	0	1
C =	1	1

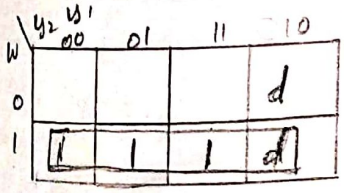
D is not there so 4th valuation $y_2 y_1 = 10$ is not needed

Fig. Improved State Assignment:

	present state		Next State w=0		Next State w=1		output
	y_2	y_1	Y_2	Y_1	Y_2	Y_1	Z
A	0	0	0	0	0	1	0
B	0	1	0	0	1	1	0
C	1	1	0	0	1	1	1
	1	0	d	d	d	d	d

K-map :-

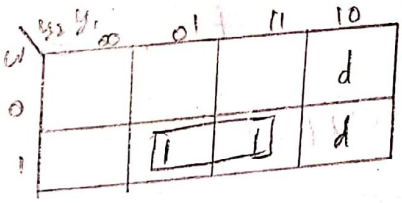
Y_1 =



Y_1 = w

w	y_2	y_1
1	0	0
1	0	1
1	1	1
1	1	0

Y_2 =

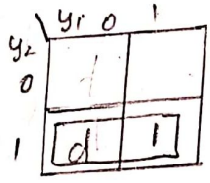


$Y_2 = w\ y_2\ y_1$

w	y_2	y_1
1	0	1
1	1	1

$Y_2 = w y_1$

Z =



Z = y_2 y_1

y_2	y_1
1	0
1	1

$Z = y_2$

State minimization :-

The purpose of state minimization is to reduced the no. of states in a sequential circuit. so that the circuit requires fewer FFs.

partitioning minimization procedure :-

Q) Minimize the given state table using the partitioning procedure

present state	Next state		output
	w=0	w=1	z
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0

A)

P_1 = (ABCDEFG)

o/p is 0

o/p is 1

P_2 = (ABD) (CEFG)

P_3 = ( ) ( ) (F)

Now Consider all 0 successors & 1. successors

ABD o/p when w=0 => BDB - in same block

w=1 => CFG " " "

CEFG o/p when w=0 => FFEF - " " "

CEFG w=1 => ECDG - in different block

D present state 'F'

$P_3 = (AB)(D)(CEG)(F)$

Now consider all 0 successors & 1. successors

ABD o/p when $w=0 \Rightarrow BDB$ - in same block

$A(B)D$ $w=1 \Rightarrow CFG$ " diff block

F is from diff block, it present state is B

can be equivalent A & C

CEG o/p when $w=0 \Rightarrow FFF$ - in same block

$w=1 \Rightarrow ECG$ - " " "

$\therefore P_4 = (AD)(B)(CEG)(F)$

Now Consider all 0 successors & 1. successors

AD o/p when $w=0 \Rightarrow BB$ - in same block

$w=1 \Rightarrow CG$ - " " "

CEG o/p when $w=0 \Rightarrow FFF$ - in same block

$w=1 \Rightarrow ECG$ " " "

$\therefore P_5 = (AD)(B)(CEG)(F)$

Since $P_5 = P_4$ and no new blocks are generated, it follows that states in each block are equivalent.

Take the 1st letter of each block

A represent both the states A and D

C " states C, E and G

Fig: Minimized state table

present State	Next state $w=0$	Next state $w=1$	output z
A	B	C	1
B	A	F	1
C	F	C	0
F	C	A	0

$D=A$

$E=C$

$G=C$

UNIT-V

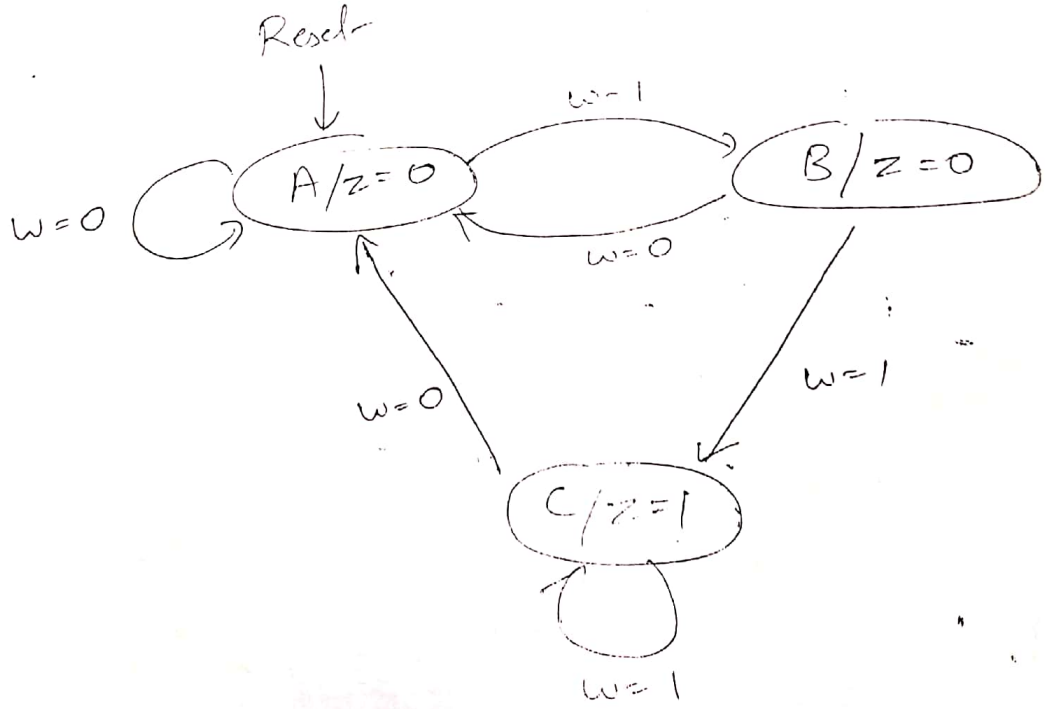
⇒ Synchronous Sequential Circuits:

* → Basic Design Steps:

1. Obtain the specification of the desired circuit.
2. Derive the states for the machine by first selecting a starting state. Create a state diagram.
3. Create a state table from the state diagram.
4. Perform state minimization a procedure that minimizes the number of states. (if required i.e. if many states)
5. Decide on the number of state variables needed to represent all states & perform the state assignment. (there can be many st. assig
6. Choose the type of flip-flops to be used in the circuit.
7. Implement the circuit as indicated by the logic expressi

Example:

Q. An FSM is defined by following state diag. Derive a circuit that realizes the FSM using D ff.



te table:

table indicates all transitions from each present state the next state for different values of the input signal. e o/p z is specified w.r.t. the present state.

Fig① State table for the seq. ckt

Present State	Next State w=0	Next State w=1	Output z
A	A	B	0
B	A	C	0
C	A	C	1

z is '0' in A
'0' in B
'1' in C

Each state variable may be implemented in the form of ff.

Since there are 3 states to be realized, it is sufficient to use 2 state variables y_1 and y_2

$2^2 = 4$ A, B, C
3 < 4
1 is unused

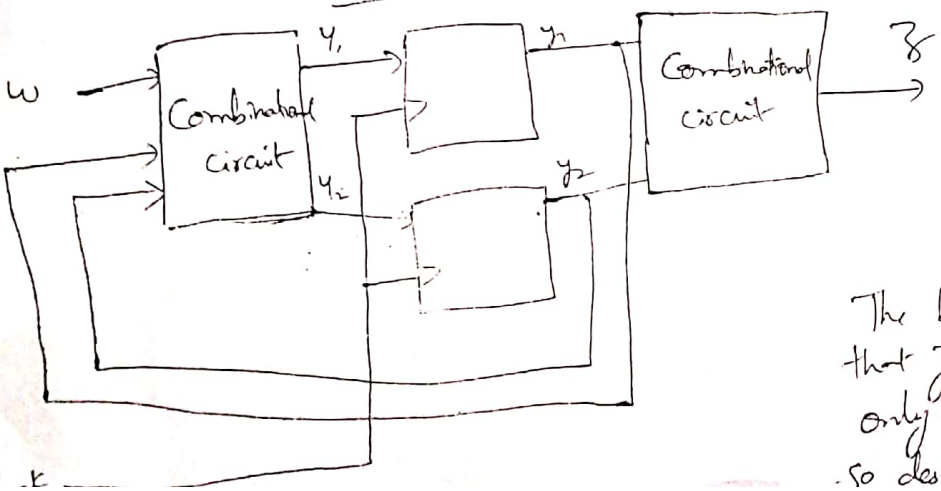
Choice

K-m
4

State Assignment:

from the above ② fig the output z is determined only by the present state of the circuit. Thus, the block diag. shows that z is a function of only y_1 and y_2 (Moore type)

fig. A general sequential circuit with i/p w, o/p z and 2 state ffs.



The block diag. shows that z is a func. of only y_1 and y_2
. So design is Moore type

The possible assignment is given in fig. where the states A, B and C are represented by $y_2 y_1 = 00, 01$ and 10 respec.

A = 0 0
B = 0 1
C = 1 0

As D is not there so 4th variation $y_2 y_1 = 11$ is not needed

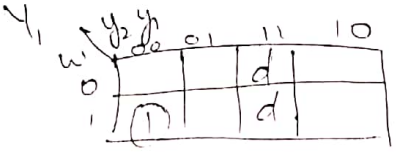
Fig. State assigned table for the sequential circuit in fig (2)

	Present State $y_2\ y_1$	Next State $w=0$ $Y_2 Y_1$	Next State $w=1$ $Y_2 Y_1$	Output z
A	0 0	0 0	0 1	0
B	0 1	0 0	1 0	0
C	1 0	0 0	1 0	1
	1 1	d d	d d	d

Choice of ffs and derivation of next state & output expressi

K-map

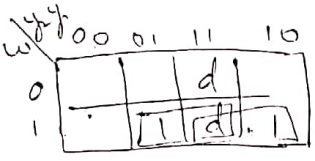
Y_1



$Y_1 = w\bar{y}_1\bar{y}_2$

Y_1 is '1' at 1st line where $w=1$, $y_2 y_1 = 00$
Y_1 is 'd' at 4th line where $w=0$, $y_2 y_1 = 11$ and $w=1$, $y_2 y_1 = 11$

Y_2



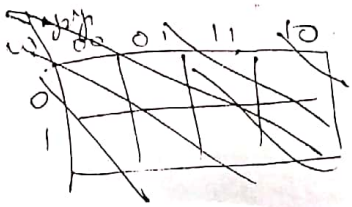
$Y_2 = wy_1 + wy_2$
$= w(y_1 + y_2)$

z

2 var. K-map (y_2 & y_1)

$z = y_2$

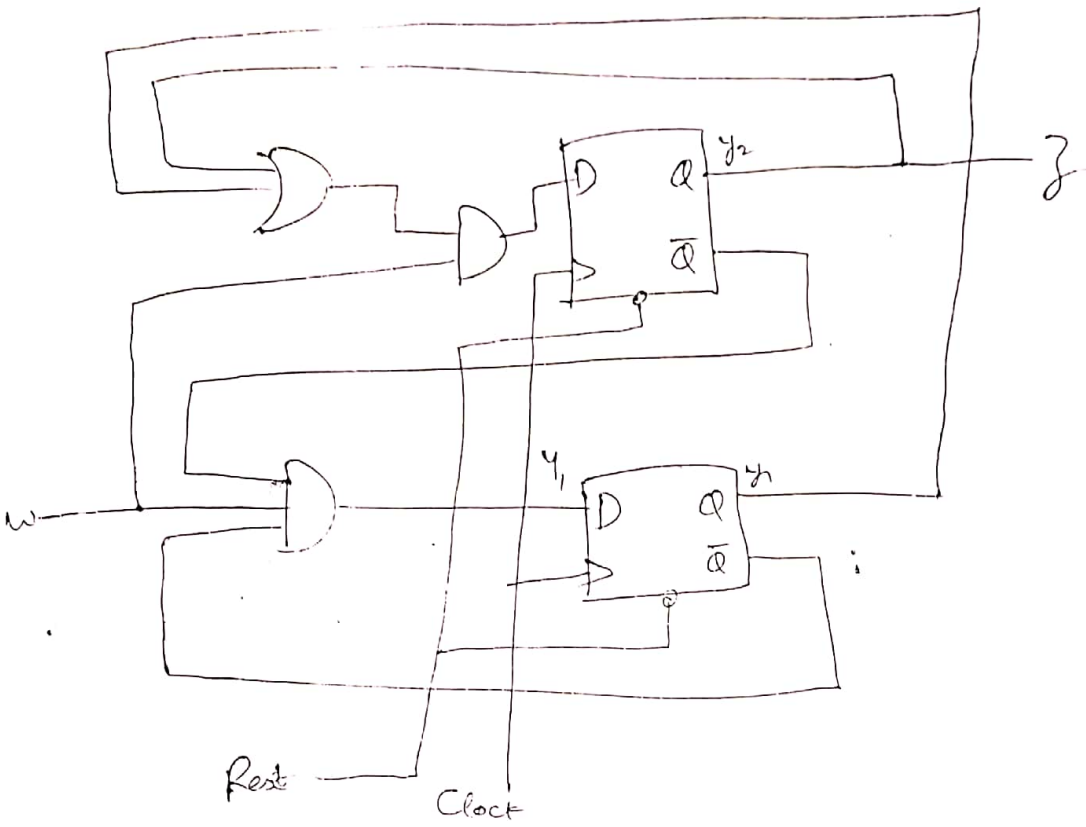
z



$Y_1 = w\bar{y}_1\bar{y}_2$
$Y_2 = w(y_1 + y_2)$
$z = y_2$

for D-ff
$D_1 = Y_1$ and $D_2 = Y_2$

Fig. Final implementation of the sequential circuit



xplain state assignment problem with an ex.
refer previous question till state table

Fig. gives one more possible alternative. In this case the states A, B and C are

	y_2	y_1
A =	0	0
B =	0	1
C =	1	1

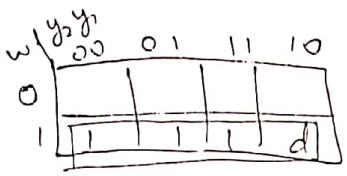
D is not there so 4th valuation $y_2y_1 = 10$ is not needed

	Present state y_2	y_1	Next state w=0 Y_2	Y_1	w=1 Y_2	Y_1	Output z
A	0	0	0	0	0	1	0
B	0	1	0	0	1	1	0
C	1	1	0	0	1	1	1
	1	0	dd		dd		d

Fig. Improved state assignment for seq. ckt in fig①

K-map

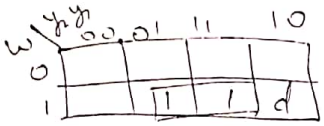
Y_1



$Y_1 = w$

$Y_1 = D_1$

Y_2



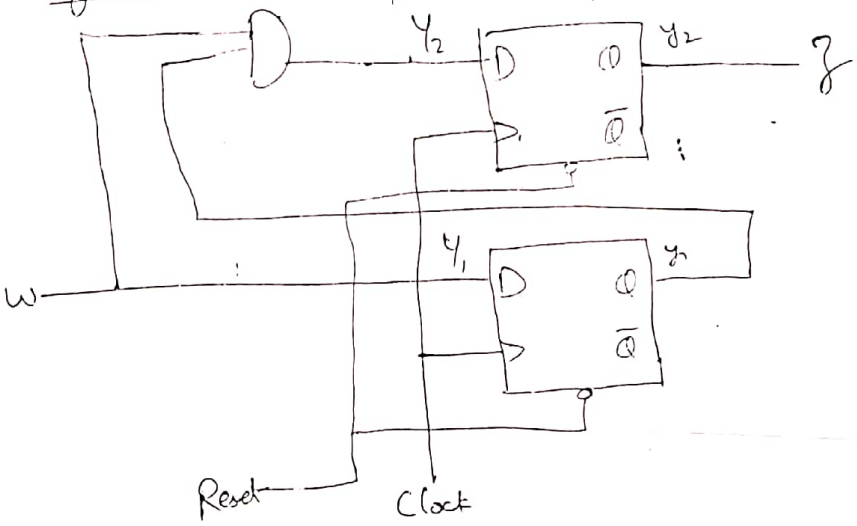
$Y_2 = wy_1$

$Y_2 = D_2$

z

$z = y_2$

Fig. Final circuit for the improved state assignment



Q. What is state minimization? How do you identify the equivalent states?

ate Minimization:

e purpose of state minimization is to reduce the number states in a sequential circuit so that the circuit requires wer flip flops.

State minimization is based on the concept of the chavioral equivalence of FSMs.

tate equivalence:

2 states S_i and S_j are said to be equivalent if and only for every possible input sequence, the same output sequence ill be produced regardless of whether S_i or S_j is the itial state.

artitioning Minimization Procedure:

linimize the given state table using the partitioning procedure.

Present State	Next State w=0	Next State w=1	O/P Z
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0

The initial partition contains all states in a single block

$P_1 = (ABCDEFG)$

The next partition seperates the states that have different outputs

ABD o/p is 1
CEFG " " 0

$\therefore P_2 = (ABD)(CEFG)$

Now consider all 0-successors and 1-successors of states in each block.

ABD o/ps when w=0 ⇒ BDB in same block

" w=1 ⇒ CFG " " "

CEFG o/ps when w=0 ⇒ FFEF in same block

" w=1 ⇒ ECDG in diff. blocks

D is from diff block

for CEFG, when w=1, D present state is 'F'

$\therefore P_3 = (ABD)(CEG)(F)$

Repeating the process yields the following

ABD o/ps when w=0 ⇒ BDB in same block

" " w=1 ⇒ CFG in diff blocks (see P_3).

F is from diff block, its present state is 'B'

CEG o/ps when w=0 ⇒ FFF in same block

" " w=1 ⇒ ECG " "

$\therefore P_4 = (AD)(B)(CEG)(F)$

Repeating the process

AD o/ps when w=0 B,B in same block

" " w=1 C,G " "

CEG " " w=0 FFF " "

w=1 ECG " "

$\therefore P_5 = (AD)(B)(CEG)(F)$

Since, $P_5 = P_4$ and no new blocks are generated, it follows that states in each block are equivalent.

Take the 1st letters of each blak
A represents both the states A and D
C " states C, E and G

Fig. Minimized state table

Present State	Next State		Output
	w=0	w=1	z
A	B	C	1
B	A	F	1
C	F	C	0
F	C	A	0

D=A
E=C
G=C

ite State Machine (FSM) representation using Moore
d Mealy state models:

oore Machine: The sequential circuits in which the output depends
ly on states are refered as moore model circuits.

Mealy Machine: The sequential circuits. in which output depends
, input as well as on states are referred as mealy model
rcuits.

Moore Model	Mealy Model
The sequential circuits in which the output depends only on states are Moore model circuits.	1. The sequential circuits in which output depends on input as well as on states are mealy model circuits.
Input changes does not effect the output.	2. Input changes may effect the o/p of the circuit.
Requires more number of states for implementing same functions	3. Requires less no of states for implementing same functions
Ex: Z=A	Ex: Y=(A+B)$\bar{x}$

Consider the sequence of values of the w and z signals during 11 clock cycles as shown in fig.

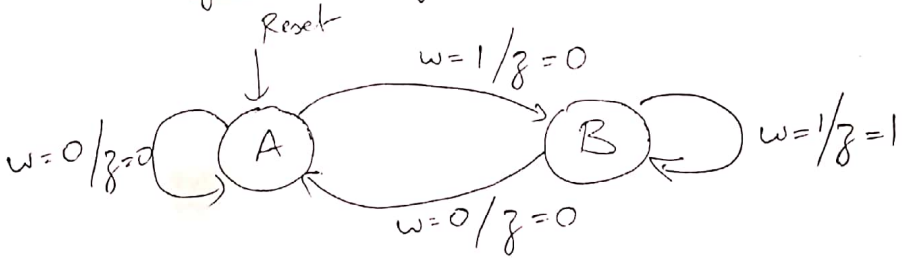
Clock cycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	1	0	0	1	1	0	0

O/p z should be equal to 1 in the same clock cycle when the second occurrence of w=1 is detected.

~~Fig.~~ State table

As long as w=0, the machine should remain in state A, producing an output z=0. When w=1, the machine has to move to a new state, B, to record the fact that an i/p 1 has occurred.

Fig. State diagram



The state table is shown in fig. The table shows that the output z depends on the present value of input w and also on present state.

Fig. State Table

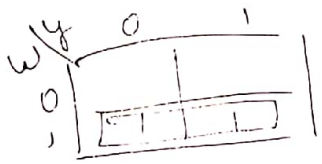
Present State	Next State w=0	Next State w=1	Output z w=0	Output z w=1
A	A	B	0	0
B	A	B	0	1

Fig. shows the state assigned table. there are only 2 states, it is sufficient to use a single state variable y.

Fig. State-assigned table for the FSM

Present State	Next State $w=0$	Next State $w=1$	Output $w=0$	Output $w=1$
y	Y	Y	z	z
A 0	0	1	0	0
B 1	0	1	0	1

Y



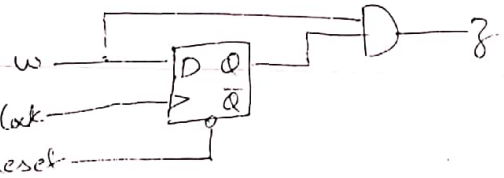
$Y = w$

w = i/p
y = present state

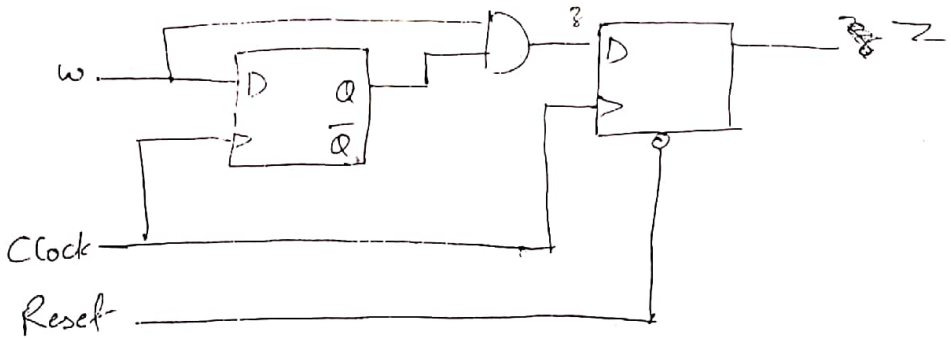
z

$z = wy$

The resultant circuit is shown in fig.



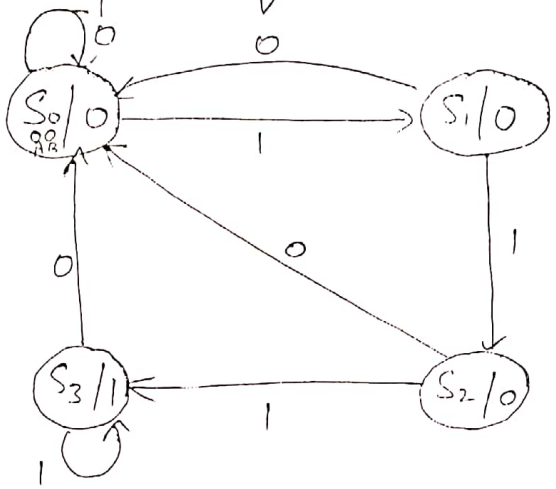
This circuit can be changed from Mealy-type to Moore type output depends on present state)



Design FSM circuit detecting '111'. It has one i/p and 1 o/p. The o/p should become '1' whenever the above sequence is detected.

Design of FSM for sequence generation & detection:

State diagram for sequence detector



From the fig., it is desired by starting with state S_0, if the i/p is 0, the circuit stays in the same state, but if the i/p is 1, it goes to state S_1 to indicate that a 1 was detected.

If the next input is 1, the change is to state S_2 to indicate the arrival of 2 consecutive 1, but if the i/p is 0 we go back to state S_0.

The 3rd consecutive 1 sends the circuit to state S_3. If more 1's are detected, the circuit stays at S_3. Any '0' i/p sends the circuit back to S_0.

In this way, the circuit stays at S_3 as long as there are 3 or more consecutive 1's received. This is a Moore Model seq. circuit since the o/p is 1 when circuit is in state S_3 and 0 otherwise.

Synthesis using D ff:

In the state table, A and B are present state values of ffs A and B. 'x' is the i/p, D_A and D_B are the i/p equations. Y is the o/p.

ate table for sequence detector

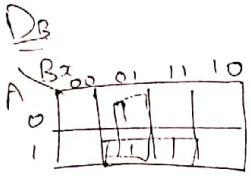
Present State		Input	Next State		Output
A	B	x	D_A	D_B	Y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1

:map

D_A

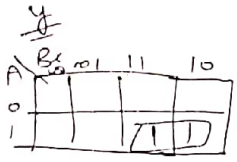
$D_A = A \cdot x + B \cdot x$

D_B



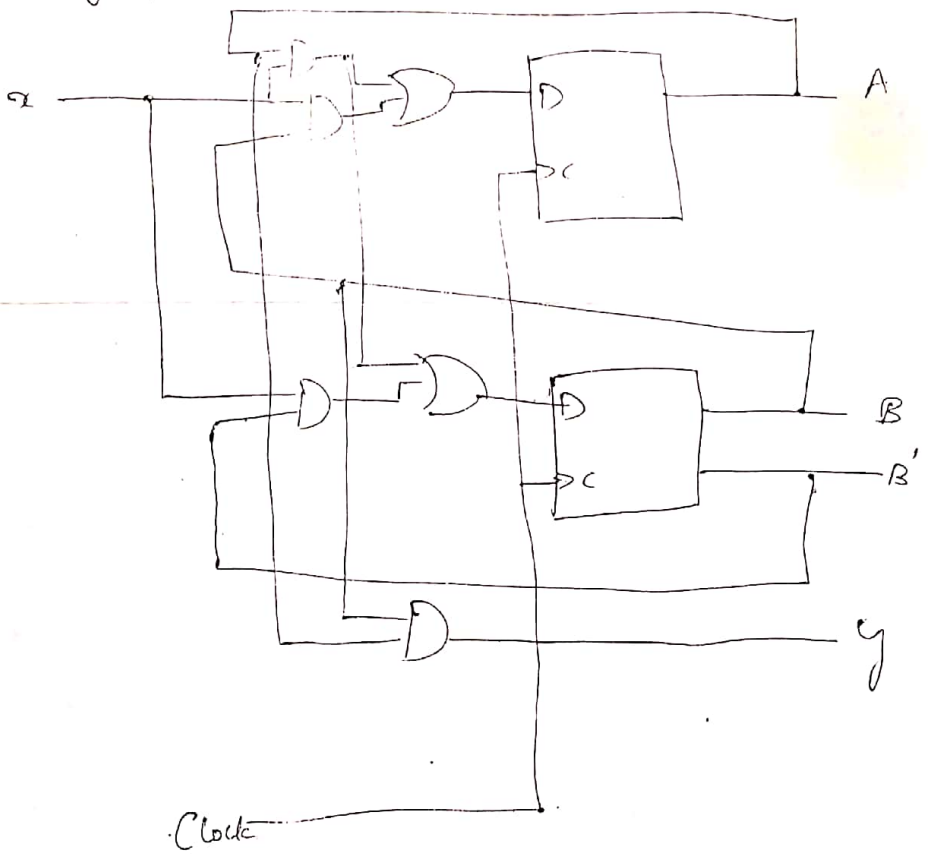
$D_B = Ax + B'x$

Y



$Y = AB$

Logic diag for sequnce detector



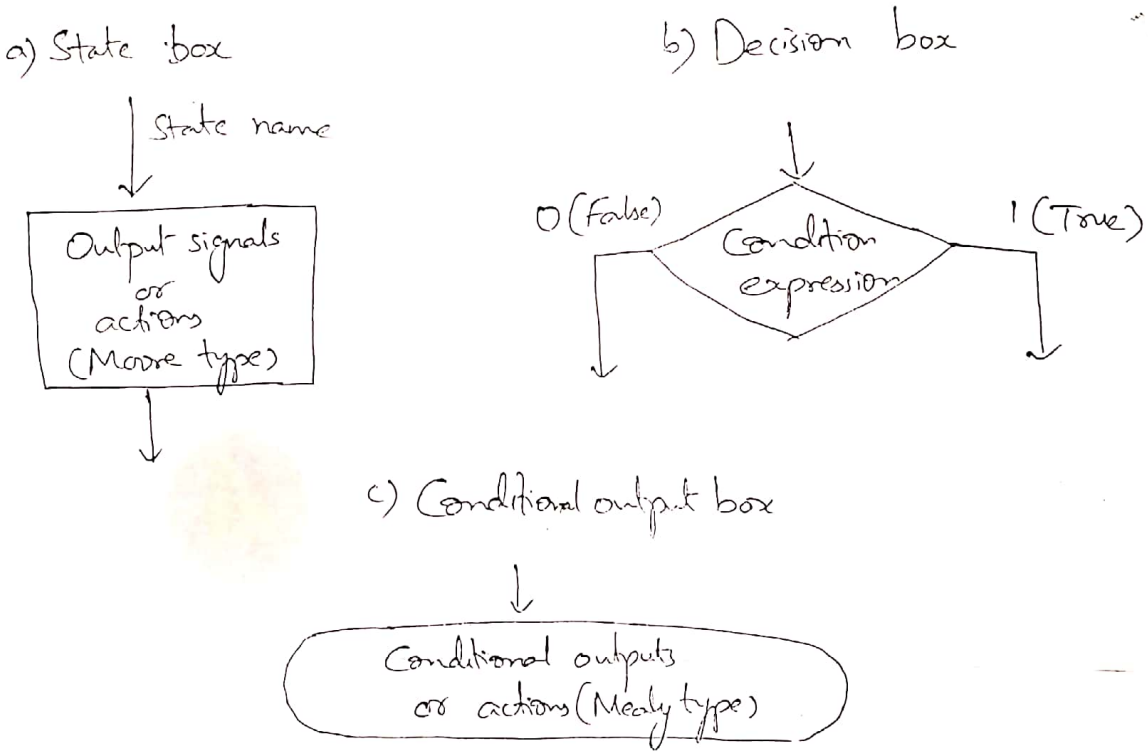
Algorithmic State Machine (ASM) Charts:

An ASM chart is a type of flowchart that can be used to represent the state transitions & generated outputs for an FSM.

The 3 types of elements used in ASM charts are:

i) State Box
ii) Decision Box
iii) Conditional output box

Fig. Elements used in ASM charts

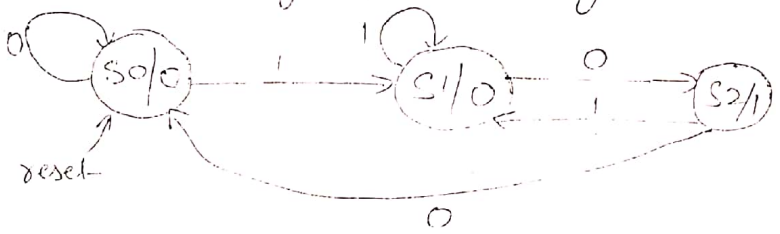


i) State box: A rectangle represents a state of FSM. It is equivalent to a node in the state diagram or row in a state table. The name of the state is indicated outside the box in the top-left corner.

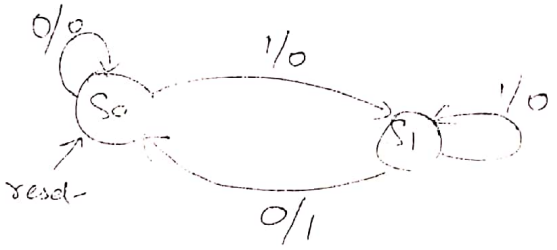
Decision box: A diamond indicates that the stated condition expression is to be tested & the exit path is to be chosen accordingly. The condition expression consists of one or more inputs to the FSM.

Conditional output box: An oval denotes the output signals that are of Mealy type. These outputs depend on the values of state variables and the inputs of FSM.

Draw the state diag. for detecting the sequence '01'



Meaning of States:
- S0 ⇒ No elements of the seq. observed
- S1 ⇒ '1' observed
- S2 ⇒ "10" observed



Meaning of states:
- S0 ⇒ No elements of the seq. observed
- S1 ⇒ '1' observed