Number Systems & Codes

⇒ Decimal number system

(b) base - 10

(b-1) no of digits → 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

* (q no of digits) . (p no of digits) → $(345.12)_{10}$

$(a_{q-1} \cdots a_1 a_0 a_{-1}) \cdot (a_{-1} a_{-2} \cdots a_{-p})$

$$N_b = a_{q-1} b^{q-1} + a_{q-2} b^{q-2} + \cdots a_0 b^0 + a_{-1} b^{-1} + a_{-2} b^{-2} \cdots a_{-p} b^{-p}$$

$$= \sum_{i=-p}^{q-1} a_i b^i$$

$a < b - 1$

	b
Decimal	10
Binary	2
Octal	2^3
Hexadecimal	16

0, 1, ... 9, A, B, C, D, E, F
(10) (11) (12) (13) (14) (15)

* $N_b = a_{q-1} b_2^{q-1} + a_{q-2} b_2^{q-2} + \cdots a_0 b_2^0$

$$\frac{N_b}{b_2} = \frac{a_{q-1} b_2^{q-1} + a_{q-2} b_2^{q-2} + \cdots a_1 b_2 + \left(\frac{a_0}{b_2}\right)}{b_2}$$

$$\frac{N_b}{b_2} = Q_0 + (a_0) \checkmark$$

$$\frac{Q_0}{b_2} = Q_1 + (a_1) \checkmark$$

Ex: 34.12_{10}

100010.00011_2

$$N_b \times b_2 = a_{-1} + \underbrace{a_{-2} b_2^{-1} + a_{-3} b_2^{-2} + \cdots a_{-p} b_2^{-(p-1)}}_{N_{b_1}'}$$

$N_{b_1}' \times b_2 = a_{-2} + \cdots$

$b_1 > b_2$ (10 → 2)

Decimal	Binary	w_4 w_3 w_2 w_1 8 4 2 1
0	0000	
1	0001	
2	0010	
3	0011	
4	0100	
5	0101	
6	0110	
7	0111	
8	1000	
9	1001	

for binary to decimal:

$$N = w_4x_4 + w_3x_3 + w_2x_2 + w_1x_1$$

$(x_4\ x_3\ x_2\ x_1)$ — 8 4 2 1

$$\therefore N = 8x_4 + 4x_3 + 2x_2 + x_1$$

$x_i \rightarrow 0/1$

Binary to octal

$001 \mid 001 \mid 011 \mid 101)_2 \rightarrow (1135)_8$

1 1 3 5

For octal:

$x_4\ x_3\ x_2\ x_1$

$8^3\ 8^2\ 8^1\ 8^0$

$$N = 8^3x_4 + 8^2x_3 + 8x_2 + 1x_1$$

$x_i \rightarrow 0, 1, 2, 3, \cdots 7$

Binary to Hexa:

$(0010 \mid 0101 \mid 1101)_2 \rightarrow (25D)_{16}$

2 5 D

Weighted code:

8421-code → Binary coded decimal code (0-9)

$$N_{10} = w_4x_4 + w_3x_3 + w_2x_2 + w_1x_1$$

Ex: wt 2421

Decimal	2421 code
0	0000
1	0001
2	0010 / 1000
3	0011 / 1001
4	0100 / 1010
5	1011 / 0101
6	1100 / 0110
7	0111 / 1101
8	1110
9	1111

⇒ Not unique

Ex: wt → 6 4 2 -3

0	0000
1	0101
2	0010
3	0001

(Non-weighted code)

# Decimal	Excess-3 (+3)
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

Cyclic code: Two codes differ in one digit.

Gray code: One popular cyclic code

1G (1 bit gray code)	2G	3 bit gray code	4 bit Gray code
0	00	000	0000
1	01	001	0001
	10	011	0011
	11	010	0010
		110	0110
		111	0111
		101	0101
		100	0100

D	G	B
0	000	000
1	001	001
2	011	010
3	010	011
4	110	100
5	111	101
6	101	110

n bit binary to n-bit gray

$G_n = B_n$

$G_i = B_i \oplus B_{i+1}$

XOR

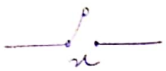
00 - 0
11 - 0
01 - 1
10 - 1

$b_4 b_3 b_2 b_1 b_0$ - B

1 0 1 1 0

$g_4\ g_3\ g_2\ g_1\ g_0$

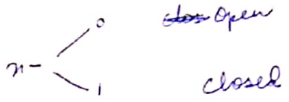
SWITCHING ALGEBRA :-



x

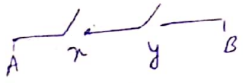
↓

Switching variable



~~close~~ Open

closed



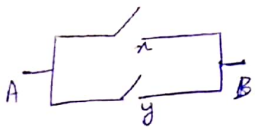
serial connection

$\equiv x.y$ ($\wedge$, $\wedge$, AND)

→ Product

$x = 0$, iff $x \neq 1$

$x = 1$; iff $x \neq 0$



$\equiv x+y$ (OR, $\cup$, $\vee$)

→ sum

x' → NOT

0	1
1	0

→ $x_1, x_2, \cdots x_n = \{0,1\}$, +, ·, ′

(NOT)

(OR) (AND)

$\{0,1\}$ $\{+, \cdot, '\}$

→ Basic properties:

(1) Idempotency :-

$x + x = x$

$x \cdot x = x$

} Useful to minimize

Proof: Induction (Take all values)

$1+1=1$ $1 \cdot 1 = 1$

$0+0=0$ $0 \cdot 0 = 0$

(2) Commutative :-

Proof: Same

$x + y = y + x$

$x \cdot y = y \cdot x$

(3) Associative :-

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

$x \cdot (y \cdot z) = (x \cdot y) \cdot z$

(4) Distributive :-

Note: $5 \times (7+3) = 5 \times 7 + 5 \times 3$

$5 + (7 \times 3) \neq (5+7) \times (5+3)$

→ Multiplication / Product (·) is distributive over sum (+), but (+) is not distributive over (·)

& In NORMAL ALGEBRA

→ In switching Algebra:

(+) distributes over (·)

& (·) distributes over (+)

∴ $x + y \cdot z = (x+y) \cdot (x+z)$

$x \cdot (y+z) = (x \cdot y) + (x \cdot z)$

] DISTRIBUTIVE

Simplification rules :-

$x + 1 = 1$

$x + 0 = x$

$x \cdot 1 = x$

$x \cdot 0 = 0$

☆ (1) Absorption :-

$x + x \cdot y = x$

$x \cdot (x+y) = x$

} Same result on exchanging (·) & (+), Duality property of Boolean Algebra

Proof :-

① $x + x \cdot y = x(1+y)$

$= x$

② $x \cdot x + x \cdot y = x + x \cdot y$

$= x$

Absorption (contd) :-

$x + x' = 1$

$x \cdot x' = 0$

(2) Consensus :- (V. Imp)

$xy + x'z + yz = xy + x'y$ $(z \to z')$

& $(x+y) \cdot (x'+z) \cdot (y+z) = (x+y) \cdot (x'+y)$

By duality

Proof :

(1) $xy + x'z + yz$

$= xy + x'z + yz \cdot 1 = x \cdot y + x' \cdot z + yz(x+x')$

$= x \cdot y + x' \cdot z + y \cdot z \cdot x + yz \cdot x'$

$= x \cdot (y + y \cdot z) + x'(z + y \cdot z)$

$= x \cdot y + x' \cdot z$

(2) $(x+y) \cdot (x'+z) \cdot (y+z)$

$= (x+y) \cdot (x'+z) \cdot (y+z + x \cdot x')$

$= (x+y) \cdot (x'+z) \cdot (y+z+x) \cdot (y+z+x')$

$= (x+y) \cdot (x+y+z) \cdot (x'+z) \cdot (y+x'+z)$ $[A \cdot (A+B) = A]$

$= (x+y) \cdot (x'+z)$

If f_1 & f_2 are 2 switching expressions :-

$f_2, f_1 = \{x_1, x_2, x_3, \ldots x_n, +, \cdot, '\}$

$f_1 - 0/1$ → Truth values

$f_2 - 0/1$

} → f_1, f_2 can be switching variables

$f_1 \cdot f_2$, $f_1 + f_2$, f_1', f_2'

↳ All switching expression.

Switching ~~expression~~ function : $f(x_1, x_2, x_3, \ldots x_n)$

x	y	z	
0	0	0	$x'y'z'$
0	0	1	$x'y'z$
0	1	0	$x'yz'$
0	1	1	$x'yz$
1	0	0	$xy'z'$
1	0	1	$xy'z$
1	1	0	xyz'
1	1	1	xyz

0 – x'
1 – x

Now, say we have

x	y	z	f
0	0	0	1 ✓
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1 ✓
1	0	1	0
1	1	0	0
1	1	1	1 ✓

What is f ?

$f = x'y'z' + xy'z' + xyz$

↳ ∴ f is not unique

New f: $x'y'z' + xy'z' + xyz$

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

$= y'z' + xyz$

→ Actually,

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

independent of x for this y & z

∴ directly we can write $y'z'$

Q: $f_1 = x'y'z' + xy'z' + xyz$ → Canonical form

$f_2 = y'z' + xyz$ → Not !

Check whether f_1 & f_2 are equivalent

↳ Use canonical form

≠ Canonical form of switching functions :–

→ All the terms contain all the variables. ☆ (either in normal or complementary) x or x'

$f(x,y,z)$ → $n=3$ → 2^n terms (product form)

$x'y'z'$, $x'y'z$, $x'yz'$, ---- ☆ MINTERMS

$f_2 = y'z' + xyz$

↳ sum of product

$f = (y+z).(x'+y'+z')$ → Product of sum

↓

8 such terms

MAXTERMS

⟹ MISSING → Shanon's expansion/decomposition theorem

DeMorgan's Theorem for complementation :-

$(x')' = x$ → Involution

$(x+y)' = x'.y'$

$(x.y)' = x'+y'$

Dual :- $f(x_1, x_2, \ldots x_n, +, ., 0, 1)$

$\downarrow$

$f_d(x_1, x_2, \ldots x_n, ., +, 0, 1)$

Dual & complement are different

All switching expressions of a 2 variable functions :-

$f(x,y) = a_0 x'y' + a_1 (x'y) + a_2 xy' + a_3 xy$

$a_i \rightarrow 0/1$

a_0	a_1	a_2	a_3
0	0	0	0
0	0	0	1
⋮	⋮	⋮	⋮

a_0	a_1	a_2	a_3	$f(x,y)$	Name
0	0	0	0	—	Inconsistency
0	0	0	1	$f = x.y$	AND
0	0	1	0	$f = xy'$	—
0	0	1	1	$f = x$	—
0	1	0	0	$f = x'.y$	—
0	1	0	1	$f = y$	—
0	1	1	0	$f = xy' + x'y$	XOR
0	1	1	1	$f = x+y$	OR
1	0	0	0	$f = x'y'$	NOR
1	0	0	1	$f = xy + x'y'$	EQUIVALENCE
1	0	1	0	$f = y'$	NOT
1	0	1	1	$f = y'+x$	$y \rightarrow x$ Implication
1	1	0	0	$f = x'$	NOT
1	1	0	1	$f = x'+y$	$x \rightarrow y$
1	1	1	0	$f = x'+y'$	NAND
1	1	1	1		

FUNDAMENTALS OF SWITCHING ALGEBRA

* functionally complete set of operations:

Canonical form of switching expression: SOP → x+y'z ; POS → (x+y)(y'+z)

For SOP: $(x+y'z) = x \cdot 1 \cdot 1 + y'z \cdot 1$

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

$f(x,y,z) = xyz + xyz' + xy'z + xy'z' + xy'z + x'y'z$

Canonical form

For POS: $(x+y)(y'+z)$

$(x+y+z\cdot z')(xx'+y'+z)$ ($xx' = zz' = 0$)

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

(product words)

Minterms → xyz, x'yz, ...

SOP = Σ(0,5,6,7) → x'y'z' + xy'z + xyz' + xyz

x	y	z		
0	0	0	— 0	x'y'z'
0	0	1	— 1	
0	1	0	— 2	
0	1	1	— 3	
1	0	0	— 4	
1	0	1	— 5	xy'z
1	1	0	— 6	
1	1	1	— 7	xyz

→ AND(.), OR(+), NOT(') {., +, '}

A set of equations is functionally complete if any switching operation (SOP/POS) can be represented or realized by these operations

{., +, '} / {., '} / {+, '}

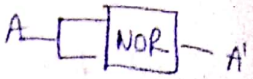
{., '} → NAND ; {+, '} → NOR → De Morgan: (A+B)' = (A'.B')

Q. Show that NOR is functionally complete :-

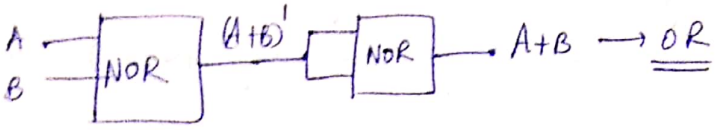
Nor → $(A+B)'$

Now, if $A = B$,

$(A+A)' = A'$ → NOT



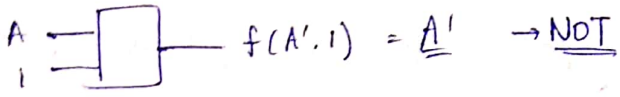
Now,

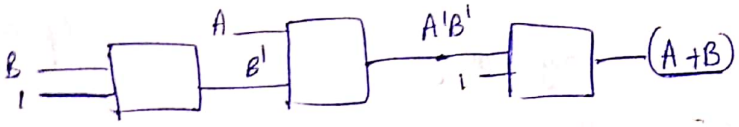


∴ {+, '} can be generated.

∴ fc.

Q. Show that $f(A'B)$ is fc with 1.





Q. Check whether the given ~~function~~ sw. fcn is functionally complete or not.

Incomplete

→ $f(x, y, z) =$

Analogy between switching algebra & Boolean algebra

→ Defn: Boolean algebra is a distributive, complemented lattice
It is commutative, associative and idempotent.
, over distributive & complement

→ A Boolean Algebra B is a set of variables $\{a, b, c\}$ & a set of operations $\{+, \cdot\}$ that satisfies the properties of idempotent, commutative, ~~adaptive~~ absorption, associative & mutually distributive.

There are two bounds ~~(0)~~ 0 (lowest) & 1 (highest)
It has a unary operation of complementation. greatest.

Complementation is unique:-

$$\begin{array}{l|l} a+b_1 = 1 & a+b_2 = 1 \\ a \cdot b_1 = 0 & a \cdot b_2 = 0 \end{array}$$

$b_1 = b_1 \cdot 1 =$ ~~(a+b_2)~~ $b_1(a+b_2) = b_1 a + b_1 \cdot b_2 = b_1 \cdot b_2$

$b_2 = b_2 \cdot 1 = b_2(a+b_1) = b_2 \cdot a + b_2 \cdot b_1 = b_2 \cdot b_1$

$\Rightarrow \boxed{b_1 = b_2}$ ∴ complementation is unique.

Q. Show that BA (Boolean Algebra) satisfy DeMorgan's law.

$(a+b)(a'b') = 0$ → will be true if $a'b' = (a+b)'$

$(a+b) + a'b' = 1$ → Will " " " "

$a+0 = a$	$a+a' = 1$
$a+1 = 1$	$a.a' = 0.$
$a.0 = 0$	
$a.1 = a$	

$(a+b).(a'b') = aa'b' + ba'b' = 0+0$

$(a+b) + a'b' = (a+b+a').(a+b+b') = (1+b)(1+a)$

$= 1$

INCOMPLETE

$\{0, 1, a, b\}$ → $\begin{cases} a+b=1 \\ a.b=0 \end{cases}$

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

Boolean algebra →

·	0	1	a	b
0	0	0	0	0
1	0	1	a	b
a	0	a	a	0
b	0	b	0	b

* Switching algebra is 2-valued Boolean algebra.
→ 2 valued Boolean algebra is isomorphic to switching algebra

* XOR → (XOR) - $x \oplus y = xy' + x'y$

① $\{A, B, C\}$

$A \oplus B = C$

$A \oplus C = B$

commutative :- $A \oplus B = B \oplus A$

associative :- $A \oplus (B \oplus C) = (A \oplus B) \oplus C$

Distributive :- $\{., \oplus\}$

$A.(B \oplus C) = A.B \oplus A.C$

check distributive for $\{+, \oplus\}$

Chapter 3 exercise of Kohavi :

3.3(d) Simplify

$a + a'b + a'b'c + a'b'c'd + a'b'c'd'e + \cdots$

$a + a'b \rightarrow a.1 + a'b$

$\rightarrow a.(1+b) + a'b$

$\rightarrow a + a.b + a'.b$

$\rightarrow a + (a+a').b$

$\rightarrow a + b$

New, $a + a = a$ $\quad \therefore a'b + a'b = a'b$ → Double karde

$\rightarrow a + b + a'(b + b'c) + a'b'c'd + \cdots$

$\rightarrow a + b + a'b + a'c + \cdots$

$\rightarrow a + b + c + \cdots$

$a + a'b$

$\rightarrow a + b$

$\rightarrow (a'b')'$

$+ (a'b'c)$

$a'' + ac$

$\rightarrow a + ac$

$a'c + a'c' + ac$

$(a'b'c')' + \cdots$

$a + b + c + \cdots$

$\rightarrow a + b + c + d + e + \cdots$

(3.4) (b) Find by inspection the complement of the following & simplify.

$$[(x+yz').(y+x'z').(z+x'y')]$$

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

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

$$= x \oplus y + x \oplus z + y \oplus z$$

(3.5) (b) Validate

$$xy + x'y' + x'yz = xyz' + x'y' + yz$$

LHS

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

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

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

$xy + x'y' + yz$

RHS

$xyz' + yz + x'y'$

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

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

$xy + x'y' + yz$

LHS = RHS

(3.6) If $AB' + A'B = C$ then show that $Ac' + A'C = B$

Ans To prove:

↳ $Ac' + A'C = B$

$Ac' + A'C \rightarrow A.(AB'+A'B)' + A'(AB'+A'B)$

XNOR

$A(AB+A'B') + A'(AB'+A'B)$

$AB + 0 + 0 + A'B$

$B(A+A')$

$= B$

XNOR

$\overline{A \oplus B}$

$= AB + A'B'$

(2.7) Find A, B, C, D by solving the following eqn.

$A' + AB = 0$

$AC = AC$

$AB + AC' + CD = CA$

$A'+B=0 \Rightarrow A'=0, B=0$

$\Rightarrow A=1, B=0.$

$AC=0 \Rightarrow C=0$

$0+1+0=A$

$\Rightarrow A=1$

(A, B, C, D)

(1, 0, 0, 1)

(2.22) A safe has 5 locks v, w, x, y, z, all of which must be unlocked for the safe to open. The keys to the locks are distributed among 5 executives in the following way:

A has key to locks	v & x
B	v & y
C	w & y
D	x & z
E	v & z

(a) Determine number of executives required to open the safe

(b) Who is "essential executive" without whom the safe can't be opened?

(c) Find all combinations of executives that can open the safe.

Exec \ key	v	w	x	y	z
A	✓		✓		
B	✓			✓	
C		✓		✓	
D			✓		✓
E	✓				✓

{w, y / C ; v, x / A ; z / D or E}

CAD / CAE → read.

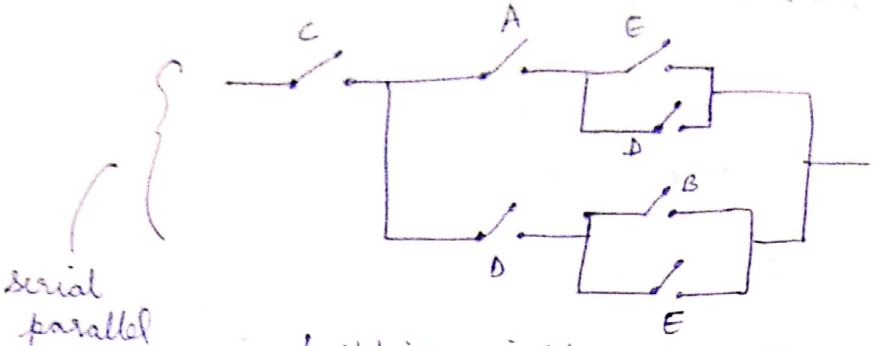
(3)

Only C has w.

∴ C is essential executive

Valid combns

$f(A,B,C,D,E) = CAE + CAD + CBD + CDE$



Serial parallel network

Switching circuit for this $f(A,B,C,D,E)$

Minimization of switching expression

Used algebraic properties to minimize the switching expression.

$$f(x,y,z) = xyz' + x'yz' + yz$$
$$= yz' + yz$$
$$= y \hookrightarrow \text{Irredundant \& minimal}$$

→ Minimal expression → Cannot get another expression with fewer literals.

→ Irredundant expression. → Cannot be further reduced

1.) Minimum no. of literals.

2.) Minimum no. of terms.

$$f = \Sigma(2,0,4,3,7,5)$$

Truth Table

Input			output	
x	y	z		
0	0	0	0	— 1
0	0	1	1	— 0
0	1	0	2	— 1
0	1	1	3	— 1
1	0	0	4	— 1
1	0	1	5	— 1
1	1	0	6	— 0
1	1	1	7	— 1

$f(x,y,z) = x'y'z' + x'y'z + xy'z' + x'yz + xyz + xyz'$

$= x'z'(1) + x'y(1) + xy'(1) + x'yz'$

$= x'z' + x'y + xy' + yz$

① $= x'z' + yz + xy'$ → Irredundant

② $= x'y + y'z' + xz$ → Irredundant

Karnaugh maps — Method of minimization

↳ $A.a + A.a' = A$

$A(a+a') = A.1 = A$

$f(a,b,c,d)$ say

$f(d,b,c)$ say

One bit vary

x \ yz	00	01	11	10
0	1	0	1	1
1	1	1	1	0

xy \ z	0	1
00	1	0
01	1	1
11	0	1
10	1	1

x \ yz	00	01	11	10
0	0	1	3	2
1	4	5	7	6

Adjacent → Terms are varying in 1 literal.

Cube → Searching for 2 adjacent cells.

↳ 2^m cells → $m = 1, 2, 3, \ldots$

If f is a n-variable fn & we can identify a 2^m cube then the minimized term have $(n-m)$ literals.

$xz' + yz + x'y'$

$y'z' + xz + x'y$

Say, we have →

x \ yz	00	01	11	10
0	0	1	1	1
1	1	0	1	1

$y + x'z + xz'$

Q $f = \Sigma\ 4, 5, 6, 8, 12, 13, 14, 15$

ab \ cd	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

ab \ cd	00	01	11	10
00	0	0	0	0
01	1	1	0	1
11	1	1	1	1
10	1	0	0	0

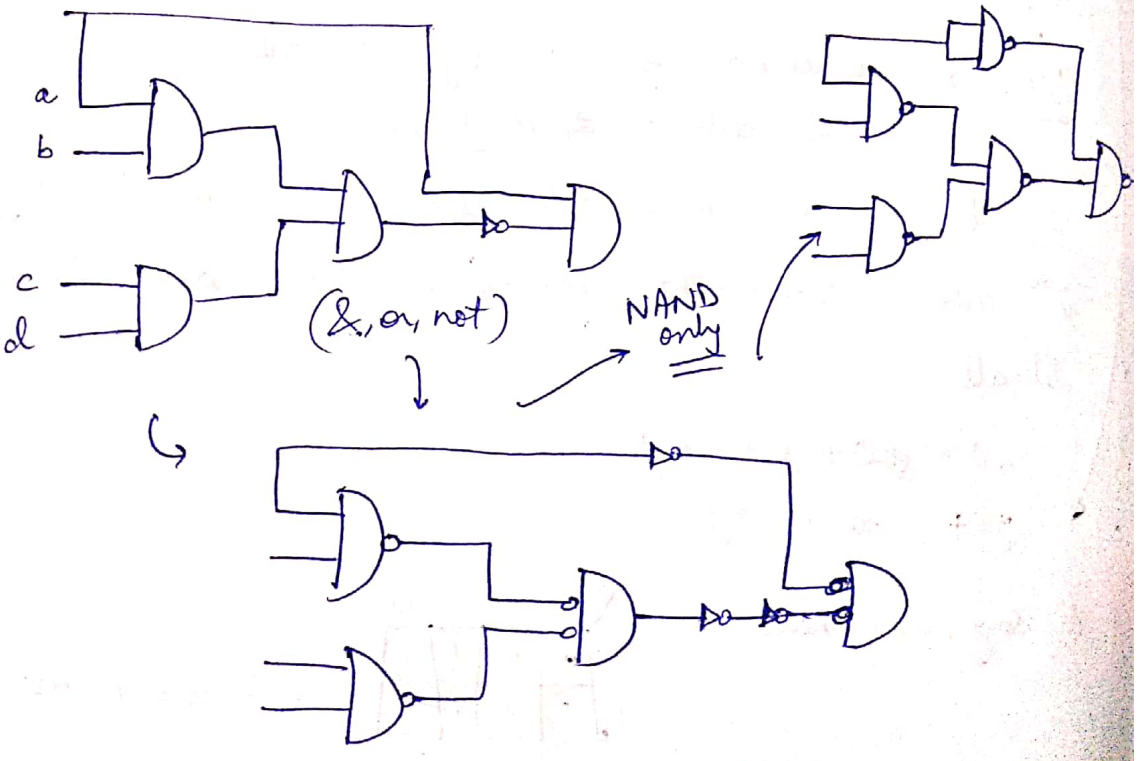
$f \longrightarrow ac'd' + bc' + bd' + ab$

AND-OR expression

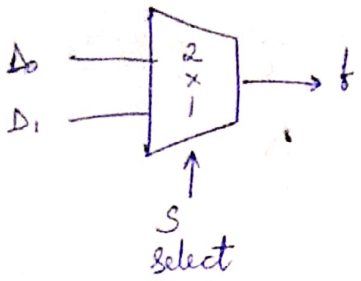
ab \ cde	000	001	011	010	110	111	101	100
00								
01								
11								
10								

⇒ NAND $(A.B)' = A' + B'$ OR, NOT

⇒ NOR $(A+B)' = A'.B'$ &, NOT



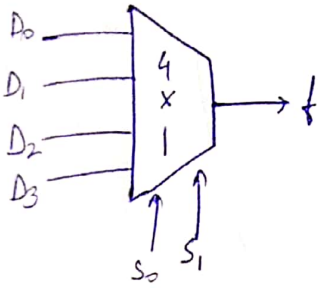
Multiplexer :- Data selector



$S = 0$, D_0 selected
$S = 1$, D_1 selected

$$f = S'D_0 + SD_1$$

If n - input variables, $S = \log_2 n$



$$f = S_0'S_1'D_0 + S_0'S_1D_1 + S_0S_1'D_2 + S_0S_1D_3$$

Adder

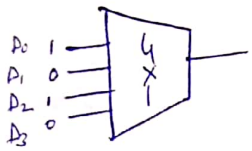
$$S = A \oplus B \oplus C$$

$A \oplus (BC' + B'C)$
$= A'(BC' + B'C) + A(BC + BC')$

$$f(A,B,C) = ABC + AB'C' + A'B'C + A'BC'$$

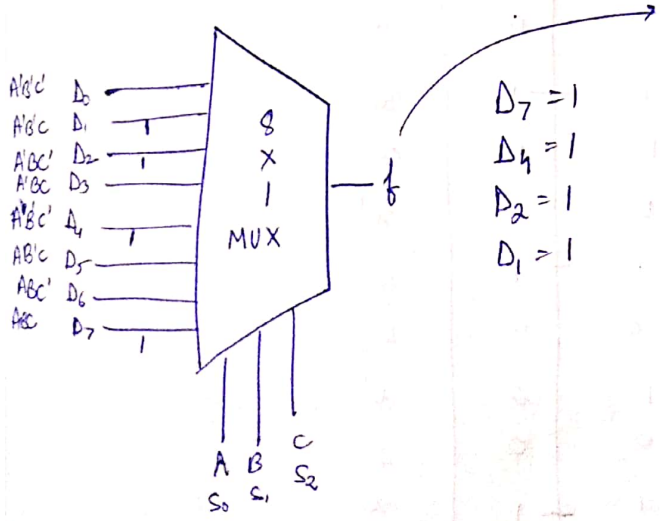
Consider 4x1 MUX

$$f = S_0'S_1'D_0 + S_0'S_1D_1 + S_0S_1'D_2 + S_0S_1D_3$$



$$f = S_0'S_1' + S_0S_1'$$

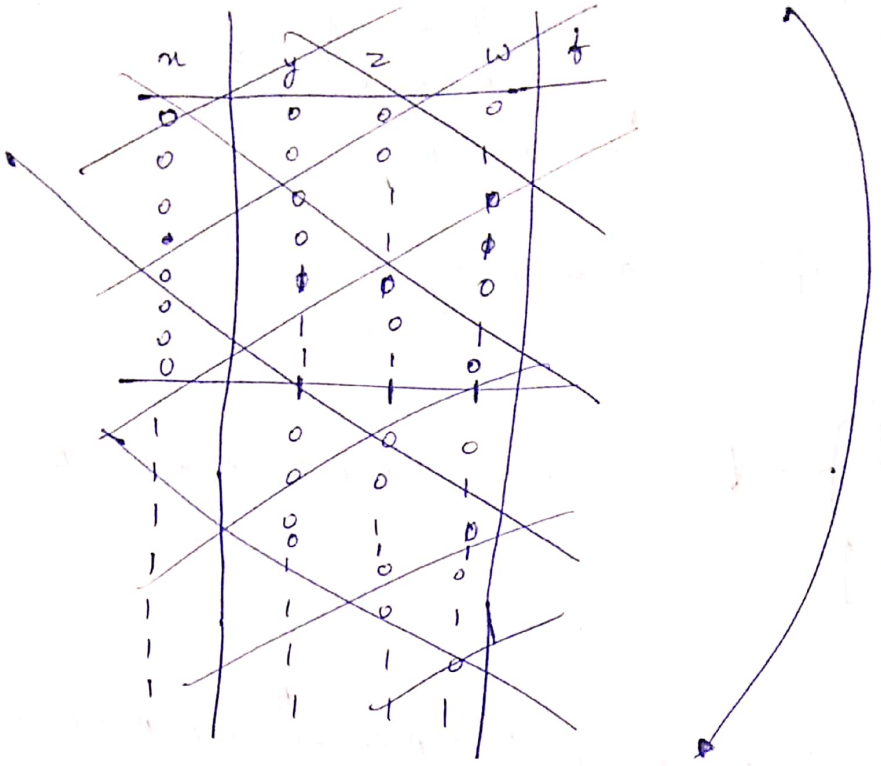
$$f = ABC + AB'C' + A'B'C + A'BC'$$



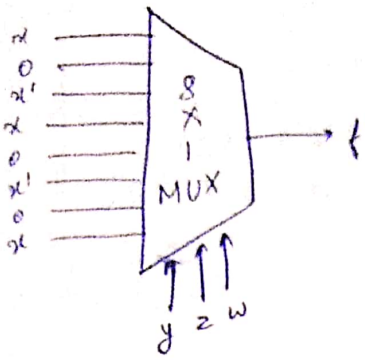
$D_7 = 1$
$D_4 = 1$
$D_2 = 1$
$D_1 = 1$

→ For n-variable fn to be realized, we need $2^n \times 1$ MUX (n-select 1)

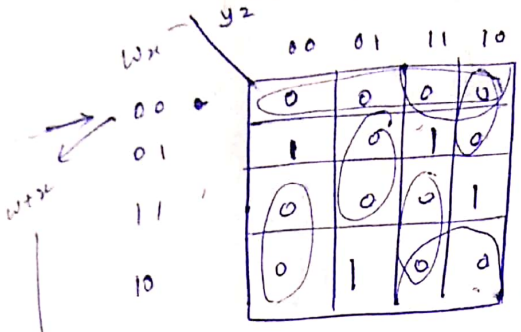
$f(x,y,z,w) = x'y'z'w' + x'yz'w + xy'z'w' + xy'zw + xyzw$



x	y	z	w	f	f
0	0	0	0	0	x
0	0	0	1	0	0
0	0	1	0	1	x'
0	0	1	1	0	x
0	1	0	0	0	0
0	1	0	1	1	x
0	1	1	0	0	0
0	1	1	1	0	x
1	0	0	0	1	x
1	0	0	1	0	0
1	0	1	0	0	x'
1	0	1	1	1	x
1	1	0	0	0	0
1	1	0	1	0	x'
1	1	1	0	0	0
1	1	1	1	1	x



Minimization with product of sum expression :-



$f = w'xy'z' + w'xyz + wx'y'z + wxyz'$

→ 4 product terms in 16 literals

SOP cheaper than POS, here

$(w'.x')' = (w')' + (x')'$

$= w + x$

Using POS → $f = (w+x).(x+y').(w'+y+z).(w'+x'+z').(w+y'+z).(x'+y'+z)$

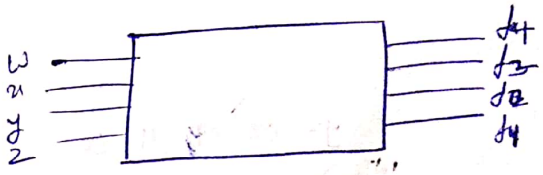
* Something about DONT care condition → for invalid inputs, put any value, which reduces terms

Q. Implement BCD to excess-3 converter → Only for 0–9

Decimal	BCD (w x y z)	Excess 3 (f_4 f_3 f_2 f_1)
0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 1 0 1
3	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 0 0
6	0 1 1 0	1 0 0 1
7	0 1 1 1	1 0 1 0
8	1 0 0 0	1 0 1 1
9	1 0 0 1	1 1 0 0

4 bit

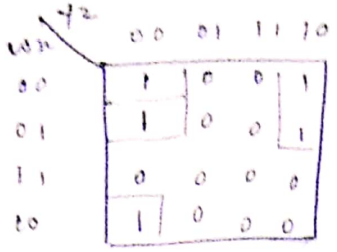
so 10,11,12,13,14,15 inputs can still be there. (here, they are invalid)



P.T.O

K-Maps for f_1, f_2, f_3, f_4

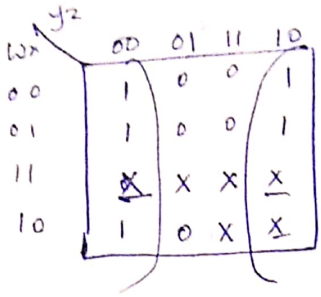
f_1 :-



$f_1 = w'z' + x'y'z'$

Without dont care

With dont care :

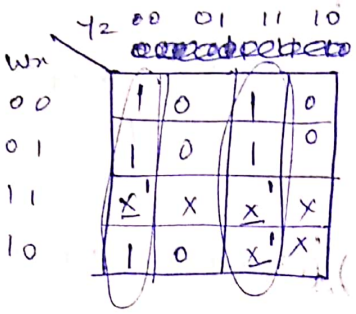


Put X = 1 for —,
1 term only.

$f_1 = z'$

Do for f_2, f_3, f_4 with dont care.

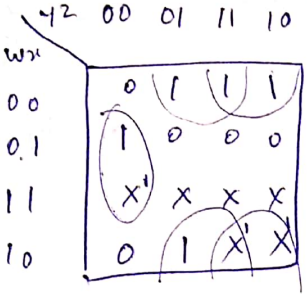
f_2 :



Put X = 1 for —,

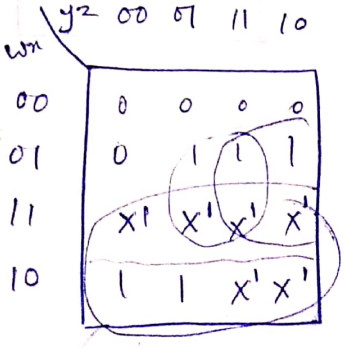
$f_2 = y'z' + yz$

f_3 :



$f_3 = x'y + x'z + xy'z'$

f_4 :



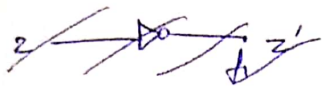
$f_4 = xz + xy + w$

$\therefore f_1 =$ [illegible] z'

$f_2 = y'z' + yz$

$f_3 = xy'z' + yx' + x'z$

$f_4 = xz + xy + w.$



Map-entered variable method for k-map :-

wx \ yz	00	01	11	10
00	1	(A)	A	0
01	0	1	C	0
11	(A')	1	1	0
10	B	1	B'	1

means f depends on A'

$f(w, x, y, z)$

↓

$f(w, x, y, z, A, B, C)$

→ A & A' are treated as different variables

$f = wy'z + w'x'y'z' + w'xy'z + wx'y'z' + wxyz$

Now,

for A → make it 1

other k-map variables → don't care (1's only)

other map-entered variables → 0 (A', B, B', C)

wx \ yz	00	01	11	10
00	x	1	1	0
01	0	x	0	0
11	0	x	x	0
10	0	x	0	x

→ $f_A = A.w'x'z'$

clearly, $f_C = C.xz$ (centre square)

$f_{B'} = B'wz$

$f_B =$

⇒ Map-entered variable. (use K-map)

Step ① Put 0 in all the cells containing map-entered variable. Find the minimal expression using K-map.

Step ② Now the fn f depends on A) Make all other map entered variables 0. & make all 1's as dont care. (Dont touch the zeroes) Then get a minimal expression using product of $A \cdot f_A$

Step ③ Repeat ② for other map-entered variables.

$$f = f_{K\text{-Map}}(w, x, y, z) + A \cdot f_A + A' \cdot f_{A'} + B \cdot f_B + B' \cdot f_{B'} + \ldots$$

Quine-McClaarsky Method:

Switching or Boolean expression

Minimal expression

Irredundant expression → cant be reduced further.

Prime implicants:

Let $f(x_1, x_2, \ldots x_n)$ & $g(x_1, x_2, \ldots x_n)$ are two switching ~~expressions~~ functions.

Now, f covers g, if f contains a 1 everywhere where g does. → $f \supseteq g$

Now, if f covers g & g covers f, then f and g are equivalent.

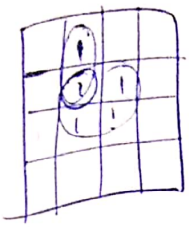
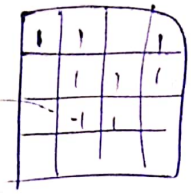
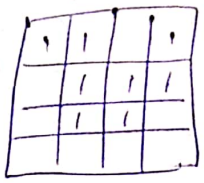
Let $h(x_1, x_2, \ldots x_n)$ be product term.

If f covers h, then it is said $\boxed{h \to f}$, h is one implicant of f.

(*) A prime implicant (p) is a product term that implies the function f & dropping any literal from p produces another product term that does not imply f

Essential prime implicant:

One prime implicant is essential if it covers a cell (min term) which is not covered by any other prime implicant

→ Sum (Union) of all essential prime implicants is one of the irredundant expressions.

Theorem:

An irredundant Boolean expression is the sum (union) of all essential prime implicants

Proof: Let f' be an irredundant expression which has one pre-implicant (Prime)??

If dropping p ⇒ f' is still irredundant ⇒ literals can be reduced ⇒ p expression can be reduced.

Essential PI ⇒ Irredundant

Prime Implicant ⇒ Minimal exp.

Quine Mecluaskey

$f = \sum_m (0,1,2,5,7,8,9,10,13,15)$

Step 1

No. of 1's	group	m	w x y z
(0)	1	0	0 0 0 0
(1)	2	1	0 0 0 1
		2	0 0 1 0
		8	1 0 0 0

(2)

PTO

Q → $\Sigma(0,1,2,5,7,8,9,10,13,15)$

Step 1

Index	Minterm	w	x	y	z
0	0	0	0	0	0
1	1	0	0	0	1
	2	0	0	1	0
	8	1	0	0	0
2	5	0	1	0	1
	9	1	0	0	1
	10	1	0	1	0
3	7	0	1	1	1
	13	1	1	0	1
4	15	1	1	1	1

Index ↓
No. of 1's in the term

Step 2

Group of Min term	w	x	y	z	
0, 1	0	0	0	–	✓
0, 2	0	0	–	0	✓
0, 8	–	0	0	0	✓
1, 5	0	–	0	1	✓
1, 9	–	0	0	1	✓
2, 10	–	0	1	0	✓
8, 9	1	0	0	–	✓
8, 10	1	0	–	0	✓
5, 7	0	1	–	1	✓
5, 13	–	1	0	1	✓
9, 13	1	–	0	1	✓
7, 15	–	1	1	1	✓
13, 15	1	1	–	1	✓

Step 3

	w	x	y	z	
0, 1, 8, 9	—	0	0	—	$A = x'y'$
0, 2, 8, 10	—	0	—	0	$B = x'z'$
Repeat → 0, 8, 1, 9	—	0	0	—	
1, 5, 9, 13	—	—	0	1	$C = y'z$
5, 7, 13, 15	—	1	—	1	$D = xz$

No more minimizations possible. $f = A + B + C + D$

Decimal representation Quin - McClaskey:

1) Difference of ~~three~~ two minterms with different indices is power of 2. (1 bit difference)

2) If the minterm with smaller index is larger than the minterm with larger index, then it cannot be combined even if the difference between minterms is power of 2.

3) Minterms with same indices are not covered.

Q. $f(v, w, x, y, z) = \Sigma(13, 15, 17, 18, 19, 20, 21, 23, 25, 27, 29, 31) + \sum_d(1, 2, 12, 24)$

→ Initially, we consider don't care as 1.

Step 1

Index	Minterms
1	1
	2
2	12
	17
	18
	20
	24
3	13
	19
	21
	25
4	15
	23
	27
	29
5	31

Step 2 (2^k → gives pos^n of → *)

	1, 17	(16)	− A
	2, 18	(16)	− B
17, 19 (2)	12, 13	(1)	− C
	17, 21	(4)	
	17, 25	(8)	
	18, 19	(1)	− D
	20, 21	(1)	− E
	24, 25	(1)	− F
21, 23 (2)	13, 15	(2)	
	13, 29	(16)	
	19, 23	(4)	
	19, 27	(8)	
	21, 29	(8)	
	25, 29	(4)	

index of 17 < 13

(17, 13) → X

	15, 31	(16)
	23, 31	(8)
	27, 31	(4)
	29, 31	(2)

Step 3 Now, we check if differences match or not

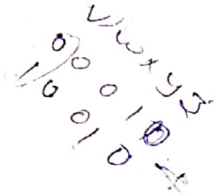
17, 19, 21, 23	(2, 4)	✓
17, 21, 25, 27	(2, 8)	✓
17, 21, 25, 29	(4, 8)	✓
13, 15, 29, 31	(2, 16)	G
19, 23, 27, 31	(4, 8)	✓
21, 23, 29, 31	(2, 8)	✓ (Already covered)
25, 27, 29, 31	(2, 4)	✓

Step 4

17, 19, 21, 23, 25, 27, 29, 31 → (2, 4, 8) — H

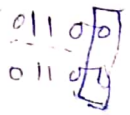
↳ Max combination we can make

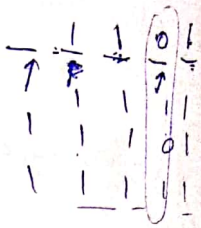
A → H are prime implicants



1, 17	(16)	A
2, 18	(16)	B
12, 13	(1)	C
18, 19	(1)	D
20, 21	(1)	E
21, 25	(4)	F
13, 15, 29, 31	G	(2, 16)
17, 19, 21, 23, 25, 27, 29, 31	H	(2, 4, 8)

NOT essential prime implicants





Now, to find essential prime implicants:

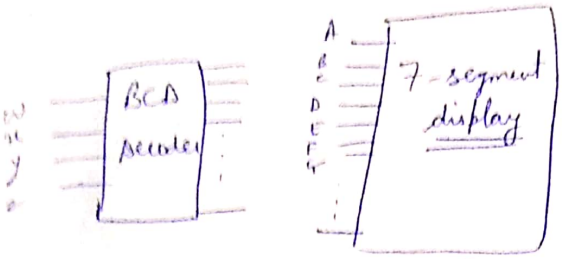
	13	15	17	18	19	20	21	23	25	27	29
v w x y z (16 8 4 2 1)											
w'x'y'z' A			X								
w'xyz' B				X							
v'wx'y C	X										
v'w'x'y D											
vw'x'y E				X	X						
vwx'y' F						(X)	X				
wxz G		X	(X)				X		X		X
vz H			X		X		X	(X)	X	(X)	X

E, G, H → essential prime implicant

B/D koi bhi for 18

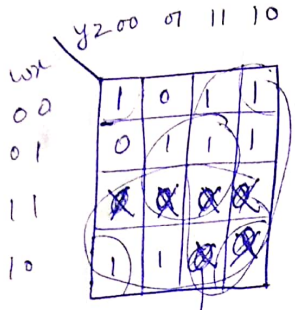
7-segment display

BCD to 7-segment display



Decimal	BCD				A	B	C	D	E	F	G
	w	x	y	z	f_1	f_2	f_3	f_4	f_5	f_6	f_7
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	0	0	1	1

f_1



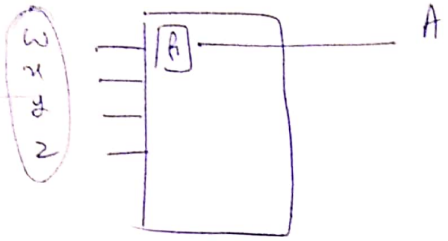
$f_1 = x'z' + y' + w + xz'$ → with dont care

Dont care only if all inputs are 0-9. Otherwise 0. (~~scribbled~~)

Without don't care:

wx \ yz	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	0	0	0	0
10	1	1	0	0

$f_1 = w'y + xz + wx'y' + x'y'z$



Quine – McCluskey method of minimization:

$f = \sum (1, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 18, 19, 20, 21, 22, 23, 25, 26, 27)$

→ Minimal expression
→ Prime implicants
→ Essential prime implicants

we have already found prime implicants (assume)

		1	3	4	5	6	7	10	11	12	13	14	15	18	19
(1,3,5,7) w'x =	A			X	X	X	X								
✓ v'x	B			X	X	X	X			X	X	X	X		
vx'y	C													X	X
vw'y	D												X	X	
wx'y	E							X	X						
v'wy	F							X	X			X	X		
x'yz	G		X						X						X
w'yz	H		X				X						X		
v'yz	I		X			X	X		X				X		
✓ v'w'z	J	X	X	X	X								X		
vwx'z	K														

One entry in column val mark, those p.i are e.p.i ☆

A, B, J, K

	10	11	18	19	26
C	—	—	X	X	X
D	—	—	X	X	
E	X	X	—	—	X
F	X	X	—	—	—
G	—	X	—	X	—
H					
I		X			

C, E ↑

$\Rightarrow f(v,w,x,y,z) = A + B + \cancel{J} + K + C + E$

v	w	x	y	z
–	0	1	–	–
–	0	1	–	–

16 2 1

	21	22	23	25	26	27
A		X	X			
B						
C					X	X
D						
E		X	X		X	X
F					⊗	⊗
G						X
H						
I		X	X			
J		⊗		⊗	⊗	
K				X	X	

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

vw x'z

A – K prime implicants

Q

$f(v, w, x, y, z) = \sum (0, 1, 3, 4, 7, 13, 15, 19, 20, 22, 23, 29, 31)$

(2)

	0	1	3	4	7	13	15	19	20	22	23	29
A						x	x					
B					x		x				x	
C			x		x			x			x	
D										x	x	
E									x	x		
F				x					x			
G			x	x								
H	x			x								
I	x	x										

A (13, 15, 29, 31)
B (7, 15, 23, 31)
C (3, 7, 19, 23)
D (22, 23)
E (20, 22)
F (4, 20)
G (1, 3)
H (0, 4)
I (0, 1)

(3)

	0	1	4	20	22
D					x
E				x	x
F			x	x	
G		x			
H	x		x		
I	x	x			

No essential PI here, so →

Now to reduce:

$P = (H+I).(G+I).(F+H).(E+F).(D+$

(H covers 0, I covers 1, F covers 4, E covers 20, 22)

$(I+G.H)(E+DF)(F+H)$

$= (FI + HI + FGH + GH).(E+DF)$

$= (FI + HI + GH).(E+DF)$

$= EFI + EHI + EGH + DFI + DFHI + DFGH$

$f = A + C + EFI$
$\quad\;\; " \quad EHI$
$\quad\;\; " \quad EGH$
$\quad\;\; " \quad DFI$
$\quad\;\; " \quad DFGH \rightarrow X$ → more literals.

Combinational circuit design:

Present output depends on present input

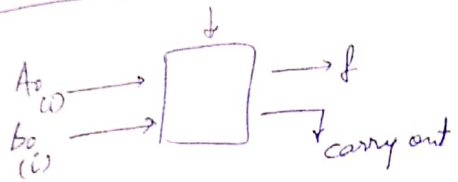
Sequential circuit design :-

Present output depends on present input & some past history.

Arithmetic Logic unit :-

+, -, *, / <, >, &, |

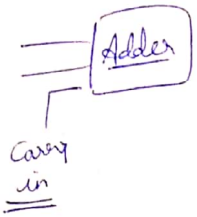
⇒ Design of 1 bit ALU. &, |, Sum



16 bit
i (0 → 15)

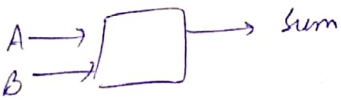
&

OR

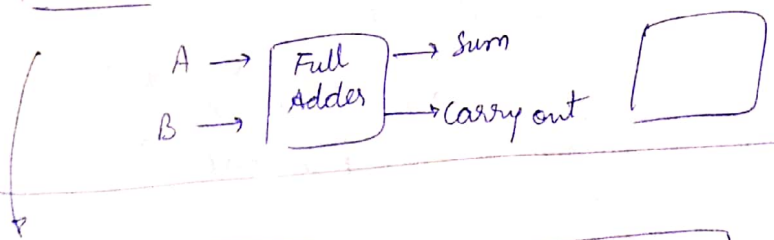


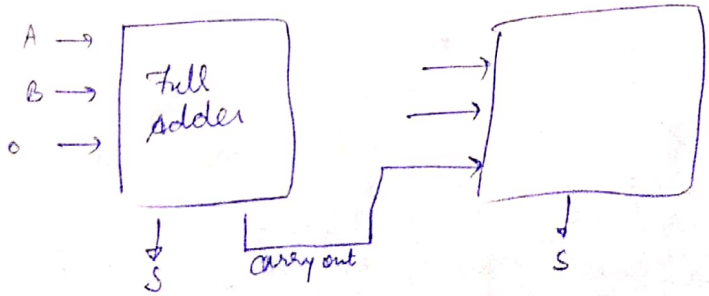
A	B	Carry out	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

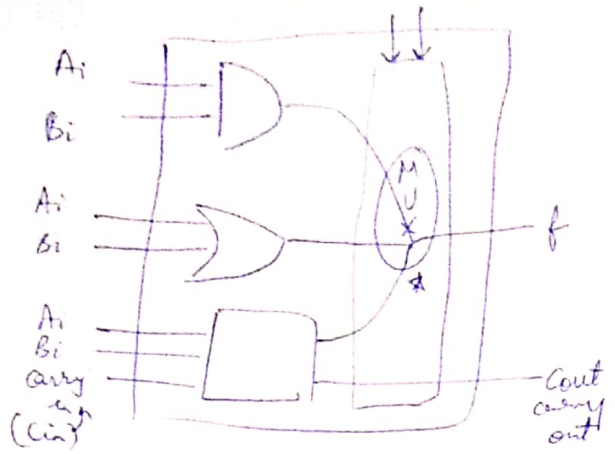
Half Adder:



Full Adder:







Sum = $A \oplus B \oplus C$ ★ Make Karnaugh Map & get

A	B	C_{in}	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Make Karnaugh Map to get

Cout = $AB + BC + CA$.

∴ Adder → Sum = $A \oplus B \oplus C$

Cout = $AB + BC + CA$

Control	
0 0	&
0 1	or
1 0	Add/Sub (with add/sub control / inv)
1 1	xor

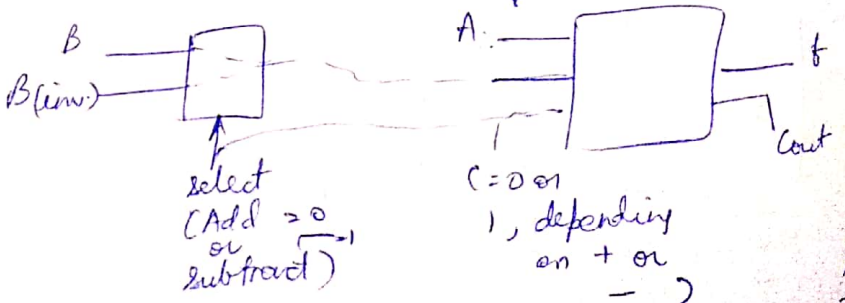
A + B

→ A + (−B)

↳ A + (2's comp of 1's)

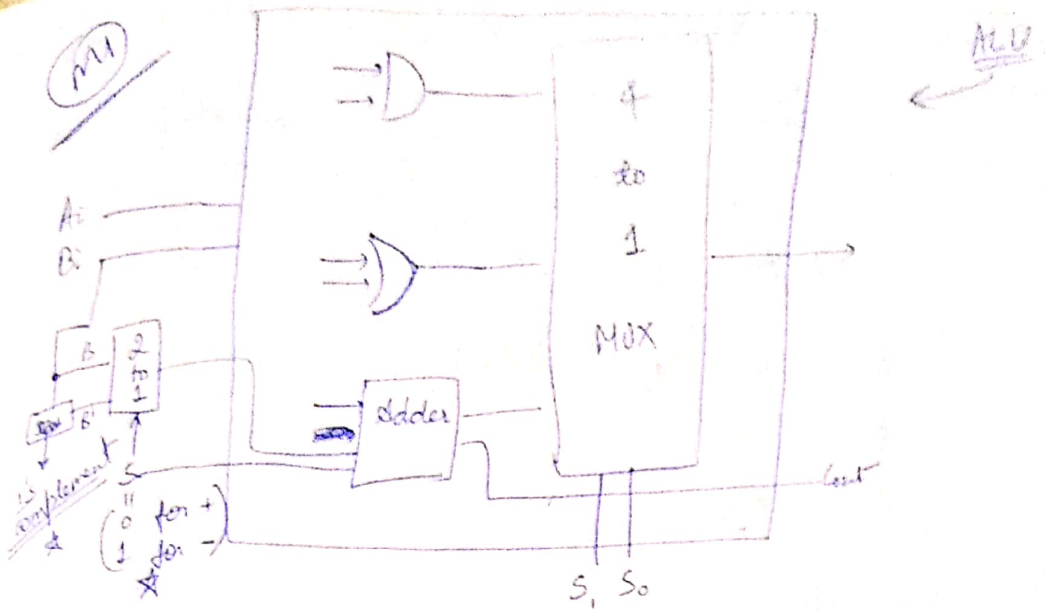
↳ A + (1st comp of B + 1)

A + invB + 1



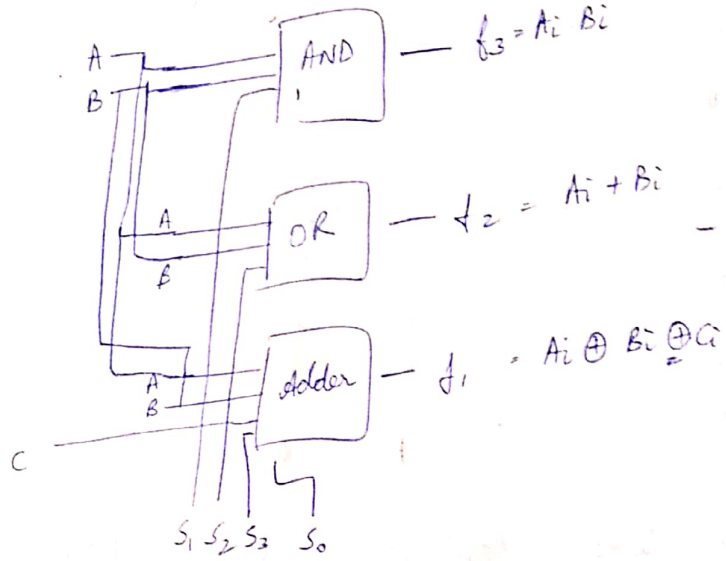
$a_0 AB + a_1 \bar{A}B + a_2 A\bar{B} + a_3$

M1



S_1	S_0	
0	0	And
0	1	or
1	0	add / sub
1	1	XOR

M2



$$f = S_3 \cdot f_3 + S_2 \cdot f_2 + S_1 \cdot f_1 + S_0 f_0$$

Read about 2 var fns

$S_0'S_1'$ - - -

MSI
LSI - few
VLSI - >10000

Combinational circuit design :

(Comparator <, =, >

→ i^{th} stage 1 bit ALU.

$(x_i \odot y_i)$

$x = y$	=	$f_2 = x_1y_1 + x_1'y_1'$
$(x > y)$	>	$f_1 = x_1y_1'$
$(x < y)$	<	$f_3 = x_1'y_1$

⇒ 2-bit comparator

x	y	x>y	x=y	x<y
x_1x_2	y_1y_2	f_1	f_2	f_3
00	00	0	1	0
00	01	0	0	1
00	10	0	0	1
00	11	0	0	1
01	00	1	0	0
01	01	0	1	0
01	10	0	0	1
01	11	0	0	1
10	00	1	0	0
10	01	1	0	0
10	10	0	1	0
10	11	0	0	1
11	00	1	0	0
11	01	1	0	0
11	10	1	0	0
11	11	0	1	0

⇒ We cannot take dont care, same reason as prev. time

→ We can make Karnaugh map we can make for all f_1, f_2, f_3 @ once only.

x_1x_2 \ y_1y_2	00	01	11	10
00	2	3	3	3
01	1	2	3	3
11	1	1	2	1
10	1	1	3	2

2 = 1 for f_2
3 = 1 for f_3
1 = 1 for f_1

$\rightarrow f_1 = x_1y_1' + x_2y_1'y_2' + x_1x_2y_2' = x_1y_1' + x_2y_2'(x_1 + y_1')$

$\rightarrow f_2 = x_1'x_2'y_1'y_2' + x_1'x_2y_1'y_2 + x_1x_2'y_1y_2 + x_1x_2y_1y_2'$

$= x_1'y_1'(x_2'y_2' + x_2y_2) + x_1y_1(x_2y_2 + x_2'y_2')$

$= (x_1'y_1' + x_1y_1)(x_2'y_2' + x_2y_2)$

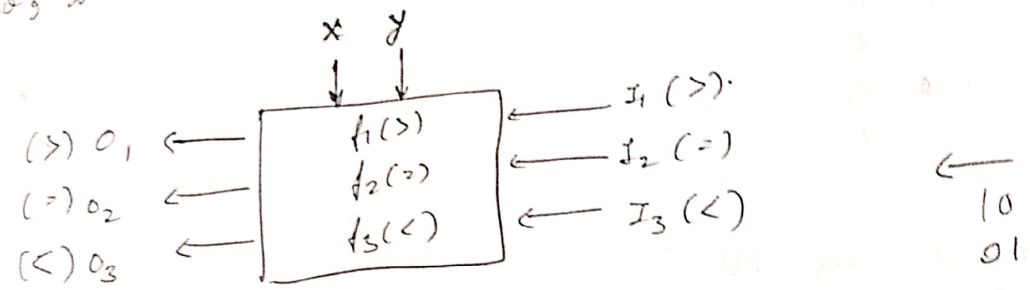
$\rightarrow f_2 = (\overline{x_1 \oplus y_1}).(\overline{x_2 \oplus y_2})$

$\Rightarrow f_3 = x_1'y_1 + x_1'x_2'y_2 + x_2'y_1y_2$

$= x_1'y_1 + x_2'y_2(x_1' + y_1)$

$\Rightarrow$ So, we are not getting any definite repetition here, since we require knowledge of previous input (MSB side) [memory

So, we construct like this:

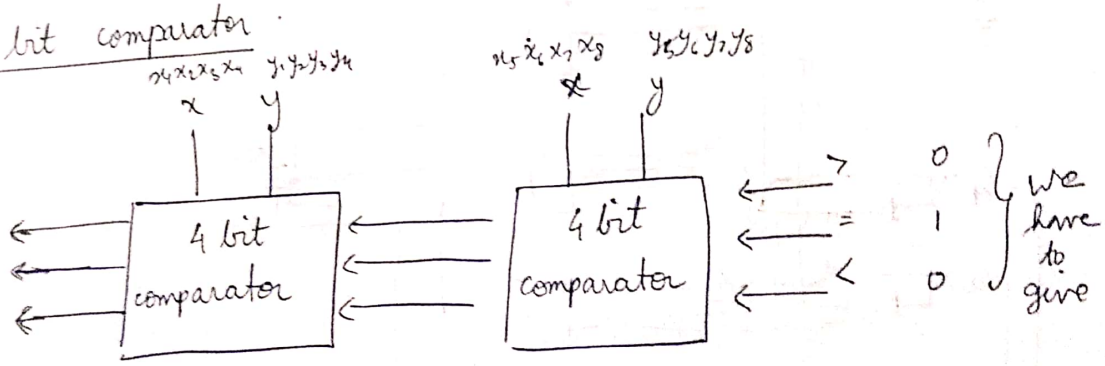


$O_1 = f_1 \cdot \textcircled{*} + f_2 \cdot I_1$

$O_2 = I_2 \cdot f_2$

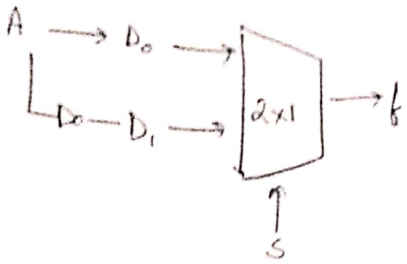
$O_3 = f_3 + f_2 \cdot I_3$

8 bit comparator



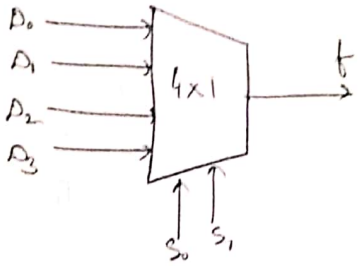
For 4 bit comparator, the simplification is better, so we use like this.

Multiplexer can be used to realize all gates



$S=0 \Rightarrow f = D_0$
$S=1 \Rightarrow f = D_1$
$f = D_0S' + D_1S$

$f = AS' + A'S$
$\therefore f = XOR(A,S)$



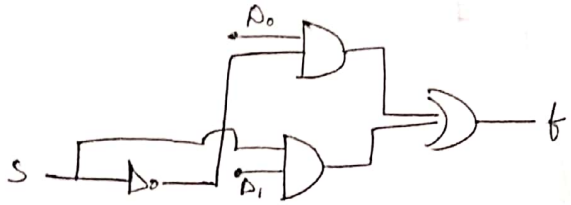
$f = S_0'S_1'D_0 + S_0'S_1D_1 + S_0S_1'D_2 + S_1S_0D_3$

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

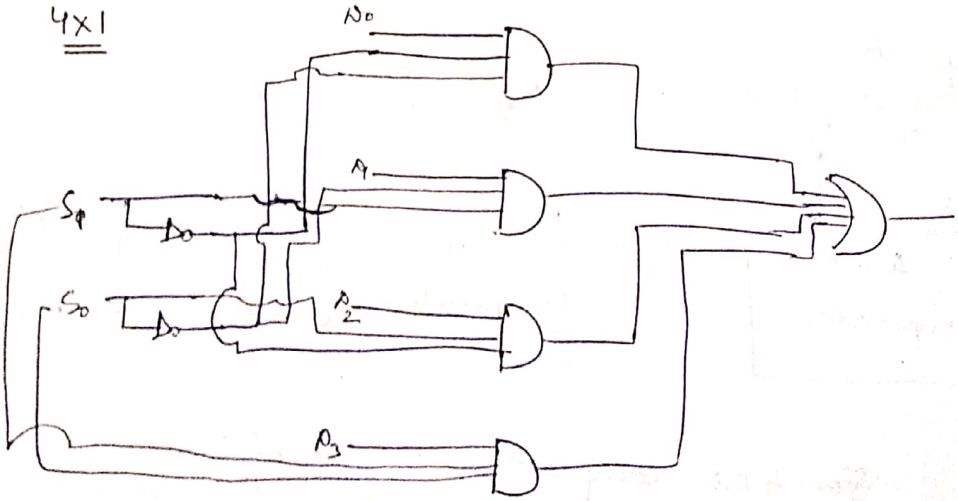
for 8×1 MUX:
$f = S_0'S_1'S_2'D_0 + S_0'S_1'D_1 + \ldots$
8 terms

Circuit for MUX:

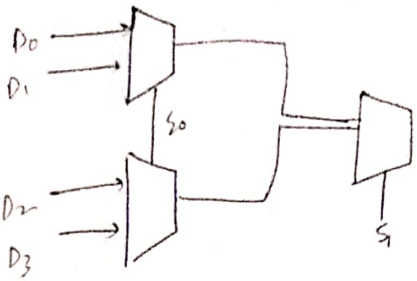
2×1

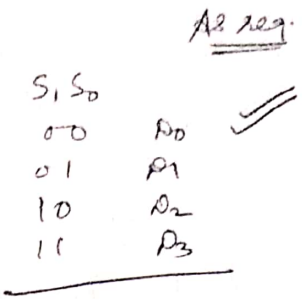


4×1



4×1 MUX using 2×1 MUX :



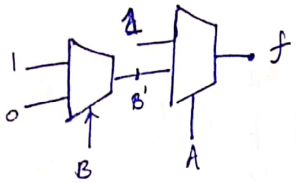


Combinational circuit design :

→ Multiplexer
→ Priority encoder
→ Decoder

Realize all gates using multiplexer.

⇒ 2 input ~~NAND~~ (NOR)?



(A.B)

A	B	f
0	0	0
0	1	0
1	0	0
1	1	1

$\overline{(A+B)}$

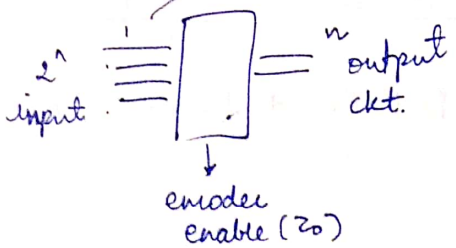
$\overline{A}.\overline{B}$

A	B	f
0	0	1
0	1	1
1	0	1
1	1	0

NAND

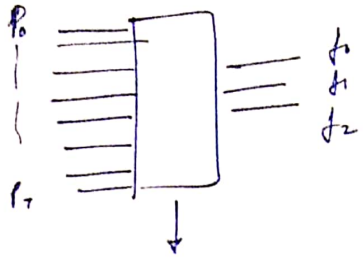
Priority encoder :-

Say, I want to give higher priority to some input lines.



Input lines – P_i
If $i > j$, then input P_i has higher priority than P_j.

Ex: $n = 3$



I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	f_0	f_1	f_2
1	0	0	0	0	0	0	0	0	0	0
ϕ	1	0	0	0	0	0	0	0	0	1
ϕ	ϕ	1	0	0	0	0	0	0	1	0
ϕ	ϕ	ϕ	1	0	0	0	0	0	1	1
ϕ	ϕ	ϕ	ϕ	1	0	0	0	1	0	0
ϕ	ϕ	ϕ	ϕ	ϕ	1	0	0	1	0	1
ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	1	0	1	1	0
ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	1	1	1	1

ϕ → dont care

↳ Jade wale ko higher priority dena hai, so lower wale dont care.

$f_0 = I_4 I_5' I_6' I_7' + I_5 I_6' I_7' + I_6 I_7' + I_7$

$f_0 = I_4 + I_5 + I_6 + I_7$ $\quad (I_6 + I_7)' = I_6' I_7'$

$f_1 = I_2 I_3' I_4' I_5' I_6' I_7' + I_3 I_4' I_5' I_6' I_7' + I_6' I_7 + I_7$

$= I_4' I_5' I_6' I_7' (I_2 + I_3) + I_6 + I_7$

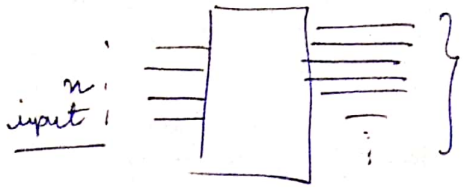
↳ $(I_4' I_5' (I_2 + I_3) + I_6 + I_7)$?

$f_2 = I_1' I_2' I_3' I_4' I_5' I_6' I_7' + I_3' I_4' I_5' I_6' I_7' + I_5 I_6' I_7' + I_7$

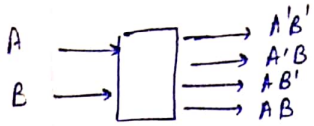
- Minimize

Make combinational circuit using &, 1, not. Ex: 

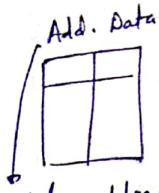
Decoder:



$n \rightarrow 2^n$



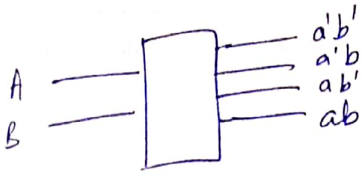
for memory → Memory address decoder. →

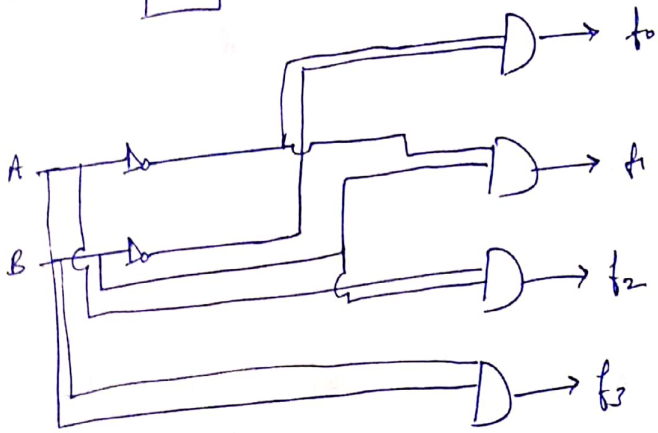


n-bit address, → 2^n loc's possible
→ 2^n data

decoder only.

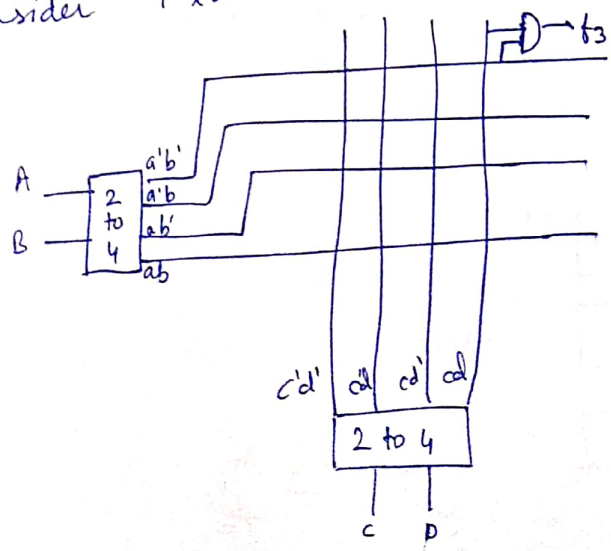
→ 2 to 4 decoder





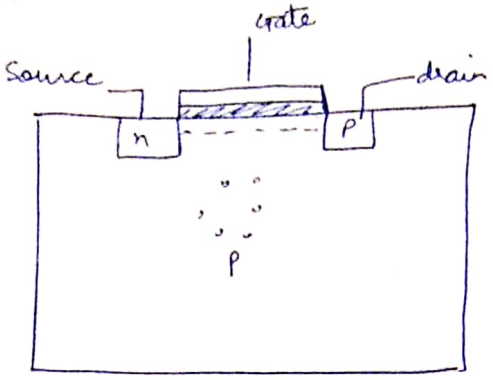
Now, we need VLSI type design (i.e, extendable design)

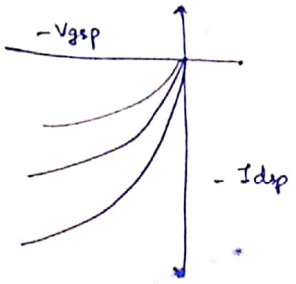
∴ consider 4 to 16 decoder :—

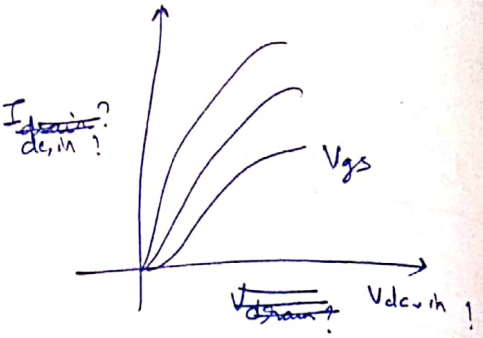


CMOS :

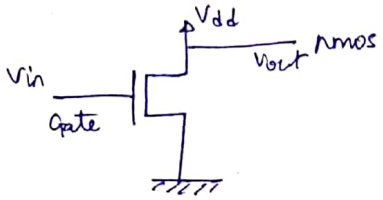
n-mos :







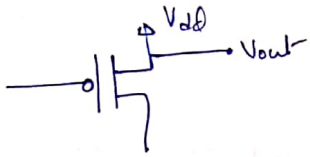
Switch :- nmos, pmos, cmos can be used.



If $V_{in} = 0 \Rightarrow$ n-mos is off $\Rightarrow V_{out}$ = high

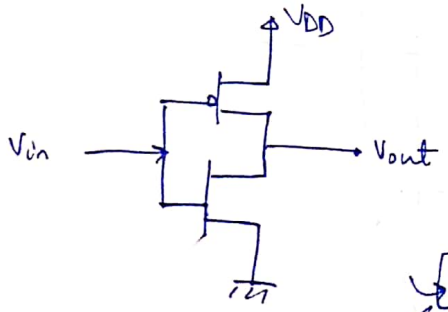
$V_{in} = 1 \Rightarrow$ n-mos on, $\Rightarrow V_{out} = 0$ (ground)

Power consumption is there if V_{DD} se ground connection we get.



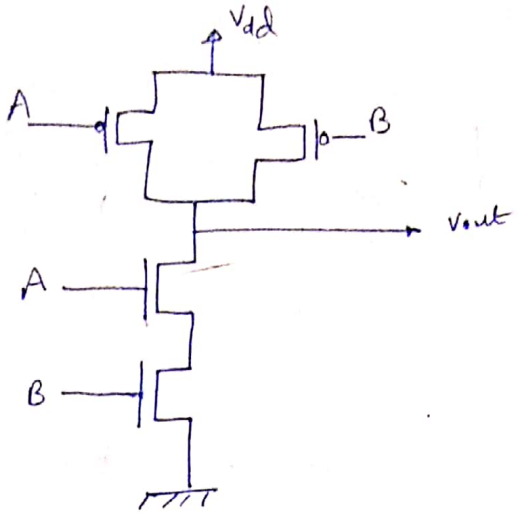
→ $V_{in} = 1$, $V_{out} = 1$.
$V_{in} = 0$, $V_{out} = 0$.

CMOS



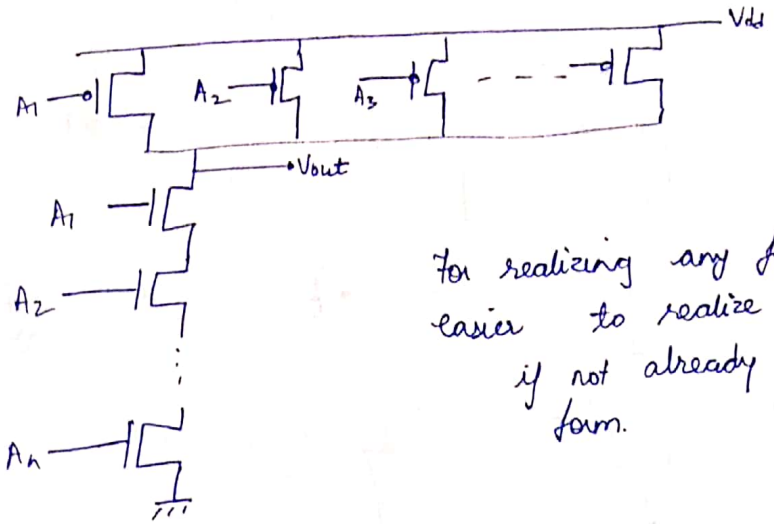
NOT by def.

2-input NAND :



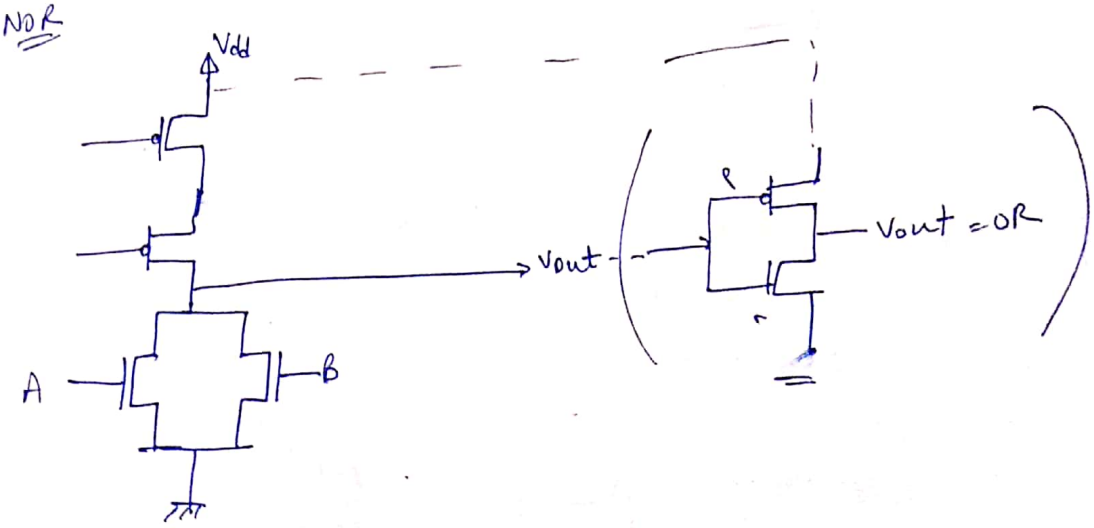
A	B	Vout
0	0	1
0	1	1
1	0	1
1	1	0

NAND

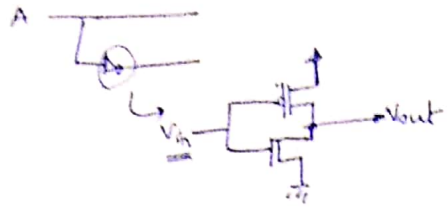


For realizing any fn, it is easier to realize its N-counterpart, if not already in that form.

NOR



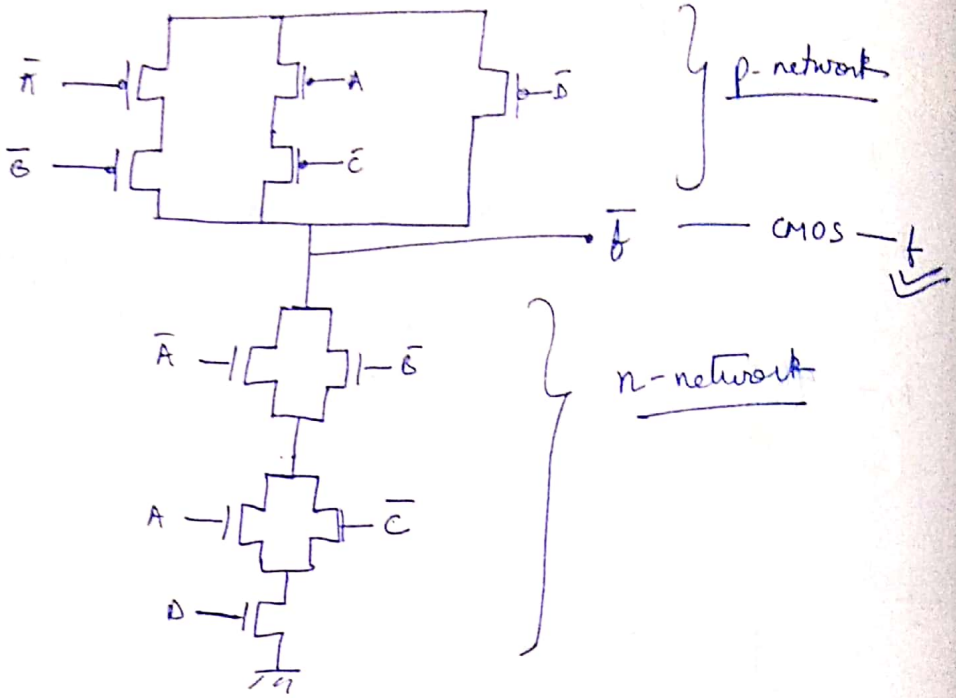
A ka $\bar{A}$ lene ke liye bhi CMOS use karna



→ $f = AB + \bar{A}C + D$

↳ $(\bar{f}) = \overline{(AB + \bar{A}C + D)} = (\bar{A}+\bar{B}).(A+\bar{C}).\bar{D}$

Implement this

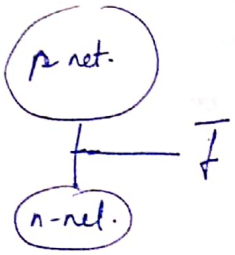


p-network is dual of n-network

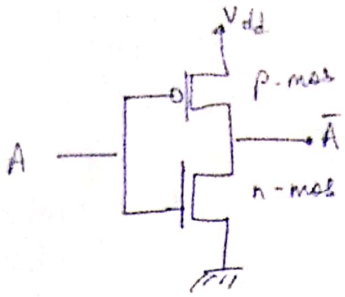
& — OR

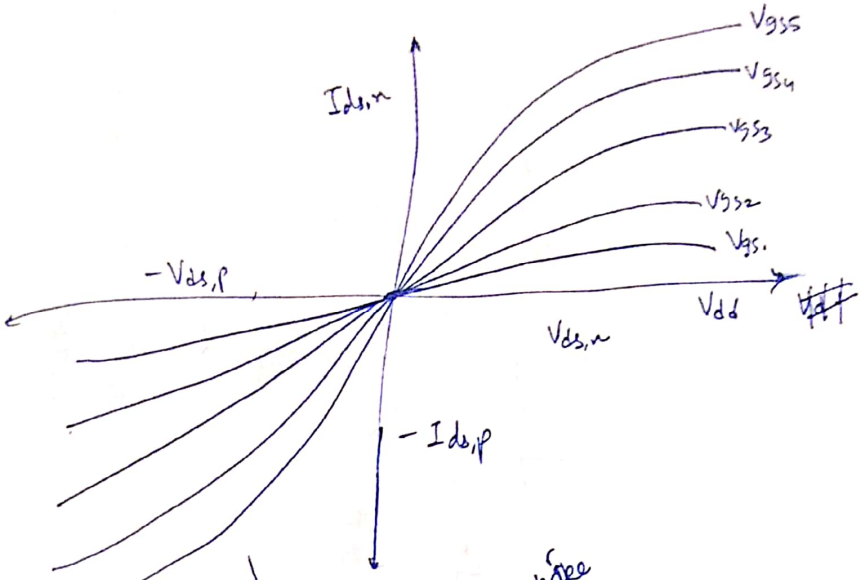
OR — &

Generally (f) →

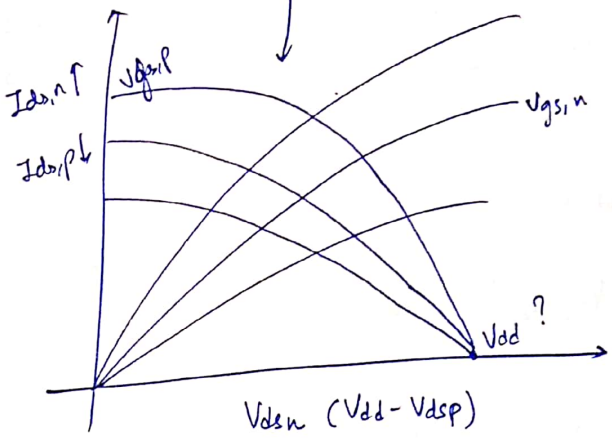


DC characteristics of NOT (CMOS inverter) :-



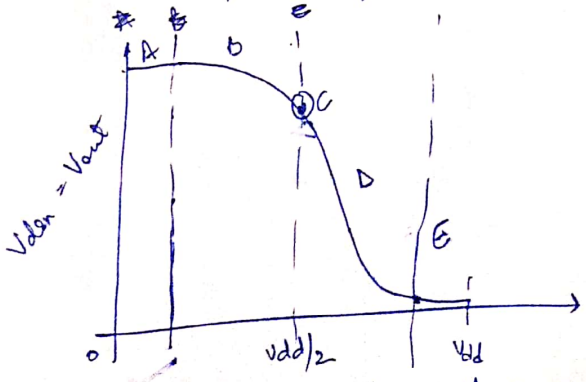


V_{dd} add karke abs. val.

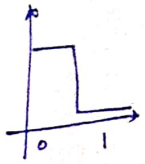


$V_{gs,n} (= V_{gs,p}) = V_{in}$

$I_{ds,n} = I_{ds,p}$



→ Both p mos & n-mos ON.



→ Contd. PTO

Region A: pmos is on (saturated)
nmos is in cutoff (off)

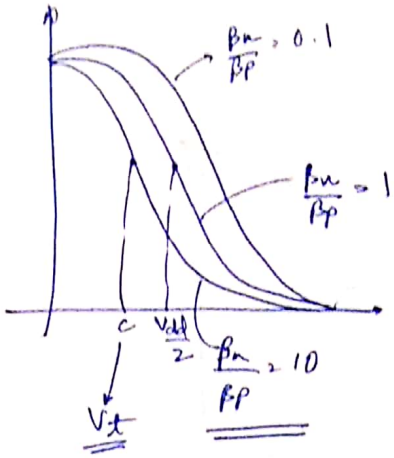
Region B, C, D: pmos on, nmos on, region

Region E: n is on (saturated)
P is in cutoff

} check?

C — transition pt.

$$\Rightarrow \beta_n = \left(\frac{\mu\epsilon}{t}\right)\left(\frac{W}{L}\right)$$



$C = \frac{V_{dd}}{2}$

$0 - V_t \rightarrow 0$
$V_t - V_{dd} \rightarrow 1$

$(0 - 2.5 \rightarrow 1,\ 2.5 - 5 \rightarrow 0)$

Noise Margin: (NM)

$$NM_{L_{low}} = |V_{IL\,Min} - V_{OL\,Min}$$

$$NM_{H_{High}} = |V_{OH\,Max} - V_{IH\,Max}$$

Adder

A 1 0 1 1 0

B 1 0 1 1

1 0 0 0 1

Cout S

$S = A \oplus B \oplus C_{in}$

$C_{out} = AB + BC_{in} + C_{in}A$

$S = A'BC_{in}' + A'B'C_{in} + AB'C_{in}' + ABC$

Reduce / Minimize to get $A \oplus B \oplus C_{in}$

$C_{out} = A'BC_{in} + AB'C_{in} + ABC_{in}' + ABC$

$= AB + BC + CA \cong (A+B).C + AB$ → 1 & reduced

We need &, OR, A, A' (complemented input).

Challenges

- Reducing the hardware ⇒ Reduce area
- Time
- Power

Main prob.

⇒ for $S = (A'BC_{in} + \cdots)$ ⇒ 4 - 3 inp & gates, 1 - 4 inp OR gates

(1 3inp & = 2 2inp &)

$= (A+B+C)(AB + C.(A+B))' + ABC$

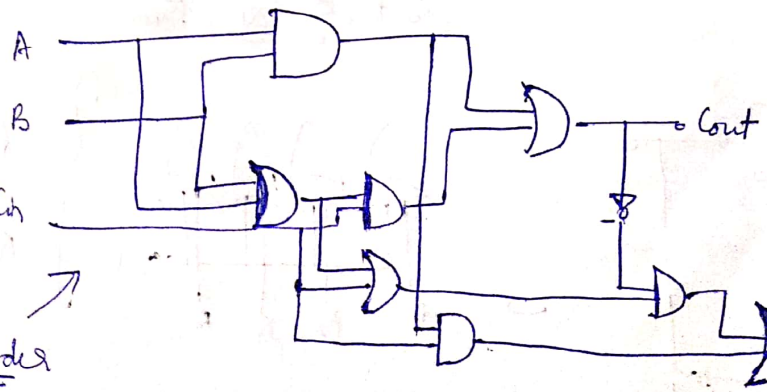
$= (A+B+C)(A'+B')(A'+C')(B'+C') + ABC$

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

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

Trying to represent such that we can use terms from Cout to reduce hardware

Now, Cout ⇒

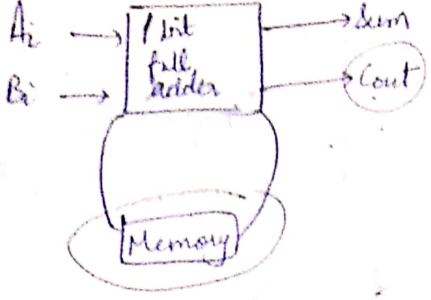


1 bit full adder

⇒ 4 2-inp &

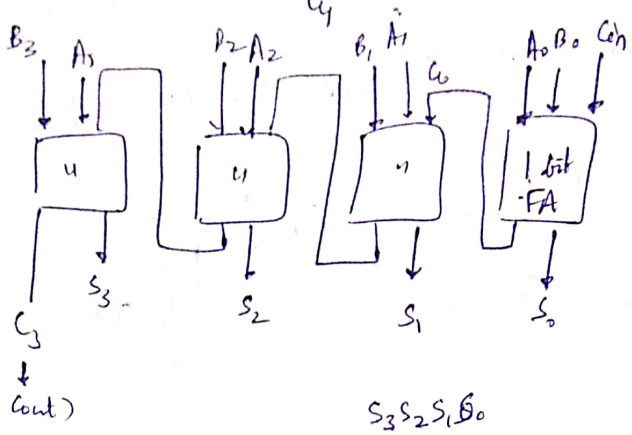
⇒ 4 2-inp OR

⇒ 1 NOT



Serial adder :—

C_3 C_2 C_1 C_0
A_4 A_3 A_2 A_1 A_0
B_4 B_3 B_2 B_1 B_0
S_4 S_3 A_2 S_1 S_0



= 0 (Add)
1 (Sub)

→ 4 bit

$S_3 S_2 S_1 S_0$
Sum

Carry propagates ("ripples")

4 bit → Ripple adder carry adder

Time required ? =

t_{add} → Addition time

t → Time to generate the carry

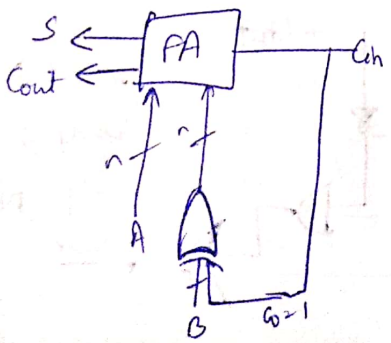
t_{prog} → Carry propagation time

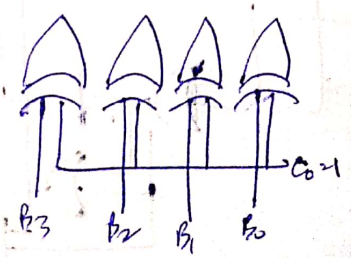
$$t = 4 t_{add} + (4-1) t_{prog}$$

$(n-1)$ → dominates

∴ n-bit ripple carry: $n t_{add} + (n-1) t_{prog}$

Sort of $(2n)$ time





For sub B⊕1 = $\bar{B}$

→ fast adder
→ Carry look ahead adder.

CLA Adder

$C_{out} = AB + BC + CA$

Carry generate $A \cdot B$ (g_i)

Carry propagate $A + B$ (p_i) (Can be XOR also)

$G = A_i B_i + G_{i-1}(A_i + B_i)$

$G = g_i + p_i G_{i-1}$

$C_0 = A_0 B_0 + C_{in}(A_0 + b_0) = g_0 + p_0 C_{in}$

$G = g_1 + p_1 C_0$

$= g_1 + p_1 (g_0 + p_0 C_{in})$

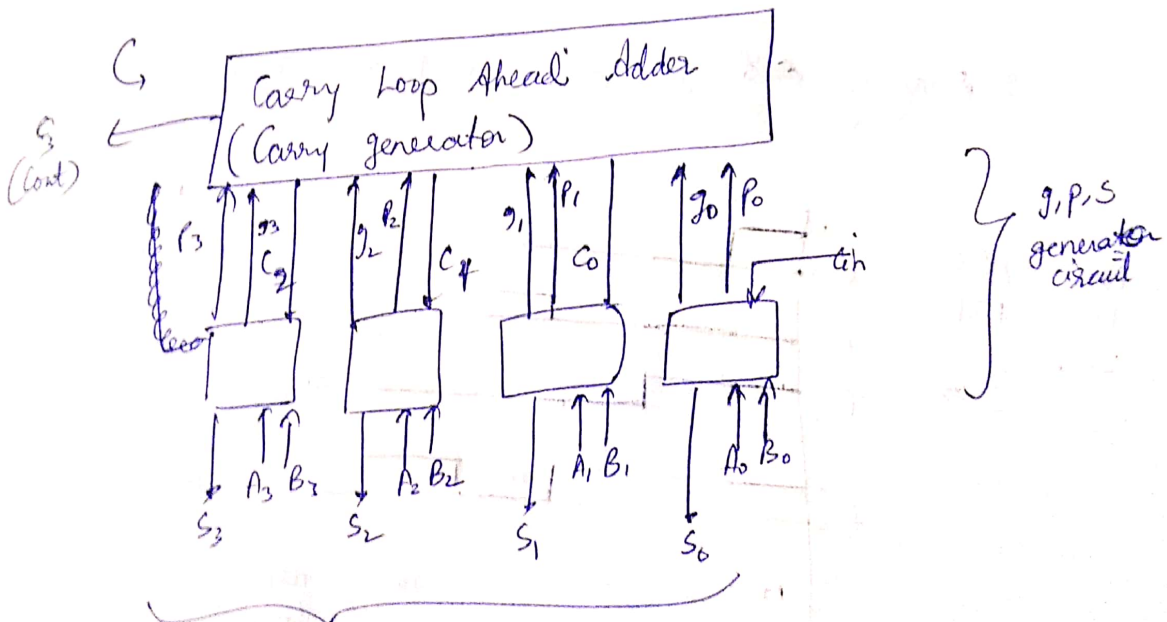
$= g_1 + p_1 g_0 + p_1 p_0 C_{in}$

$C_2 = g_2 + p_2 C_1$

$= g_2 + p_2 g_1 + p_1 p_2 g_0 + p_2 p_1 p_0 C_{in}$

$C_3 = g_3 + p_3 C_2$

$= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 C_{in}$



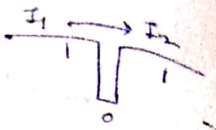
Parallelly all process occurs.
Pure combinational &, or circuit
V. fast.

Hazards in Combinational circuits :-

Change in the input

I_1 $f = 1$

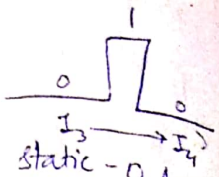
I_2 $f = 1$



Static -1 hazard

I_3 $f = 0$

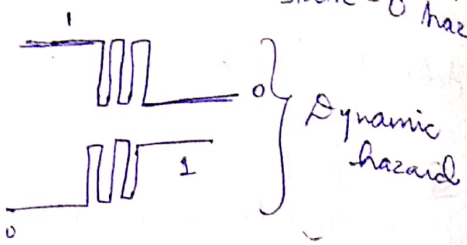
I_4 $f = 0$



Static - 0 hazard

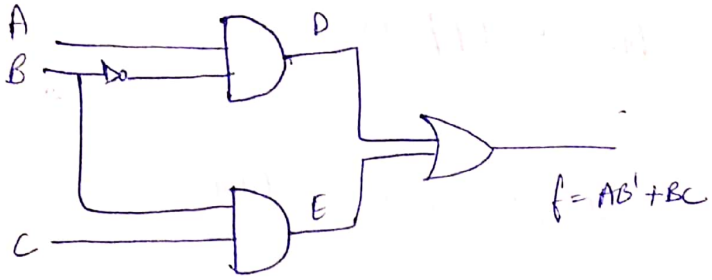
$I_a \rightarrow I_a'$ $f : 1 \rightarrow 0$

$I_b \rightarrow I_b'$ $f : 0 \rightarrow 1$



Dynamic hazard

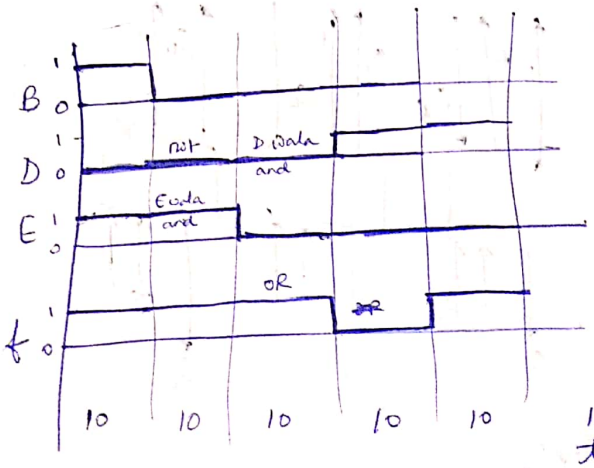
→ Happens due to differences of delay



→ Delay of ~~each~~ each gate is 10 ns. (say)

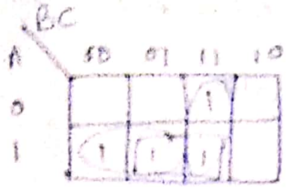
ABC → ABC

111 101



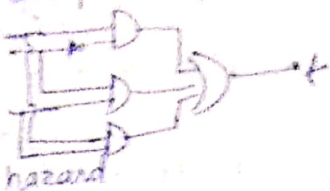
111

K-map → DOUBT Hazardous & not hazardous



$AB + BC + (AC)$

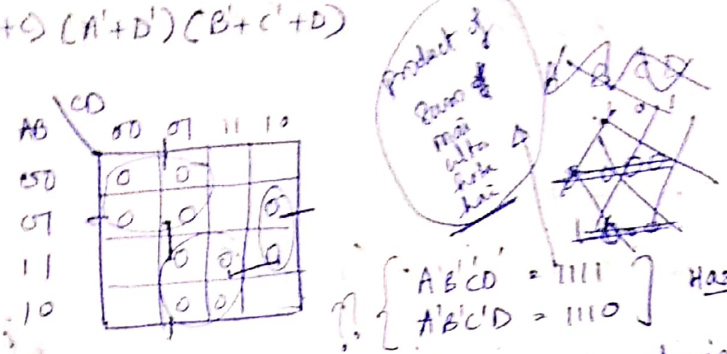
(AB) → delay



Find Adjacent 1's which are not covered by the same term.
For that one, we will have hazard
To unko ek me lelo

Product of sum comparison

$f = (A+C)(A'+D')(B'+C'+D)$



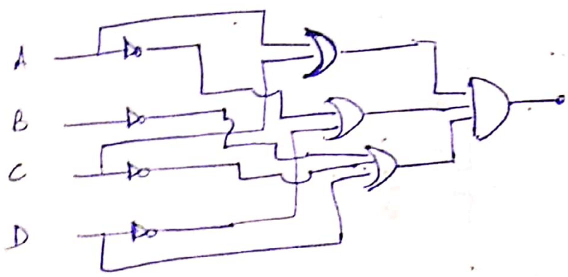
Product of sum & mai ulta hota hai

$A'B'CD' = 1111$
$A'B'C'D = 1110$
Hazard

Kuch terms add karne padegenge to eliminate hazard

$f = (A+C)(A'+D')(B'+C'+D)(C+D')(A+B'+D)(A'+B'+C')$

Ye add kara to avoid hazards



$(A+C)(A'+D')(B'+C'+D)$

Realizing combinational circuits with NAND/NOR logic

1) Two level ckt 2) Multi level ckt

$f = A + BC' + B'CD$ → SOP - NAND-OR ckt

$f = (f')' = ((A + BC' + B'CD)')'$

$= ((A')' \cdot (BC')' \cdot (B'CD)')'$

↳ NAND - NAND

$f = (A' \cdot (B' + C) \cdot (B + C' + D'))'$ → OR - NAND

$f = (A + BC' + B'CD)$

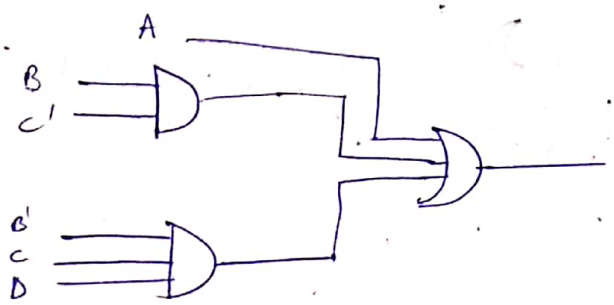
$= (A')' + ((BC')')' + ((B'CD)')'$

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

↳ NOR - OR

How to →

$f = A + BC' + B'CD$



SOP

→ Replace all gates by NAND gates.

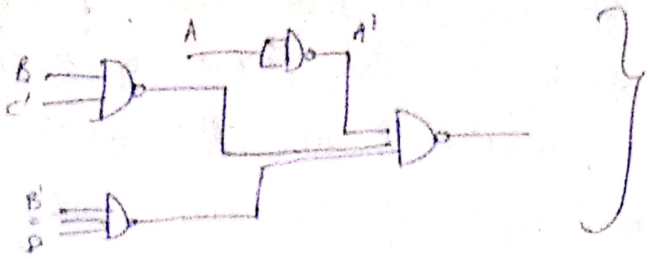
→ If any single literal exists at the input of the last OR, then complement the literal.

??
(Kya tha)

$F_{SOP} = l_1 + l_2 + \dots l_m + P_1 + P_2 + \dots P_n$

$l_1, l_2, \dots l_m$ → single literals

$P_1, P_2, \dots P_n$ → product terms



} Two level realization aisa ho jaye

fan-in & fan-out kya hai

Multi-level relations with NAND-NOR

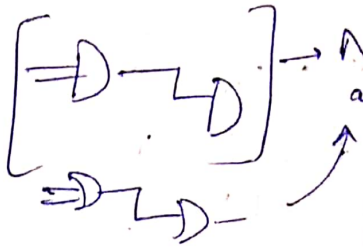
$F = a'[b' + c(d + e') + fg'] + hi'j + k$

→ Realize AND-OR ckts (SOP)

→ Realize the ckt with x-inp AND-OR $(1 \le x \le 4)$

Realization:

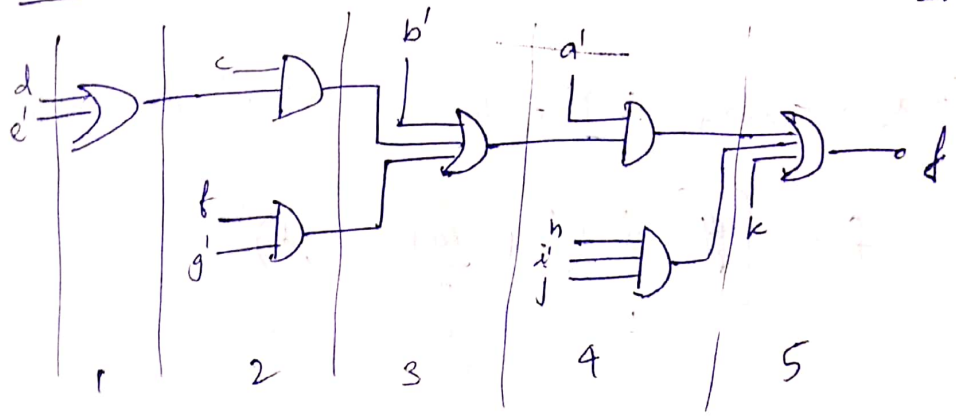
✓ [output of AND (OR) gate cannot be fed to the input of AND (OR) gate.



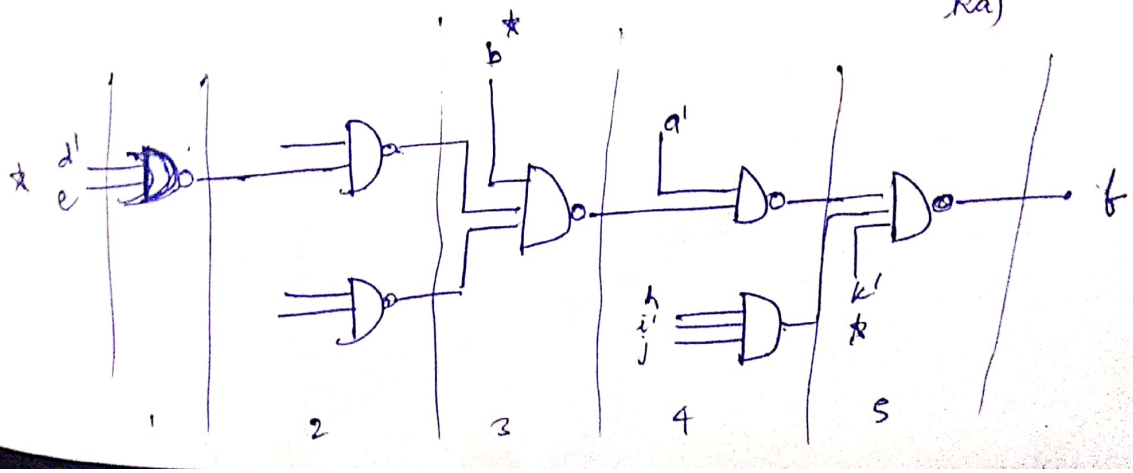
AO ya OA type circuit.

→ OA type?

1)

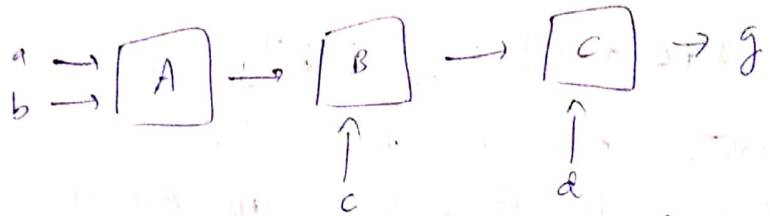


2) Replace gates by NAND gate, & wo complemented input dena OR pe (individual literals ka)

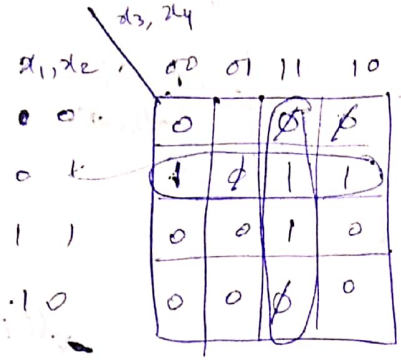


→ After replacing all gates by NAND gates, if any single literal exists at the input of the odd level gates (OR), complement all the single literals. Don't change the input of the even level gates.

Q $g(x_1, x_2, x_3, x_4) = \sum(4, 6, 7, 15) + \sum_{\phi}(2, 3, 5, 11)$



Find the relⁿ b/n x_i & a, b, c, d & determine A, B, C



4 ϕ → $2^4 = 16$ possible fns

$(x_1'x_2 + x_3x_4)$ ⇒ Min fn → isme is form may nahi aa sakta

fog $g = f_3(f_2(f_1(a,b), c), d)$

A, B, C