A chip is a system completely realised on a piece of Si.
(On a Si disc of dia. ≈ 18 inches, 'chips' chopped off').

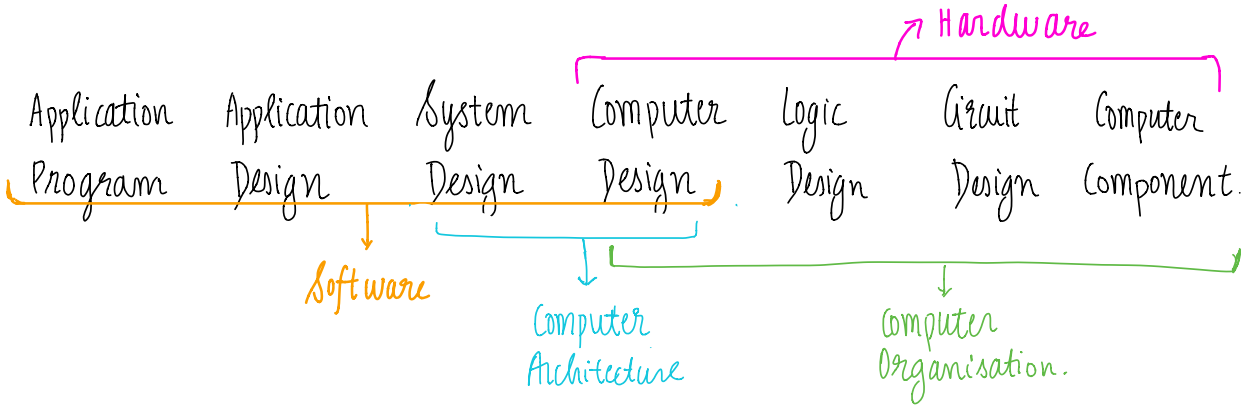
DC
Abhay
#NoPlag

. **Computer Architecture:**

A functional description of requirements and design for various parts of a computer.

. **Computer Organisation:**

How operational attributes, features are linked to realise computer architecture.

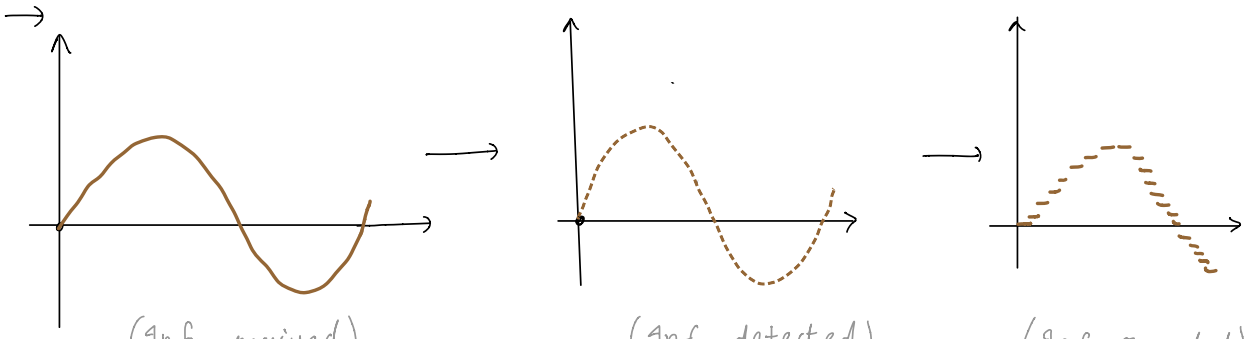


SIGNALS:

Any variation in physical quantity wrt time.

. **Analog Signals:**

The diff. b/w successive values of time and amplitude is not discernable.



(Info received)
ANALOG

(Info detected)
DISCRETE TIME,
ANALOG DATA

(Info recorded)
DISCRETE TIME,
DISCRETE DATA

→ All observed signals are discrete time signals.

. Digital Signal:

If difference b/w successive values of time and amplitude of discrete signal is an integral multiple of a least count.
(values are rounded off to nearest multiple of least count).

→ Accuracy $\propto$ no. of amplitude levels
$\propto$ sampling time.

. Digital v/s Analog:

→ Flexibility: (accuracy of digital signals can be varied).
→ Upgradability: (digital req. partial redesign, analog req. complete redesign)
→ Delay: (response time of analog $\ll$ digital).

. BINARY NO. SYSTEM:

→ In hexadecimal system, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F are digits.

. Conversion:

→ To convert decimal → binary,

$(5)_{10}$ (remainder)

$\frac{5}{2} \longrightarrow 1$

$\frac{2}{2} \longrightarrow 0$

1

this way.

$\therefore \quad 101$

integer

0.625

$0.625 \times 2 \longrightarrow 1$

$0.25 \times 2 \longrightarrow 0$

$0.5 \times 2 \longrightarrow 1$

0

this way.

$\therefore 0.101$

=X=

→ Analog signal Processing → ADC → Digital signal Processing → DAC → Analog Signal Processing.

=X=

→ 4 bits : nibble, 8 bits : byte, 2^{10} : Kilo, 2^{20} : Mega, 2^{30} : Giga...

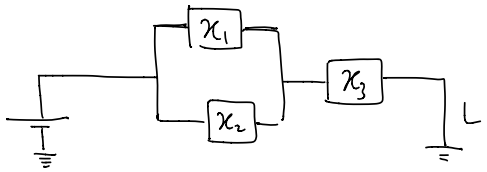
CTM

→ A byte can accomodate one keyboard character.

. LOGIC FUNCTIONS:

'yes' or 'No'.

→ Union, Intersection, complement : logic operations.



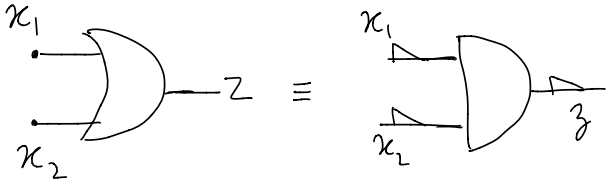
$$L = (x_1 + x_2) \cdot x_3$$

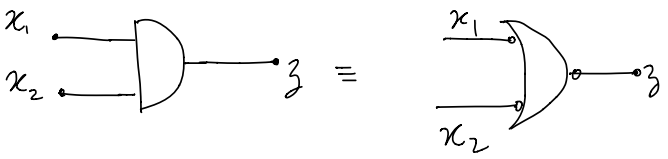
. Precedence of Operations:

→ Order ( ), NOT, AND, OR {BNAO}.

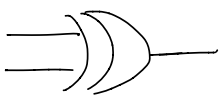
Basic logic gates. → NAND, NOR are universal

→ Negative Logic symbol:





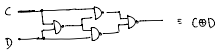
→ XOR:



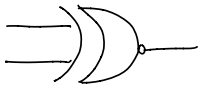
XOR

$$Z = x_1 \oplus x_2 \equiv x_1\bar{x}_2 + \bar{x}_1 x_2$$

CTM



→ XNOR:



$$Z = \overline{x_1 \oplus x_2}$$
$$= x_1 x_2 + \bar{x}_1 \bar{x}_2$$

CTM

• Logical Analysis:

Consists of Truth Table, Logical function and Timing diagram.

$Z = (x_1 + x_2) \cdot x_3$ etc.

• BOOLEAN ALGEBRA:

→ Mathematical methods that simplify circuits rely primarily on Boolean Algebra.

• Huntington's Postulates:

i) Closure wrt + & •

ii) $x + 0 = x$, $x \cdot 1 = x$.

iii) $x + y = y + x$, $x \cdot y = y \cdot x$.

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

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

v) $x + x' = 1$, $x \cdot x' = 0$.

vi) There exist at least 2 elements $x, y \in B$, s.t. $x \neq y$.

NOTE: → No inverse elements, ∴ no subtraction & division.

. Duality:

Every algebraic expression deducible from Boolean Algebra is valid if operators and identity elements are interchanged (see postulates written in pairs).

. Basic Theorems:

→ $x + x = x$, $x \cdot x = x$ ⟶ [Proof: $(x+x).1$, $(x+x).(x+x')$, $x.(x+x')$ → Identity element] [Proof: $x.x$, $x.x + x.x'$, $x.(x+x')$]

→ $x + 1 = 1$, $x \cdot 1 = x$ ⟶ [$(x+1).(x+x')$, $x + 1.x'$, $x + x' = 1$] [$x.1 + x.x'$, $x.(1+x')$, x]

→ $(x')' = x$ ⟶ [$x + x' = 1$ & $x.x' = 0$ ∴ $(x')' = x$]

→ $(x+y)+z = x+(y+z)$, $x(yz) = (xy)z$.

→ $(x+y)' = x'y'$ [To prove, show that: $(x+y).x'y' = 0$ & $(x+y) + x'y' = 1$; $x'y'.x + x'y'.y$; $0+0$; $(x+y+x').(x+y+y')$; 1.1] $(xy)' = x' + y'$.

→ $x + xy = x$. [$x(1+y)$] , $x.(x+y) = x$. [$x.x + x.y$] or dual.

= X =

→ A Boolean function corresponds to a circuit.
∴ Simplification of the expression ⟶ simplification of circuit.

. Each variable in each term constitutes an input, and is called a literal.

. Consensus Theorem:

$yz(x+\bar{x})$

→ $xy + x'z + yz = xy + x'z$ — CTM

→ $(x+y).(x'+z).(y+z) = (x+y).(x'+z)$ — use duality.

→ $A \oplus 0 = A, \quad A \oplus 1 = \bar{A}$ — CTM

. Generalised Form of De Morgan's Theorem:

→ $(A+B+C\ ---\ )' = A'.B'.C'\ ---$

→ $(ABCD\ ---\ )' = A'+B'+C'\ ---$

i.e; the complement of a func. is obtained by taking it's dual and complementing each literal.

. LOGIC SYNTHESIS:

→ Implementing a given function using AND, OR, NOT
or NAND only
or NOR only.

. Normal Term:

Product/sum term in which no literal appears more than once.

- **Minterm:** For a function of 'n' variables, minterm is a product term of 'n' literals. (denoted by $m_0, m_1, \ldots$)

- **Maxterm:** For a function of 'n' variables, maxterm is a sum term with 'n' literals. (denoted by $M_1, M_2, \ldots$)

- **Canonical Sum of Product (CSOP):**
  Every product term is a minterm.

- **Sum of Products:**
  Set of AND terms connected by sum

- **Canonical Product of sum (CPOS):**
  Every sum term is a maxterm.

- **Product of Sum:**
  Set of OR terms connected by product.

∴ Given a truth table, we can form a func.

$$F = \underset{\text{(CSOP)}}{\Sigma\, m(i,j,k\ldots)} = \underset{\text{(CPOS)}}{\Pi\, M(i,j,\ldots)}.$$

→ Decimal equivalence for numbering is followed:

eg;

	x	y	z
0 :	0	0	0
1 :	0	0	1
2 :	0	1	0

....

✱ To convert $f = a+d$ to CPOS.

$$= a+d+b\cdot\bar{b}+c\cdot\bar{c}$$

$$= a+b\cdot\bar{b}+d+c\cdot\bar{c}$$

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

$$= \left[(a+b).(a+\bar{b})+d+c\right].\left[(a+b)(a+\bar{b})+d+\bar{c}\right]$$

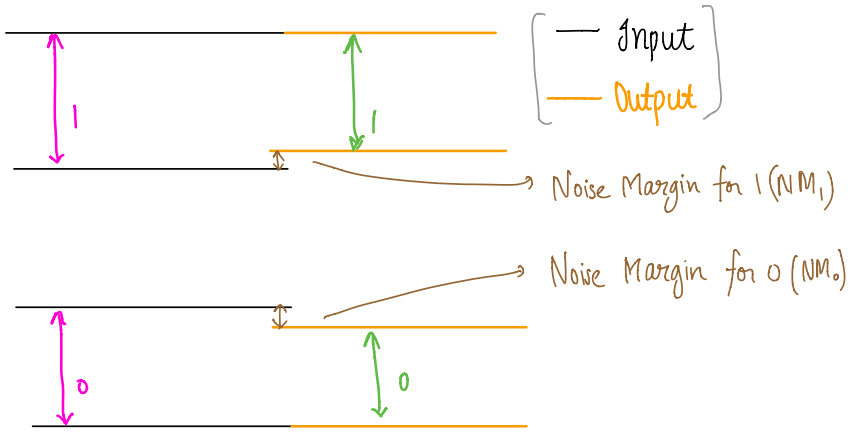
$$= (d+c+a+b).(d+c+a+\bar{b})..(a+b+d+\bar{c})(a+\bar{b}+\bar{c}+d).$$

• Logic Circuit:

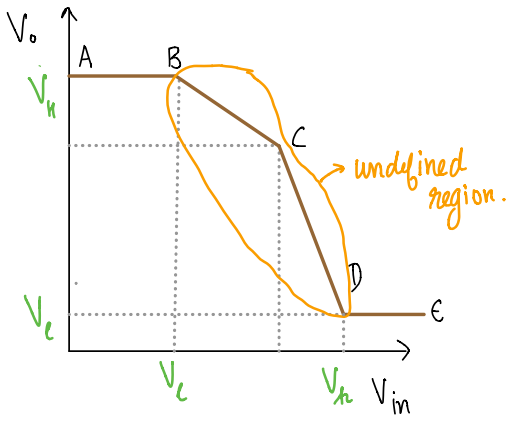
Any circuit in which the signals are constrained to have some no. of discrete values is called logic circuit.

. If only 2 values: binary logic.

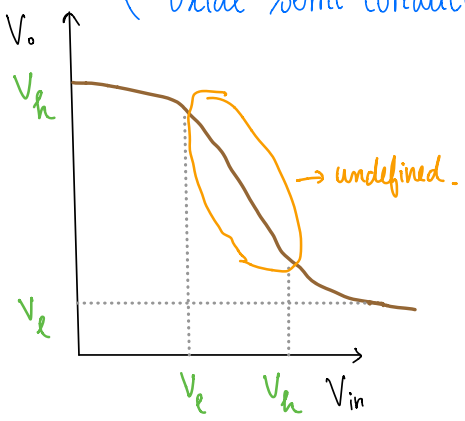
. Binary levels:



. TTL: (Transistor-Transistor logic)



. CMOS: (Complementary Metal Oxide Semi-conductor)



→ $r^2 r^1 r^0 . r^{-1} r^{-2}$

radix point

. Half Adder:

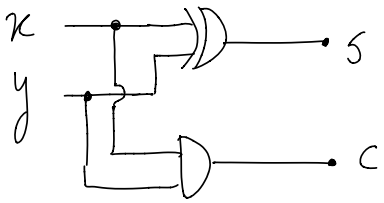
x	y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

CTM

$S = x \oplus y$

$C = xy$.

∴



• Full Adder:

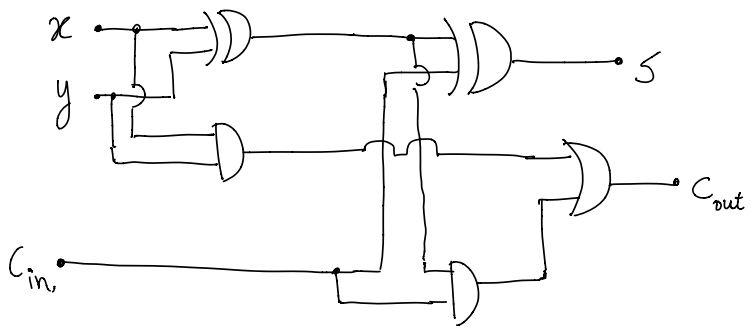
C_{in}	x	y	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

CTM

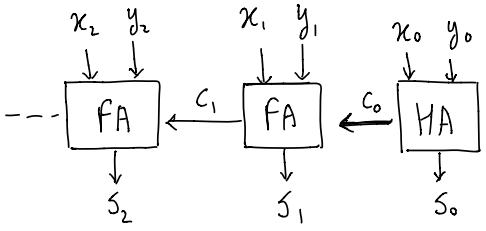
$$S = C_{in} \oplus (x \oplus y).$$

$$C = x.y + C_{in}.(x \oplus y).$$

$\therefore$



• n-Bit Full Adder:

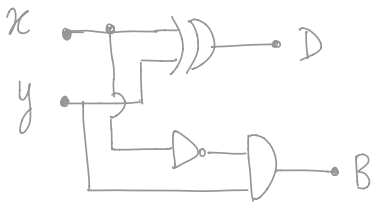


. Half-Subtractor:

x	y	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

CTM

$$D = x \oplus y$$
$$B = \bar{x} . y$$



. Full Subtractor:

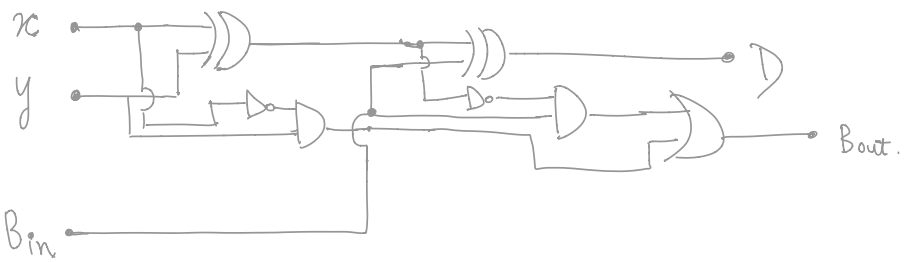
B_{in}	x	y	D	B_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	1

CTM

$$D = B_{in} \oplus (x \oplus y)$$

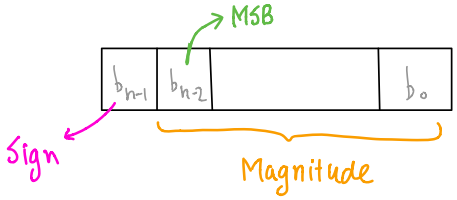
CTM

$$B_{out} = (\overline{x \oplus y}) . B_{in} + \bar{x} . y$$



. SIGNED NUMBERS:

i) Sign & Magnitude Representation:



. Easy to understand, but diff. for hardware realisation.

→ Complements are used to simplify operations, leading to less expensive circuits.

ii) (r-1)'s complement representation:

→ (r-1)'s complement of a no. is taken by complementing every bit. (for eg; in decimal, 9's complement of 2 is 7).

→ It is actually subtracting the no. from (r-1)(r-1)---- (e.g; 999-048 = 951).

e.g; 176 - 048
176 + 951 - 999
1127 - 1000 + 1
128

e.g; 176 - 1348
= - (1348 - 176)
= - (1348 + 9823 - 9999)
= - (11172 - 10000)
= - 1172.

→ In case of binary, the complement is obtained by subtracting from 1111.....

iii) r's Complement:

→ r's complement is obtained by adding 1 to (r-1)'s complement.

e.g; 176 - 048

176 + 952 - 1000.

= 1128 - 1000

= 128.

e.g. 048 - 176

- (176 - 048)

= - (176 + 952 - 1000).

= - (1128 - 1000).

→ To find r's complement for binary nos.;
Start from LSB Keep all zeroes and first 1, after which, take complement of all digits.

Shortcut:

→ To find r's complement of any no. of radix 'r',
Keep all least significant zeroes. Subtract the first non-zero least significant bit from r and rest all from (r-1).

→ Ignore radix point while finding complement.

→ Algorithm for Subtraction Using r's Complement:

If M - N. M, N are unsigned numbers.

→ $M + (r^n - N) - r^n$

$= M + N' - r^n$

$M \geq N$

- sum produces carry, discard it.

$M < N$

- no carry.
- → Take complement of result of $(M+N')$ & put −ve.

→ Algorithm for Subtraction Using $(r-1)$'s Complement:

- $M - N$

$= M + ((r^n - 1) - N) - (r^n - 1)$.

$= M + N' - (r^n - 1)$.

→ $M + N'$ produces carry,
- Do end around carry.

→ No carry.
Take $(r-1)$'s complement of $M+N'$.

$= X =$

→ Signed r's and $(r-1)$'s complement is obtained by complementing the no., including the signed bit.

→ In case of signed binary nos., if there is no carry, the no. is obtained in complement form.

. BINARY CODES:

. Discrete, distinct among group info. can be represented as binary code. Computer bits usually represent coded info., not plain binary digits.

. Binary codes consist of bits, each combination representing an element from the set that is coded.

. Decimal Codes:

→ If 4 bits, 16 diff. combinations. 10 needed for decimal. Different arrangements yield diff. codes.

. Straight binary assignment = BCD. (used for decimals, encoded in form of bits)

→ BCD, 2421 are **weighted codes.**

→ 2421, excess 3, 8,4,-2,-1 code are self complementing.

→ Excess 3 is obtained by adding 3 to BCD.

. Diff. symbols give same value to 2421, hence only those chosen which are self complementary.

- **Binary → Gray:**

→ Add 0 to left of MSB.

→ XOR concecutive digits to obtain gray code (MSB first).

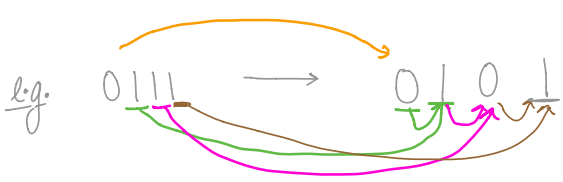
e.g. 0110 → 00110

0 1 0 1 → Gray code.

- **Gray → Binary:**

→ MSBs are identical.

→ To obtain a given index, consider XOR of gray code at that index and previous binary bit.



- **KARNAUGH MAP:**

→ K-Map is a diagram made of squares, each square representing one minterm | maxterm of the function that is to be minimised.

→ 2 minterms | maxterms, OR/AND → 1 variable removed.

4 → 2

8 → 3

⋮ ⋮

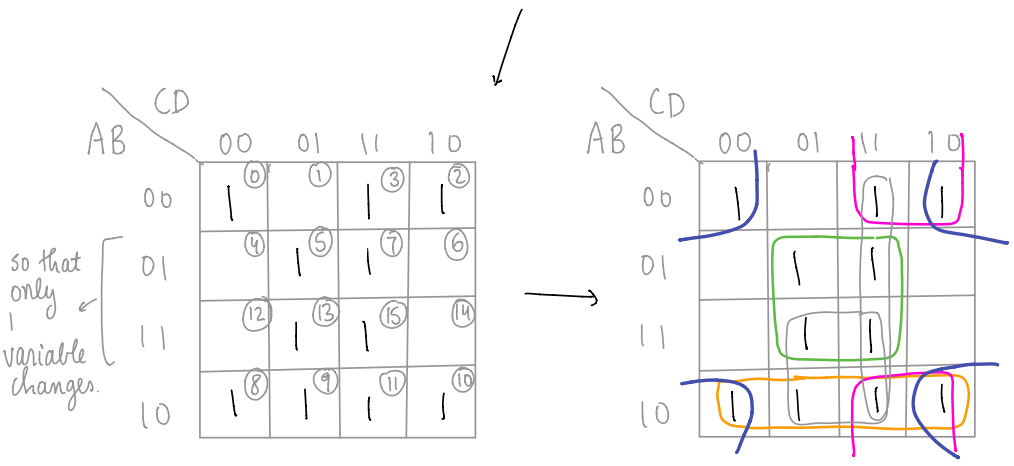
→ squares must be combined horizontally or vertically or squared. [including cyclic pairs], in powers of 2 only (2, 4, 8, 16...). Such combinations called adjacent squares.

→ **Prime Implicant:**

Term obtained by combining maximum possible adjacent squares.

. If the prime implicant is the only one covering 1/more minterms, it's called essential prime implicant.

e.g.; $f(A,B,C,D) = \sum m(0,2,3,5,7,8,9,10,11,13,15)$.



Hint: just check variables with same value across all squares. → $\bar{B}\bar{D} + BD + A\bar{B} + \bar{B}C$

$\therefore f = \bar{B}\bar{D} + BD + A\bar{B} + \bar{B}C$.

→ For POS:

i) By Minterms:

. Complement the func. by replacing all 0's with 1's & vice-versa
. Form SOP.
. Take complement. ($\overline{SOP}$ = POS).

ii) By Maxterms:

. Fill 0's in maxterms given for func.
. Form P.I & EPI as usual.
. Write func. in POS form

NOTE:
[A=0, B=1 ⟶ $(A+\overline{B})$.]
↳ when written in POS.

. Don't Care:

→ For certain func., outputs are not specified for certain inputs. We DON'T CARE what the output is for those inputs. and assume it to be 0 or 1 based on our convenience. (so as to simplify as many terms as possible in K Maps).

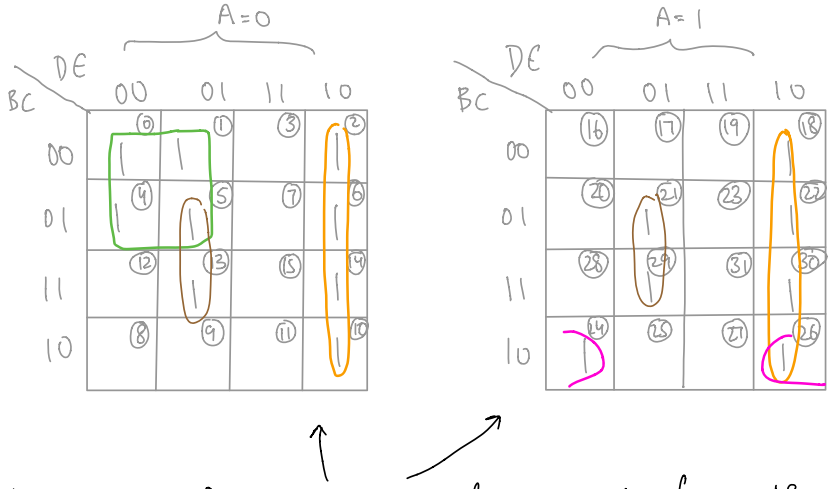
. Same don't care condition can be assumed 1 in SOP & 0 in POS. or vice-versa.

• 5 Variable K-Map:

→ Shannon's Theorem:

$$f(x_1, x_2, \ldots x_n) = \bar{x}_1 \cdot f_1(x_2, x_3 \ldots x_n) + x_1 f_1^*(x_2, \ldots x_n).$$

e.g.:



. Find overlap of 2 squares, eliminate A from those terms.

. Write other terms as usual.

$$f = D\bar{E} + \bar{A}\bar{B}\bar{D} + C\bar{D}E + AB\bar{C}\bar{E}.$$