DATE : / /
PAGE NO. :

Digital Logic

Five Modules : 4-6 Marks | Easy Subject.

Module 1, → Boolean Algebra and Minimization.

→ There are two types of systems : Analogue i.e. Continuous
Digital i.e. Discrete.

→ We (in CS) don't care about how Gates are actually made, we only care about their behaviour.

→ Boolean Algebra : Given by George boole in 1843
↳ Two valued System (0 & 1)
↳ Inspired from prop. logic.
↳ Given by Socrates, Aristotle.

→ After almost 100 years of boolean alg., shanon, used it to model the Digital Systems.

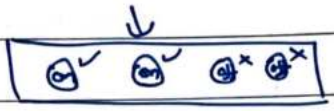
→ Every data that can be stored in comp. can be repr. as 0 & 1. and this is how actually stored in RAM. RAM contain capacitors which when charged repr. 1, else repr. 0.

→ Physically we assign a range of charge like (4V to 5V) to high (i.e. 1) else low (i.e. 0). at 0V to 1V. and transition occurs b/w 1V to 4V.

GO CLASSES

DATE : / /
PAGE NO. :

A=12 $\xrightarrow{\text{Compiler}}$ A= 1100 → In memory
↓
Program



Series of transistors repr. 12

→ Boolean Algebra ⇌ Switching Algebra

→ Collection of n bits have 2^n possible states.

→ Binary switch has two states

→ We use boolean variables, like X or Y, to repr. i/p or o/p of a switching circuit. Each var. can either take 0 or 1.

If physical entities input signal is low then bool var. is 0, and if it's high then it's 1. These symbols 0 & 1 do not have any numeric value, they are just symbols.

→ Binary v/s. Boolean In boolean alg. 0 & 1 don't have a numeric value; they just repr. diff states of circuit. Like in gates 0 means low voltage & 1 repr. range of high voltages. In switches 0 means open switch, and 1 means close circuit.

Binary is used to repr. a numerical value. We can add, divide, etc). It's a no. system.

GO CLASSES

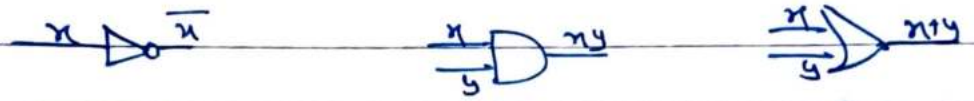
DATE : / /

PAGE NO. :

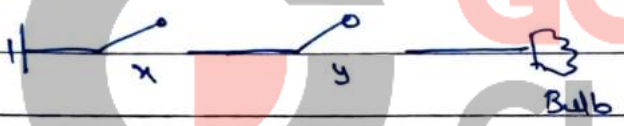
→ Bit ⇌ Binary Digits i.e 0 & 1

→ Boolean Algebra ≡ Propositional Logic.

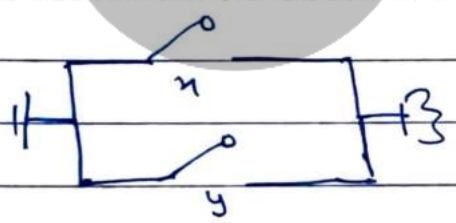
↳ Basic operations are (AND, OR, Not)
+ · −



Not Gate AND Gate OR Gate.



→ AND using switches



→ OR using switches

→ Boolean expression contains 0, 1, $\underbrace{x, y, \ldots}_{\text{boolean vars}}$ +, ·, $\bar{x}$ ...

↳ ① 0 & 1

↳ ② Boolean variables

↳ ③ If X & Y are bool. exp then
$\bar{X}$, $\bar{Y}$, X+Y, X·Y, (X) ...

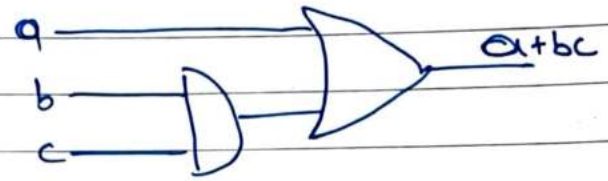
→ Each boolean expression corresponds directly to a circuit of logic gates.

GO CLASSES

Inverter $\rightleftharpoons$ Not Gate

DATE : / /
PAGE NO. :

Ex: $a+bc$ correspond to

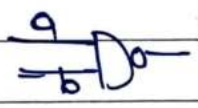


→ Truth Table (also known as table of combinations) specifies the values of a boolean expression for every possible combination of values of variables in the expression.

→ Precedence NOT > AND > OR.

So $a+bc$ means $a+(bc)$ & not $(a+b)\cdot c$

→ There are four more standard boolean operations:

a) NAND ($\uparrow$)  $a \uparrow b = \overline{a \cdot b}$ (Not of And)

b) NOR ($\downarrow$) $a \downarrow b = \overline{(a+b)}$ (Not of OR)

(NAND & NOR gates are universal gates bcz, they alone are functionally complete)

a	b	$\bar{b}$	ab	$a+b$	$a \uparrow b$	$a \downarrow b$
0	0	1	0	0	1	1
0	1	0	0	1	1	0
1	0	x	0	1	1	0
1	1	x	1	1	0	0

Truth table

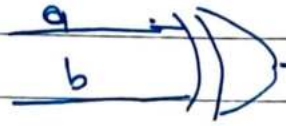
GO CLASSES

DATE : / /

PAGE NO. :

c) Exclusive OR (EXOR) ($\oplus$)

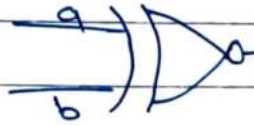
Also called Equivalence Operator



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

$a \oplus b = 1$ iff $a \neq b$

d) EXNOR ($\odot$)



$a \odot b = ab + \bar{a}\bar{b} = \overline{a \oplus b}$

a	b	$a \oplus b$	$a \odot b$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

GO CLASSES

→ When we arrange truth table such that values of all variables in decimal system is sorted in inc. order.

00	00	Not in std. form.
01	10	
10	11	
11 Standard	01	

→ AND & OR gates can be repr. for many inputs as $a \cdot b \cdot c$ & $a+b+c$ respectively. (Not writing truth table)

DATE: / /
PAGE NO.:

⇒ **Literal:** Each appearance of a variable in its original or complemented form in an expression.

Ex: $a\bar{b}c + a\bar{c}b + ab + ca$

This has total 10 literals and 3 variables

⇒ **Expression:** A set of literals (possibly with repeat) combined with logic operations.

Ex: $AB + CD + \bar{A}\bar{B} + \bar{C}B$

⇒ **Equation:** Expression 1 = Expression 2

⇒ **Function of variables:** Equation where LHS is defined by RHS

Ex: $f(a,b,c) = (a+b(c)\,\bar{a}$

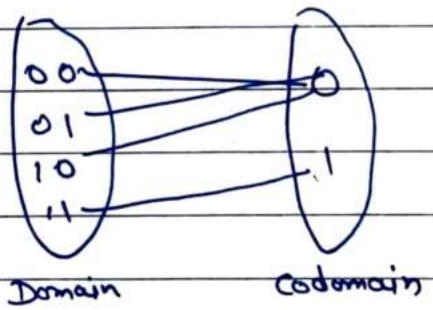
GO CLASSES

Boolean Expression == Boolean Function : Both has domain, codomain & mapping

⇓

So boolean exp. is called boolean func.

Ex: $f(a,b) = ab$



★ **Complement of a boolean function:**

Ex:

a	b	$f = \bar{b}$	$\bar{f} = b$
0	0	1	0
0	1	0	1
1	0	1	0
1	1	0	1

if $f(a,b) = 0$ then $\bar{f}(a,b) = 1$.

$\bar{f}$ is called complement of f.

DATE : / /

PAGE NO. :

* If $exp1 = exp2$ then $(exp1)^d = (exp2)^d$

NOTE $f = \bar{f}$ is not at all (NEVER) possible.
So $f \neq \bar{f}$

* **Boolean Algebra Law** : Same (isomorphic) to that of PL, set, Group theory, Bool. lattice.

→ Let C, A & B be any boolean expression:

→ **Domination | Annulment Law:**

$A + 1 = 1$

$A \cdot 0 = 0$ (Dual of above)

(other in this)

$A \oplus 1 = \bar{A}$

$A \oplus 0 = A$

$A \odot 1 = A$

$A \odot 0 = \bar{A}$

→ **Complement Law:**

$A + \bar{A} = 1$

$A \cdot \bar{A} = 0$

→ **Commutative Law:**

$A + B = B + A$
$A \cdot B = B \cdot A$
} Holds in all op. except implication

→ **De morgan's Law:**

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

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

→ **Identity Law:**

$A + 0 = A$

$A \cdot 1 = A$

→ **Idempotent Law:**

$A + A = A$

$A \cdot A = A$

$A \oplus A = 0$

$A \oplus \bar{A} = 1$

$A \odot A = 1$

$A \odot \bar{A} = 0$

→ **Double Comp:**

$\bar{\bar{A}} = A$

→ **Associative Law:**

$(ab)c = a(bc)$
$(a+b)+c = a+(b+c)$
} Hold in all op. except implication

(Can be proved by taking cases.

So

$(A \uparrow B) \uparrow C \neq A \uparrow (B \uparrow C)$

$(A \downarrow B) \downarrow C \neq A \downarrow (B \downarrow C)$

GO CLASSES

DATE : / /

PAGE NO. :

→ Distributive Law:

$a(b+c) = ab+ac$

$a+(bc) = (a+b)(a+c)$

NOTE AND, OR are not complement of each other as if $f=ab$ then $\bar{f} \neq a+b$

⇒ Complementing Boolean Expression using Demorgan's Law: If bool. exp. only have literals, AND, OR, NOT then demorgan's law can be use to find complement of this boolean expression.

$$f(a, b, \bar{a}, \bar{b}, +, \cdot, 0, 1)$$

$$\bar{f}(\bar{a}, \bar{b}, a, b, \cdot, +, 1, 0)$$

Ex $f = \overline{(x+\bar{y})\cdot\bar{c}}$, $\overline{(x+\bar{y})\cdot\bar{c}}$

$\bar{f} = (x+\bar{y})\cdot\bar{c}$, $(\bar{x}\cdot y)+c$

Simplify $\bar{f}$: $\overline{\overline{A\bar{B}C} + \overline{ACD} + B\bar{C}}$

$= \overline{\overline{A\bar{B}C}} \cdot \overline{\overline{ACD}} \cdot \overline{B\bar{C}}$

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

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

$= A\bar{B}CD$

GO CLASSES

DATE : / /
PAGE NO. :

$f = (\bar{v}w + x)y + \bar{z}$

$\bar{f} = ?$

$\Rightarrow \bar{f} = \overline{(\bar{v}w + x)y + \bar{z}}$

$= [\overline{(\bar{v}w + x)} + \bar{y}] \cdot \bar{\bar{z}}$

$= [(\overline{\bar{v}w} \cdot \bar{x}) + \bar{y}] \cdot z$

$= ([(v + \bar{w}) \cdot \bar{x}] + \bar{y}) \cdot z$

$= (vx + \bar{w}x + \bar{y}) \cdot z$

Ans.

* ⇒ Some more useful (really useful properties)

a). Absorbtion Law:

i) $a + ab = a$. * ✓

ii) $a \cdot (a + b) = a$ (Dual of above eq.)

Most imp (**) iii) $a + \bar{a}b = a + b$. * ✓

b) Consensus Property:

$AB + \bar{A}C + BC = AB + \bar{A}C$ **

↳ Remember as B is with good of A & C is with bad of A & they B & C are doing conspiracy being together, so remove them.

GO CLASSES

DATE : / /

PAGE NO. :

⇒ Boolean algebra laws are given in a pair to show that boolean algebra satisfies duality

i.e. if an identity is true then dual of it will be true

Ex: $a+ba = a$

$a\cdot(b+a) = a$ (Dual of above identity)

⇒ Dual of a function can be given if bool. exp. only has OR, AND, literal, NOT, then:

$$f(a, b, \bar{a}, \bar{b}, \cdot, +, 0, 1)$$
$$\downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow$$
$$f^d(a, b, \bar{a}, \bar{b}, +, \cdot, 1, 0)$$

Ex: $f: a+b+\bar{c}\cdot\bar{d}$

$f^d = a\cdot b\cdot(\bar{c}+\bar{d})$

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

$\bar{f}^d = a+b+cd$

So, we do not complement the literal while finding the dual of any boolean expression.

NOTE: for some function $f = f^d$ is possible to be true but not for all functions.

Such fun. are called self dual functions

Ex: $f = a$, $f^d = a$,

$f = \bar{f}$ is not at all possible

$\exists f \; f = f^d$ & $\forall g \; f \neq f^c$

GO CLASSES

DATE : / /

PAGE NO. :

Qu: If $f^d = \bar{f}$ is possible.?

$\Rightarrow$ Yes possible, for Ex $f = a\bar{b} + b\bar{a}$ i.e $a \oplus b$.

$\bar{f} = (\bar{a}+b)\cdot(a+\bar{b})$

$f^d = (a+\bar{b})\cdot(\bar{a}+b)$

So $\bar{f} = f^d$ ✓

→ If $E = F$, then $\bar{E} = \bar{F}$ ie
if $E = F$ then $E^d = F^d$ is also true.

} Can be used to prove theorem directly.

Some practise:

$\boxed{\alpha + \bar{\alpha}\beta = \alpha + \beta.}$

$\boxed{\alpha + \alpha\beta + \alpha\gamma + \alpha z = \alpha.}$ → Absorbtion Law

a) $\bar{A}(\overline{ABCD})$

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

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

$= \bar{A}$ (Absorbtion Law)

b) $A(\bar{A}+B) + AB$

$= A\bar{A} + AB + A\bar{B}$

$= 0 + A(B+\bar{B})$

$= A.$

c) $ABC + \bar{A}BC + \bar{A}\bar{B}C$

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

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

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

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

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

c) $BCDE + BC(\overline{DE}) + (\overline{BC})DE$

$= BC(DE + \overline{DE}) + (\overline{BC})DE$

$= BC + \overline{BC}\,DE$

$= BC + DE$

d) $\bar{X}YZ + \bar{X}Y\bar{Z} + XZ$

$= Z(\bar{X}Y + X) + \bar{X}Y\bar{Z}$

$= Z(Y+X) + \bar{X}Y\bar{Z}$

$= YZ + XZ + \bar{X}Y\bar{Z}$

$= Y(Z + \bar{Z}\bar{X}) + XZ$

$= YZ + Y\bar{X} + XZ$

$= Y\bar{X} + XZ$

(Consensus Law)

GO CLASSES

⇒ Simplification is imp. bcz. simplified bool. exp. will give simple circuit from gates. (i.e. less gates required).

⇒ Literal count is a measure that tells us size of bool. exp.

Ex: $a\bar{b}c + ab + \bar{c}b + \bar{b}\bar{c} + \bar{a}b\bar{c}$ has 12 literals & 3 variables.

⇒ By simplification/ minimization we mean to reduce #literals.

Que: Prove: $(a+b)\cdot(a+\bar{a}\bar{b})\cdot c + \overline{(\bar{a}\cdot(b+\bar{c}))} + \bar{a}\cdot b + abc = a+b+c$

⇒ Case I: $a=1$

$LHS = (1+b)\cdot(1+\bar{a}\bar{b})\cdot c + \overline{(0\cdot(b+\bar{c}))} + 0\cdot b + 1\cdot bc$

$= 1\cdot 1\cdot c + (\bar{0}) + 0 + 0$

$= 1 + 1 + 0 + 0 = \underline{1}$

$RHS = 1 + b + c = \underline{1}$

Case II: $a=0$

$(0+b)\cdot(0+1\bar{b})\cdot c + \overline{(1\cdot(b+\bar{c}))} + 1\cdot b + 0\cdot bc$

$LHS = b\cdot\bar{b}c + \overline{b+\bar{c}} + 1$

$= 0 + \overline{b+\bar{c}} + b$

$= (\bar{b}\cdot\bar{\bar{c}}) + b$

$= b + \bar{b}c$

$= \underline{b+c}$

$RHS = 0 + b + c = \underline{b+c}$

∴ Hence Proved.

⇒ Can also be done using boolean al. simplification.

GO CLASSES

DATE : / /

PAGE NO. :

NOTE Cancellation laws do not hold in boolean algebra.

Ex: $\cancel{0}\cdot 1 = \cancel{0}\cdot 0$ } Can't cancel

$\Rightarrow 1 \neq 0$

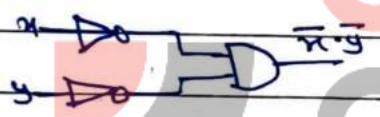
Somethings related to XOR:

$\overline{a \oplus b} = a \odot b = \bar{a} \oplus b = a \oplus \bar{b} = \bar{a} \odot \bar{b}$

$a \oplus b = \overline{a \odot b} = \bar{a} \oplus \bar{b} = \bar{a} \odot b = a \odot \bar{b}$

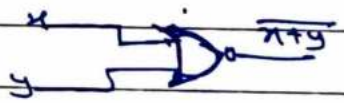
Here 1 ij diff. Here also 1 ij diff.

Que: Which of the following circuits are equivalent?

a) 

b)

c)

d) 

$\Rightarrow$ a = d and b ≡ c

→ # boolean functions possible.

a). for 1 var.

x	f1	f2	f3	f4
0	0	0	1	1
1	0	1	0	1

So 4 functions are possible

b) for 'n' variable

(2^{2^n}) functions possible.

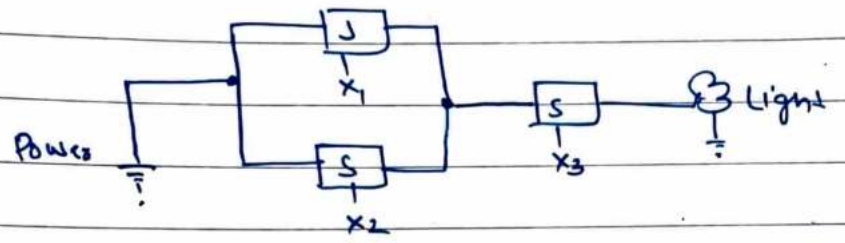
c) for 0 variables

$\Rightarrow$ True i.e. always 0 or 1.

DATE : / /

PAGE NO. :

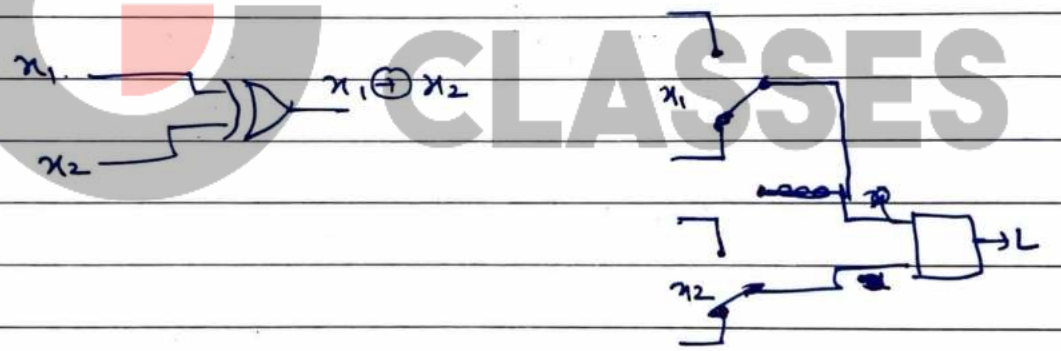
Qu: What will be logical behaviour of below circuit?



=> $(x_1 + x_2)\, x_3$.

Qu: Design circuit for staircase light control. using 2-way switch. (Both switch off/on, then off else on)

=>



★ Standard forms of boolean expressions:

→ There are many ways to express a boolean expression, it's useful to have a standard or canonical way. It can be derived by truth table. (It might'nt be simplest form).

→ Over 2 var, we can only have 16 (2^{2^2}) function, but some function can be written in various ways if some std. repr. is not followed.

DATE : / /

PAGE NO. :

There are two principle standard forms:

a) Sum-of-products (SOP) b) Products-of-sums (POS)

Let's understand them:

→ **Product term:** AND of literals, or, a single literal.

Ex: AB, $\bar{A}$, $B\bar{C}$, are product term, but $a+b$, $\overline{ab}$ are not.

→ **Sum term:** OR of literals, or, a single literal

Ex: $a+b$, a, $\bar{b}$, $\bar{a}+b$ are sum term, but ab, $\overline{ab}$, $\overline{a+b}$ are not.

→ **SOP:** Sum/OR of product terms

$P_1 + P_2 + P_3$ Ex: $a+bc+\bar{c}b$

→ **POS:** Product/AND of sum terms

$(S_1)(S_2)(S_3)$ Ex: $(a+b)(\bar{b})(\bar{a}+c)$

→ we are talking about forms of same exp. like (aa) & $(a+c)$, both are same exp. logically, but, in diff forms.

→ abc, $a+b+c$ are both SOP & POS.

Que: Is following term in SOP, if not convert.

$\overline{AB}C\bar{D} + A\bar{B}D + \bar{C}\bar{D} + A\bar{D}$

⇒ Not in SOP bcz of $\overline{AB}C\bar{D}$

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

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

↳ Now in SOP.

GO CLASSES

DATE : / /
PAGE NO. :

→ **Minterms:** A product term of n literal in which every variable must appear exactly once in original or complemented form.

→ **Maxterm:** A sum term of n literal in which every variable must appear exactly once in any form.

Ex: if f(a,b) then:

	Product term	Sum term	Minterm	Maxterm
a	✓	✓	×	×
$a+b$	×	✓	×	✓
$a\bar{b}$	✓	×	✓	×
$a+\bar{b}$	×	✓	×	✓
$\bar{a}+a+b$	×	✓	×	×

For 1 variable we have 2 minterms & 2 maxterms. $(a \,\&\, \bar{a})$

For 3 variable we have 2^3 i.e 8 minterms & 8 maxterms

abc, $ab\bar{c}$, $a\bar{b}\bar{c}$, $a\bar{b}c$ ↔ $a+b+c$, $a+b+\bar{c}$, $a+\bar{b}+c$, $a+\bar{b}+\bar{c}$

$\bar{a}bc$, $\bar{a}b\bar{c}$, $\bar{a}\bar{b}c$, $\bar{a}\bar{b}\bar{c}$ $\quad$ $\bar{a}+b+c$, $\bar{a}+\bar{b}+c$, $\bar{a}+b+\bar{c}$, $\bar{a}+\bar{b}+\bar{c}$

→ In all of the definitions above SOP, POS, sumterm, minterm---, order of literals does not matter.

→ For 'n' variables we have 2^n minterms & maxterms possible.

→ Every minterm is '1' i.e. True for exactly one row.

Ex:

a	b	ab	$a\bar{b}$	$\bar{a}b$	$\bar{a}\bar{b}$
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

∴ only for 1 row.

GO CLASSES

So, we can say ab is minterm associated with row $a=1, b=1$ & $a\bar{b}$ is minterm associated with row $a=1, b=0$.

Ex: $f(a,b,c)$

a	b	c	associated minterm		associated maxterm	
0	0	0	$\bar{a}\bar{b}\bar{c}$	(m_0)	$a+b+c$	(M_0)
0	0	1	$\bar{a}\bar{b}c$	(m_1)	$a+b+\bar{c}$	(m_1)
0	1	0	$\bar{a}b\bar{c}$	(m_2)	$a+\bar{b}+c$	(M_2)
0	1	1	$\bar{a}bc$	(m_3)	$a+\bar{b}+\bar{c}$	(M_3)
1	0	0	$a\bar{b}\bar{c}$	(m_4)	$\bar{a}+b+c$	(M_4)
1	0	1	$a\bar{b}c$	(m_5)	$\bar{a}+b+\bar{c}$	(M_5)
1	1	0	$ab\bar{c}$	(m_6)	$\bar{a}+\bar{b}+c$	(M_6)
1	1	1	abc	(m_7)	$\bar{a}+\bar{b}+\bar{c}$	(M_7)

Short Notation of minterms — short notation of maxterms

→ Similarly, every maxterm is 0, for exactly 1 row.

Ex: $f(a,b)$

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

So, we can also see associated maxterms in above table.

Also called index

Note: For short notation of minterm (m_i) or maxterm (M_i), fixed order of variables is considered, (alphabetical if not given)

0 → minterm → Complemented
0 → maxterm → Not complemented
1 → minterm → Not Complemented
1 → maxterm → Complemented.

DATE : / /
PAGE NO. :

⇒ Relationship b/w minterm & maxterm (m_i & M_i) is that both are complement of each other.

Ex: $m_2 = x \cdot \bar{y}$, $M_2 = \bar{x} + y$

$\overline{m_2} = \overline{x \cdot \bar{y}} = \bar{x} + y = M_2$

so $\overline{m_2} = M_2$ & $\overline{M_2} = m_2$

M_i is complement of m_i.

NOTE: For a given row of truth table, the minterm is formed by including x_i if $x_i = 1$ & by inc. $\bar{x}_i$ if $x_i = 0$ and the maxterm is formed by including x_i if $x_i = 0$ & by inc. $\bar{x}_i$ if $x_i = 1$.

Ex: $\bar{a}\bar{b}\bar{c}d\bar{e}$ & $\bar{a}+b+c+\bar{d}+e$ are associated with row $(a=0, b=0, c=0, d=1, e=0)$ i.e. row 18. (m_{18} / M_{18})

Que: For n variables, how many minterms are there?

⇒ Exactly 2^n (Not almost, misconception: that minterm f^n value is 1).

⇒ Expressing boolean functions using ~~minterms~~ maxterms/minterms: Any bool. fun. is True for some minterms & false for some ~~minterms~~ maxterms (Row). Similarly True for some maxterms & false for some maxterms.

- All the minterms having value 1 are called true (1) minterms.
- " " " " " 0 " " false (0) "
- " " maxterms " " 1 " " true (1) maxterms
- " " " " " 0 " " false (0) "

DATE : / /

PAGE NO. :

Ex:

x	y	f_1
0	0	0
0	1	1
1	0	0
1	1	1

→ f_1 is 0 for m_0

→ f_1 is 1 for m_3

1 minterms (or maxterms) m_1 & m_3

0 minterms (or maxterms) m_0 & m_2

Imp.

Any boolean function f can be expressed as a ~~function~~ summation (OR) of those minterms for which f = 1. OR can be expressed as a product (AND) of those maxterms for which f = 0.

$$f = \Sigma_m (f=1) \quad \text{or} \quad f = \Pi_M (f=0)$$

OR — list of 1-minterms — AND — list of 0-maxterms

So above $f_1 = \bar{x}y + xy$ or $f_1 = (x+y)(\bar{x}+y)$

→ Note that minterms are not 1-minterm, each row is a minterm i.e. a product term having all variables. However only 1-minterm are used to express f.

→ Minterms of function are those minterms which are used to express function i.e. when f=1 & Maxterm of function are those maxterm when f=0.

So, function's minterms & maxterms are mutually exh. & exclusive.

GO CLASSES

DATE : / /

PAGE NO. :

So if i say m_i, m_j & m_k are minterms of $f(a,b,c)$ then m_i, m_j & m_k are 1 and remaining minterms are 0 for $f(a,b,c)$.

Ex.

a	b	c	f_1	f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	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

Cannonical SOPs

$f_1 = \Sigma m(1, 4, 7) = \bar{x}\bar{y}z + x\bar{y}\bar{z} + xyz$

$f_2 = \Sigma m(3, 5, 6, 7) = \bar{x}yz + x\bar{y}z + xy\bar{z} + xyz$

minterms of f_1 are 1, 4 & 7, maxterms of f_1 are 0, 2, 3, 5, 6

" " f_2 " 3, 5, 6, 7, " " f_2 " 0, 1, 2, 4

Cannonical POSs

$f_1 = \Pi M(0, 2, 3, 5, 6) = (\bar{x}+\bar{y}+\bar{z})(x+\bar{y}+z)(x+\bar{y}+\bar{z})(\bar{x}+y+\bar{z})(\bar{x}+\bar{y}+z)$

$f_2 = \Pi M(0, 1, 2, 4) = (\bar{x}+\bar{y}+\bar{z})(x+y+\bar{z})(x+\bar{y}+z)(\bar{x}+y+z)$

⇒ Cannonical SOPs: Each fun. can be repr. by CSOPs by ORing all minterms of that function i.e. 1 minterm

Sum of minterms ≡ Cannonical SOP.

⇒ Cannonical POs: Each fun. can be repr by CPOSs by ANDing all maxterms of that functions → 0 maxterms

Product of maxterms: Cannonical POs.

DATE : / /

PAGE NO. :

→ for any boolean fun. we have unique CSOP & CPOS. It can have many SOPs & POSs. (NOT unique)

Ex:

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

CSOP = $\bar{a}b + a\bar{b} + ab$

$f = a+b$

↳ This is SOP but NOT CSOP.

SOP → Sum of Product Terms (may not have all literals)

CSOP → Sum of minterms

→ $f = \bar{f}$ is never possible.

→ Is CSOP complement of CPOS? Not at all. → CSOP & CPOS both are repr. same func, then how can they be complements.

→ Is CSOP dual of CPOS? No + at all.

→ CSOP & CPOS are equivalent & just principle unique repr. of a function.

Ex: $f = a+b$

CSOP of f = $\bar{a}b + a\bar{b} + ab$ — ①

CPOS of f = $a+b$. — ②

dual of f = $a \cdot b$ — ③

Comp. of f = $\bar{a} \cdot \bar{b}$ — ④

$1 \equiv 2, 1 \not\equiv 3, 1 \not\equiv 4, 2 \not\equiv 3, 2 \not\equiv 4, 3 \not\equiv 4.$

GO CLASSES

DATE : / /
PAGE NO. :

→ In propositional logic, SOP is disj. of conj. i.e. Disjunctive Normal form (DNF) & POS is Conj. Normal form (CNF).

Boolean Algebra	Propositional Logic
POS	CNF
SOP	DNF
CPOS	CCNF (Principle CNF)
CSOP	CDNF (Principle DNF)

→ $f = \Sigma_m(f=1) = \Pi_M(f=0)$

$\bar{f} = \Sigma_m(\bar{f}=1) = \Sigma_m(f=0) = \Pi_M(\bar{f}=0) = \Pi_M(f=1)$

(used in que)

→ for f, if minterms indices of f = S & all indices are N then maxterm indices of f = N−S.

Ex: $f(a,b,c) = \Sigma(1,5,6) = \Pi(0,2,3,4,7)$

minterms → $\Sigma(1,5,6)$; maxterms → $\Pi(0,2,3,4,7)$

Ques: Express the boolean function $f = A + \bar{B}C$ as sum of minterms

→ M1:

$A \cdot 1 \cdot 1 + \bar{B} \cdot C \cdot 1$

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

$ABC + A\bar{B}C + AB\bar{C} + A\bar{B}\bar{C} + A\bar{B}C + \bar{A}\bar{B}C$

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

$= \Sigma(1,4,5,6,7)$

M2:

A means 1 --, $\bar{B}C$ means -01

100, 101, 111, 101, 110, 001

$= \Sigma(1,4,5,6,7)$

My Pref

Use any method.

DATE : / /

PAGE NO. :

Que: Express $f = xy + \bar{x}z$ as product of maxterms

Ans: find $f = \Sigma m(1,3,6,7)$ by prev. method

& then $f = \Pi m(0,2,4,5)$

So, best method is find minterms first & then write other terms as maxterms & then POS.

Que: Express the complement of the following function in CSOP:

a) $F(A,B,C,D) = \Sigma(0,2,6,11,13,14)$

$\bar{F} = \Sigma(1,3,4,5,7,8,9,10,11,12,15)$

b) : $F(x,y,z) = \Pi(0,3,6,7)$

$\bar{F} = \Sigma(0,3,6,7)$

GO CLASSES

★ Minimization of Boolean Expression using K-Map

Bool. exp. can gen. be simplified by using the algeb. laws as done previously. However it has many problems:

a) No easy systematic procedure / algorithm.

b) Hard to tell if it is min or can be simplified further.

c) Result depends on order in which we apply boolean law. to minimize.

d) Error Prone.

For Ex: $f = \bar{a}\bar{b}\bar{c} + \bar{a}\bar{b}c + \bar{a}b\bar{c} + a\bar{b}c + ab\bar{c} + abc$

$f = \underline{\bar{a}\bar{b} + \bar{b}c + b\bar{c} + ab}$

↳ Irredundant i.e. not minimum nor can be further minimised.

$f = \underline{\bar{a}\bar{b} + b\bar{c} + ac}$

↳ minimum

DATE : / /
PAGE NO. :

So we use Karnaugh Map (K Map) to get minimum size expression from a function. It provides systematic approach by providing visuals to combine terms to get minimum expression. K map can give us minimum SOP or minimum POS.

Minimum SOP (or POS) has minimum # terms & each term has minimum # literals. It is not necessarily unique. We may get min. SOP/POS with diff terms but same literal counts in them.

★ **K-Map:** It can be thought of special version of truth table designed to ease the minimization w/o having to use boolean algebra theorems & equation manipulations.

K-map can be used to minimize expression with upto 4 variables easily.

→ **Idea behind K-Map:**

function $f(a,b,c)$ → unique SOP
↓
Combine minterms ← Basically SOP minterms
↳ Minimum Expression

Que. Which minterms can be combined?

a) $m_4, m_6 = a\bar{b}\bar{c} + ab\bar{c}$
$= a\bar{c}$

b) $m_4, m_7 = abc + \bar{a}\bar{b}c$
↳ Can't be combined.

GO CLASSES

DATE : / /
PAGE NO. :

→ m_i & m_j can only be combined if only one variable is changed.

Ex $ab\bar{c} + \bar{a}b\bar{c} = bc$ (only a is changed)

→ In Kmap, truth table is represented in such a way (in matrix/map format) that adjacent cells differ only by one variable.
↳ NOT Diagonally

Ex: $f(a,b,c)$

c \ ab	00	01	11	10
0	0	2	6	4
1	1	3	7	5

→ 100, 101 } only c changing.
→ 001, 111 → only b changing

Ex2: $f(a,b,c,d)$

cd \ ab	00	01	11	10
00	m_0	m_4	m_{12}	m_8
01	m_1	m_5	m_{13}	m_9
11	m_3	m_7	m_{15}	m_{11}
10	m_2	m_6	m_{14}	m_{10}

So. f can be repr. as k-map / CSOP / CPOS / Truth table, etc.
↳ just another repr. of any function.

In STT (std. Truth table), adjacent rows don't differ by 1 variable (Ex 01 & 10)

→ Each cell in k-map ≡ a minterm ≡ a maxterm

K-map is thought of 3D i.e. folded i.e. edges are touching i.e. 'kind' of sphere

DATE:

PAGE NO.:

→ **Adjacent cells:** Two cells are adjacent iff they differ exactly by a single variable.

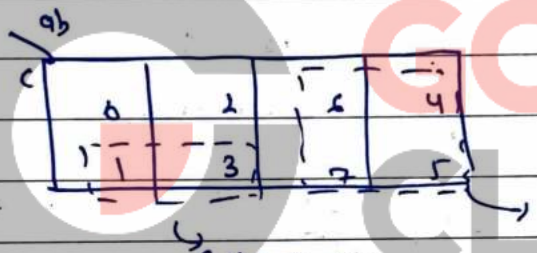
Ex:

c \ ab	00	01	11	10
0	0 (000)	2 (010)	6 (110)	4 (100)
1	1 (001)	3 (011)	7 (111)	5 (101)

, 0,2 are adjancent 0 & 3 are not adjacent.
4,0 are adjacent 5 & 0 are not.

→ **Cube in k-map:** Group of 2^m cells so that every cell in that group is adjacent to exactly m cells in that group. $(m \geq 0)$

Ex:



0,4 is also a cube of size 2

cube of size 2^2,

cube of size 2^1

6 is adj to 4 & 7, (i.e. 2 cells)
4 " " " 6 & 5,
5 " " " 7 & 4, &
7 " " " 5 & 6.

1 is adj. to 3 & 5
3 is adj to 1

{0,1,4,5}
→ cube of size 2^2.

{0,2,6,4,1,3,7,5}
→ cube of size 8, i.e. each cell is adj. to 3 cells.

0 is adj. to 1,4,5.
6 " " to 2,4,7
5 " " " 4,7,1.

You know cubes. now

GO CLASSES

DATE : / /

PAGE NO. :

Que: Why 2^m cells in a cube in K-map?

→ Because we want minimum SOP. Ie One product term with r ~~t~~ literals cover 2^m minterms where r+m is # var. in fun.

Ex: $f(a,b,c)$.

We have product term ab, minterms are abc & $ab\bar{c}$ → 2^1

product term $\bar{a}$ then minterms $\{a\bar{b}\bar{c}, a\bar{b}c, abc, ab\bar{c}\} \Rightarrow 2^2$

(Similar case for sum term also).

So, this is why we cover cubes of size 2^m & check only those var. which are not changing in cube.

Idea

2 minterms can be reduced to 1 product term iff they change by single variable. (2 minterms can't be reduced to 1 product term). 4, 8 minterms can be reduced to 1 product term iff all minterms have 1 unchanged variable.

Ex: $\underbrace{abc + a\bar{b}c}_{\downarrow} + \underbrace{ab\bar{c} + a\bar{b}\bar{c}}_{\downarrow}$ (a is unchanged)

$ac + a\bar{c}$

a

↳ This product term is covering 4 minterms as $f(a,b,c)$ has 3 vars.

So, we create cube of size 2^m so that we can write product term (or sum term) for that cube, & then write min. exp.

DATE : / /
PAGE NO. :

Size ↑ cube ↑ literals in product (or sum) term ↓

mSOP ⇌ minimum SOP.

→ for Writing product terms from cubes to find min SOP, we make cubes of 1-minterms,

finding mSOP:

a). make K-map.

b). make as large as possible cubes where fun is 1. (can overlap)

c). Cover all 1's by min. (max sized) cube.

d) write product terms of all cubes we found above.

e) Sum these terms & its mSOP.

Ex:

b \ a	0	1
0	0	0
1	1	1

→ b

(as only a is changing & b is unch.)

mSOP = b.

b \ a	0	1
0	1	0
1	1	0

$\bar{a}$

mSOP = $\bar{a}$

b \ a	0	1
0	1	0
1	0	1

$\bar{a}\bar{b}$, ab

mSOP = $ab + \bar{a}\bar{b}$

b \ a	0	1
0	1	1
1	1	1

$\bar{b}$, $\bar{a}$

∴ mSOP = $\bar{a} + \bar{b}$

b \ a	0	1
0	1	1
1	1	1

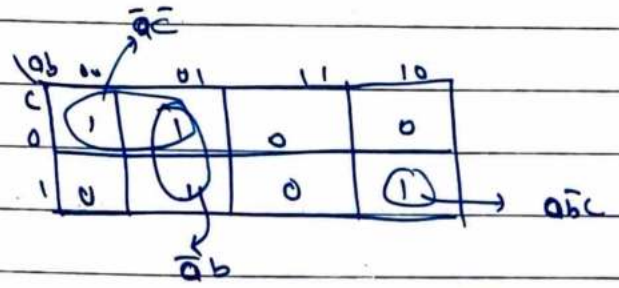
mSOP = 1

DATE : / /

PAGE NO. :

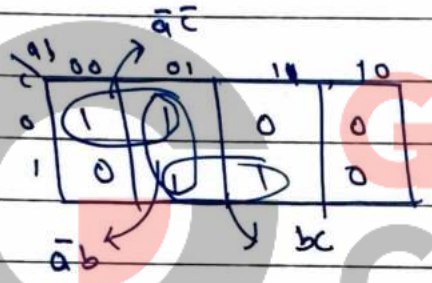
In cube with 2^m cells, m variables change so product term has n-m literals where n is # var. in function.

Ex: 1



$f = \bar{a}b + \bar{a}\bar{c} + a\bar{b}c.$

Ex: 2



$f = \underline{\bar{a}\bar{c} + \bar{a}b + bc}$

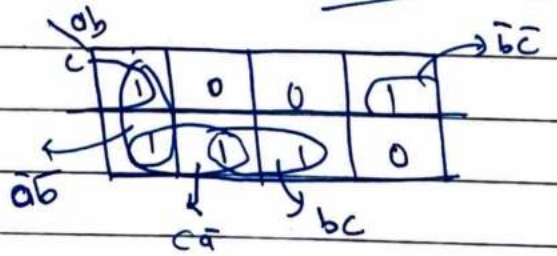
↳ min : No.

$\bar{a}b$ cube is redundant.

$f = \underline{\bar{a}\bar{c} + bc}$

↳ MSOP

Ex: 3



$f = c\bar{a} + \bar{b}\bar{c} + bc + \bar{a}\bar{b}$

(Not min)

$f = \bar{a}\bar{b} + c\bar{a} + bc + \bar{b}\bar{c}$ OR $c\bar{a} + bc + \bar{b}\bar{c}$

Both are MSOPs.

Not unique.

GO CLASSES

	a	b	f
m_0	0	0	x
m_1	0	1	y
m_2	1	0	z
m_3	1	1	w

=

b \ a		
	x	z
	y	w

Preffered.

=

b \ a		
	x	y
	z	w

Que: Simplify bool. func. using kmap. $f(a,b,c) = \Sigma(0,2,3,4,6)$

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

$\rightarrow \bar{c}$

$\rightarrow \bar{a}b$

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

GO CLASSES

Rules for minimisation using k-map:

a). First cover those 1 cells, by cubes that cannot be combine with other 1 cells

b). Cover 1 cells with min. # cubes such that each cube is as large as possible.

c) Cube within another cube is not used in min SOP

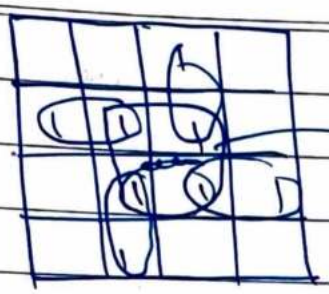
d) If there is more than one way of covering the map with minimal # cubes, select the cover with larger cubes (i.e less # literals in min SOP)

e) A cube contained in any combination of other cubes already selected in the cover is redundant by virtue of consensus theorem (i.e. $ab + \bar{a}c + bc$)

↳ Red undant.

DATE:
PAGE NO. :

Minimal exp. can be one among minimum SOP or mPOS or can be both min. exp.



→ Redundant (Larger cube)

⇒ Using k-map to get minimised POS expression

Here we cover zeroes in k-map with cubes so write sum terms & then product them.

Que. Find mPOS if $f(a,b,c) = \Sigma(0,1,3,7)$

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

$(\bar{b}+c)$

$\bar{a}+b$

↳ Not changing (Comp. for 1)

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

Que. If $f(a,b,c,d) = \Sigma(0,3,4,8,10,12)$, find mPOS?

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

$\bar{d}+c$

$\bar{a}+\bar{d}$

$\bar{c}+a+d$

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

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

→ There is no relation b/w #mSOPs & #mPOSs.

GO CLASSES

DATE : / /

PAGE NO. :

Qu: If we make every cube of size 1, then which type of SOP will we derive?

⇒ Canonical SOP.

★ **Functions with Don't Cares Combinations:** Partial function (Incompletely Specified fun)

For a bool. funⁿ, sometimes or can occur some input combinations can NEVER occur, for these input comb. we don't care about the fun. value. Such combinations are called Don't Care Combinations.

For Ex: $f(a,b) = 1$ if $a=1$

T.T.

a	b	f_1
0	0	X
0	1	X
1	0	1
1	1	1

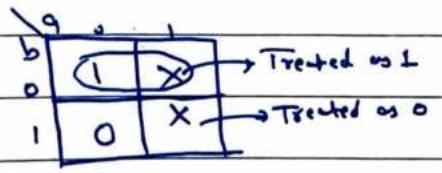
] Don't Care combinations as we don't care what happens when a=0.

f_1	f_2	f_3	f_4
0	0	1	1
0	1	0	1
1	1	1	1
1	1	1	1

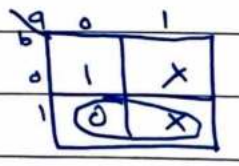
4 fun. f is repr.

Each don't care can be specified as 0 or 1 (like joker card), a fun. with k don't cares corresponds to a class of 2^k distinct functions. Our aim for mSOP is to use any value of don't care which makes cube large.

Ex: $f(a,b) = \Sigma(0) + d(2,3)$



mSOP = $\bar{b}$



mPOS = $\bar{b}$

DATE : / /

PAGE NO. :

Ex:2 Find mSOP for

$f(w,x,y,z) = \Sigma m(0,2,4,5,8,14,15)$

$d(w,x,y,z) = \Sigma m(7,10,13)$

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

→ Not req. / redundant.

$mSOP = \bar{x}\bar{z} + \bar{w}x\bar{y} + wxy$

NOTE Largest cube may n't be present in mSOP / mPOS.

★ **Covering Functions:** f covers g ~~only~~ iff whenever g=1, then f=1 OR whenever f=0 then g=0.

i.e.

g	f
1	1
0	0/1
0	0
~~1~~ 0	~~1~~ 1

f covers g here.
(Remember del).

Qu: $f = a+b$ & $g = ab$. ? g cover f or f cover g?

⇒ when f=1 (0,1), g=0 so g don't cover f.
but when g=1 (1,1), f=1, so f covers g.

If f cover g & g cover f then f=g.

f covers g ≡ g implies f ≡ g=1 → f=1 ≡ f=0 → g=0

Covering fun. is defined from its POV. (minterm)

DATE: / /

PAGE NO.:

Implies ka ulta covers

$f = x + \bar{y} + \bar{z}$, $g = x + \bar{z}$

here $f=0$ means $g=0$, so f covers g.

NOTE If $f(a,b,c) = \Sigma(0,5,7)$,
then covering fun. of f:
$g(a,b,c) = \Sigma(0,5,7) + d(1,2,3,4,6)$

Not valid for Π

OR

If $f(a,b,c) = \Pi(0,5,7)$
then covering fun. of f:
$g(a,b,c) = \Sigma(1,2,3,4,6) + d(0,5,7)$.

→ mpos / mSOP for "covering fun of f" in "f" (check above note & make k-map).

→ Bigger product term implies its subset product term i.e. subset covers bigger product term.

Ex. $\alpha\beta\gamma \rightarrow \alpha\beta$ so $\alpha\beta$ covers $\alpha\beta\gamma$

→ Smaller ~~product~~ sum term implied its subset sum term i.e. subset term covers bigger sum term.

Ex: $\alpha+\beta \rightarrow \alpha+\beta+\gamma$ so $\alpha+\beta+\gamma$ covers $\alpha+\beta$.

GO CLASSES

K-map is eff. for ≤4 var. only.

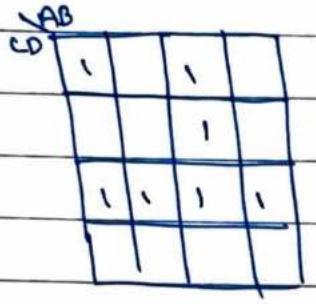
DATE : / /

PAGE NO. :

Implicants (Imp.), Prime Implicants (PI), Essential Prime implicants (EPI)

→ Every cube of 1's is implicant.

Ex:



Here we have

7 → 1 sized imp. / cube.

6 → 2 size imp.

1 → 4 sized imp.

total 14 implicants.

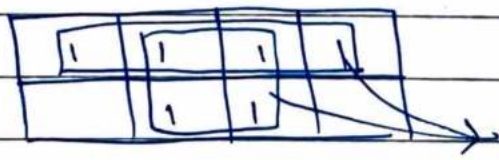
→ Best way to count implicants is grab each cell one by one & write all implicants from that ~~cube~~ cell & never see that cell again or any cube covering that cell as it's already covered.

→ Implicant is a product term which implies f.

Imp = 1 → f = 1 Obviously

→ Prime Implicants: cube of 1's which is as big as possible. (Subset of imp.)

Ex:



Imp: 6 + 7 + 2 = 15

Prime Imp: 2

Prime Imp is an imp. (from map POV) which is not fully contained within any other implicant.

→ Product term (as small as possible) which implies f, is PI. (don't have redundant literal)

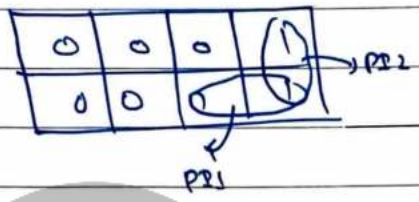
GO CLASSES

DATE : / /
PAGE NO. :

→ **Essential Prime Implicant (EPI):** It is a PI that includes atleast one '1' that is not in any other PI. It's a PI that has one cell atleast (1) which is not part of another PI. (Covers 1 cell uniquely)

Non EPI are those PI where every cell (of 1) of cube is part of another PI.

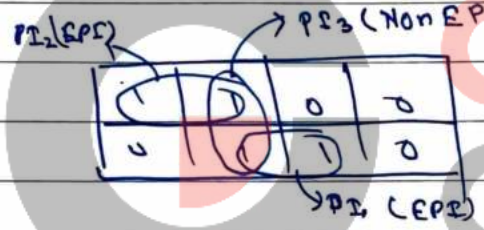
Ex:1



Imp = 5. (3+2)

PI1 & PI2 both are EPI.

Ex:2

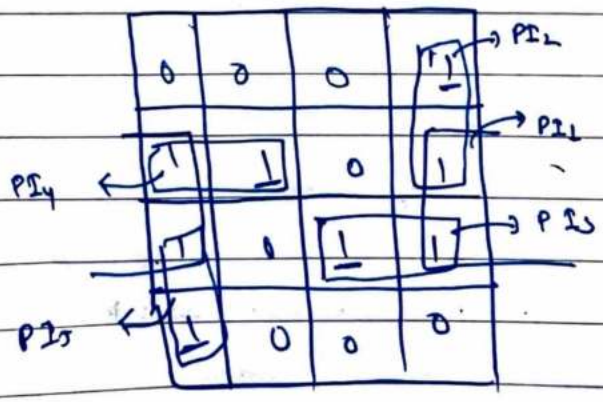


PI3 is not EPI.

→ In MSOP we have to mandatorily take all EPIs & if EPIs are covering all 1's then we have unique MSOP. (Converse is not True)

Que: Determine EPI & Non EPI

$\Sigma m(1,2,3,5,8,9,11,15)$



size1 size2 size4

Imp: 8+8+1 = 17.

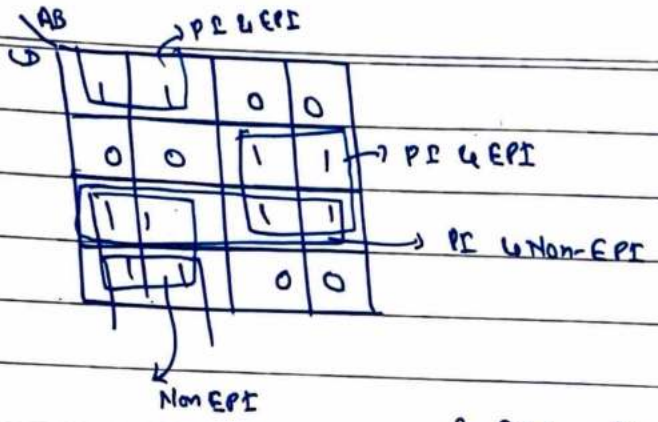
PI2, PI3, PI4 & PI5 are EPIs & PI1 is not EPI.
↳ will not be in MSOP also.

& While making cubes, always look at corners & folds)

DATE: / /

PAGE NO.:

Ex:3



2 EPI will be in mpos & any 1. nonEPI

So mpos: $\bar{A}\bar{D} + AD + CD$

OR

$\bar{A}\bar{D} + AD + \bar{A}C$.

→ In msop we have all EPIs (must) & some non EPIs. (but all are PIs).

→ For don't cares, we consider them as 1 always to count implicants & also for PI we consider them 1. For EPI we don't consider it (same def. for EPI i.e. should uniquely cover atleast one '1' cell).

Ex: 4

X	1		X
0	X	1	X

Imp: 6+6 = 12

PI = 6

EPI = 0.

★ **Cyclic K-map:** Has exactly 2-msols. and both msops have 0 common term. Thus a is no EPI, every PI has same size & each 1 cell is covered by exactly 2 PIs.

DATE : / /

PAGE NO. :

Ex: 1

c \ ab			
1	1	0	1
0	1	1	1

All condition satisfied.

MSOP1: $\bar{a}\bar{c} + cb + a\bar{b}$

MSOP2: $\bar{b}\bar{c} + \bar{a}b + ac$

Ex: 2

1			1
1	1		
	1	1	
		1	1

✓

NOTE: Terms Imp, PI, EPI are defined only for SOP expressions (never cover 0 with them).

→ Some author (non standard) define similar concept to implicant for 0 cells also called Implicate.

SOP	POS
Implicant	Implicate
Prime Implicant	Prime Implicate
EP. "	Essential " "

Ex: $f = \Sigma m(2,3,5,7)$

c \ ab			
0	1	0	0
0	1	1	1

Implicate = 4+3 = 7

Implicants = 4+3 = 7

DATE : / /

PAGE NO. :

Dual & Complement of a function

$$f^d(a, b, \bar{a}, \bar{b}, +, \cdot, 1, 0)$$
$$\Updownarrow$$
$$f(a, b, \bar{a}, \bar{b}, \cdot, +, 0, 1)$$
$$\Updownarrow$$
$$\bar{f}(\bar{a}, \bar{b}, a, b, +, \cdot, 0, 1)$$

So in dual of function, variables remain unchanged. (Also Not)

Ex: Dual of eqn $(a)(a+b) = a$ is $(a)+(ab) = a$

Dual of "$(\bar{x}+\bar{x}) = \overline{xy}$" is "$(\bar{x}\bar{y}) = \overline{(x+y)}$"

$$f^d(a, b, \bar{a}, \bar{b}, +, \cdot, 1, 0)$$
$$\Updownarrow$$
$$\bar{f}(\bar{a}, \bar{b}, a, b, +, \cdot, 1, 0)$$

→ So only literals change when $f^{dual} \leftrightarrow \bar{f}$.

Ex if $f^d = a+b$

then, $\bar{f} = \bar{a}+\bar{b}$

& $f = ab$

→ So, for $f^d(0,0) = \bar{f}(1,1)$

$f^d(1,0) = \bar{f}(0,1)$

so on.

GO CLASSES

DATE : / /
PAGE NO. :

Ex:

a	b	$\bar{a}+b$ $f(a,b)$	$\bar{f}=a\bar{b}$	$f^d=\bar{a}b$
0	0	1	0	0 $(\bar{f}(1,1))$
0	1	1	0	1 $(\bar{f}(1,0))$
1	0	0	1	0 $(\bar{f}(0,1))$
1	1	1	0	0 $(\bar{f}(0,0))$

So.

a	b	f	$\bar{f}$	f^d
0	0	x	$\bar{x}$	$\bar{w}$
0	1	y	$\bar{y}$	$\bar{z}$
1	0	z	$\bar{z}$	$\bar{y}$
1	1	w	$\bar{w}$	$\bar{x}$

So In dual we basically rev. the complemented row. (if TT in std. form).

Que: $f=(\bar{v}w+x)v+\bar{z}$

Verify $f^d(0,1,0,0,1) = \bar{f}(1,0,1,1,0)$

$f^d = (((\bar{v}+w)\cdot x)+y)\cdot\bar{z}$
$= (((1+1)\cdot 0)+0)\cdot 0$
$= 0$

$\bar{f} = (((v+\bar{w})\cdot\bar{x})+\bar{y})\cdot z$
$= (((1+1)\cdot 0)+0)\cdot 0$
$= 0$

→ In std. T.T. we write input comb. in order of their decimal value.

GO CLASSES

DATE: / /

PAGE NO.:

Qu: How to find dual exp. if T.T. is given for dual.

⇒ from T.T., find $\bar{f}$ & then f's expression and from expression of f, find dual.

Ex:

a	b	f^d	$\bar{f}$	f
0	0	0	1	0
0	1	1	1	0
1	0	1	1	0
1	1	1	0	1

$f = ab$

$f^d = a+b$.

Note that $f^d(t) = \bar{f}((2^n-1)-t)$ for f over n variables

or

$\bar{f}(t) = f^d(2^n-1-t)$

If $f(x) = 0$ then $\bar{f}(x) = 1$

if $\bar{f}(x) = 1$ then $f^d(2^n-1-x) = 1$

if $f(x) = 0$ then $f^d(2^n-1-x) = 1$.

★ Neutral Boolean function: A bool. fun. 'f' over n var. is neutral iff f=1 for half of input combinations and f=0 for remaining half.

Ex. $f = a \oplus b$, $a \odot b$.

neutral functions = $\frac{{}^{2^n}C_{2^{n-1}}}{**}$ where n var. function

DATE : / /
PAGE NO. :

⇒ $f = T$ → Never , $f \neq T$ → Always , $f = f^d$ = sometimes.

★ **Self Dual Functions:** function f is self dual iff $f = f^d$.

a	b	f	f^d
0	0	a_0	$\overline{a_3}$
0	1	a_1	$\overline{a_2}$
1	0	a_2	$\overline{a_1}$
1	1	a_3	$\overline{a_0}$

DH

So, for self dual fun $a_0 = \overline{a_3}$, $a_1 = \overline{a_2}$, $a_2 = \overline{a_1}$ & $a_3 = \overline{a_0}$

So $a_0 \neq a_3$, $a_1 \neq a_2$, $a_2 \neq a_1$ & $a_3 \neq a_0$

Self Dual function ⟶ Neutral

Ex:

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

Neutral but not self dual.

Half of the ~~elements~~ rows have 2 choices so.

self dual fun = $(2)^{2^{n-1}}$ for n variables

★ In self dual fun.

$f(x) \neq f(2^n - 1 - x)$ $\forall 0 \leq x \leq 2^n - 1$

Decimal value

DATE : / /

PAGE NO. :

Que: $f = \Sigma(0,2,3,6)$ is f self dual?

→ **Yes.**

$$\left.\begin{array}{l} f(0) \neq f(7) \\ f(1) \neq f(6) \\ f(2) \neq f(5) \\ f(3) \neq f(4) \end{array}\right\}$$ Exactly 1 among pair is 1.

★ Another special fun. is those fun whose comp. is same as dual (No specific name).

↳ Orthogonal Function? (No) } Misconception.

↳ No such word

Ex:

a	b	f	$\bar{f}$	f^d
0	0	0	1	1
0	1	1	0	0
1	0	1	0	0
1	1	0	1	1

So $\bar{f} = f^d$.

Que. Is such function neutral always?

⇒ **No.** Ex:

a	b	f	$\bar{f}$	f^d
0	0	0	1	1
0	1	0	1	1
1	0	0	1	1
1	1	0	1	1

same

$\bar{f} = f^d$ but not neutral.

for such fun $f(x) = f(2^n - 1 - x)$ $\forall x \; 0 \le x \le 2^{n-1} - 1$ } Palindromic string.

#such fun $= 2^{2^{n-1}}$ }

↳ Half rows have 2 choices

DATE : / /
PAGE NO. :

★ XOR, XNOR functions & their properties:

NOTE Cancellation law does not hold in boolean algebra.

Ex NOR is also known as Eq. oper / Coincidence opⁿ. It is same as biimplication of prop. logics.

$$\boxed{a \oplus b = \bar{a} \odot b = a \odot \bar{b} = \overline{a \odot b} = \bar{a} \oplus \bar{b}}$$

↳ Think of what you are changing & how is it affecting us

★ XOR with more than 2 inputs: Mod 2 sum

It is comm & associative

→ So XOR is 1 iff odd no. of 1's in i/p.

$1 \oplus 1 \oplus 1 \oplus \underbrace{\quad}_{\text{any 0}} = 1$

$a \oplus a \oplus a \oplus a = 0$

↳ bcz odd 1's not possible.

⇒ XNOR is 1 iff even no. of 0's in i/p.

$0 \odot 0 \odot 0 \odot 0 \odot \underbrace{\quad}_{\text{any 1s}} = 1$

~~$0 \oplus 0 \oplus 0 \oplus 0 +$~~

⇒ XOR is odd 1 checker & XNOR is even 0 checker.

→ XOR is equal to XNOR when odd. no. of i/p are considered

Ex: $a \oplus b \oplus c = a \odot b \odot c$

↳ always bcz. { LHS is 1 when 1 or 3 1's
RHS is 1 when 0 or 2 0's
same always.

GO CLASSES

DATE : / /

PAGE NO. :

★ Positive Logic System & Negative Logic System:

Binary logic is always 0s & 1s but electricity don't comes as 0 or 1, it comes in volts. So we assign volts to 0's & 1's.

In positive logic we assign 0s to low voltage & 1's to high voltage.

In negative logic we assign 0s to high voltage and 1's to low voltage.

Ex:

Digital Circuit			+ive Logic System			-ive Logic System		
X	Y	F(X,Y)	X	Y	F	X	Y	F
L	L	L	0	0	0	1	1	1
L	H	H	0	1	1	1	0	0
H	L	H	1	0	1	0	1	0
H	H	H	1	1	0	0	0	0

So +ive → -ive logic

AND → OR,

OR → AND,

1 → 0 (High)

0 → 1 (Low)

a → a

Conclusion is +ive logic system & -ive logic system are duals of each other.

GO CLASSES

Made By - Karan Agrawal (GATE CS 2024 AIR 102)

DATE: / /
PAGE NO.:

M2 in DLD 2024 course

Module-3 Number System

Number SYSTUMM: Simply, way to count things or way to represent numbers.

Ex: Roman System, Unary, Binary, Decimal etc

- Roman System → using english alphabets
- Unary → Babies count 1 symbol
- Binary → for computers 2 symbols (bits)
- Decimal → for Humans 10 symbols (digits)

Decimal System:

We start to repeat symbols as soon as we finish them.

Ex: In decimal no. system:

0, 1, ---, 7, 8, 9, 10, 11, 12, --- 20 --- 99, 100, 101 --- 799, 800 ---

Base of no. system is defined as the total no. of symbols available in the no. systems.

So base of decimal system is 10.

Base equals no. of diff values the set has before repeating itself.

Every digit has a weight:

Ex:

2	2	7	0	6
↑	↑	↑	↑	↑
10^4	10^3	10^2	10^1	10^0

In general,

$(abcde)_x$ (base is x)

$= (a*x^4) + (b*x^3) + (c*x^2) + (d*x^1) + (e*x^0)$

In $(256)_{10}$ 5 is telling us how many 10's we have skipped and 2 is telling us how many 100's we have skipped.

Counted.

DATE : / /
PAGE NO. :

★ Binary System:

Also called computer number system.
2 symbols called bits i.e. 0 & 1.

Binary	Decimal
0	0
1	1
1 0	2
1 1	3
1 0 0	4
1 0 1	5
1 1 0	6
1 1 1	7
1	1
1	1

No symbol now repeat

repeat again

& so on.

Here in $(1010)_2$, 2nd symbol used after counting
2 items 1 time and 3rd " " " "
4 " 0 times " 4th " " " "
8 " 1 times.

⇒ **Binary to Decimal:**

$$(1011)_2 = 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = 8+0+2+1$$
$$= (11)_{10}$$

~~Binary to Decimal~~

$$(a_1\ a_2\ a_3)_2 = a_1*2^2 + a_2*2^1 + a_3*2^0$$

DATE : / /

PAGE NO. :

⇒ **Decimal to Binary:** Do reverse engineering

as $(11)_{10} = (1011)_2$

$= 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0$ — ①

Divide ① with 2

$= 1*2^2 + 0*2 + 1$ — ②

mod ulus of ① with 2 is last digit of binary.

So $11\%2 = 1$

& $11/2 = 5$ = ⑤

Now if we divide ⑤ by 2

$= 1*2 + 0$ — ③

& mod ③ by 2, it's 2nd bit of binary.

$5\%2 = 1$

$5/2 = 2$ — ④

and so on we keep on dividing & convert decimal to binary.

Shorthandedly:

→ Remainder (Mod)

2	11	1
2	5	1
2	2	0
2	1	1
	0	

↓ Reverse order

$(11)_{10} = (1011)_2$

This's how it's working.

GO CLASSES

DATE : / /
PAGE NO. :

Remainder obtained at each state is one of the desired bit and least sig. bit is obtained first.

Ex:2 $(156)_{10} = (?)_2$

	Quotient	Remainder
$156 \div 2$	78	0 — LSB
$78 \div 2$	39	0
$39 \div 2$	19	1
$19 \div 2$	9	1
$9 \div 2$	4	1
$4 \div 2$	2	0
$2 \div 2$	1	0
$1 \div 2$	0	1 — MSB

(10011100)

$= (\cancel{00}\ \cancel{11}\ \cancel{00}\ \cancel{11})_4$

Generally

$\Rightarrow$ $(abcd)_B = (N)_{10}$ where B is base of abcd.

$(N)_{10} = (a*B^3 + b*B^2 + c*B^1 + d)_{10}$

Quotient of N/B is Q_1 $(a*B^2 + b*B + c)$

Remainder of N/B is d

Quotient of Q_1/B is Q_2 $(a*B + b)$

Remainder of Q_1/B is c

Quotient of Q_2/B is Q_3 (a)

Remainder of Q_2/B is b

Quotient of Q_3/B is 0 (0)

Remainder of Q_3/B is a

& we get ~~binary~~ in base B (abcd)

DATE : / /
PAGE NO. :

★ **Octal System:** Here we have base 8 i.e. 8 symbols (0,1,2,3,4,5,6,7).

0, 1, 2, ... 7, 10, 11, 12 ... 17, 20 ... 77, 100, 101 ... 777.

$(67)_8 = (6*8^1 + 7*8^0)_{10}$

$= (48+7)_{10} = 55$

★ **Hexadecimal System:** Base is 16.

16 symbols are (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F)

$(10F)_{16} = (1*16^2 + 0*16^1 + 15*16^0)_{10}$

$= 256 + 15$

$= (271)_{10}$

⇒ **Base 'r' to Decimal:**

for any base r system $r>1, r \in N$, we have r symbols and

$(abcd)_r = [(a*r^3) + (b*r^2) + (c*r) + (d*r^0)]_{10}$

Also if we have radix point

$(abcd.ef)_r = [(a*r^3) + (b*r^2) + (c*r) + (d*r^0) + (e*r^{-1}) + (f*r^{-2})]_{10}$

Ex: $(101.01)_2 = (1*2^2 + 0*2^1 + 1*2^0 + 0*2^{-1} + 1*2^{-2})$

$= 4 + 1 + \frac{1}{4}$

$= (5.25)_{10}$

Radix point in decimal no. system is called decimal point.

Ex:2 $(11F.0A)_{16} = (1*16^2 + 1*16 + 15*16^0 + 0*16^{-1} + 10*16^{-2})$

$= \left(256 + 16 + 15 + 0 + \frac{10}{256}\right)_{10}$

GO CLASSES

DATE : / /

PAGE NO. :

⇒ **Decimal to Base "r"** Already seen.

Short hand:

$N = (ar^3 + br^2 + cr + d)$

r	N

Quotient	Divisor	Remainder
	ar^3+br^2+cr+d	d
	ar^2+br+c	c
	$ar+b$	b
	a	a
	0	

So $N = (abcd)_r$

In case of radix point:

$N = (.ab)_{10} = (?)_r$

Let $(.ab)_{10} = (.xyz)_r$

$N = (.ab)_{10} = \left(\frac{x}{r} + \frac{y}{r^2} + \frac{z}{r^3}\right)_{10}$

Now $N * r$

$= x + \frac{y}{r} + \frac{z}{r^2}$ — (2) fractional/ Decimal part

Integer part $= x$

Now (2) $* r$

$= y + \frac{z}{r}$ — (3) fractional/ Decimal part

Integer part $= y$

Now (3) $* r$

$= z$

↳ Integer part.

GO CLASSES

DATE : / /

PAGE NO. :

Ex: $(.152)_{10} = (?)_8$

$.152 * 8 = 1.216$

$.216 * 8 = 1.728$

$.728 * 8 = 5.824$

$.824 * 8 = 6.592$

⋮

$= (.152)_{10} = (.1156\ldots)_8$

Ex:2 $(.25)_{10} = (?)_2$

$.25 * 2 = 0.5$

$.5 * 2 = 1.0$

$= (.01)_2$

Ex:3 $(.3)_{10} = (?)_2$

$.3 * 2 = 0.6$

$.6 * 2 = 1.2$

$.2 * 2 = 0.4$

$.4 * 2 = 0.8$

$.8 * 2 = 1.6$

$.6 * 2 = 1.2$ Repetition.

Keep repeating.

$= (.01001)_2$

Basically for decimal to base conversion we handle decimal & fraction separately.

For Ex: $(10.625)_{10} = (?)_2$

2	10	0
2	5	1
2	2	0
2	1	1
	0	

$.625 * 2 = 1.250$

$.25 * 2 = 0.5$

$0.5 * 2 = 1.0$

So $= (1010.101)_2$

$=$

DATE : / /
PAGE NO. :

Que: $(23.31)_4 \rightarrow (?)_7$

$\Rightarrow$ $(23.31)_4 = (2*4^1 + 3*4^0 + 3*4^{-1} + 1*4^{-2})_{10}$
$= (11.8125)_{10}$

$(11.8125) = (?)_7$

7	11	4
7	1	1
	0	

$.8125 * 7 = 5.6875$
$.6875 * 7 = 4.8125$
$.8125 * 7 = 5.6875$

$= (14.\overline{54})_7$.

~~Ans.~~

Que: Why octal & hexadecimal systems? What's is it's importance?

$\Rightarrow$ As we know need of binary in computer system (High & Low voltage).

But, binary is a long sequence. ~~and~~ Binary can be easily converted to ~~binary~~ octal by grouping 3 bits each as $(2^3 = 8)$

Ex: $(011\ 011\ 010\ 101\ 100\ 101)_2 = (33254 5)_8$

Similarly hexadecimal can also be got from binary.

Ex: $(0100\ 1010\ 1011\ 0111)_2 = (8A77)_{16}$ (append 200)

(Always start to make groups from left of radix to right of radix, and append any no. of zeroes to complete grp) size 3/4

DATE : / /

PAGE NO. :

Ex :- $(\underbrace{0000}_{Grp2}\underbrace{bcde}_{Grp1} \cdot \underbrace{fg00}_{Grp1})_2 = (\ ?\ )_{16}$

As binary can be converted back easily from octal & hexadecimal. we convert decimal to octal/hd. (less # steps) & then easily get binary from octal/hd.

$$(Src)_{10} \rightarrow (Mid)_{8/16} \rightarrow (Dstn)_2$$

So, basically we reduce # steps & this way beneficial for computers.

$(1101 \cdot 1011)_2 \Rightarrow (\underbrace{00}_{\text{Append zero}}1101 \cdot 101100)_2 = (15 \cdot 54)_8$

★★ Binary Arithmatic : similar to decimal arithmatic of schools.

★ Addition

if digit $x+y <$ base. write as it in

if digit $x+y \geq$ base write base $-(x+y)$ & 1 as carry

Ex:

$$\begin{array}{cccc} & 1 & & 1 \\ 2 & 3 & 0 & 4 \\ 9 & 8 & 7 & 6 \\ \hline 12 & 1 & 8 & 0 \end{array}$$

$10 \geq$ Base. 0 & 1 carry

$1+0+7 <$ Base

$9+2+1 \Rightarrow 12$

$8+3=11$ & 1 carry, 1 written

DATE : / /

PAGE NO. :

Similarly for Binary:

```
  1 1 1 1
 10 11 0 1
 11 01 1 1
-----------
11 001 0 0
```

2 ≥ Base

write 2-2 & 1 carry

1+1+1 = 3 ≥ Base.

write 3-2 & 1 carry

In worst case we have to add 1+1+1 = 3 so 1 carry & sum 1.

Two digit + One Carry

x	y	$carry_{in}$	Sum	Carry 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

So addition is Easy-Peasy.

GO CLASSES

DATE : / /

PAGE NO. :

☆ Subtraction:

Subtraction were awesome if we don't have to borrow.
We can guarantee NO BORROW if we subtract from all 9s

Ex:

$$\begin{array}{r} 999 \\ 784 \\ \hline 215 \end{array}$$ → No borrow

Another way to subtract.

$82-64 \Rightarrow 82+100-100-64 \Rightarrow 82+(100-64)-100$

$\Rightarrow 82+\underbrace{(99-64)}_{\text{Easy subtraction as No borrow}}-100+1 \Rightarrow 82+(99-64)-100+1$

$\Rightarrow \underbrace{82+35}_{\text{Addition is anyway easy}}-100+1 \Rightarrow \underbrace{118}_{\text{ignore carry}}-100 = \underline{18}$ Ans.

This subtraction is done by complementing the number so that we don't have to borrow.

⇒ **Complements of a number in base 'r'.**

- (r-1)'s complement → Diminished radix Complement
- r's Complement → Radix Complement.

In decimal we have 9's & 10's complement.
" Octal " " 7's & 8's "
" Binary " " 1's & 2's "

GO CLASSES

$\frac{-1}{b}$ $\Leftarrow$ $\left\{ \frac{-0}{1} \quad \frac{-1}{0} \right.$

DATE : / /

PAGE NO. :

If we have n digits in no. N of base r then $(r-1)^s$ complement is $(r^n-1)-N$ and r's complement is $r^n - N$.

Ex: $N = (82)_{10}$

9's comp = $100-1-82$

$= 99-82$

$= 17$

10's comp = $10^2 - 82$

$= 100-82$

$= 18$

NOTE: r's complement = (r-1)'s complement +1 and in (r-1)'s comp. $\underbrace{rrrr\ldots r}_{n \text{ times}} - N$.

Ex:2 $N = (0101101)_2$

1's comp = $1111111 - 0101101$

$= 1010010$

2's comp = 1's comp + 1

$= 1010010 + 1$

$= 1010011$

In binary $1-x$ is $\bar{x}$ So, to get 1's complement just flip the bits.

Ex: 1's comp. of $(110110100)_2$ is $(001001011)_2$.

GO CLASSES

DATE: / /
PAGE NO.:

In r's complement

let $N = (\underbrace{edcba}_{\text{anything}}\overset{\text{Non zero}}{0}00)_r$

then r's comp. : $(r-1-e)(r-1-d)(r-1-c)(r-1-b)(r-a)000$

Ex: 10's comp. of $(8201129400)_{10}$
is 1798870600

Ex: 2 2's comp of $(110100\underset{\text{as it is}}{0})_2$
Comp. $(001011\underset{\text{as it is}}{0})_2$

Shortcut for 2's comp is put all no. till first 1 as it's & then complement other bits

⇒ Subtraction of unsigned no. using r's complement

Let M, N be unsigned no.
M - N.

Case 1 $M \geq N$

Ex: $82 - 64 = 82 + (99-64) + 1 - 100$
No borrow

$= 82 + 35 + 1 - 100$

$\Rightarrow 82 + \underset{\text{10's comp.}}{36} - 100$

$\Rightarrow 118 - 100$ (delete leading 1.

$\Rightarrow 18.$

So if $M \geq N$, then we discard end carry.

N
+ r's comp of N
result

Carry Discarded.

DATE : / /
PAGE NO. :

Case 2: $M < N$

Ex: $20 - 78 = 20 + (99-78) + 1 - 100$

$\Rightarrow 42 - 100$

$= -(100-42)$ → 10's comp. of 42.

$= -58.$

So no end carry in $M < N$

$$\begin{array}{r} M \\ -\ N \\ \hline \text{result} \end{array}$$

→ Correct answer will be $-$ (10's comp. of result).

Summary: M, N are unsigned no, r is base & we want $M-N$ using r's complement.

We find $M + (r\text{'s comp of } N)$

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

$= M - N + r^n$

Case 1: $M \geq N$ ↔ End Carry Exist

Discard it & you have answer.

Case 2: $M < N$ ↔ End carry don't exist

$-(r\text{'s comp. of } (M-N+r^n))$ & this is answer.

Ex: $239 - 82$

$\Rightarrow 239 + 918$ → 10's comp. of 82

$\Rightarrow 1157$

∴ End carry

$= 157$ Ans.

Ex2: $82 - 239$

$\Rightarrow 082 + 761$ → 10's comp of 239

$\Rightarrow 843$

∴ No end carry

$\Rightarrow -157$ → 10's comp of 843

GO CLASSES

DATE : / /
PAGE NO. :

Now in Binary

Ex:1

$$\begin{array}{r} 1111010 \\ -\ 1101011 \\ \hline \end{array} \Rightarrow \begin{array}{r} 1111010 \\ +\ 0010101 \\ \hline 10001111 \end{array}$$

2's comp of N

end carry → discard → ans. is 0001111 ✓

Ex:2

$$\begin{array}{r} 1101011 \\ 1111010 \\ \hline \end{array} \Rightarrow \begin{array}{r} 1101011 \\ 0000110 \\ \hline 1110001 \end{array}$$

2's comp

No end carry → 0001111 → − 0001111 ✓

⇒ **Subtraction of unsigned no. using (r−1)'s complement.**

Ex: 82 − 24 ⇒ 82 + (100−24) − 100

⇒ 82 + (99−24) + 1 − 100

⇒ 82 + (9's comp of 24) + 1 − 100

⇒ 82 + 75 + 1 − 100

⇒ 82 + 76 − 100

⇒ 158 − 100

⇒ 58

If m ≥ n, we have end carry, we add it (i.e add 1) & discard the carry.

If m < n, we don't have end carry, we just 9's comp it again.

Ex: 24 − 82 = 24 + (9's comp of 82) + 1 − 100

⇒ 24 + 17 + 1 − 100

⇒ (41) − 99

⇒ − (99 − 41) ⇒ − (9's comp. of 41) ✓

GO CLASSES

DATE : / /

PAGE NO. :

Summary: If M, N are unsigned no., r is base & we want M−N using (r−1)'s complement

we find M + (r−1's comp. of N)

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

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

Case 1: $M \geq N$ ⇔ End carry

↳ Add 1 (or End carry) & discard carry. Ans

Case 2: $M < N$ ⇔ No End carry

↳ −(r's comp. of $(M-N)+(r^n-1)$).

$-((r^n-1) - [(M-N)+(r^n-1)])$

$= -[(r^n-1) - (r^n-1) - (M-N)]$

$= -(-(M-N)) = M-N.$

Ex: 239 − 82

⇒ 239 + 917

⇒ 1156 (Add end carry

⇒ 156+1

⇒ 157

Ex:2 82 − 239

⇒ 82 + 760

⇒ 842 (No endcarry

⇒ −(157) ↳ r's comp of 842

(Basically for r's & (r−1)'s comp. do not memorise steps, try them with decimals by your own & make your own rules).

(Easy hai, looks hard).

DATE : / /

PAGE NO. :

Now in Binary (using 1's complement)

Ex1: 1101 − ~~0100~~ 0011

⇒ 1101 + 1100

1's comp. of 0100

⇒ 10001

end carry

⇒ 0001 + 1

⇒ 0010 ✓

Ex2 0011 − 1101

⇒ 0011 + 0010 → 1's comp. of 1101

⇒ 0101

No end carry

∴ −(1010)

1's comp. of 0101 ✓

★ r's & (r−1)'s complement for Radix Point.

Same method:

Ex:1 $(39.502)_{10}$

10's → 60.498

9's Complement → 60.497

Ex:2 $(10100.1011)_2$

2's → 01011.0101

1's comp → 01011.0100

⇒ Complement is needed bcz. "sub. with borrow" is hard & addition is easy (for humans) & in from h/w POV, we will be able to do all work by single h/w only i.e adder. (No subtraction diff. h/w req.).

GO CLASSES

DATE : / /
PAGE NO. :

★ Representations of Signed Binary Numbers : only for Binary.

Decimal = +199 , -199

↳ How to write it to D.C. using 0&1 only.

Signed Binary No.

- Sign-Magnitude System
- Sign. Complement System
  - 1's comp. System
  - 2's comp. system

GO CLASSES

a). **Signed Magnitude System:** Use extra bit to store sign
0 → "+" & 1 → "-".

For Ex: 0100 means +4 & 1100 means -4. in sign. mag. system.

9 in sign. mag. system with 4 bits only is NOT possible. It will be 01001.
↳ Mandatory.

$(111)_2$
Sign. Mag. → $(-3)_{10}$
Unsigned → $(7)_{10}$

→ 0 in Sign. mag. has two rep. 00 & 10.

DATE : / /

PAGE NO. :

b). 1's Complement No. System:

SM System ≡ 1's Comp. System ≡ 2's comp. System
↳ for $n \geq 0$.

for $n \leq 0$, in 1's comp. system (−N) is stored as 1's complement of N.

Ex: 9 in 1's comp. system 01001
−9 " " " " (01001)'s 1's comp = 10110.

Here also MSB is 1 for −ive no. always like SM system.

Que: Repr. −8 in 4 bits in 1's comp. system.
⇒ (1000)'s 1's comp = 0111 } Mistake.
↳ It is +7 in 1's comp. system.

⇒ Can't Rep. bcz
+8 = (01000) → Req. 5 bits.

Here 0 can be repr. as 00 or 11 (1's comp of +0).

c). 2's Complement Number System:

for $n \leq 0$, in this no. system (−N) is stored as 2's complement of N.

Ex 9 in 2's comp 01001
−9 " " " 10111

Here also MSB is 1 for −ive no. & 0 for +ive no.

GO CLASSES

1's comp. of 1's comp is actual no.
2's " " 2's " " " "

DATE: / /
PAGE NO.:

So, In all systems +ive integers have same binary, but for -ive no. they differ.

Qu: What do following no. repr:

a) $(01010)_2 \Rightarrow (10)_{10}$

b) $(11010)_2 \Rightarrow$ Depends on no. system used.

- → If unsigned : 26
- → If signed
  - → Sign mag. : -10
  - → 1's comp. no. system : -5
  - → 2's comp. no. system : -6

NOTE: 2's/1's comp. operation is ~~op~~ unary opⁿ on any no. & 2's/1's comp. no. system is a storage system to store signed numbers. Don't confuse b/w them.

Ex: 00101 $\xrightarrow{\text{1's comp}}$ 11010

what it repr in 2's no. sy. → +5

NOTE: 2's comp. no. system is weighted no. system (i.e. every position is associated some weight). like binary, decimal no. systems.

In 2's comp. no. system

$$\boxed{abcde = -(a*2^4) + (b*2^3) + (c*2^2) + (d*2^1) + (e*2^0)}$$

Ex: 10110 = $-2^4 + 2^2 + 2^1$ ⇒ -10

In 2's comp. no. system

DATE: / /
PAGE NO.:

So In 2's comp. no. System MSB has negative weight & all other bits have positive weight. (You know it).
Also in 2's comp. system extending MSB do not change it.

Ex: 0...0011 = 00011 = 000011 = 011 = 3.
101 = 1101 = 11101 = 111111...101 = -3
Don't Matter.

→ Sign-Mag. & 1's comp. No. System are non-weighted.

→ With 3 bits we should be able to repr. 8 diff. no. but with SM & 1's comp. no. sys. we are able to repr. only 7 diff. no. bcz '0' has got two representations. So 1 comb. is wasted in SM & 1's comp. no. system.

	Unsigned	Sign-Mag.	1's comp.	2's comp
0 0 0	0	0	0	0
0 0 1	1	1	1	1
0 1 0	2	2	2	2
0 1 1	3	3	3	3
1 0 0	4	-0	-3	-4
1 0 1	5	-1	-2	-3
1 1 0	6	-2	-1	-2
1 1 1	7	-3	-0	-1

Same in all

In sign-mag = 0000 = 1000 = (0)10.
In 1's comp = 0000 = 1111 = (0)10.

In 2's comp. NS every no. has unique representation.

DATE : / /

PAGE NO. :

★ Range of diff. no. systems:

a). **Unsigned** : for 3 bits : 0 to 7

for n bits = 0 to $2^n - 1$

b) **S.M.Ns** : for 3 bits : -3 to 3

for n bits : $-2^{n-1} - 1$ to $2^{n-1} - 1$

c). **1's comp N.s** : for 3 bits : -3 to 3

for n bits = $-2^{n-1} - 1$ to $2^{n-1} - 1$

d) **2's comp. N.s** : for 3 bits = -4 to 3

for n bits : -2^{n-1} to $2^{n-1} + 1$

GO CLASSES

	in sm.	in 1's comp	2's comp.	unsigned
Min.	1111111...11	100...00	100...00	00000...00
Max.	01111...11	01111...11	01111...11	11111...11

⇒ Now, we have already seen addition & subtraction in case of unsigned numbers.

DATE: / /

PAGE NO.:

Subtraction &

★ Additions of Signed Binary Numbers.

Q). No. in Signed Magnitude System: (Addition).

Similar to Decimal Addition we do

i.e. $(+x)+(+y) = +(x+y)$

$(-x)+(-y) = -(x+y)$

$(+x)+(-y) = -(y-x) \mid (x-y)$ } choose sign from greater mag. no.

if $|y|>|x|$; if $|x|>|y|$

Ex: $12+13=25$; $12-13=-1$; $13-12=1$; $-13-12=-25$.

Similarly, for Binary No.

Compare sign bit

- a1: both -ive, → add both and put -ive symbol
- a2: both +ive → add both & put +ive symbol
- a3: both diff → Compare Magnitude
  - → subtract bigger from lesser & use symbol of higher magnitude.

Ex: $\overset{(-3)}{1011} + \overset{(7)}{0111}$

diff symbol $111 > 011$

$$\begin{array}{r} 111 \\ -011 \\ \hline 100 \end{array} \rightarrow 4$$

⇒ 0100 result

Note that all i/p will be in same no. system & o/p will also be in same no. system

DATE : / /
PAGE NO. :

b) **Subtraction of Signed no. in SM system.**

$A - B \Rightarrow A + (-B)$

Ex: $\overset{-3}{1011} - \overset{7}{0111}$

$\Rightarrow 1011 + 1111$ (Change symbol)

$\Rightarrow$ Addition of no. in SM system

$\Rightarrow 1(011+111)$

Sign bit $\Rightarrow$ $\underset{-10}{11010}$

Cost: Sign bit comparison
If diff then mag. comparison & subtraction
If same then addition
$\Rightarrow$ $O(n)$ time & Addition + Subtraction hardware.

GO CLASSES

1). **Addition of sig. no. in 2's Complement System:**

Add. numbers normally & discard carry

Ex:1
$$\begin{array}{rl} 00101 & (5) \\ +\ 11100 & (-4) \\ \hline 100001 & \Rightarrow 0001 \quad (1) \end{array}$$
Discard

Ex:2
$$\begin{array}{rl} 11011 & (-5) \\ 11100 & (-4) \\ \hline 110111 & \Rightarrow 10111 \quad (-9) \end{array}$$
Discard

Ex:3
$$\begin{array}{rl} 00101 & (5) \\ 11010 & (-6) \\ \hline 11111 & \Rightarrow (-1) \end{array}$$
No carry

Basically i/p is 2 no. in 2's comp. no. system, add them & discard carry. O/p is sum in 2's comp. no. system.

DATE: / /

PAGE NO.:

Logic for 2's comp system $\vec{M}, \vec{N}$

$\Rightarrow (2^n - M) + (2^N - N)$

$= [2^{m'} - (M+N)] \rightarrow$ sum M+N in 2's comp.

Carry

d) Addition of 2 sig. no in 1's comp system: Add the no. and also add the carry. (if exist)

Ex: 00101 (5)
00100 (4)
01001 (9)

Ex2: 00101 (5)
11011 (-4)
10000
+1
0001 (1)

Que: Add -8 & 19 in

a). Sign mag. system

$\Rightarrow$ m = 101000
N = 010011
M+N = +(N-M)
= 01011

b) In 2's comp. system

$\Rightarrow$ m = 111000
N = 010011
M+N = 1001011
Discard
M+N = 1011

c). In 1's comp system

$\Rightarrow$ M = 110111
N = 010011
M+N = 100101
+1

M+N = 00101

d) Unsigned system

$\Rightarrow$ NON SENSE

GO CLASSES

DATE : / /
PAGE NO. :

e) Subtraction of two no. in 2's complement system:

If A & B are in 2's comp. system
then A-B = A + 2's comp. of B.

Discard the end carry.

Ex: 1 +4 : 00100 = A
+9 : 01001 = B
A-B = A + (B)'s 2's comp
= 00100
10111
11011 = -5

Ex: 2 -4 = 11100 = A
+9 = 01001 = B
A-B
= A + 2's comp. of B
11100
+ 10111
1 10011
Discard
= 0011 -13

f) Subtraction of two no. in 1's complement system:

If A & B are in 1's comp. system
then A-B = A + 1's comp of B
and wrap around the carry.

GO CLASSES

DATE : / /

PAGE NO. :

Ex:1 A = 11011 = -4

B = 11010 = -5

A-B = A + 1's comp. of B

= 11011
00101
10000
1
00001 ✓

wrap around carry

Ex:2 A = 00100 = 4

B = 11010 = -5

A-B = A + 1's comp of B

= 00100
00101
01001 ✓

→ So, in signed comp. system (1's & 2's both) addition & subtraction is done using single same circuit i.e. adder & also no. comp. required.

→ 2's complement system is better than 1's complement no. system for signed number bcz:

a. No need to wrap around carry.

b. No two rep. of 0.

c. Better range

★ Overflow for Addition operation: We store bits in registers in computer.

Size of registers is fixed. i.e. it can have n bits only.

When two n bits data is operated we may get result of n+1 or more bits of data, but register can only store n bits. So in this case MSBs OVERFLOW

DATE : / /
PAGE NO. :

Rocket Crashed → due to overflow.

Ex: we have 4 bits registers
we add two unsigned no. $10 \to (1010)_2$ & $13 \to (1101)_2$
result is 23 which need 5 bits to be stored.
which leads to overflow.

Overflow depends on:
- → Representation of Data (UNsigned, Signed, etc)
- → Operation (Addition, Sub, mul, etc)

NOTE: Only addition don't cause overflow, other operations can also do. BUT, in GATE syllabus we study overflow for addition only, in unsigned and signed (all 3 no. system) representations.

A) Overflow in case of unsigned addition: Overflow def. is same, just the detection condⁿ changes acc. to opⁿ (addition) & representation.

" If carry comes out of MSB " ⇒ Overflow
(MSB → Left Bit.)

Ex: $1 + 1 = 10$
overflow for 1 bit register.

Ex: 2 bits. (Range 0 to 3)
2+2 → OF
2+1 → NO OF
-3+-4 → Nonsense.
4+1 → Non sens
Already out of range

DATE : / /
PAGE NO. :

So in unsigned no. for let's say 4 bits

$A = a_3\ a_2\ a_1\ a_0$

$B = b_3\ b_2\ b_1\ b_0$

$r_4\ r_3\ r_2\ r_1\ r_0$

OF occurs iff $r_4 = 1$

B) Overflow in case of signed no. addition:

Imp: Overflow can NEVER occur if numbers have opposite sign (Obvious)

B1 Overflow in sign magnitude addition: If carry from magnitudes' sum.

Let

$A = S_A A_1 A_2 A_3 A_4$

$B = S_B B_1 B_2 B_3 B_4$

If $S_A \neq S_B$ OF can't happen.

If $S_A = S_B$, then OF occurs iff magnitudes of both gives carryout of MSB.

Ex:- 1110 + 0011 [−6 + (+3)]
↳ No OF as $S_A \neq S_B$

Ex2: 1110 + 1011 [−6 + (−3)]
↳ $S_A = S_B$

```
  110
  011
 1001
```

Carry out → Overflow

GO CLASSES

DATE : / /

PAGE NO. :

B2 **OF in 2's Compt. addition:** There is no overflow if:

a). Both operand has ~~same~~ diff MSB (i.e. diff signs), OR

b). Both operand are +ive & sum is +ive, OR.

c) " " " -ive " " " -ive.

For both 1's & 2's Complement sum

OF if:

$$\begin{array}{r} +A \\ +B \\ \hline -R \end{array} \quad \text{or} \quad \begin{array}{r} -A \\ -B \\ \hline +R \end{array}$$

Overflow

Ex: 0100 (4) +ive

+ 0100 (4) +ive

1000 (-8) -ive

So overflow

OF condition:

$$\boxed{a_{msb}\, b_{msb}\, \overline{S_{msb}} + \overline{a_{msb}}\, \overline{b_{msb}}\, S_{msb} = 1}$$

msb of sum

If True, then OF.

Ex:2

1101 (-3)

1100 (-4)

1 1001 (-7) No OF.

Discard

Ex:3

1011 (-5)

1010 (-6)

1 0101 (5) OF

Discard

We have to discard carry, its a part of addition.

DATE: / /

PAGE NO.:

Always check OF in final result i.e. discard Cout in 2's comp & wrap around Cout in 1's comp.

B3 OF in 1's Comp. addition:

→ Same three conditions of No overflow

Ex:1

0101	(5)	NO OF
1001	(-6)	↳ Also diff sign
1110	(-1)	

Ex:2

1010	(-5)	NO OF
0110	(+6)	
10000		
↳ 1		
0001	(1)	

Ex:3

1010	(-5)	OF
1011	(-6)	
10011		
↳ 1		
0100	(4)	

Here also,

$$\overline{a_{msb}}\ \overline{b_{msb}}\ C_{out} + a_{msb}\ b_{msb}\ \overline{C_{out}} = 1$$

↳ OF

An alternative method for detecting overflow in 2's comp addition is as: "An OF occurs iff cout of the MSB is not equal to Cin of MSB".

Bcz Consider only MSBs

					Cout	Cin	Cout	Cin	Cout	Cin	Cout	Ci
	1	0	1	1	1	0	1	1	0	0	0	0
	0	1	1	1	1	1	1	1	0	0	0	0
			0	1	0	1	1	1	0	0	0	1
	OF can't occur		OF		NoOF		NoOF		OF			

So OF iff Cin & Cout are diff

So OF iff $C_{in} \oplus C_{out} = 1$ in 2's comp. only

GO CLASSES

DATE : / /
PAGE NO. :

★ **Sign Extension:** Extra bits to fill when copying content of n bit register to >n bit register. So that value remains same

a). **For unsigned numbers:** Copy zeroes

$$1101 = 01101 = \underbrace{00\ldots0}_{\text{any zeroes}}1101$$

So $\underbrace{a_n a_{n-1} \ldots a_1}_{n \text{ bits}} \xrightarrow{\text{Copy}} \underbrace{\overbrace{00\ldots0}^{x} a_n a_{n-1} \ldots a_1}_{n+x \text{ bits}}$

b). **For signed no. in sign magnitude repr:**

$$01001 = 001001 = 00\ldots01011$$
$$11001 = 1001001 = 1\underbrace{0\ldots0}_{\text{any zeroes}}1001$$

So. $\underbrace{a_n a_{n-1} \ldots a_1}_{n \text{ bits}} \xrightarrow{\text{Copy}} \underbrace{a_n \overbrace{00\ldots0}^{x} a_{n-1} a_{n-2} \ldots a_1}_{n+x \text{ bits}}$

c) **For signed no. in ~~sign mo~~ 1's Comp. repr.:** Copy MSB

$$0110 = 00110 = 0\ldots0110$$
$$10001 = 110001 = 1\ldots10001$$

So $\underbrace{a_n a_{n-1} \ldots a_1}_{n \text{ bits}} \xrightarrow{\text{Copy}} \underbrace{a_n \overbrace{a_n a_n a_n \ldots a_n}^{x} a_{n-1} \ldots a_1}_{n+x \text{ bits}}$

GO CLASSES

DATE : / /
PAGE NO. :

d). For signed no. in 2's comp. repr.: Copy MSB

0011 = 00--011

1011 = 11--1011

So, $\underbrace{a_n a_{n-1} \ldots a_1}_{n \text{ bits}} \xrightarrow{\text{Copy}} \underbrace{\overbrace{a_n a_n \ldots a_n}^{n \text{ bits}} a_{n-1} \ldots a_1}_{n+n \text{ bits}}$

★★ Binary Codes: Code in general is a system of rules to convert information - such as letter, word, image or gesture into another repr. sometimes shorter or secret or for making it suitable to travel on channel, etc.

Binary codes are pattern of 0's & 1's only. i.e. code containing 0 & 1 only.

Ex: aabc can be written as 00 00 01 10

For n diff elements we need atleast $\lceil \log_2 n \rceil$ bits. i.e. k bits can give 2^k diff elements.

There are diff binary codes, each having its own advantages.

Binary codes are imp. to store data in computer/ckts.

We study binary codes despite having binary no. (which is also a binary code) bcz, we can't repr. letters, symbols in binary no., we need some coding. Also have many application like error control, enc., etc.

GO CLASSES

Ascii of 'A' is 65 – 90
" of 'a' is 97 – 122
" of '0' is 48 – 57

DATE: / /
PAGE NO.:

Binary codes for decimal digits are:

Decimal Digit	2-out-of-5 code	Excess 3 code	8421 Binary Coded Decimal (BCD)	2421 Code.	84-2-1 Code
0	00011	0011	0000	0000	0000
1	00101	0100	0001	0001	0111
2	00110	0101	0010	1000, 0010	0110
3	01001	0110	0011	1001, 0011	0101
4	01010	0111	0100	1010, 0100 (1's comp)	0100 (1's comp)
5	01100	1000	0101	1011	1011
6	10001	1001	0110	1100	1010
7	10010	1010	0111	1101	1001
8	10100	1011	1000	1110	1000
9	11000	1100	1001	1111	1111
	Provides 1 bit error detection i.e. 2 Hamming dist.	SCC code (BCD+3)	Binary of each digit (unassigned digit also)	SCC. Code & weighted	SCC & weighted code.

49 in BCD is 0100 1001 & in binary is 110001

→ Unused comb. of BCD are referred invalid.

→ BCD No. are Decimal no. represented using bits only. It is not binary number.

→ BCD are also known as 8421 code. i.e. weight at every position. Ex: 0101 = 4+1 = 5

→ In 2421 Code. Some no. like 2 has 2 possible encodings i.e. 1000 & 0010, but we prefer 0010 i.e. will low decimal eq. is treated as binary till 4 & after that use self complementing property.

DATE : / /

PAGE NO. :

→ **Self Complementary Code**: when in decimal x & 9-x has codes y in and 1's comp. of y. those codes are SCC.

Ex: 2421 code
↳ 2 is 0010
9-2 i.e. 7 is 1101

BCD codes are not SCC.
↳ 2 is 0010
9-2=7 is not 1101

→ Weighted codes has weight at every position.

→ Excess 3 code are also SCC. but unweighted.

$(5)_{10} = (0101)_2 \xrightarrow{+3} 1000$

$(9)_{10} = (1001)_2 \xrightarrow{+3} 1100$

→ BCD (8421), 2421, 84-2-1, Excess 3, 2-out of 5, these are all per decimal digit codes. and each has its own advantage
↳ bcz. for each digit → fix code

→ If a code is SCC, then sum of weight is 9.

SCC → sum of weight 9.

★ **Gray Code**: It has many applications such as counter but here with each next count only 1 bit changes.

Ex: 2 → 3
10 → 11
↳ So only need to flip one bit.

This is main property of gray code.

GO CLASSES

DATE : / /
PAGE NO. :

→ Gray codes are not per digit decimal codes.

→ Gray Codes ⇌ Reflected Binary Codes (RBC) ⇌ Reflected Binary (RB)
diff to this

★ Rec. Construction of Gray codes

Let G_n: Gray code with n bits

G_1:	G_2:	G_3:	Decimal
0	0 0	0 0 0	0
1	0 1	0 0 1	1
	1 1	0 1 1	2
	1 0	0 1 0	3
		1 1 0	4
		1 1 1	5
		1 0 1	6
		1 0 0	7

Reflect. Reflect

Here for G_n write G_{n-1}, and then reflection of G_{n-1} then add 0 in starting G_{n-1} and 1 at reflected G_{n-1}.

But recursive method might be time consuming. So we have efficient method to convert decimal/binary to gray & vice versa.

⇒ Binary No. to Gray Code.

Let binary m: $b_3 \quad b_2 \quad b_1 \quad b_0$

Gray Code of m: $b_3 \quad b_3 \oplus b_2 \quad b_2 \oplus b_1 \quad b_1 \oplus b_0$

DATE : / /

PAGE NO. :

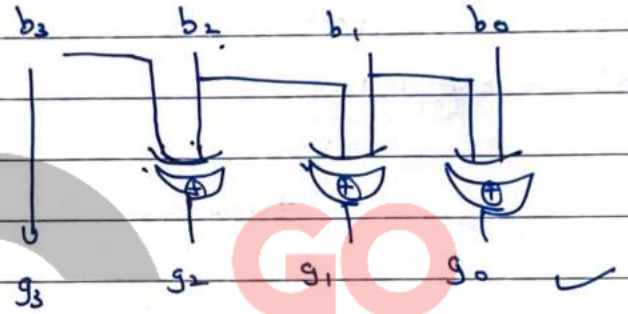
Ex: $m = 29 = (11101)_2$

$= 1 \quad 1 \oplus 1 \quad 1 \oplus 1 \quad 1 \oplus 0 \quad 0 \oplus 1$

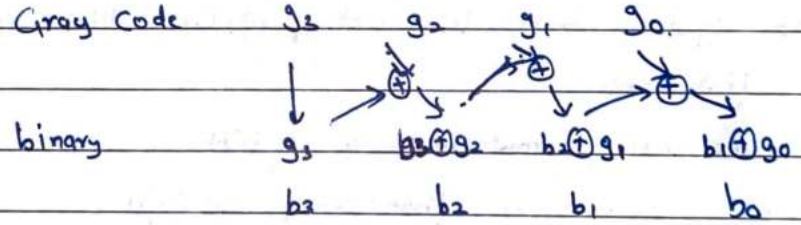
$= 10011$

↳ Gray code of 29.

So.



⇒ Gray Code to Binary No.:



Ex: Gray code is 10011

Binary is $1 \quad 1 \oplus 0 \quad 1 \oplus 0 \oplus 0 \quad 1 \oplus 0 \oplus 0 \oplus 1 \quad 1 \oplus 0 \oplus 0 \oplus 1 \oplus 1$

$= 11101$

↳ 29.

Now we know Decimal ⇄ Gray.

→ Gray of m & Gray of m+1 has only 1 bit changed.

→ No code is wasted here (all per. are used).

GO CLASSES

★ Another way to find Gray Codes:

n^{th} gray code is $n \oplus \left\lfloor \frac{n}{2} \right\rfloor$

bitwise XOR.

Ex: Gray code of 29

$m = 29 = 11101$

$\left\lfloor \frac{m}{2} \right\rfloor = 14 = \oplus\ 01110$

10011 → Gray code of 29.

GO CLASSES

★ Properties of Gray Codes:

→ Hamming distance is 1 i.e. Gray code differ by 1 bit only

→ Code is cyclic bcz. last entry of G_n differ by 1 bit from first entry.

→ G_n uses all permutations of n bits

→ G_n is same as first half of G_{n+1}

DATE : / /

PAGE NO. :

☆ Hazards in DL

→ whenever unwanted (or undesirable) transitions in the O/P signal of digital circuit, then we call them hazard.

For Ex: Glitches we saw in async. counter

Hazard.

- Static
  - Static 1 → SOP
  - Static 0 → POS
- Dynamic. (in multilevel ckt) → Complex to resolve, Value goes in transient state before settling
- Essential (in seq. ckt) → in async seq-circuit (like we saw in Counters)

☆☆ Static Hazard: Those hazard when signal level should have been constant BUT it changes for a small amount of time

For Ex: If signal is 1, it goes from 1 to 0 for small time, (due to static hazard). or vice versa.

NOTE In any ckt either both static-1 & static-0 hazard will be there or none. (Any one any not possible).

GO CLASSES

DATE : / /
PAGE NO. :

* Static-1 Hazard: It comes in SOP terms.

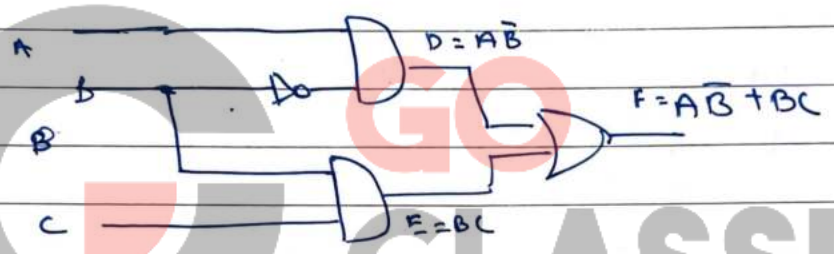
for F_1.

$F(A,B,C) = \Sigma(3,4,5,7)$

A \ BC	00	01	11	10
0	0	1	1	0
1	1	1	1	0

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

⇓ Circuit



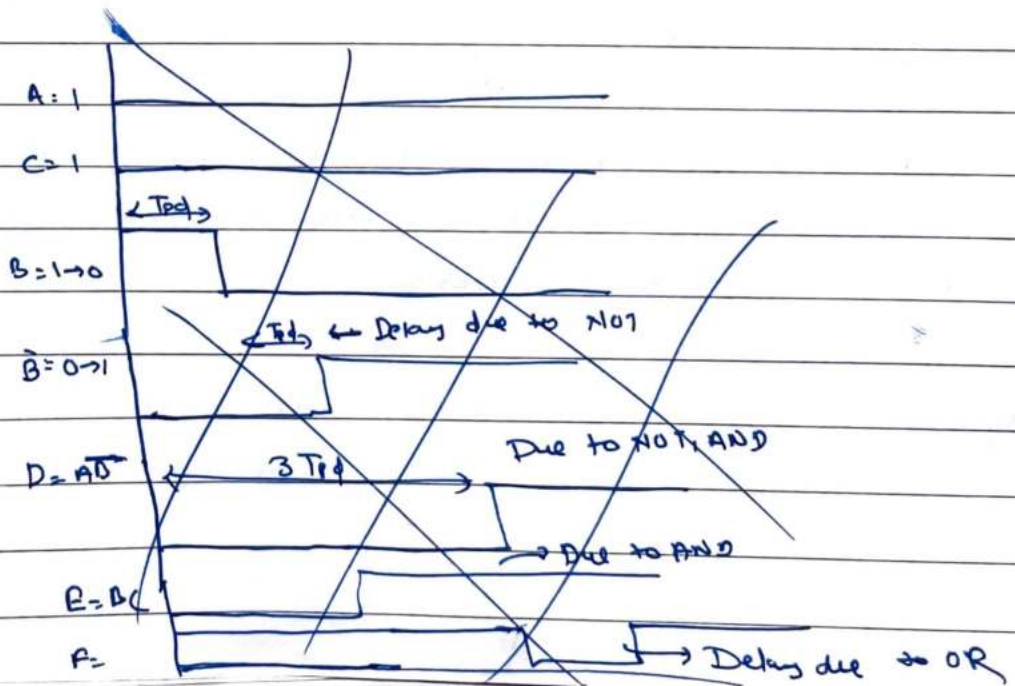
Assume Tpd of each gate is same.

initially if $A = C = 1$, then $B \to 1$ to 0 then

Case 1: $A = B = C = 1$, $F = 1$

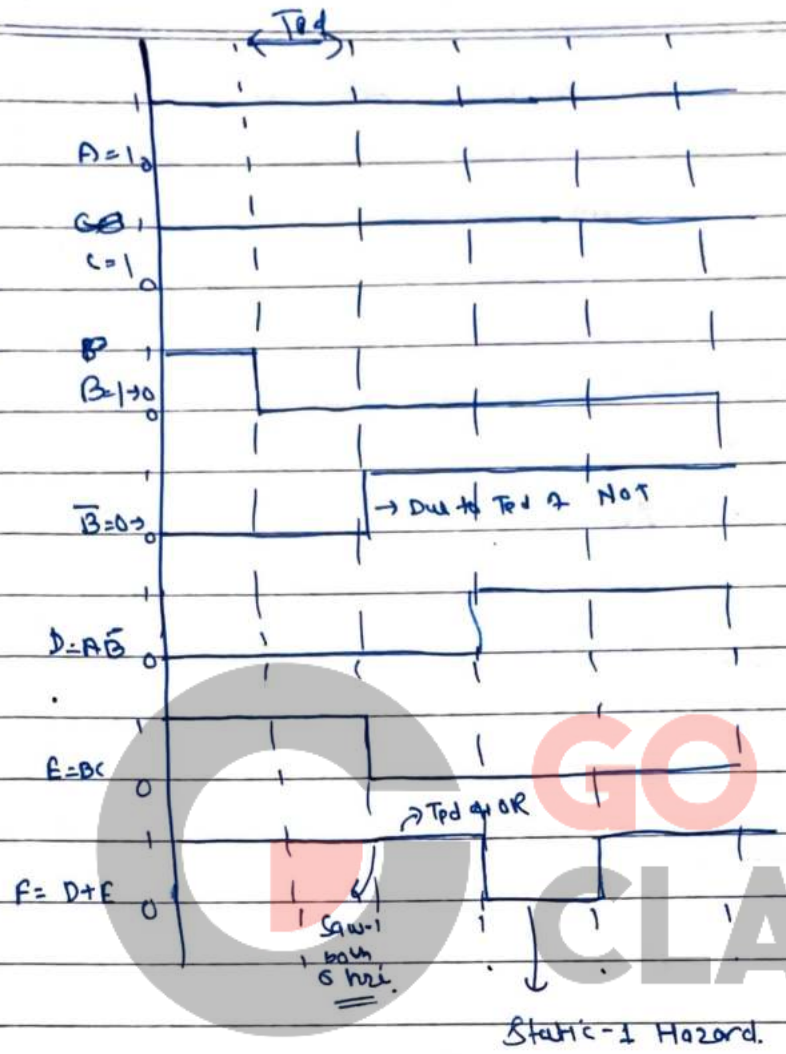
Case 2: $A = C = 1$ & $B = 0$ then $F = 1$

So, in both case O/p is 1, but See Timing Diagram:



DATE : / /

PAGE NO. :



GO CLASSES

* **Static 0 Hazard:** Similarly static 0 Hazard occurs in POS terms

* **Elimination of static hazard:**

In prev. ex:

A \ BC				
	0	0	1	0
	1	1	1	0

$f = A\bar{B} + BC + AC$ → Add this term also.

So we have to add redundant term to remove static hazard.

Similar Consensus Theorem

DATE : / /

PAGE NO. :

→ If there exist a pair of 1's which are not in same cube, it means there is static hazard & each such pair is static 1 hazard.

Que: Adding which term in below circuit will remove static-1 hazard?

$F = X\bar{Y} + YZ$

⇒ Ans = XZ

→ Gate CSE 2006 PYQ. ←

GO CLASSES

3in DLD 2024

Module-3 Combinational Circuits.

PAGE NO. :
DATE : / /

★ Compact Truth table representation.

We know that:

a	b	f(a,b)
0	0	x
0	1	y
1	0	z
1	1	w

and

$f = x\bar{a}\bar{b} + y\bar{a}b + z a\bar{b} + wab$

Now, from this, we can easily create CTT given expression.

Ex: $f = \bar{a}b + ca\bar{b} + \bar{c}ab$

then C.T.T. is

a	b	f(a,b,c)
0	0	0
0	1	1
1	0	c
1	1	$\bar{c}$

This is C.T.T, std. TT will have 8 rows as f(a,b,c).

→ CTT is just another representation of the function.

We can easily convert std. T.T. to compact truth table as:

Ex:

a	b	c	f(a,b,c)
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

← Std TT. →

a	b	f(a,b,c)
0	0	$\bar{c}$
0	1	1
1	0	$\bar{c}$
1	1	0

CTT

OR

a	c	f(a,b,c)
0	0	1
0	1	b
1	0	$\bar{b}$
1	1	0

CTT2

PAGE NO. :
DATE : / /

Given Exp → CTT ✓
↳ STT ✓

Given CTT → Exp. ✓
↳ STT. ✓

Given STT → Exp ✓
↳ STT ✓

You know all

Table can be made more compact.

For Ex: for prev. ex only

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

OR

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

or

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

GO CLASSES

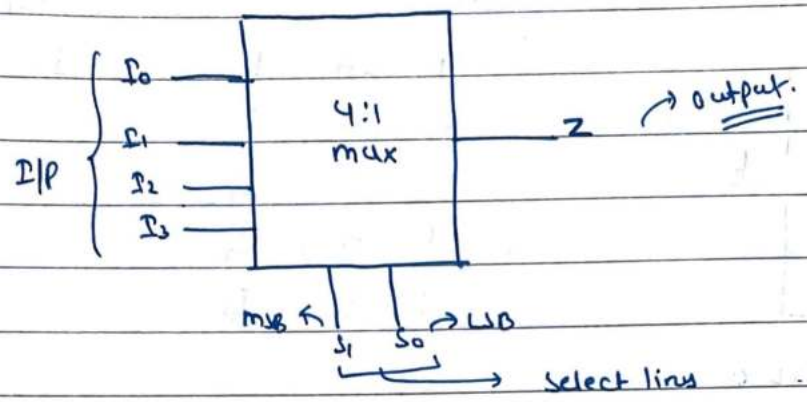
Note that exp. written by any above table will be same.

→ CTT is used to represent mank circuits like mux, encoder, etc

PAGE NO. :
DATE : / /

☆ Multiplexer: $n \times 1$ circuit with $\log_2 n$ select lines ≡ bridge

Ex:

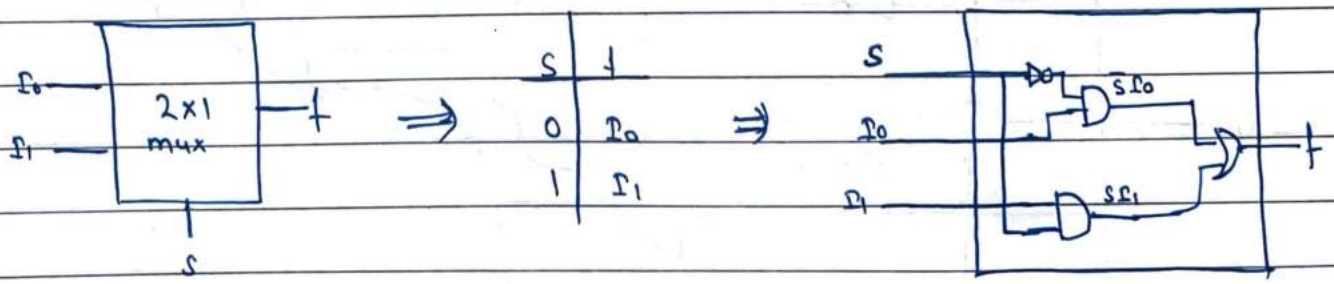


Z will be I_2 if S_0 is 0 & S_1 is 1 ✓

mux (≡ multiplexer) is a data selector i.e. selects one i/p on the basis of select lines.

→ $2^n \times 1$ mux has n select lines.

→ It is the fixed repr. of mux to be $2^n \times 1$ not $2^n \times 2$ b/c, it is def. like this.



Block Diagram

Truth table
⇓
Ex:
$\bar{S}I_0 + SI_1$

Actual Circuit.

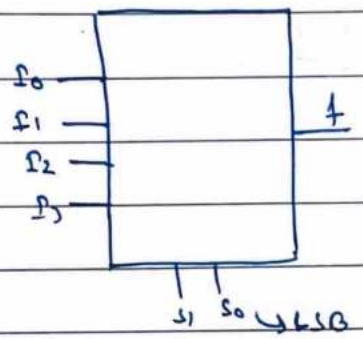
Std. TT. will have 4 rows.

GO CLASSES

PAGE NO. :
DATE : / /

Similarly, for 4x1 mux :

Block Diagram:



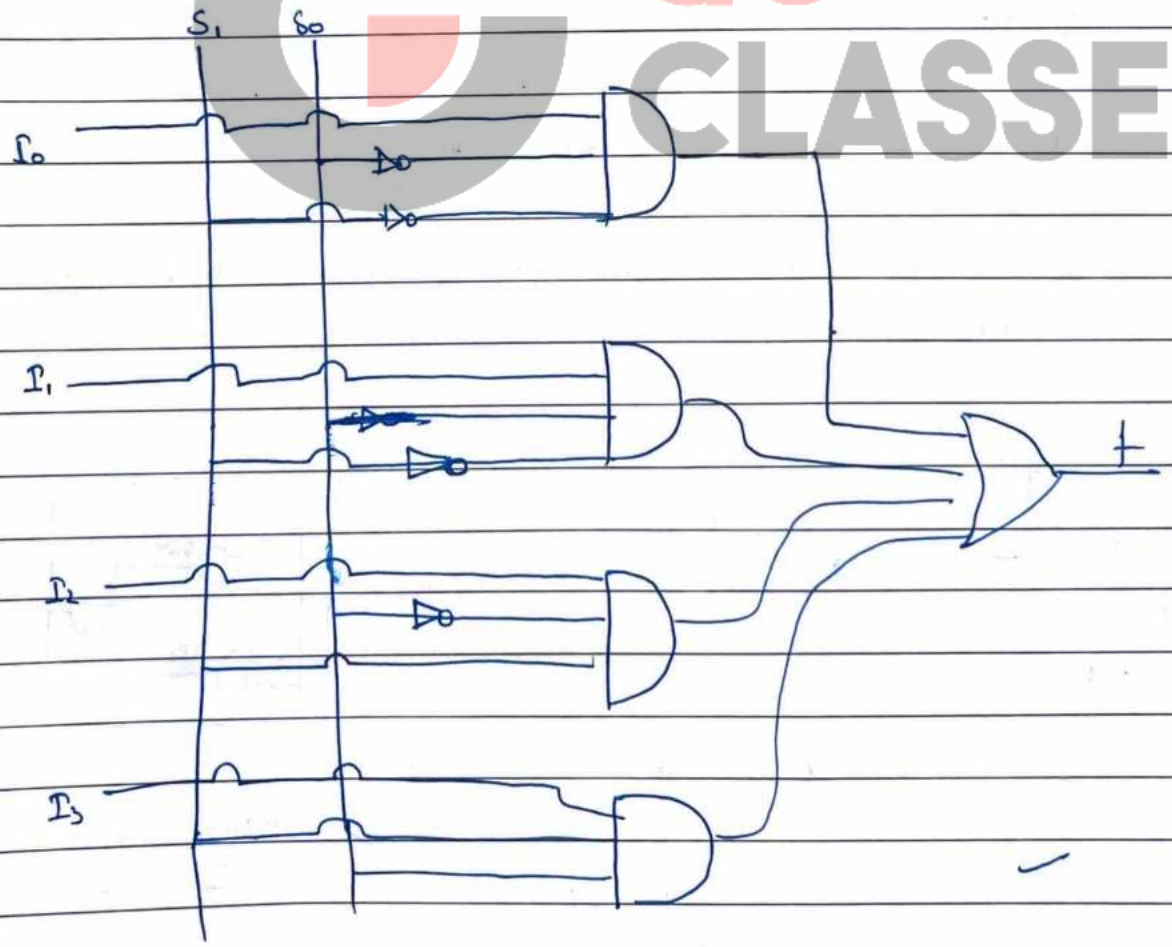
C.T.T

S_1	S_0	f
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Exp.

$f = \bar{S_0}\bar{S_1}I_0 + \bar{S_0}S_1 I_2 + S_0\bar{S_1}I_1 + S_0 S_1 I_3$

Actual Circuit:



If $S_0 = 1$, $S_1 = 0$ then I_1 will be at f.

GO CLASSES

PAGE NO. :
DATE : / /

Generally,

$$f = \sum_{k=0}^{2^n-1} m_k I_k$$

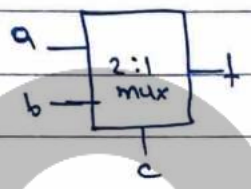
where m_k is k^{th} minterm.

& I_k is k^{th} input.

So k^{th} i/p will be selected as o/p iff k^{th} minterm is True.

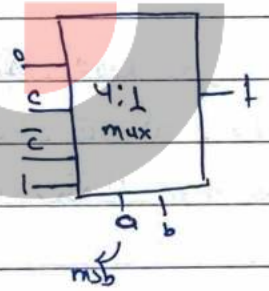
⇒ **Realisation / Implementation of functions using mux:**

Ex:



$f = \bar{c}a + cb.$

Ex:2



$f(a,b,c) = \bar{a}\bar{b}0 + \bar{a}bc + a\bar{b}\bar{c} + ab$

$= bc + a\bar{c}$

↳ (on simplifying)

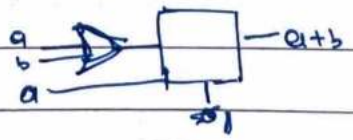
This is how we can easily get which function is implemented on seeing the mux.

mux → fun

Now we will see fun → mux :

⇒ **OR using mux:**

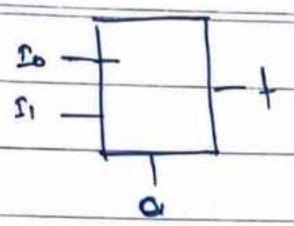
Attempt-0



No sense as using OR gate.

GO CLASSES

PAGE NO. :
DATE : / /



we want $f = a+b$.

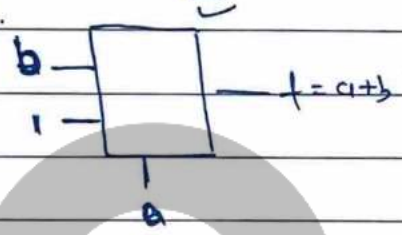
f is $\bar{a} I_0 + a I_1$

If we take I_1 as 1 & I_0 as b then,

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

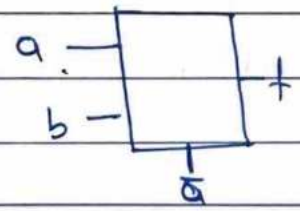
$= a+b$.

So.



How to guess this? best way is get SOP of function you want to implement i.e $a+b$ here = $ab + \bar{a}b + a\bar{b}$, now for $\bar{a}$ we have b and for a we have $(b+\bar{b})$ i.e 1.

Many other ways to implement:

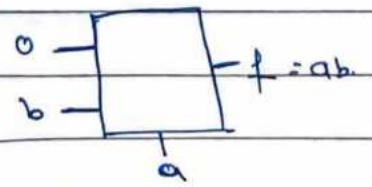


$f = \bar{a}a + \bar{a}b = \bar{a} + \bar{a}b \Rightarrow a+b$

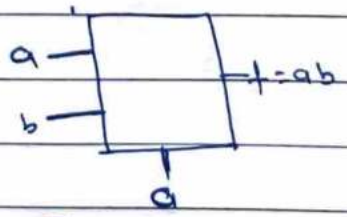
but it's not ideal as an extra inverter is req. to input $\bar{a}$.

⇒ AND gate using mux:

$f = ab$ ↳ already in SOP so 0 for $\bar{a}$ & b for a



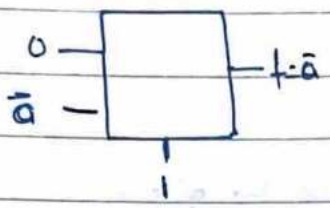
OR



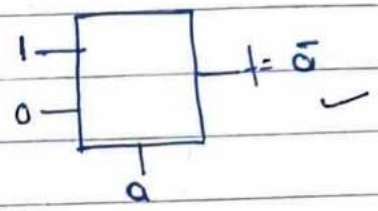
$\Rightarrow a\bar{a} + ab = ab$.

PAGE NO. :
DATE : / /

⇒ Inverter using mux:



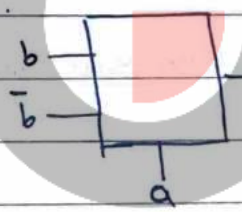
↳ Not a good ckt. as already req. inv



⇒ XOR using mux:

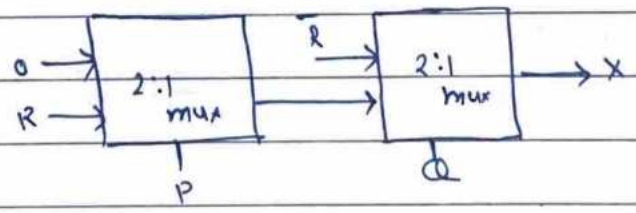
$f = a\bar{b} + \bar{a}b$

so.



$f = a\bar{b} + \bar{a}b = a \oplus b.$

Que: What's the expression of X?



$x = \bar{Q}\bar{R} + Q(PR)$

$= \bar{Q}\bar{R} + PQR.$

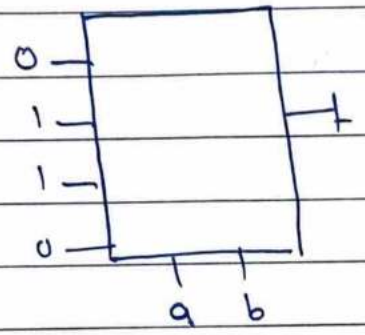
GO CLASSES

Karan

PAGE NO :
DATE : / /

Que: Implement $a \oplus b$ using 4:1 mux:

$\Rightarrow$ $f(a,b) = a\bar{b} + \bar{a}b$



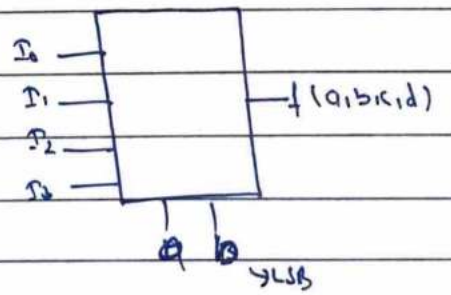
Directly can be given by truth table easily.

→ If asked with 8x1 mux, put one select line (MS) as 0 and last 4 i/p as don't cares.

$f = a \oplus b \oplus c$ is one if exact odd ones are there i.e. 1 or 3

So $f = \bar{a}\bar{b}c + \bar{a}b\bar{c} + \bar{a}b\bar{c} + abc$.

Que: Implement $f(a,b,c,d) = ad + bc + \overline{a+b+c}$ by following mux:



$f = ad + bc + \bar{a}\bar{b}\bar{c}$ (method-1)

when $a=0, b=0$, $I_0 = \bar{c}$

when $a=1, b=0$, $I_2 = d$

when $a=0, b=1$, $I_1 = c$

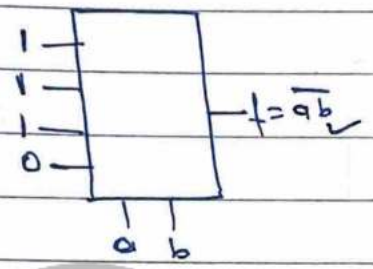
when $a=1, b=1$, $I_3 = d+c$

OR can do with CSOP also: (Method-2)

PAGE NO. :
DATE : / /

NOTE Single $2^n:1$ mux is enough (w/o any other gate) for any n-variable function as seen in XOR ex. on prev. page. We just write T.T. as i/p.

Ex: for $f(a,b) = \overline{a}b$, only 4:1 mux is needed



put var on select line & truth value on input lines.

→ Single $2^{n-1}:1$ mux is not always enough for n-var function.

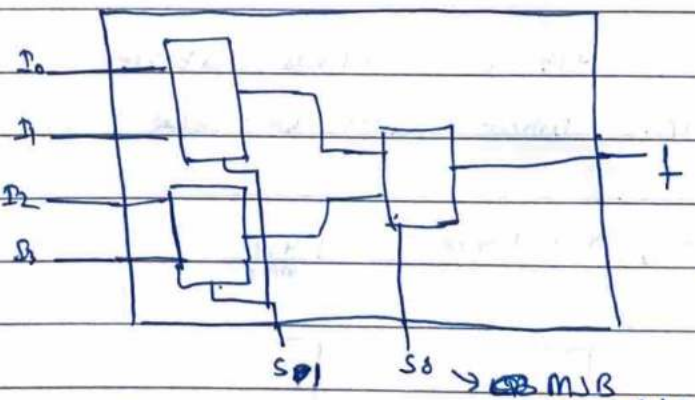
Ex: Can't implement $X \oplus Y$ by 2:1 mux w/o inverter.

However some n var. fun. can be implemented but not all.

NOTE Single $2^{n-1}:1$ mux + Single Inverter are enough for any n-variable function. (Think of Compact truth table).

☆ Building larger mux from smaller mux:

→ 4x1 mux using 2x1 mux only



→ MSB !!!

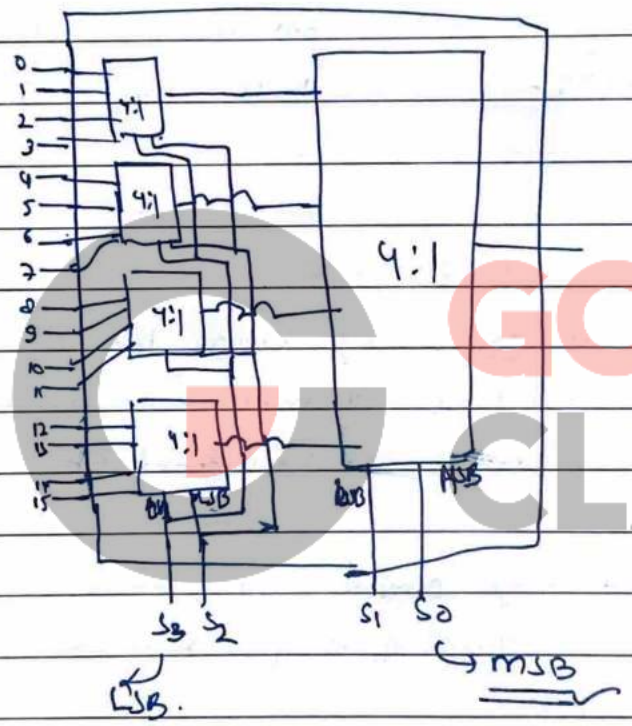
→ LSB will not work

PAGE NO :
DATE : / /

Check all settings like So MSB is req. So LSB will not work.

→ 8x1 mux can be derived from 7 (4+2+1) 2x1 muxs.

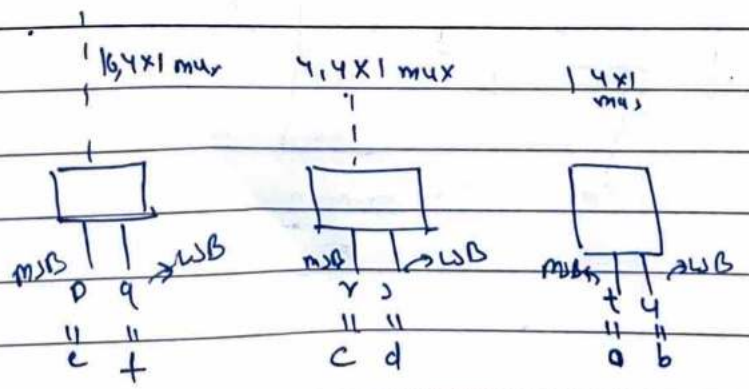
→ 16x1 mux from 4x1 muxs



order will be
S1 S0 S3 S2
MSB LSB

How to choose select lines? Just try what you get in diff settings & what were you expecting.

Qu: In 64:1 mux with 6 select lines abcdef with a as msb and f as LSB. What will be value of p q r s t u below:



(work on this)

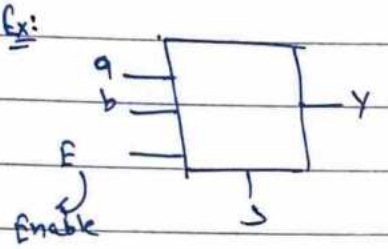
GO CLASSES

PAGE NO :
DATE : / /

⇒ Multiplexer with Enable Input:

Enable i/p is present in every comb ckt. It's like on/off switch, Circuit works if enable is 1, else ckt will not work.

Ex:



T.T.

E	S	Y	
0	X	0	→ mux is idle, if E=0
1	0	a	→ mux working normally
1	1	b	

↳ Compact T.T.
Std. will have 2^4 rows.
as Y is f(a, b, E, S)

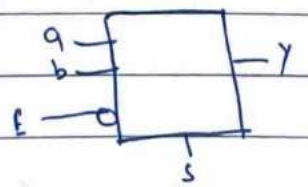
Now, in every exp. we calculate we can mul. with E so that it work only if E is 1.

Here $Y = E(\bar{S}a + Sb) = E\bar{S}a + ESb$

There are two types of enable:

→ Active low enable: Circuit work if E=0, and if E=1 circuit is idle.

→ Active high enable: Circuit work if E=1, (Circuit is active when enable is high), and if E=0 circuit is idle. Explained this above. Default



mux with active low enable.

→ Easily can add one more line in circuit to implement enable. When CPU don't want ckt to work always, it only needs when enable is 1.

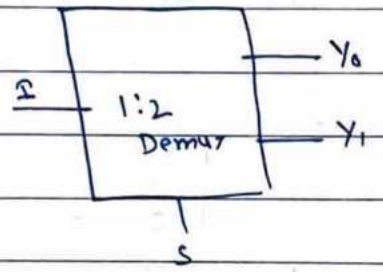
PAGE NO. :
DATE : / /

☆ Demultiplexer (Demux): Another combinational circuit.

Opposite of mux. Not much imp.

1 i/p and 2^n o/p lines.

Ex:

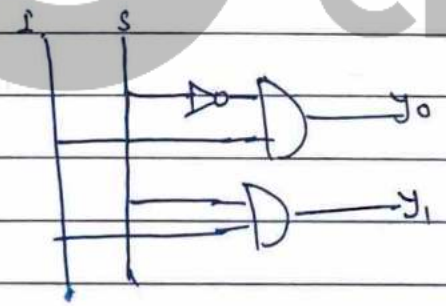


T.T:

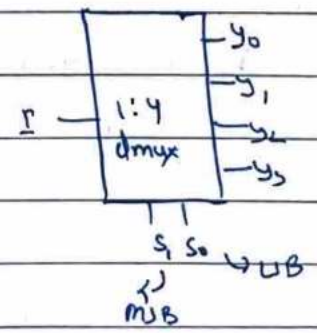
S	Y_0	Y_1
0	I	0
1	0	I

Exp: $Y_0 = \bar{S} I$, and $Y_1 = S I$

Imp:



Similarly:



S_1	S_0	y_0	y_1	y_2	y_3
0	0	I	0	0	0
0	1	0	I	0	0
1	0	0	0	I	0
1	1	0	0	0	I

↳ CTT.

$y_0 = \bar{S}_1 \bar{S}_0 I$, $y_1 = S_0 \bar{S}_1 I$, $y_2 = \bar{S}_0 S_1 I$, $y_3 = S_0 S_1 I$

GO CLASSES

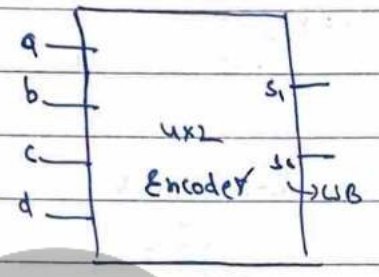
PAGE NO. :
DATE : / /

★ **Encoder:** Encoding means giving Codes. So basically, encoder does encoding. i.e. binary encoding.

Gives code to 2^n peoples using n bits.

It is of size $2^n : n$ and no concept of select lines.

Ex:



$2^n \rightarrow$ I/P lines.

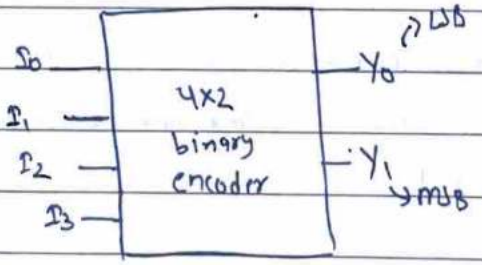
$n \rightarrow$ O/P

When d will be 1 then S_1 & $S_0 = 1$

c " " 1 " S_1 & $S_0 = 0$

There are two types of encoders:

1). **Binary Encoders:** Here, we allow only one i/p to be 1 at a time i.e. a=1 & b=1 is not possible.



Now, truth table is

I_0	I_1	I_2	I_3	Y_1	Y_0 (LSB)
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1
Remaining comb. can't occur	X	X	X	X	X X

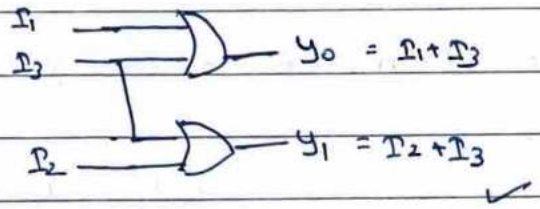
GO CLASSES

PAGE NO. :
DATE : / /

One hot encoded → exactly #1 bit is

$Y_1 = I_2 + I_3$, $Y_0 = I_1 + I_3$

So, actual imp:



K-map for Y_0 :

I_1I_0 \ I_3I_2				
	X	0	X	1
	0	X	X	X
	X	X	X	X
	1	X	X	X

$y_0 = I_3 + I_1$

$y_0 = \overline{I_2}\,\overline{I_0}$

k-map for y_1

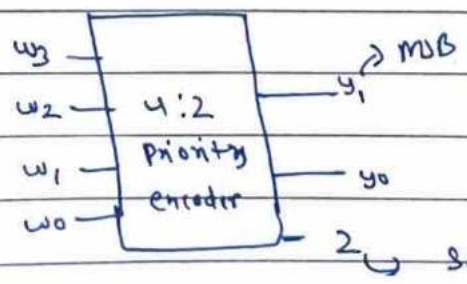
I_1I_0 \ I_3I_2				
	X	1	X	1
	0	X	X	X
	X	X	X	X
	0	X	X	X

$y_1 = I_2 + I_3$

$y_1 = \overline{I_0}\,\overline{I_1}$

→ An useful example of encoder is Octal to binary converter (8X3 encoder), i/p is given in one-hot encoded form (i.e. out of 8 bits 1 bit is 1.).

b) Priority Encoder: Here if more priority i/p is 1, then rest do not matter.



Some one need activation.

let w_0 = lowest priority & w_3 = highest " .

GO CLASSES

PAGE NO.:
DATE: / /

Here we have an extra o/p which is 0 only when all inputs are 0.

(x → don't care)

T.T.

w_3	w_2	w_1	w_0	y_1	y_0	z
1	x	x	x	1	1	1
0	1	x	x	1	0	1
0	0	1	x	0	1	1
0	0	0	1	0	0	1
0	0	0	0	x	x	0

$y_1 = w_3 + w_2$, $y_0 = w_3 + \overline{w_2} w_1$

→ Here z is not enable i/p. It is o/p which is 1 when someone need attention or y_0 & y_1 can be zero when no need attention or noone need attention. So to dist. this we have z.

$z = w_3 + w_2 + w_1 + w_0$. ✓

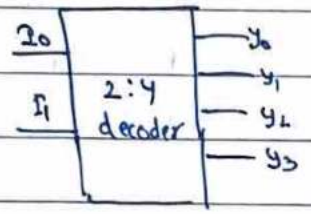
↳ 1 if anyone attent is 1.

PAGE NO. :
DATE : / /

Imp. mux & decoder

★ **Decoders:** If we are encoding we also need to decode. It has n inputs and 2^n outputs. Exactly one of 2^n o/p is True.

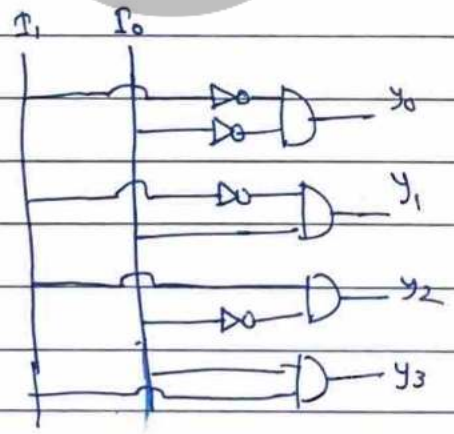
Ex:



T.T 2

I_1	I_0	Y_0	Y_1	Y_2	Y_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

$y_0 = \bar{I_0}\bar{I_1}$, $Y_1 = \bar{I_1} I_0$, $Y_2 = I_1 \bar{I_0}$, $y_3 = I_1 I_0$



Decoder ~~will~~ can also have enable i/p (active low or active high)

→ Decoder is analogus to Receptionist who tells you path to dept, once you tell dept. code.

GO CLASSES

PAGE NO :
DATE : / /

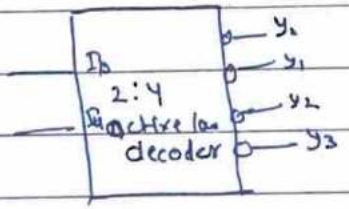
→ Practical ex. of decoder is binary to octal encoder.
i/p is 3 digit binary o/p is particular decimal as one hot encoding.

⇒ **Active low decoder:** Default decoder was active high decoder where exact one o/p was one.
In active low decoder exactly one o/p will be 0 and all other are 1.

T.T.

I_1	I_0 (→LSB)	Y_0	Y_1	Y_2	Y_3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

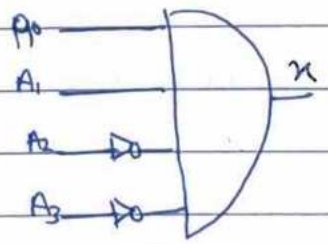
It can be implemented using NAND gates
It is repr. as:



NAND decoder ⇌ Active low dec.

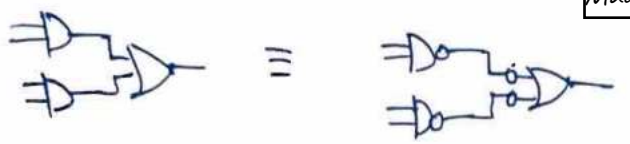
Que: Which binary no. is detected by following circuits give A_0 is LSB & A_3 is MSB:

a)



⇒ no. ↓
0011
↳ bcz. for this x=1

GO CLASSES

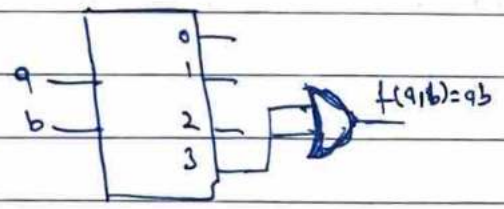


DATE: / /

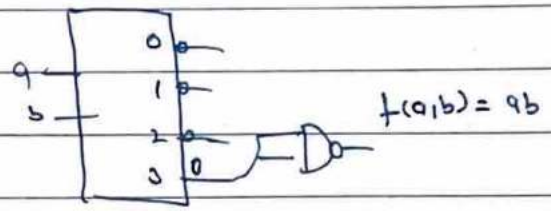
⇒ **Function realization using decoder:**

Ex: $f(a,b) = ab$ using

a) active high decoder

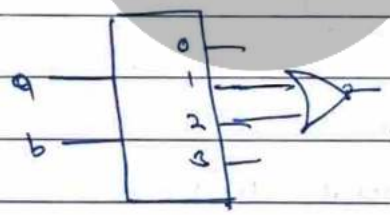


b) active low decoder



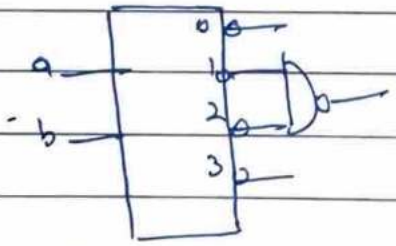
Remember to use always OR with active high decoder i.e. AND decoder and use NAND with active low i.e. NAND decoder.

Que: Implement $f(a,b) = a \oplus b$ using a) active high decoder



$f = a\bar{b} + b\bar{a} = a \oplus b.$

b) active low decoder



$f = \overline{\overline{a\bar{b}} \cdot \overline{b\bar{a}}} \Rightarrow a\bar{b} + b\bar{a} = a \oplus b$

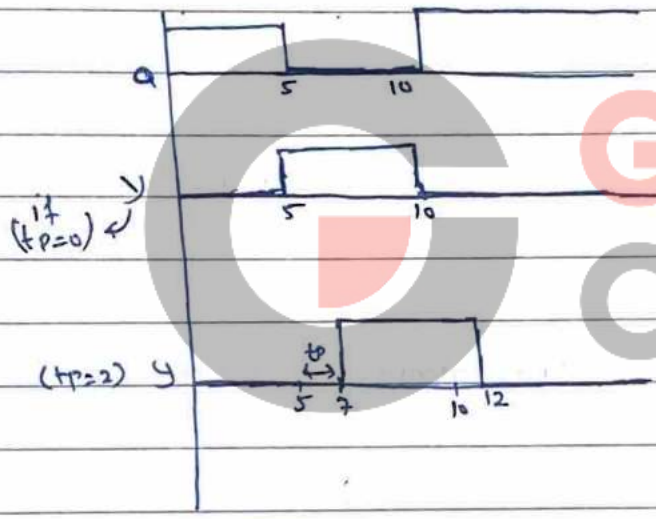
So, for implementing n var function we need $n \times 2^n$ decoder and 1 NAND or OR Gate.

PAGE NO :
DATE : / /

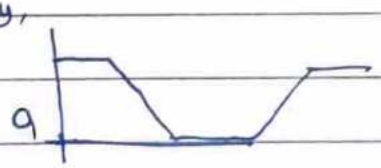
★ **Propagation delay:** When we change i/p, then not instantly o/p changes, it takes very little delay to react changes. bcz in circuit there are physical elements that work. This delay is called propagation delay of gate.

Ex: let $t_p = 2ns$ (prop. delay). for

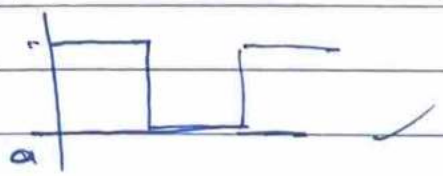
then timing diagram will be:



Practically,



this happens, but as told in M-1, we consider range of voltage as high and range as low. So, theortically it will be eq. to:



GO CLASSES

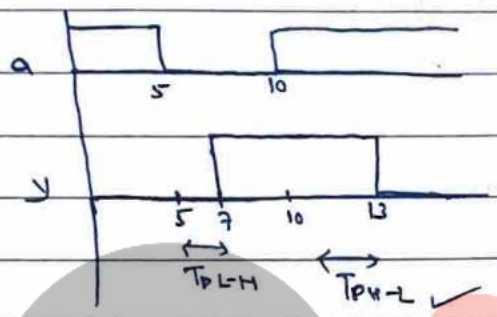
PAGE NO :
DATE : / /

→ Practically T_P when going from low to high & T_P when going from high to low can be different

Ex:

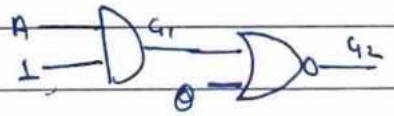
$T_{PL-H} = 2\,ns$ & $T_{PH-L} = 3\,ns$

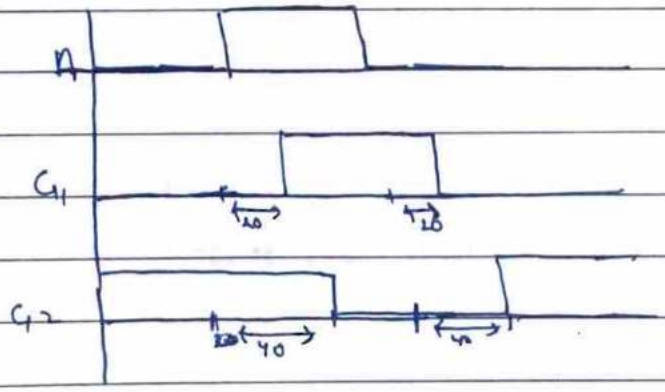
then



But theoretically we'll assume $T_{PL-H} = T_{PH-L} = T_P$. Untill unless given in question.

Ex: T_P for each gate is 20 ns, then timing diagram for below circuit is as:





PAGE NO :
DATE : / /

⇒ **Propagation delay of a circuit:**

Imp definitions:

→ Maximum prop. delay is the longest delay b/w an i/p changing value and the o/p changing value.

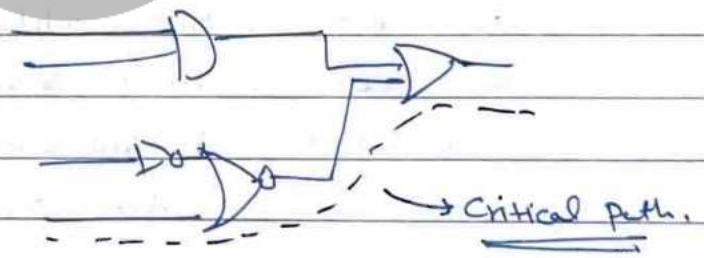
→ The path that causes this delay is called the critical path. It imposes limit on max. speed of circuit.

→ Pd of circuit of Pd of critical path.

So Pd of circuit is max. delay possible (on longest route).

Ex: Prev. ex's Prop. delay is 20+20 = 40ns.

Ex:2 Every gate's delay is 10ns, then delay of prev circuit is:



30ns

→ We assume Pd for wire to be 0, but practically it exists.

→ For delay just shift waveform to right by Pd time.

GO CLASSES

PAGE NO.:
DATE: / /

☆ Adders (Last comb. ckt. of syllabus)

→ We have seen some fundamental circuits i.e. mux, demux, encoder and decoder. Now we'll design most fundamental thing required and that is adder.

→ We will see addition of unsigned numbers. After that we'll use same circuit for signed numbers also. (Small part of lecture).

→ Adder forms basic component of ALU. It has four basic elementary operation possible : $0+0=0$, $1+0=1$, $1+1=10$ & $0+1=1$

→ We may need to add 3 bits if carry is 1 & that will be done using full adders.

☆ **Half Adder** : It deals with 2 bits addition. i.e. take two bits as i/p & gives sum & carry bits as output. In full adders we use prev. carry but here we don't care about that.

Add two bits → Use half adder

Add 2 bits & a carry → Use full adder

Add 2 - multibit number → Use a) Ripple Carry adder
b) Carry lookahead adder

GO CLASSES

PAGE NO.:
DATE: / /

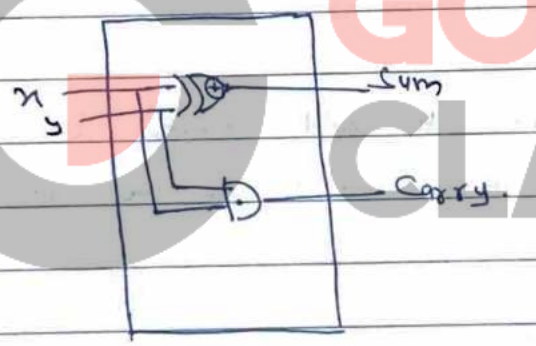
Truth table of half adder:

I/P		O/P	
X	Y	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$S = X\bar{Y} + Y\bar{X} = X \oplus Y$

$C = XY$

So, implementation of half adder is so easy



Half adder

Qu: Add. 011 & 101 using half adder

⇒ Can't do bcz

$$\begin{array}{r} 011 \\ +\ 101 \\ \hline 0 \end{array}$$

only LSB can be added.

3 bit reg

Qu: If "t" is the prop. delay of AND, OR, NOT, EXOR gate etc, then delay of half adder is ?

⇒ t (for S → t & for C → t)

GO CLASSES

PAGE NO. :
DATE : / /

→ For last que, we can't take AND-OR implementation of XOR unnecessarily to ↑ the prop. delay.

Que: Delay of AND, OR, NOT gate i.e. basic gates is d, then delay of half adder is?

⇒ $3d$ i.e. d for carry and $3d$ for XOR i.e. sum.

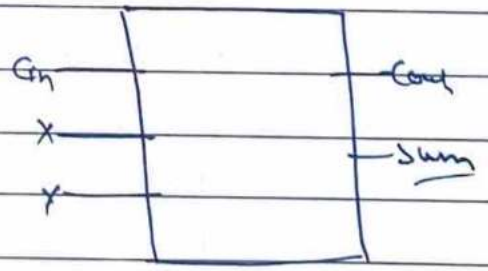
Que: In above que, if i/p available in original & complemented form then delay is?

⇒ $2d$ ($2d$ for sum and d for carry)

★ **Full adder:** Circuit to add 3 bits (2 of no & 1 of carry)
It takes 3 i/p & gives two o/p

Truth table:

i/p			o/p	
x	y	Cin	Sum	Cout
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



Block diagram

Cout is 1 when atleast two input are 1 &
Sum is 1 when odd i/p are 1.

$+ \equiv \oplus$ when both one is not possible

PAGE NO. :
DATE : / /

$S \equiv \bar{X}\bar{Y}C_{in} + \bar{X}Y\bar{C_{in}} + X\bar{Y}\bar{C_{in}} + XYC_{in} \equiv X \oplus Y \oplus C_{in}$

$C_{out} = XY + YC_{in} + XC_{in}$.

For Sum we have two equations shown above.

For Cout we have many equations:

* $C_{out} = XY + C_{in}X + C_{in}Y$ — (I)
$C_{out} = XY + C_{in}(X+Y)$ — (II)
* $C_{out} = XY + C_{in}(X \oplus Y)$ — (III)
$C_{out} = XY \oplus C_{in}(X \oplus Y)$ — (IV)
$C_{out} \neq XY \oplus C_{in}(X+Y)$ — (V)

✓ ✗

→ Equation (V) is not valid for cout,
because if $X=1, Y=1, C_{in}=1$ then $1 \oplus 1 = 0$, but it must be 1.

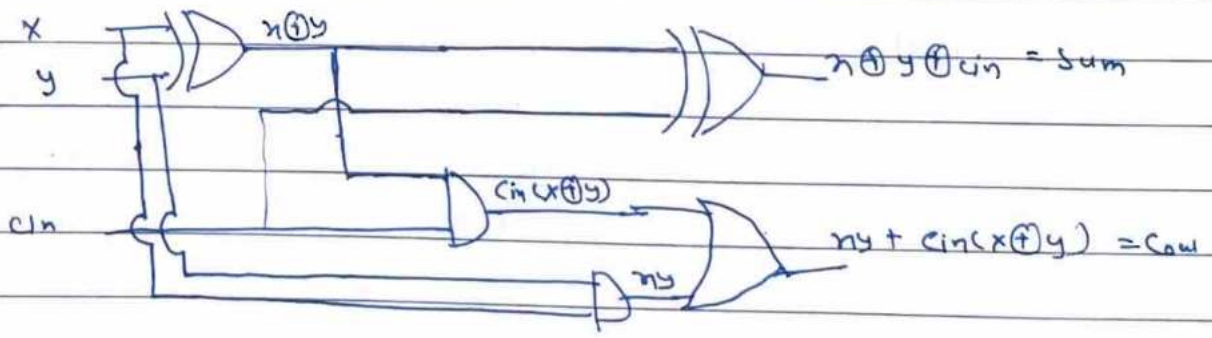
→ Note that xy & $(x \oplus y)C_{in}$ can't be true at same time bcz if $xy=1$ then $x \oplus y = 0$, so that's why $xy \oplus (x \oplus y)C_{in}$ works.

→ Any fun. can be written as exor of true minterms the way we wrote OR of true minterms

Ex: $f = \Sigma m(1,3) = \bar{a}b + ab = \bar{a}b \oplus ab$.

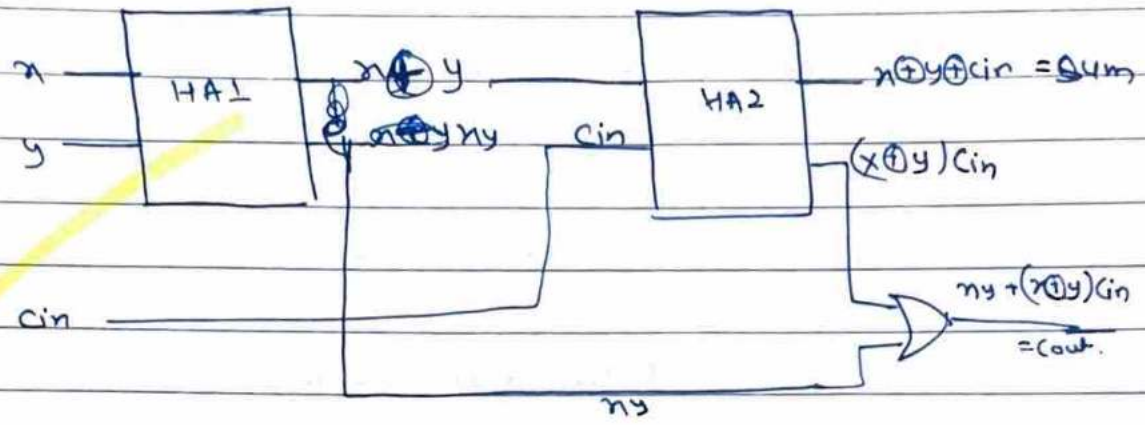
This is because two minterms cannot be 1 at same time & so OR & exor will behave same.

→ AND, OR, EXOR implementation



PAGE NO. :
DATE : / /

→ Two half adder & 1 OR implementation



→ Simple AND, OR, Not implementation can easily be made from expression derived. (2 level for Cout & 3 level for sum)

Que: Can we use only HA to implement FA?
⇒ Yes, we 3 HA (1 in place of 1 OR), Cout = xy ⊕ (x⊕y)cin.

Note: Sum = x⊕y⊕Cin = x⊙y⊙Cin.

★ Ripple Carry Adder: Uses many FAs to add multibit binary number i.e. does actual addition.

⇒ Delay of FA:

- Only basic gates (AND, OR, NOT, NAND, NOR), & each gates' delay is t then its is 3t (2t for carry & 3t for sum)
- (Think from Basic Equation of Sum & Carry)

GO CLASSES

PAGE NO.:
DATE: / /

→ We design binary adder by cascading full adder with Cout of prev. f.adder as cin of current FA. Ripple carry & carry lookahead are two designs of n-bit parallel adder.

→ Cin

a b c d

w x y z

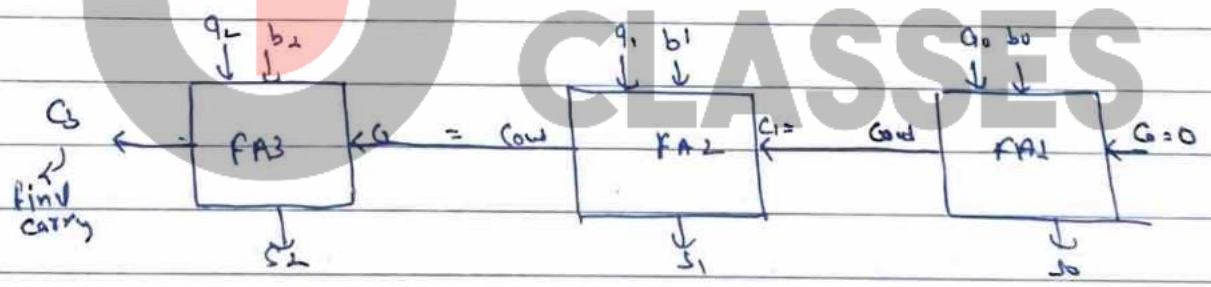
Can use HA

Need FA.

→ RCA (Ripple Carry adder) resembles our human addition computation.

Ex: For 3 bit numbers say

$a_2 a_1 a_0 + b_2 b_1 b_0$



This is how RCA works.

→ In RCA, carry ripples through the FAs to Cout finally. n-bit binary adder needs just n FAs.

→ Initial S_0, S_1, S_2 are invalid i.e. unavailable for us then after some delay S_0 becomes available, then S_1 & finally S_2 & C_3.

→ S_2, C_3 depends on C_2, C_2 depends on C_1 & C_1 depends on C_0. So, all bits becomes available only after carry propagates through the whole circuit.

PAGE NO :
DATE : / /

Que: Assume every FA is created using basic Gates; then what's the delay of RCA if every gate has delay of "δ"?

⇒ We will get S_0 after 3δ & C_0 at 2δ.

Now, we get S_1 after 3δ + 2δ and C_1 at 2δ + 2δ.

↳ to get C_0 used in S_1

3+2+2

Similarly S_2 at 7δ & C_2 at 2+2+2 = 6δ

Finally S_3 at 6δ + 3δ = 9δ and C_4 at 8δ.

for C_2 for sum

C_n at 2nδ and S_{n-1} at (2n+1)δ

Que: FA is implemented using 2HA & 1 OR gate. Delay of AND, OR, Not is δ and that of XOR is 2δ. then RCA delay is?

⇒ Time to get sum: 4δ (Critical path, 2 XOR of HAs)

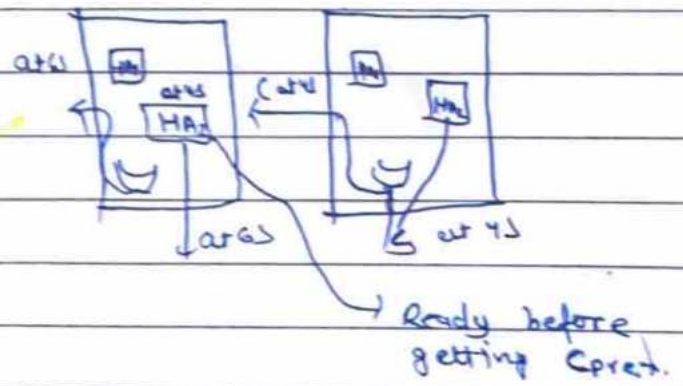
Time to get carry: 4δ (Critical path → 1XOR, 1AND, 1OR)

Now S_0 at 4δ, C_1 at 4δ.

S_1 at 6δ, C_2 at 6δ

↳ bcz already 4δ, comp. will be done, only left is dep. of C_2

~~S_{n-1} : (2n)δ , C_n : (2n)δ~~



PAGE NO.:
DATE: / /

⇒ Drawbacks of RCA:

- Total delay is proportional to # bits.
- Performance degrades for large n.
- Bottleneck is carry which propagates seq. from one stage to next.
-

→ Drawbacks of RCA can be overcomed if we some how gen. all carry bits in advance before actual addition, then after the initial delay, all bits can be summed up in parallel. Time compl. shifts from $O(n)$ to $O(1)$, but h/w comp. ↑.

★ **Carry Look Ahead (CLA) adder**: In this we gen. all carries initially and then sum up all bits in one go.

Initially we have a_3, a_2, a_1, a_0, b_3, b_2, b_1, b_0 and c_0.
Now, we'll create a separate circuit for carry calculation

Let's write T.T. first

a	b	Cin	Cout
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

GO CLASSES

PAGE NO :
DATE : / /

→ when $a=1$ & $b=1$, then no matter what's the C_{in}, C_{out} will surely be 1. So $a=1, b=1$ are called carry generator (G). $G = ab$.

→ when $a=1, b=0$ or $a=0, b=1$, C_{in} gets propagated i.e. $C_{out} = C_{in}$. This is called carry propagator. (P)

$P = a\bar{b} + b\bar{a} = a \oplus b$.

Not used

→ when $a=0, b=0$, then no matter what C_{in} is C_{out} will be 0, this is called carry killer. (K)

$k = \bar{a}\bar{b}$

So, we get C_{out} when either carry propagates or carry is generated. And to know $G = ab$ & $P = a \oplus b$, we don't need any other i/p.

$G_0 = a_0 b_0$, $P_0 = a_0 \oplus b_0$

$G_1 = a_1 b_1$, $P_1 = a_1 \oplus b_1$ and so on.

Que: If only basic gates can be used with delay δ, then delay of P_3?

⇒ 3δ. (only OR, NOT, AND delay)

→ Initially at $t=0$ a_i, b_i, C_0 are available then we can get G_i at $t=\delta$ & P_i at 3δ.

So at δ we have all G_i's & at 3δ we have all P_i's if, δ is delay of basic gates and of XOR is 3δ.

GO CLASSES

PAGE NO. :
DATE : / /

→ Now we have a_i, b_i, P_i and G_i. Now we need to generate c_{i+1} from them. i.e Cout from stage i.

$$C_{i+1} = (a_i b_i) + c_i (a_i \oplus b_i)$$

Formula we already know from FA.

$$\boxed{C_{i+1} = G_i + C_i P_i}$$

Intuitively also for C_{i+1} either gen. at i or prop. C_i.

$C_1 = G_0 + P_0 C_0$ → calculated from initial data

$C_2 = G_1 + P_1 C_1$ → G_1, P_1 available.

$C_2 = G_1 + P_1 (G_0 + P_0 C_0)$ → Cal. from initial data.

⋮

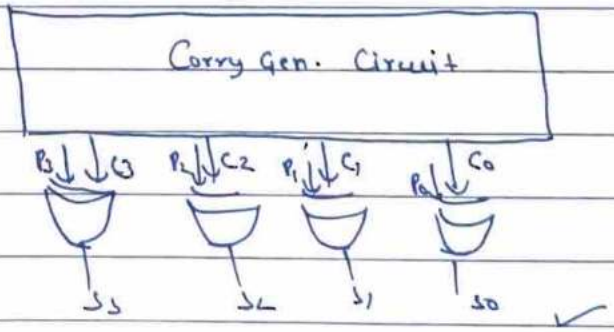
$$C_4 = \underbrace{G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0}_{\text{from initial data.}}$$

→ So we can gen. all carries initially so that we don't have any dependency from prev. stage.

Now $S_0 = a_0 \oplus b_0 \oplus C_0 = P_0 \oplus C_0$

$S_1 = a_1 \oplus b_1 \oplus C_1 = P_1 \oplus C_1$

So, CLA may look as:



PAGE NO.:
DATE: / /

→ C_4 is 1 if, either C_3 generates it i.e. G_3,

OR, C_2 gen. and P_3 propagates $G_2 P_3$,

OR, C_1 " " $P_2 P_3$ ", $G_1 P_2 P_3$

OR, C_0 " " P_1, P_2, P_3 ", $G_0 P_1 P_2 P_3$

OR, C_0 is 1 and everyone propagates, $C_0 P_1 P_2 P_3$

So $C_4 = G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_3 P_2 P_1 + C_0 P_0 P_1 P_2 P_3$

⇒ Some observations about carry equations:

- C_i has $i+1$ terms, so fan-in req. for OR gate of C_i should be $i+1$. and # AND gates is i (as any term is G_{i-1})
- C_i has $i+1$ literals at most in 1 term so fan-in req for AND gate of C_i is $i+1$

So, fan-in of OR → $i+1$, fan-in of AND → $i+1$

OR → 1, # AND → i

⇒ Delay of CLA:

Q→ When fan-in is unlimited and delay of basic gates is d.

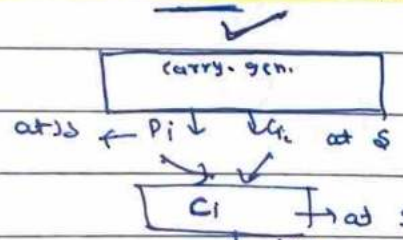
Here we get P_i at 3d & G_i at d.

Now we get C_{i+1} after 2d as 1 level of AND & 1 level of OR as fan-in is unlimited

So, at 5d we have P_i, G_i & C_i.

Now we need to XOR C_i & P_i, which will take 3d.

So at 8d we have our sum



XOR it with P_i to get S_i : 8d

PAGE NO. :
DATE : / /

NOTE C_0 is not always zero, bcz, we only manufacture adder of fix bits (say 8 bit adder), then we add 16 bits by using it twice, so, while using it 2nd time C_{in} may not be zero.

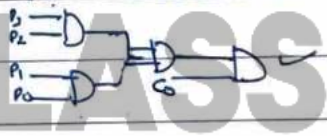
Q⇒ Delay of ~~Gin~~ CLA (4 bit) when fan-in is 2 and delay of basic gates is d.

→ we'll get G_i at 5 and P_i at 3d. (like before), as just need to do xor, AND.

→ Now we'll get C_4 after:

C_4's critical path will be $B_3 P_2 P_1 P_0 C_0$, so

3 levels of AND



and then 5 OR gate i/p. → 3 levels for this

So get C_4 after 6S.

So finally S_4 will be available at $\underbrace{3S}_{\text{for } P_i} + \underbrace{6S}_{\text{for } C_i} + \underbrace{3S}_{\text{for } S_i} = 9S$

Approx $\Theta(\log n)$ levels for AND, OR gates as fan-in can't be unlimited. Base of log is fan-in.

→ GRE que. In Google doc.

GO CLASSES

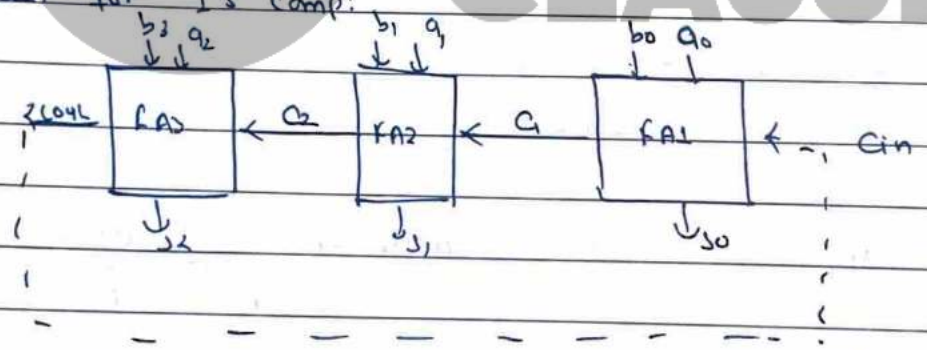
PAGE NO. :
DATE : / /

So far, we have seen adders for unsigned addition i.e. $O(n)$ for RCA & $O(\log n)$ for CLA if fan-in is limited.
Now, for signed addition.

a) 1's Comp: I/P in 1's comp (A, B)
Sum A+B using adder, adder(A, B)
We get S & Cout, sum term, adder(S, Cout)
This is A+B in 1's comp form.
$2 * O(n)$

b) 2' Comp: I/P in 2's comp (A, B)
Sum A, B using adder add(A, B).
Ignore Cout, S is final result in 2's comp. form
$O(n)$

→ Circuit for 1's comp:



wrapping end carry.

→ Not a seq. circuit, just using 1 circuit twice.

GO CLASSES

DATE: / /
PAGE NO.:

M-4 in DLD Lectures 2024

Module-4 (Functional Completeness)

In boolean algebra we have seen many operations. Using these operations, we can write bool. exp. of any boolean function.

But, do we really need all of them to write exp. of any boolean function?

No, not all required.

→ Set of boolean operation which is sufficient to write expression of every boolean function is called "Functionally Complete" set of boolean expression.

Ex: {NOR}, {AND, OR, NOT}, etc.

→ Set of logic gates which are enough to implement any digital circuit is called "functionally complete" set of gates.

→ Similar def. for functionally comp. set of set theory operations & functionally complete set of prop. logic operations

⇒ Proof that {AND, OR, NOT} is func. comp. in boolean alg.

As we can write CSOP / CPOS for any function i.e. we can express any func. using {AND, OR, NOT}, So it's functionally complete.

⇒ Proof that {AND, NOT} / {NOT, OR} is func. complete.

As using demorgan's law AND → OR, NOT & OR → AND, NOT, So, {AND, NOT} or {OR, NOT} is also functionally complete.

GO CLASSES

Superset of F.C. set is obviously functionally complete.

DATE: / /

PAGE NO.:

Ex: Remove OR.

$A+B \Rightarrow \overline{\overline{A+B}} \Rightarrow \overline{\bar{A}\cdot\bar{B}}$ (No OR)

Remove AND

$\bar{A}B \Rightarrow \overline{\overline{\bar{A}B}} \Rightarrow \overline{\overline{\bar{A}}+\bar{B}} \Rightarrow \overline{A+\bar{B}}$ (No AND)

So {AND, NOT}, {OR, NOT} both are functionally complete.

bcz:

Any fun. Truth Table → SOP/POS → Remove AND using demorgan's law → Only OR, NOT

→ Remove OR → Only AND, NOT

Que: Eliminate addition from f:

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

$f = x\overline{\overline{\bar{y}+\bar{z}}} + \overline{xy}$

$f = x(\overline{yz}) + \overline{xy}$

$f = \overline{x\overline{yz} + \overline{xy}}$

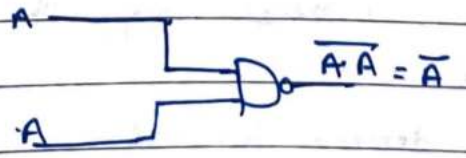
$f = \overline{\overline{x\overline{yz}}\cdot\overline{\overline{xy}}}$

$f = \overline{\overline{x\overline{yz}}\cdot xy}$

⇒ Prove that NAND Gate is an universal gate.

Proof: To prove that, any bool. fun. can be implemented using only NAND gates, we will show that AND, OR, NOT operation can be performed using only these gates.

a). NOT Gate.

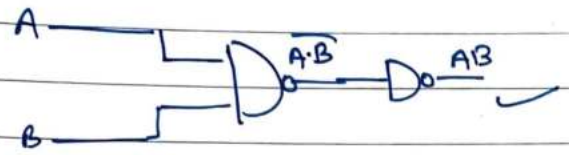


$\overline{A\cdot A} = \bar{A}$

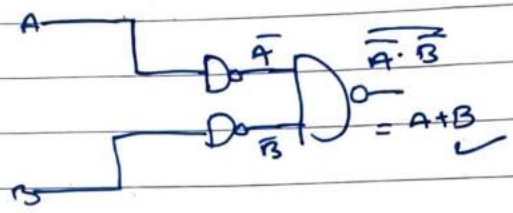
If some F.C. set is determined by some set, then this set is also F.C.

DATE: / /
PAGE NO.:

B). AND Gate.



C). OR Gate



∴ Hence, Proved.

Universal gates are those gates which can imp. any Boolean function w/o need to use any other gate type.

Ex: NAND Gate, NOR Gate.

We can only universal gates to make any digital circuit.

Que: Express $\overline{x+y}$ using only NAND operations

$\Rightarrow \overline{x} \cdot \overline{y} \Rightarrow (x \uparrow x) \cdot (y \uparrow y) \Rightarrow (x \uparrow x) \uparrow (y \uparrow y)$

$= \quad \overline{x} \cdot \overline{y} \quad \overline{x \uparrow y} \Rightarrow \overline{\overline{x}\,\overline{y}}$

$\Rightarrow \underbrace{((x \uparrow x) \uparrow (y \uparrow y)) \uparrow ((x \uparrow x) \uparrow (y \uparrow y))}_{\text{Ans}}$

$= \overline{\overline{\overline{x}\,\overline{y}}\;\overline{\overline{x}\,\overline{y}}} \Rightarrow \overline{\overline{\overline{x}\,\overline{y}}} = \overline{x}\,\overline{y}$

Similarly we can prove that NOR is also a universal Gate.

But ∵ $x+y = (x \downarrow y) \downarrow (x \downarrow y)$ } ↓ is NOR

$\overline{x} = x \downarrow x$

OR, NOT is derived from NOR

F.C.

DATE: / /

PAGE NO.:

To check if a set is f.c. always minimise operations of set and then try to create {NOT, AND} or {NOT, OR} or any other f.c. set from that.

We also have a formal algorithm for this f.c. Checking: (Saw later)

⇒ Prove that $\{\rightarrow, \neg\}$ is func. complete.

$a+b = \neg a \rightarrow b$
$\neg a = \neg a$

$(+, \neg) \xrightarrow{\text{derived from}} (\uparrow, \neg)$

↳ Already f.c. ↳ So f.c.

⇒ Prove that $\{\rightarrow, \vee\}$ / $\{\rightarrow, \wedge\}$ is not f.c.

→ By formal theorem told later.

⇒ Prove that $\{\rightarrow, 0\}$ is f.c.

$\neg a = a \rightarrow 0$

$a+b = \neg a \rightarrow b = (a \rightarrow 0) \rightarrow b$

So {OR, NOT} is derived from $\{\rightarrow, 0\}$. So f.c.

★ Properties of boolean functions:

a) 0-preserving boolean function: $f \in T_0$ is 0 preserving iff for $f(0,0 \ldots 0) = 0$.

Ex: $a+b$, ab, $a \oplus b$, a, 0 ↳ constant function

$a \rightarrow b$, $a \uparrow b$, $a \downarrow b$, $a \odot b$, $\bar{a}$ are not 0 preserving.

DATE : / /

PAGE NO. :

$\in T_1$

b) **1-preserving boolean functions:** f is 1 preserving iff $f(1,1,\ldots,1)=1$

i.e. if all i/p are 1 then o/p is 1.

Ex: $a+b, ab, a\to b, a\odot b, a, 1$ are 1 preserving

$a\uparrow b, a\downarrow b, a\oplus b, \bar{a}, 0$ are not 1 preserving.

NOTE: If we know a fun is 0 preserving, we can't say anything about 1 preserving property or vice-versa.

$\in S$

c) **Self Dual Function:** f is self dual if $f=f^d$.

Ex: $a, \bar{a}$ are self dual.

$a+b, ab, 0, 1, a\oplus b, a\odot b, a\uparrow b$ are not self dual func.

$\in L$

d) **Linear Function:** f is linear if one of the following two statements holds:

a) For each 1-value of f, the no. of 1's in corresponding i/p is odd & for every 0-value of f, the no. of 1's in corresponding i/p is even.

OR

b) For each 1-value of f, the no. of 1's in corr. i/p is even & for every 0-value of f, the no. of 1's in corresponding i/p is odd.

(f shouldn't have dummy variables)

If one of the above stmts hold, f is linear.

GO CLASSES

DATE : / /

PAGE NO. :

Ex: $f = \bar{a}$

a	f
0	1
1	0

(even 1's)
(odd 1's)
} ⇒ Linear.

$f = ab$

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

Not linear bcz. for 0-values #1's are sometime odd, sometimes even. They should be odd always (here).

$a \downarrow b$
$a \uparrow b$
$a \to b$,
Similarly $a+b$ is also not linear.

$f = a \oplus b$

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

Linear.
1-value = #1 are odd
0-value : #1 are even.

Similarly $a \odot b$ is also linear.

* NOTE: Before checking linearity of function f minimise f. bcz. dummy var. can change answer.

For Ex: $f(a,b) = a + ab$

a	b	a+ab
0	0	0
0	1	0
1	0	1
1	1	1

} Not linear.

GO CLASSES

DATE : / /
PAGE NO. :

On minimizing

$f(a,b) = a + ab = a$

a	a
0	0
1	1

} Linear Now

So, do NOT forget to remove the dummy variable (b).

Linear Boolean function $\rightleftharpoons$ Alternating function $\rightleftharpoons$ Counting function

Que: $f(a,b,c) = \bar{a}b + a\bar{b} + a\bar{b}c$.
Check linearity of f.

$\Rightarrow$ Minimise f.

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

$= a \oplus b$.

↳ Linear ✓

NOTE Constant functions 0 & 1 are Linear. ✓

E.M

e). Monotonic Boolean Functions: for function f, if inc. the i/p comb. by changing any single i/p from 0 to 1, then f value should not decrease. (i.e. 1 to 0 not allowed)

Ex:

a	b	c	f	f'
0	0	1	0	1
0	1	1	0 or 1	1

✓

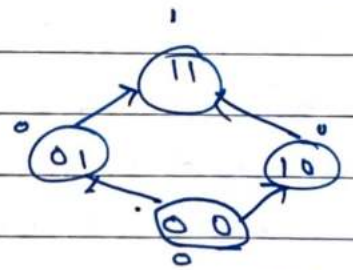
DATE : / /

PAGE NO. :

Best way to check for monotonic function is by making lattice.

Ex: Check if a b is monotonic

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



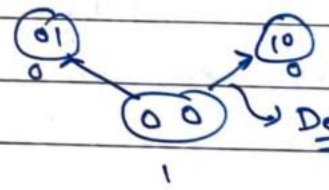
value should not dec: on this path

So, monotonic.

a+b is also monotonic.

Check $a \to b$ is monotonic or not

a	b	$a \to b$
0	0	1
0	1	1
1	0	0
1	1	1



Decreasing

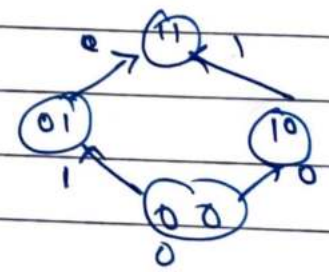
So NOT monotonic

$a \uparrow b$, $a \downarrow b$

$a \odot b$, $a \oplus b$ are also not monotonic.

Check $f(a,b) = b$ is monotonic or not

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



Montonic

Lattice for 3-var can also be created.

DATE : / /
PAGE NO. :

NOTE Constant functions 0 & 1 are monotonic as f. value can never be decreased.

Post's Functional Completeness Theorem: A set of boolean functionally complete if and only if this system doesn't belong entirely to any of T_0, T_1, L, M, S.

i.e. X is set of func.

$X \notin T_0$ AND $X \notin T_1$ & $X \notin L$ & $X \notin M$ & $X \notin S$

where T_0 is set of 0 preserving functions
T_1 " " " 1 " "
L " " " Linear "
M " " " Monotonic "
S " " " Self-Dual ".

i.e. X should have: ~~atleast one function~~ → to be functionally complete.

a) atleast one function that don't preserve zero, and,
b) " " " " " " one, and,
c) " " " " " is not linear, and,
d) " " " " " " monotonic, and,
e) " " " " " " self-dual.

So, if all f. of X are 0 preserving, OR
" " " " 1 " , OR
" " " " Linear , OR
" " " " Monotonic OR
" " " " Self dual , then
X is not F.C.

DATE : / /

PAGE NO. :

Ex: $X = \{ x \leftrightarrow y, xy, 0 \}$

Not 0 preserving ϕ

Not 1 " $\{0\}$

Not Linear $\}$

Not Monotonic $\}$

Not Self Dual $\{ x \leftrightarrow y, xy, 0 \}$

As all fun. are 0 preserving it is not F.C.

Ex: 2 $\{ (x+\bar{y}), x \downarrow y, 1 \}$ (Directly, superset of $x \downarrow y$, so F.C)

$x \downarrow y, 1, x+\bar{y}$ is not zero preserving

$x \downarrow y$ is not 1 preserving

$1, x \downarrow y$ is not linear

$1, x \downarrow y$ is not monotonic.

$x+\bar{y}, 1, x \downarrow y$ is not self dual

Ex: 3 $\{ \cap, \vee, \rightarrow, \leftrightarrow \}$

$\Rightarrow$ All are 1 preserving. So NOT F.C.

Ex: 4 Show that $\{1, A\bar{B}\}$ is F.C.

$\rightarrow$ $1 \rightarrow$ Not 0 preserving, Not self dual.

$A\bar{B} \rightarrow$ Not 1 preserving, Not monotonic, Not linear, Not self-Dual

↳ 1,0 is 1

& 1,1 is 0

So, by Emil Post's theorem,

It is functionally Comp.

GO CLASSES

DATE : / /
PAGE NO. :

Method 2.

Create NOT.

$1 \cdot \bar{A} = \bar{A}$ NOT can be created.

Create AND

$A \cdot \overline{\overline{B}} = A \cdot B$, AND can be created.

So F.C

Good ex.

Que: $f(a,b,c)$ given by T.T. below is F.C.

a	b	c	$f(a,b,c)$	f^* → Comp. & Reverse
1	1	1	0	0
1	1	0	0	1
1	0	1	1	1
1	0	0	0	0
0	1	1	1	1
0	1	0	0	0
0	0	1	0	1
0	0	0	1	1

Q). Show that f obey's req. of post's completeness theorem.

⇒ $f(1,1,1) = 0$ ✓ Not 1 preserving

$f(0,0,0) = 1$ ✓ Not 0 preserving

$f(1,1,0) = f(1,0,0) = 0$ ✓. Not linear (Minimised table) (as odd 1 & odd 0).

$f = \bar{a}\bar{b}\bar{c} + \bar{a}bc + a\bar{b}c$ ← No dummy Check using K-map.

DATE: / /

PAGE NO.:

$f(0,1,1) = 1$
$f(1,1,1) = 0$ } Not Monotonic

from f^d row } Not Self Dual.

as $f \neq f^d$

or

$f(1,1,0) \neq \bar{f}(0,0,1)$

So, obeys all req. of Post's F.C. theorem.

b). Show that f is f.c. by deriving any already know F.C. set.

Using k-map

$f(a,b,c) = \bar{a}bc + a\bar{b}c + \bar{a}\bar{b}\bar{c}$

$f(a,a,a) = \bar{a}$

↳ Not created

$f(a,a,1) = \overline{a+c}$

↳ NOR created

And {NOT, NOR} is F.C.

So, Hence Proved

NOTE ⟷ There is NO official/std. term "Partially functional comp."

GO CLASSES

DATE : / /
PAGE NO. :

Imp. NOTE In boolean function 0, 1 are not by default available to us so $\{\rightarrow\}$ is not. f.c. ~~y~~

→ is bool. function & set $\{\neg, 0\}$ is f.c.

In digital circuits, 0,1 are by default available to us. So, gate "$\rightarrow$" is functionally complete as 0 & 1 are available to us.

$\bar{x}+y$

So, in Exam take care of they are asking about f.c. of boolean function or logical gates ....

Digital Ckt (with mux, adder) / Gates → 0 & 1 available

Boolean function → 0, 1 Not available unless given.

GO CLASSES

PAGE NO. :
DATE : / /

M-5 Sequential Circuits.

★ Combinational Circuit v/s. Sequential Circuits

→ Circuits we have seen so far are comb. circuits like gates, mux, encoder, demux, decoder and adder.
O/p of combinational circuits is determined by the present value of i/p and not the prev. value of o/p.
In this, we do not care what was the o/p before, we can just see the i/p and can tell the o/p.

→ Circuits we'll see now in this module are seq. circuits such as latches and flipflops.
O/p of seq. circuit depends on current i/p as well as prev. output.
for Ex: Counter. current i/p is +1, but o/p can't be told untill you know prev. o/p.

Comp. Circuit = f (current i/p)
Seq. Circuit = f (current i/p, prev. o/p)
= f (current i/p, prev. i/ps).

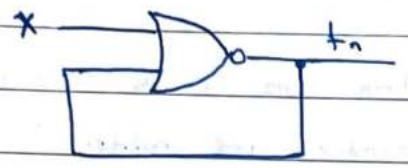
→ Seq. circuits implicity contain memory elements along with logic gates. In comb. circuit we only had logic gates.

→ Almost every useful circuit is sequential circuit.

GO CLASSES

PAGE NO :
DATE : / /

⇒ Analyze the following circuit:



⇒ Let current o/p is f and next o/p is f_n

if $x=1$ and f was 1 then f_n will be 0
" " " " " 0 " " " " 0
" $x=0$ " " " 1 " " " " 0
" " " " " 0 " " " " 1

Here o/p is surely dependent of prev. o/p. so it's a seq. circuit.
↳ whole prev. history

→ Two convention to write outputs {Q_t → current, Q_{t-1} → prev and Q_{t+1} → next} or {Q → current, Q_n → next and Q_p → prev.).

→ In seq. circuits, ~~we remember~~, we remember prev. outputs using memory element. Most basic memory element might be storing only 1 bit, lets say this is device D. So now to store n bits we have #n, Ds.

→ Now we'll device device that can store data, that can remember.

GO CLASSES

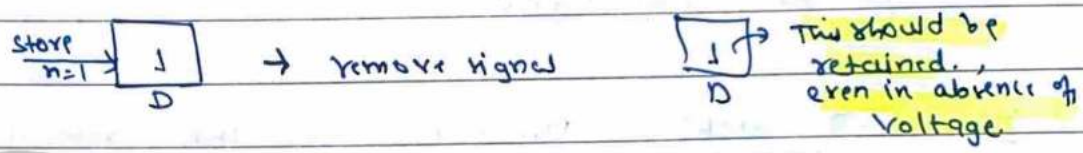
PAGE NO. :
DATE : / /

★ **Memory Elements / Storage:** We will now try to create 1-bit storage device.

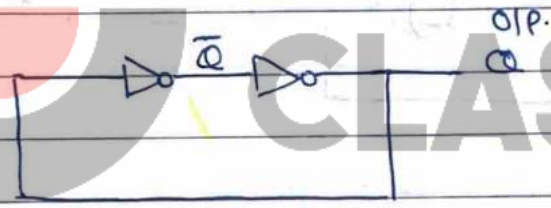
⇒ what does "Remembering information" mean?

Ans. store the info and then retain it untill it is changed further.

Ex: Say, storage device is D



→ Most basic one-bit storage device can be.



Cross Coupled Inverter. ✓

Here if 1 is stored then, 1 is retained, but, here we are not given option to change. Once storage is fixed we can't change it. If Q is 0, then can't change to 1.

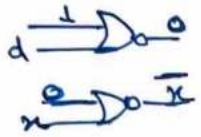
Qu: Can we say above circuit (i.e. cross coupled inverter) is 2-bit storage device as it can give Q & $\bar{Q}$.

→ No, not at all. It can't give all possible 4 comb. It is just giving us Q & $\bar{Q}$. Can't store two bits '0' & '0'. It is just we are getting "Complement" freely.

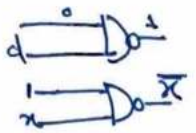
→ Disadv. of cross coupled inv. is it don't have user control.

PAGE NO :
DATE : / /

In nor



In NAND

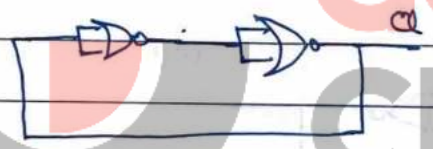


→ There are two ways to get user control on CCS, are we NAND or NOR instead of NOT with one user i/p.

→

is not storage device as to store 0, we need to apply x=1 continuously, this is not the definition of storage device. It should retain w/o any signal.

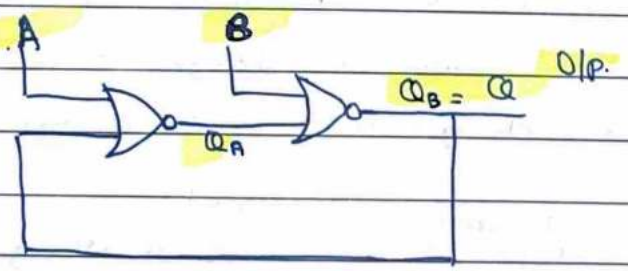
★ **S-R latch:** If we use NOR gate in cross coupled inverter, nothing will change:



≡ Eq. to cross-coupled inverter.

Here, now we need user control also.

So,



A & B are user control here.

If B=1, A=0 then whatever Q_A was, Q_B will be Q surely

So A=0, B=1 is used to store 0.

If A=0, B=0, then prev. i/p is retained here.

as $Q_A = \overline{Q_{pr}}$ and $Q_B = \overline{Q_A} = \overline{\overline{Q_{pr}}} = Q_{prev}$

So Q_B will be Q_{prev}.

and Q_B is Q, ⇒ $Q_{prev} = Q_B = Q$ ✓

PAGE NO. :
DATE : / /

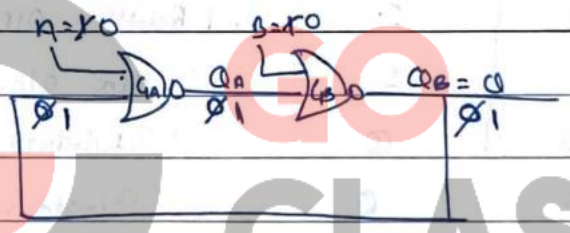
→ If $A=1$, $B=0$, then Q_A will be 0 for sure and Q_B will be 1 for sure. So $A=1, B=0$ is used to store 1.

* → Special case

If $A=1$ & $B=1$, then Q_A will be 0 and Q_B will be 0 also (As $1 \downarrow 0 = 0$). So we'll be having Q as 0 stable.

But now, if we want to retain this by giving $A=0, B=0$, chaos will happen

~~$Q_B = Q$, $Q_A = Q$, now $Q_{next} = (Q \downarrow 0) = 1$, $Q_{next} = (Q \downarrow 0) = 0$~~



If $G_A = G_B$ i.e. both NOR have same speed then as soon as we change the signal to 0 & 0 Q_B becomes = 1 $(0 \downarrow 0)$ and Q_A becomes 1 to 0 $(0 \downarrow 0)$. Now after 1 prop. delay it changes to 0 and 0, then 1 then 0 and so on it keeps on alternating.

So, the o/p is not retained by $A=0, B=0$ if prev $A=1, B=1$. It will start alternating.

→ If speed of G_A & G_B are diff and fluctuating then we can't determine what will happen next after $A=0, B=0$ if prev A & $B=1$

→ Using $A=1, B=1$ is not allowed bcz it will $\downarrow$ to 0. (→ Not a Problem.)
but it can't be retained. (→ This is the main Problem)

Set → making o/p 1
Re Set → making o/p 0

PAGE NO. :
DATE : / /

A=1, B=1 is forbidden bcz, o/p of A=1, B=1 is stable and it 0.

$\underbrace{\begin{pmatrix} A=1 \\ B=1 \end{pmatrix} \Rightarrow \begin{pmatrix} A=0 \\ B=0 \end{pmatrix}}_{\text{this transition}}$ is uncertain.

Truth table:

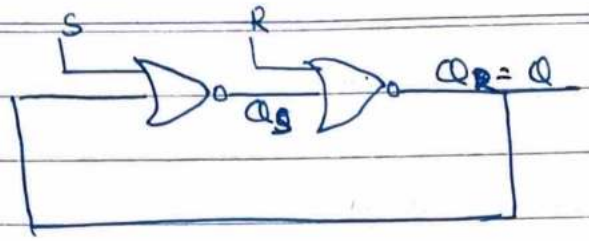
A=S	B=R.	Q_n (next o/p)	
0	1	0	(Resetting o/p)
1	0	1	(Setting o/p)
0	0	Q	(Retaining current o/p)
* 1	1	0	(forbidden)

↳ this is compact T.T.
↳ Std. TT. will have 8 rows as
$Q_n = f(A, B, Q)$.
↳ Not drawing this T.T.

→ The circuit we studied just now is "SR Latch".

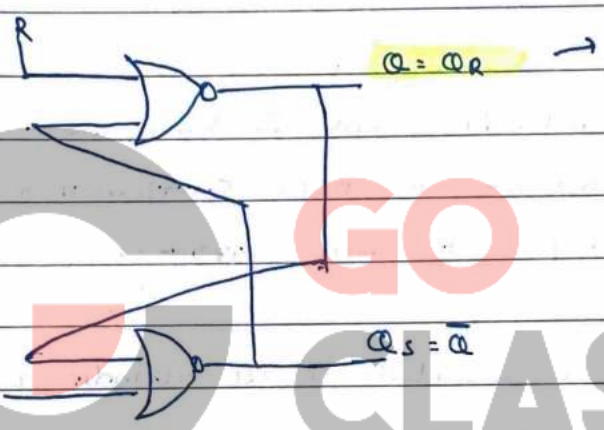
* Here we say i/p A as S (standing for set) bcz when S=1, then it sets the o/p and i/p B as R (i.e. Reset) bcz when R=1, it resets the o/p.

PAGE NO.:
DATE: / /



S-R Latch.

|||



→ Remember R at above as final o/p is taken from R

Another way to draw.

GO CLASSES

Qu: fill in following table for SR- latch:

time	S	R	Q_n
1	0	1	0
2	0	0	0
3	1	0	1
4	0	0	1
5	0	0	1
6	0	1	0
7	1	1	0
8	0	0	Undetermined.

Problem.
↳ can't say next o/p.

PAGE NO. :
DATE : / /

→ Characteristic table of S-R latch

S A	R B	Q_A	$Q_B = Q$
0	1	1	0
1	0	0	1
0	0	Q_{prev}	Q_{prev}
1	1	0	0

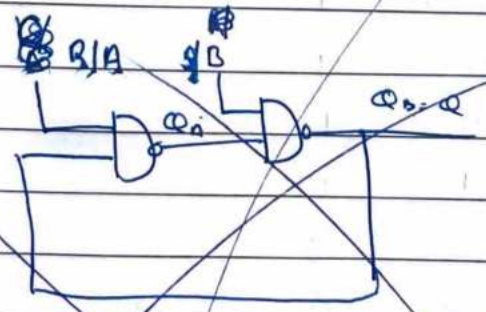
Here, note that, $Q_A = \overline{Q_B} = \overline{Q}$

So when $S \neq 1$ and $R \neq 1$ then Q_A & Q_B are inverse of each other so whenever we need complemented result we can take Q_A instead of using inverter.

for forbidden i/p $Q_A = Q_B$, but it is already forbidden so, we donot care.

→ The SR latch we have studied till now was SR latch using NOR. It can also be implement by NAND gate.

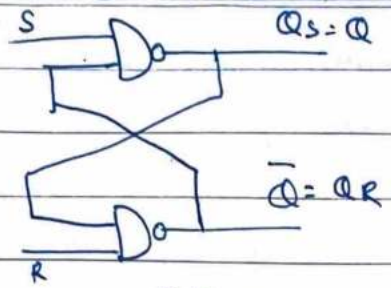
☆ S-R latch using NAND gate (Also called $\overline{S}\overline{R}$ latch)



SR latch using NAND
$\overline{S}\overline{R}$ latch.

PAGE NO. :
DATE : / /

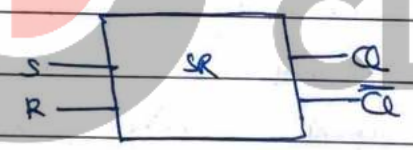
★ SR Latch using NAND Gate ($S\bar{R}$ Latch)



$\bar{S}$-$\bar{R}$ latch ✓

→ SR latch was active high i.e. (Retaining when i/p is low) and $\bar{S}\bar{R}$ latch is active low (Retaining when i/p is high)

→ In $S\bar{R}$ latch also, $Q_A = \bar{Q}_B$ whenever i/p are valid and $Q_A = Q_B$ for forbidden i/p. So if we want $\bar{Q}$, we don't need an extra inverter.



Block Diagram of SR latch.

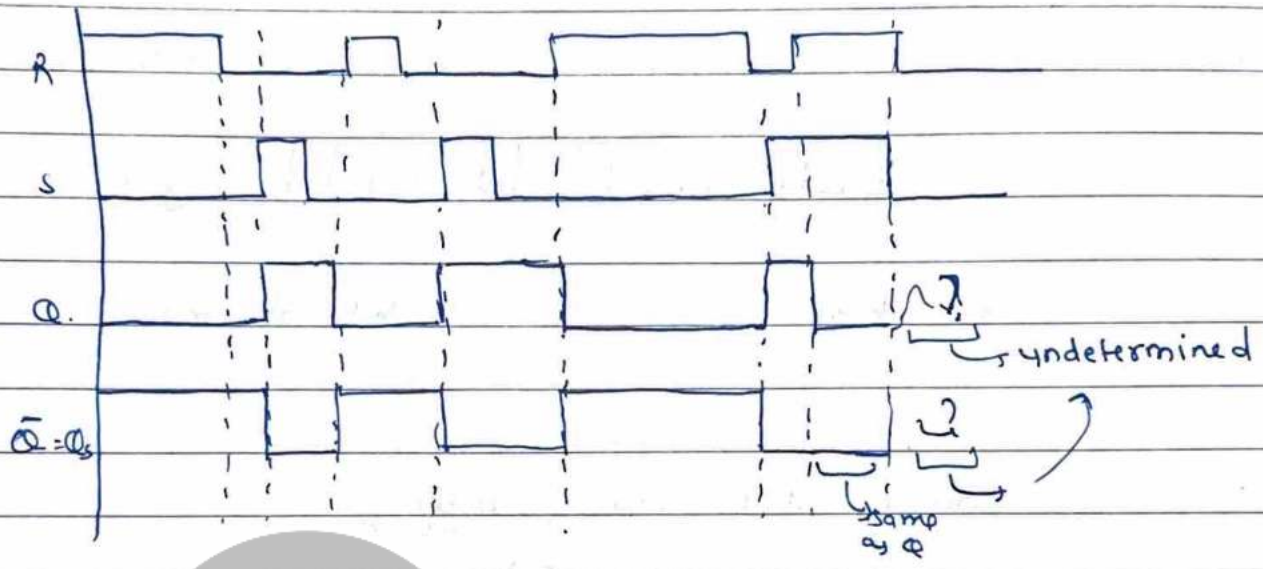
⇒ Problem with latches: As soon as i/p is changed, o/p changes. There is not any controlling i/p (like enable) to control its working. We will use something

→ Timing Diagram:

GO CLASSES

PAGE NO. :
DATE : / /

Ex: Timing diagram for S-R-latch:



★ Flipflops: The basic latch changes its states when the i/p signals change. It is hard to control when the i/p signal will change & thus hard to know when the latch may change its state. We want something like "Enable input". In this case it is called the "Clock" i/p bcz. it's desirable for the state changes to be sync.

→ For comb circuit → Enable i/p
→ For seq. circuit → Clock i/p

→ flipflops = Latch + Clock. i.e. Clocked latch.

→ Clock: Signals that control the o/p of seq. circuit. If clock signal are absent then o/p of circuit will change randomly.

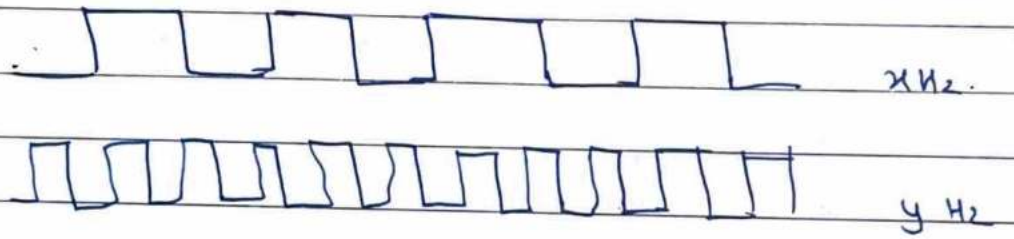
Clock is measured in frequency in Hz.
Ex: We buy 2.4 GHz PC.

GO CLASSES

PAGE NO :
DATE : / /

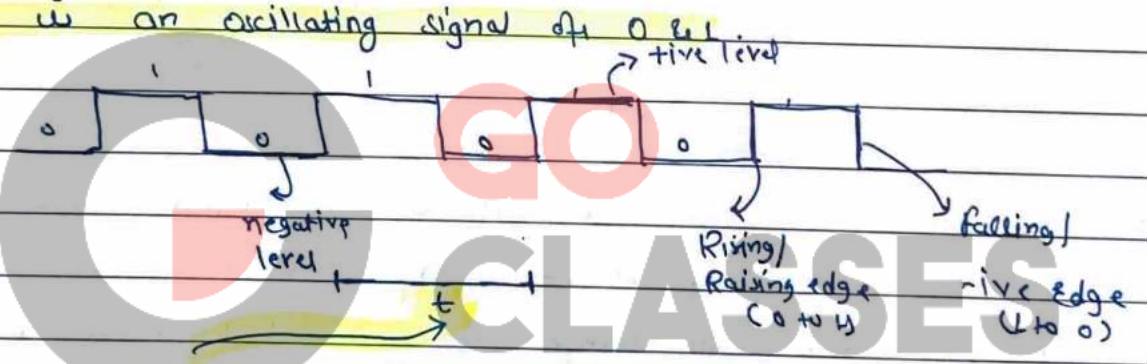
frequency is how fast signals are produced.

Ex.



then here $y > x$ i.e. 2nd clock has more frequency.

→ Clock is an oscillating signal of 0 & 1



GO CLASSES

Time period (t): Time duration b/w consecutive rising edges or (b/w consec. falling edges).

frequency: How frequently signal is changing

$$freq \propto \frac{1}{t}$$

$$f = \frac{1}{t}$$

Unit is Hz i.e. changes / second.

If t is 1s then freq. is 1 Hz.

t	freq
$1ns = 10^{-9}s$	$1GHz = 10^9 Hz$
$1\mu s = 10^{-6}s$	$1MHz = 10^6 Hz$
$1ms = 10^{-3}s$	$1KHz = 10^3 Hz$

PAGE NO. :
DATE : / /

There are 4 standard flipflops: , SR, D, T and JK FF.

★ **S-R flipflop:** This is just clocked SR latch.

It's behaviour should be:

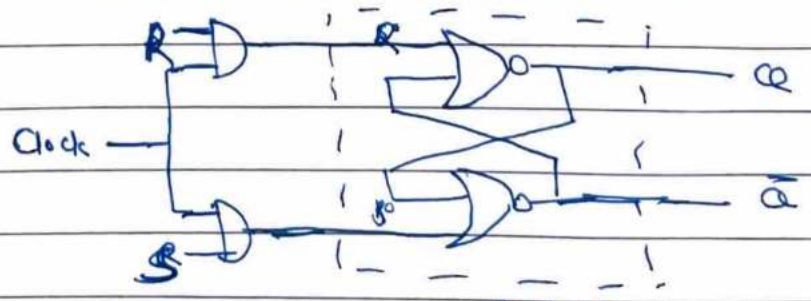
Clock	S	R	Q_{t+1}	
0	x	x	Q_t	(No change)
1	0	0	Q_t	
1	0	1	0	
1	1	0	1	
1	1	1	Never occurs	

Behaviour Table. / Char. Table

So when clock is 0, then it does not respond.
It has many implementations, but its behaviour is same for all.

⇒ **SR latch imp. of SR FF:**

In SR latch we want when clock=0 → S=0 & R=0 bcz then it will retain and when clock=1 → S & R work's
This can be done by adding ~~NOR~~ AND gate, i.e. when clock=0, i/p will be zero else as it is as:

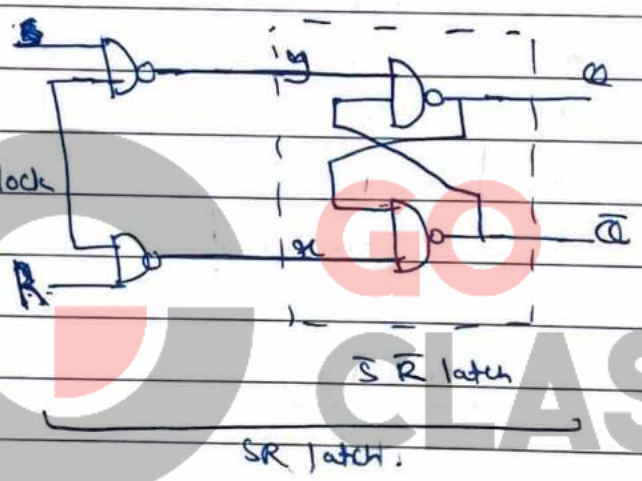


SR FF.

PAGE NO :
DATE : / /

In prev. ckt when clock=0, → $S^0=0$, $R^0=0$ else R & S are unchanged and this is what we wanted.
i.e. $S^0 = S$ & $R^0 = R$

⇒ **$\bar{S}\bar{R}$ latch imp. of SR ff:** Here we want when clock=0, then S=1 & R=1 so that it retain & when clock=1 then S & R should be vers-side. This can be done using NAND.

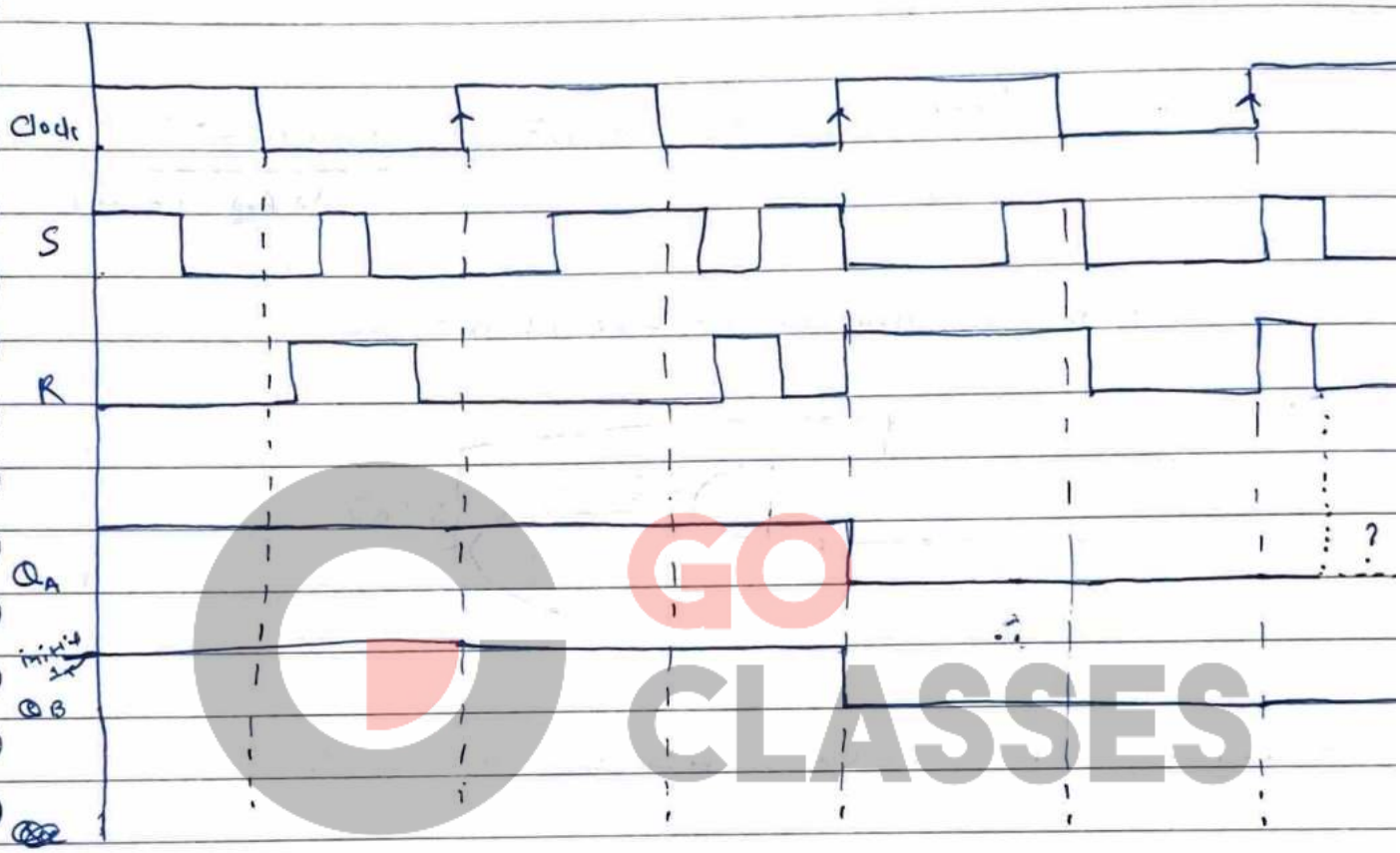


Now when Clock=0 → x=1, y=1 so Q is retained
when clock=1 → $x=\bar{R}$, $y=\bar{S}$ and this is what we wanted from char. table of SR ff.

Note that SR ff's behaviour is same unlike SR & $J\bar{R}$ latch which were active high & active low respectively.

PAGE NO.:
DATE: / /

⇒ Timing diagram for SR flipflop:



Q_A → Flipflop is +ive level triggered

Q_B → Flipflop is +ive edge triggered (told later)

→ In seq. circuit State = O/p = Stored data.

⇒ Char. / Behaviour table of SR FF:

Clock	S	R	Q_n
0	X	X	Q
1	0	0	Q
1	0	1	0
1	1	0	1
1	1	1	X

PAGE NO. :
DATE : / /

→ Truth table can be made with char. table as f(S, R, clock, Q)

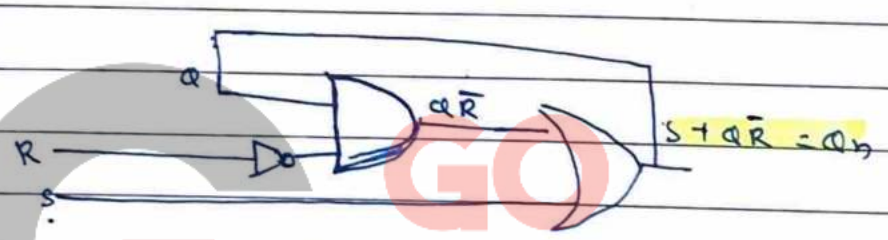
→ From T.T.:

Q_n \ SR			
0	0	X	1
1	0	X	1

$Q_n = S + Q\bar{R} = \underline{\text{Clock}(S + Q\bar{R})}$

↳ Exp (major)

So SR ff. can also be implemented as:

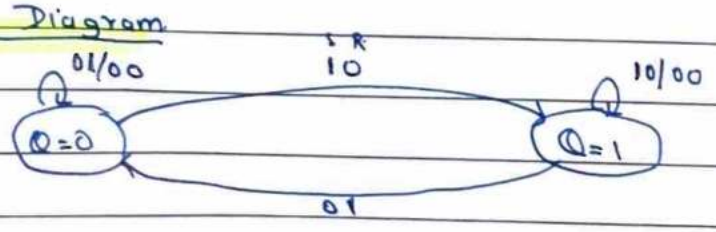


→ Excitation table: It shows behaviour from o/p POV.

Q	Q_n	S	R	
0	0	0	X	Reset or Retain 01 00
0	1	1	0	Set 10
1	0	0	1	Reset 01
1	1	X	0	Set or Retain 10 00

↳ Excitation table of SR flipflop.

→ State Diagram



↳ State diagram of SR ff.
(Can be made very easily from excitation table)

Clock is fix for all comp. of the

PAGE NO. :
DATE : / /

☆ D-flipflop Data flipflop

→ In SR ff we changed two signals just two store 1 bit. (i.e S, R) we overcome this in D-ff where we have just one signal.

→ In D-ff. simple idea is provide the bit on i/p and we will store it, no need to change two signals like SR.

→ In this simply $Q_n = Q$.

→ Characteristic Table for D ff:

Clock.	D	Q_n
0	X	Q
1	0	0
1	1	1

→ Char. Equation for D-ff:

$D = \text{Clock} \cdot Q_n$

↳ Independent of prev. O/P.

→ Truth table for D-ff:

Clock.	D	Q	Q_n
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1
0	X	X	Q

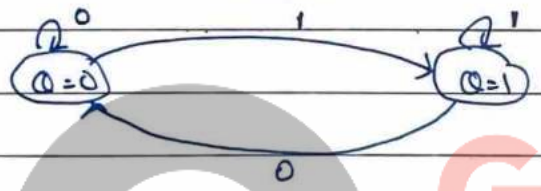
GO CLASSES

PAGE NO. :
DATE : / /

→ **Excitation table for D-ff:**

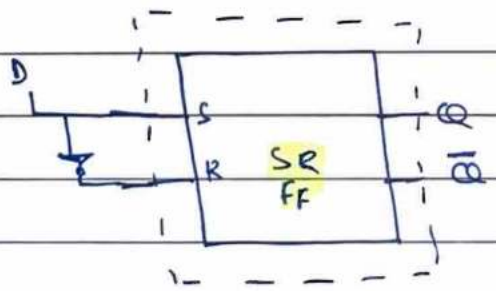
Q	Qn	D
0	0	0
0	1	1
1	0	0
1	1	1

→ **State Diagram for D-ff:**



→ **Implementations of D ff:**

a). From ~~char. equation~~ SR FF. (formal way later)



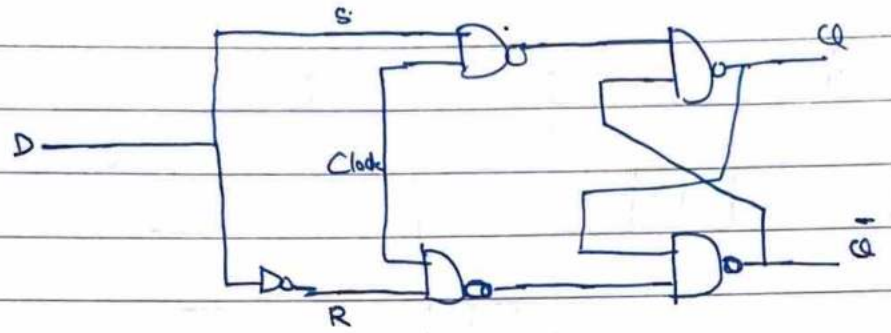
D ff.

when D=1, S=1, R=0, Q_n=1] Behaviour we wanted.
D=0, S=0, R=1, Qn=0.

GO CLASSES

PAGE NO.:
DATE: / /

More detailed imp. (from SR's NAND imp.)



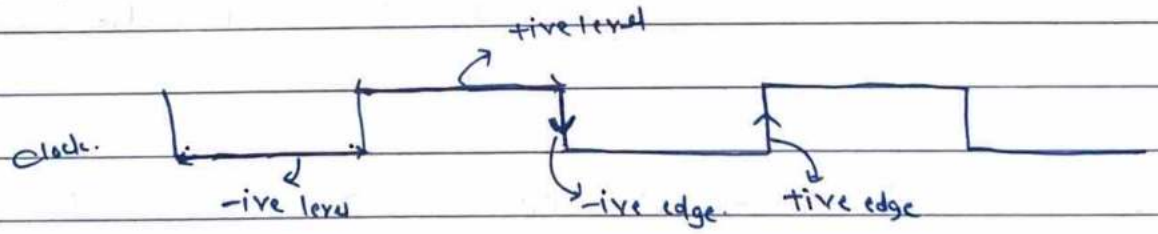
S=R is not at all possible in this circuit.

★ Clock Triggering (small detour)

→ Seq. circuit behave only when clock triggers. Now any seq. can respond to:

+ive level of clock] Level Triggered
-ive level of clock

Rising ← +ive edge of clock] Edge Triggered
-ive edge of clock] Falling ←

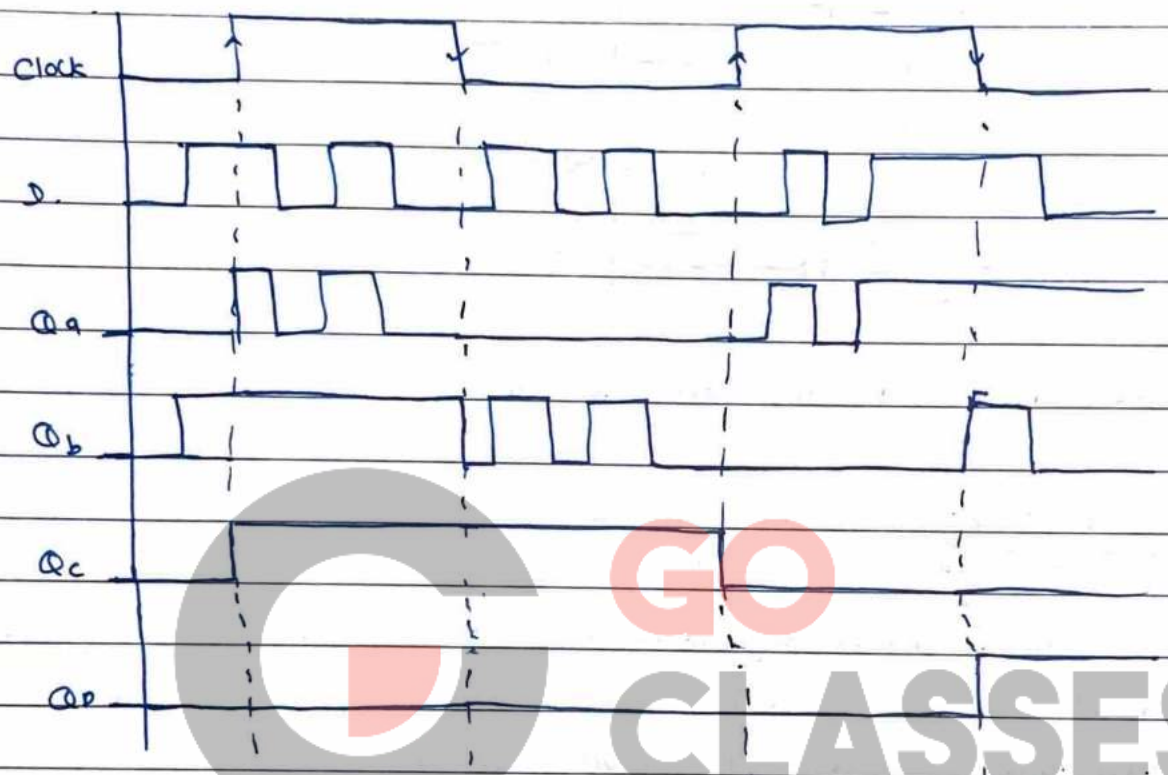


→ By default edge triggering is used bcz. we want usually that memory storage device can change its state atmost once in a cycle of clock.

→ Clock triggering is property of seq. circuit not the clock, clock is same always

PAGE NO. :
DATE : / /

→ Timing Diagram of D-FF:



Q_A → DFF is +ive level triggered
Q_B → " " -ive " "
Q_C → " " +ive edge "
Q_D → " " -ive " " .

→ for imp point ignore the clock i/p i.e. $Q_n = f(Q, D)$. But in actual it depends.

→ Any FF can have any type of clock triggering.

→ In -ive level clock triggering, O/P changes when clock=0 & in +ive " " " " " " " =1.

PAGE NO. :
DATE : / /

★ **J-K flipflop**: It is similar to SR ff, but, if i/p can be used here, it's not forbidden.

→ **Char. Table**

J	K	Q_n	
0	0	Q	(Retain)
0	1	0	(Reset)
1	0	1	(Set)
1	1	$\bar{Q}$	(Toggle)

→ only diff from JR

→

Q \ JK	00	01	11	10
Q	0	0	1	1
$\bar{K}Q$	1	0	0	1

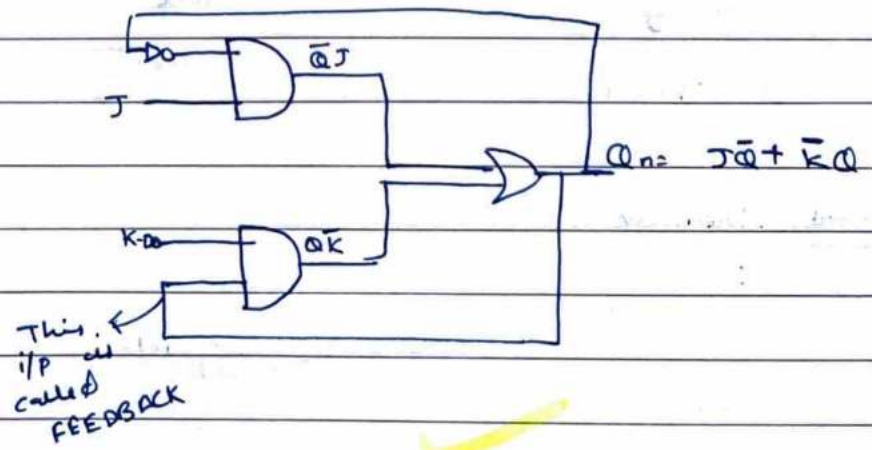
$J\bar{Q}$ → Redundant

$$Q_n = J\bar{Q} + \bar{K}Q.$$

↳ Char. Equation.

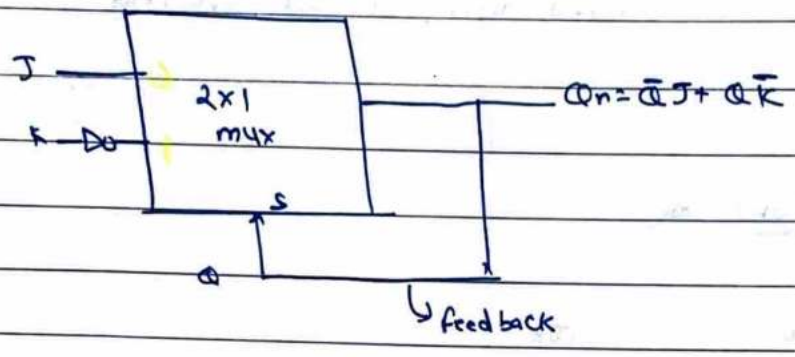
→ **Implementation**

a) Using char. equation



PAGE NO :
DATE : / /

It can also be imp as mux :

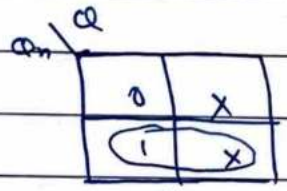


→ Truth table (easily designed from char. table)

→ **Excitation table:**

Q	Qn	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Reset | Retain
01 / 00
Set | Toggle
10 & 11
Reset | Toggle
01 | 11
Set | Retain
10 | 00



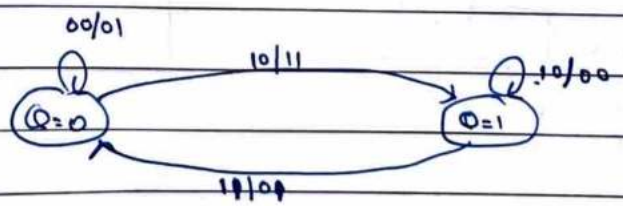
for J

$J = Q_n$

for K

$K = \overline{Q_n}$

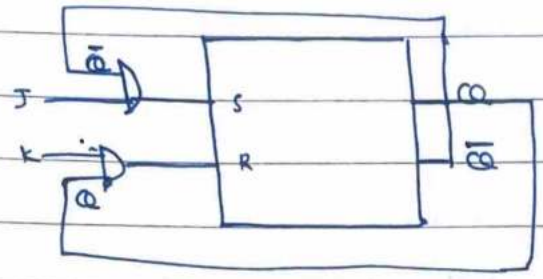
→ **State Diagram:**



PAGE NO. :
DATE : / /

★ Implementation of JK ff using SR ff.

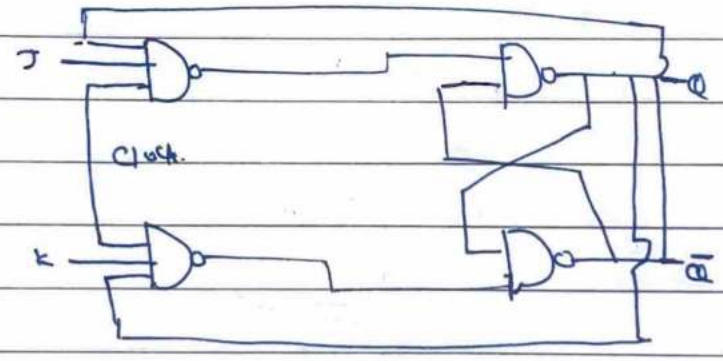
Idea: $J=1, K=1 \rightarrow Q_n = \bar{Q}$



$J=1, K=1, Q=1 \rightarrow S=0, R=1 \rightarrow Q=0$
$J=1, K=1, Q=0 \rightarrow S=1, R=1 \rightarrow Q=1$
] Togsling ✓

→ JK ff was invented by Jack Kilby

→ NAND imp. of JK.



for verifying in exam give i/p & check behaviour. (easy)

Ex:-

J	K	Q	Q_n
0	0	0	1
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Fill while revising.

GO CLASSES

PAGE NO :
DATE : / /

☆ T- flipflop: (Toggle FF)

In this if $T=1$, then $Q_n = \bar{Q}$
if $T=0$ then $Q_n = Q$.

So $T=1$ toggles the state.

→ Behaviour / Characteristic Table.

Clock	T	Q_n	
Triggered	0	Q	(Retain)
Triggered	1	$\bar{Q}$	(Toggle)
Not Triggered	X	Q	

→ State Equation.

T	Q	Q_n
0	0	0
0	1	1
1	0	1
1	1	0

So $Q_n = T\bar{Q} + Q\bar{T}$

$= T \oplus Q$.

→ Excitation Table

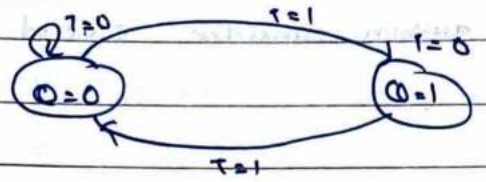
Q	Q_n	T	
0	0	0	(Retain
0	1	1	(Toggle)
1	0	1	(Toggle)
1	1	0	(Retain

So $T = Q \oplus Q_n$.

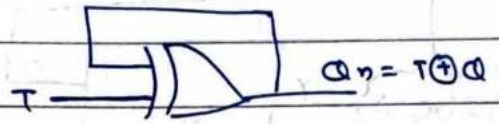
GO CLASSES

PAGE NO. :
DATE : / /

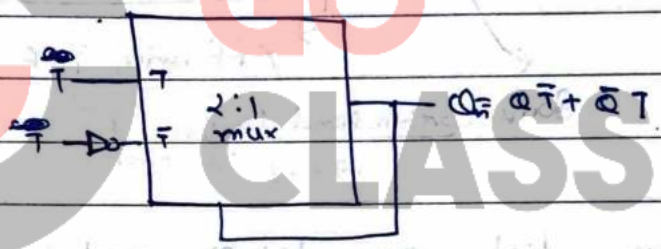
→ **State Diagram:**



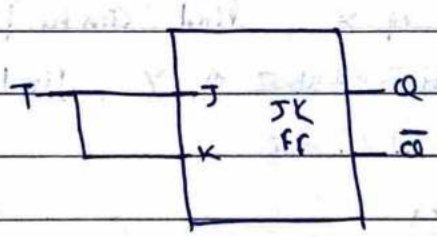
→ **Implementation:**



(Using XOR)



(using mux)



(Using JK FF)

T=1 → J=1, K=1 → Q = Q̄
T=0 → J=0, K=0 → Qn = Q.

→ Now, we've seen all 4 std ffs, now we'll convert them to one another.

GO CLASSES

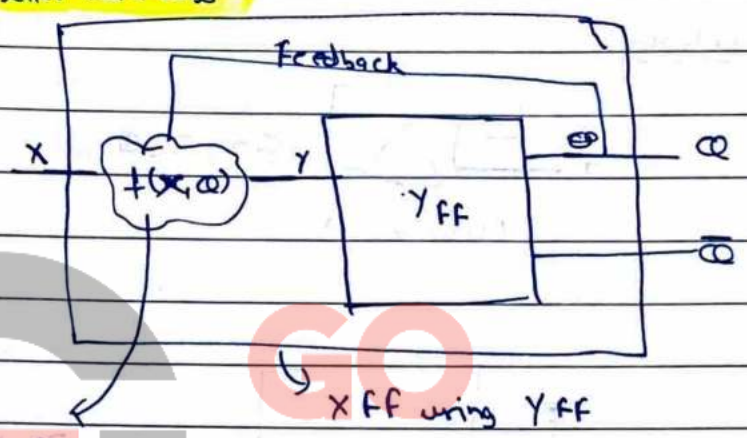
PAGE NO. :
DATE : / /

→ Some authors considered only edge Triggered latches as FF all other are latches but we & other authors consider clocked latches with any triggering as FF.

★★ **Flip Flop Conversion:** Implement flipflop "X" using flipflop "Y".

→ better than algo

Basic Idea:



So, we find some $f(X,Q)$ and give it as X.

from char. table of X, find Q_n as $f(X,Q)$
now. from excitation table of Y, find Y as $f(X,Q)$ using Q as Q_n.

→ Not Imp / remember idea

Procedure:

a. Write T.T. of desired FF.
b. Obtain value of i/p of given FF using its excitation table in above T.T.
c. Design necessary circuit.

PAGE NO. :
DATE : / /

→ **D to T. ff conversion**

Given: D. FF.

Desired: TFF

T	Q	Q_n	D
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

↳ TT of T-FF

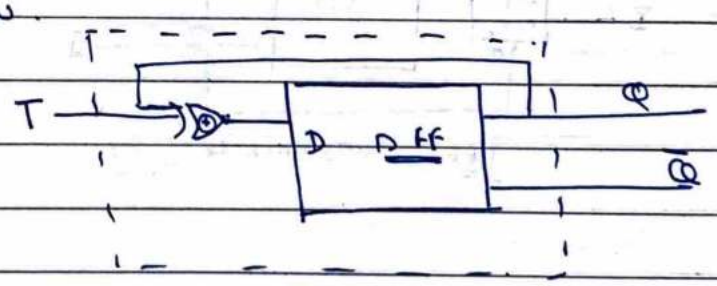
↳ Excitation table of D-FF.

Q \ T	0	1
	0	1
	1	0

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

$$D = T \oplus Q.$$

So, now.



T ff

now this XOR can be given as AND-OR, MUX, NAND, etc.

PAGE NO. :
DATE : / /

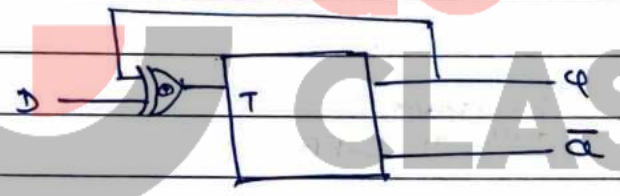
→ **T to D Conversion:**

Given : T ff.

Desired : D ff.

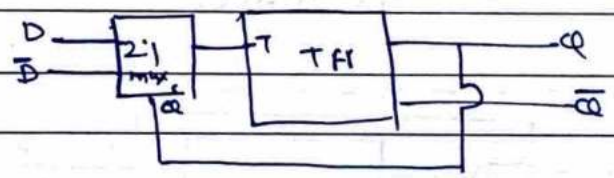
D	Q	Q_n	T
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

$T = D \oplus Q$.



D ff using T ff.

|||



DFF using mux & TFF.

GO CLASSES

PAGE NO. :
DATE : / /

→ **D to JK ff.**

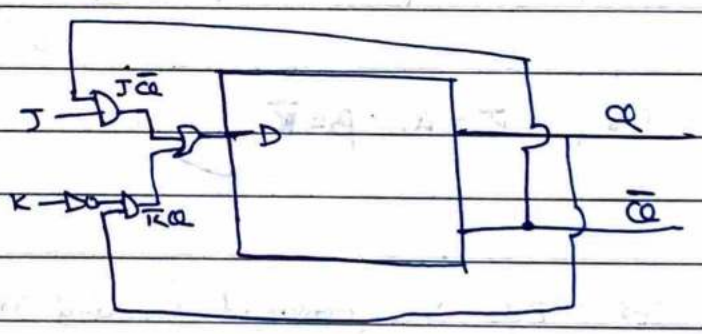
Given : D

Desired : JK

J	K	Q	Q_n (using as T_{JK})	D (using T_{JK} as D ff.)
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

Q \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1

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



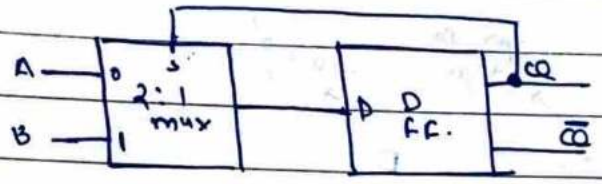
JK ff using D ff.

(can also be done using mux).

GO CLASSES

PAGE NO. :
DATE : / /

Qu: Relate AB-ff to JK-ff ?



→ When $A=0, B=0 \rightarrow Q_n = 0$ (Reset)
$A=1, B=1 \rightarrow Q_n = 1$ (Set)
$A=0, B=1 \rightarrow Q_n = Q$ (Retain)
$A=1, B=0 \rightarrow Q_n = \overline{Q}$ (Toggle)
} from given circuit

A	B	Q_n	J	K
0	0	0	0	1
0	1	Q	0	0
1	0	$\overline{Q}$	1	1
1	1	1	1	0

So, $J = A$, $B = \overline{K}$

→ Any ff can be converted to any other ff, remember the idea not the algo (If x using y then $y = f(x)$).

GO CLASSES

Rem. whose TT & whose ET.

PAGE NO. :
DATE : / /

→ **T to SR FF.**

Given: T, Desired: SR

S	R	Q	Q_n	T
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	0
1	1	0	X	] can't occur
1	1	1	X	

$T = \bar{S}RQ + S\bar{R}\bar{Q}$ (SOP)

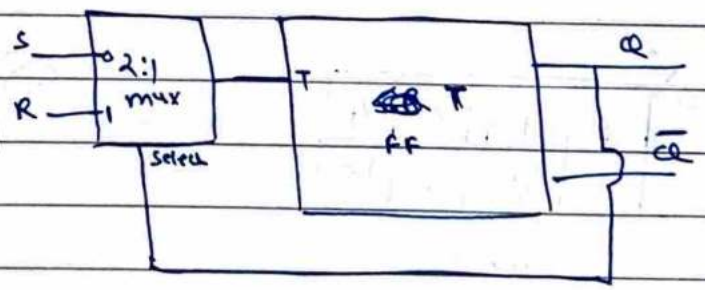
↳ Not min. for min. use K-map

SR \ Q				
	0	0	X	1 → $S\bar{Q}$
	0	1	X	0

RQ

$T = S\bar{Q} + RQ$

Now

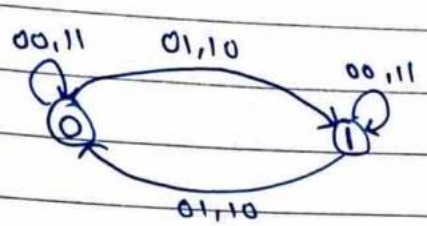


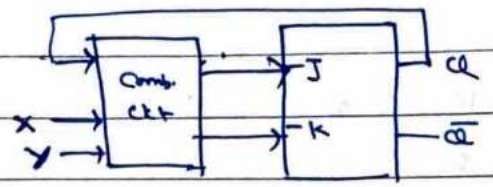
To SR FF using TFF.

PAGE NO :
DATE : / /

Gate IT 08

Ques: Consider following state diagram & it's real. from JK ff.





The comb. circuit gen. J & K in terms of x, y and Q.
Bool. exp. for J & K is?

⇒ **M-1**: Inefficient

Get T.T. of XY from state diagram & JK from exc. table

X	Y	Q	Qn	J	K
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	1	1	X
0	1	1	0	X	1
1	0	0	1	1	X
1	0	1	0	X	1
1	1	0	0	0	X
1	1	1	1	X	0

K-Map:

Q \ XY				
	0	1	0	1
	X	X	X	X

for J

$J = X\bar{Y} + Y\bar{X}$

Q \ XY				
	X	X	X	X
	0	1	0	1

for K

$K = X\bar{Y} + \bar{X}Y$

So $J = X \oplus Y$ and $K = X \oplus Y$.

Lengthy Interesting Ques

↳ Inefficient Method.

GO CLASSES

PAGE NO. :
DATE : / /

→ **Method 2 :** Option Elimination.

from State diagram

X	Y	Q_n	J	K
0	0	Q	0	0
0	1	$\bar{Q}$	1	1
1	0	$\bar{Q}$	1	1
1	1	Q	0	0

} Only 1 option matches done here

if $x=y=1$ → $Q_n = Q$ → Then J & K must be 1 & 1

So $J = x \oplus y$ and $K = x \oplus y$. (Easy & Efficient method)

→ JK ff to SR ff is easy bcz JK is already SR, just forbid J=1 & K=1 for it to be SR.

→ DIY SR ff to JK ff. (Work Revising).

J	K	Q	Q_n	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

GO CLASSES

PAGE NO :
DATE : / /

S:

$Q Q_n$ \ JK	00	01	11	10
00	0		X	
01	0			0
11				
10				

R:

$Q Q_n$ \ JK	00	01	11	10
00	X		0	
01	X			1
11				
10				

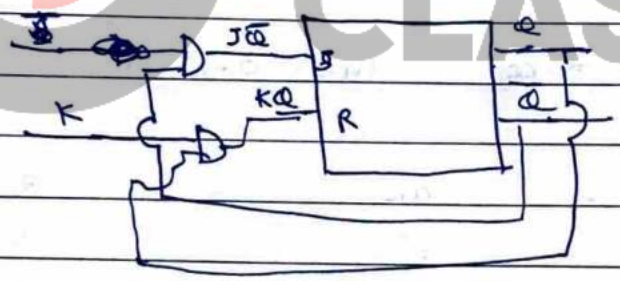
S_s

Q \ JK	00	01	11	10
0	0 0	0 2	1 1 6	1 4
1	X 1	0 3	0 7	X 5

R_s

Q \ JK	00	01	11	10
0	X 0	X 2	0 6	0 4
1	0 1	1 3	1 7	0 5

$S = \bar{Q}J$, $R = QK$



GO CLASSES

PAGE NO. :
DATE : / /

★ Special inputs in flipflops: Preset & Reset (or clear)

→ When ff is just turned on, we don't know their state. We use direct i/ps (reset & preset) to bring all ff in the system to a know starting state prior to clocked operation.

→ Preset makes the state '1' & Clear make state '0' of the flipflop.

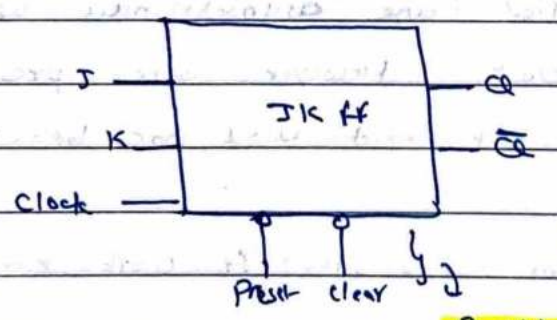
→ Direct i/ps here i.e. Preset & clear are active low i.e. work when value is 0.

→ T.T.

Preset	Clear	Clock	J	K	Q_n
0	1	x	x	x	1
1	0	x	x	x	0
1	1	Normal JK Behaviour			
0	0	x	x	x	x

Cannot occur / forbidden

→ Block Diagram:



Bubble Indicates active low

→ Preset & Clear can be used with any type of ff.

GO CLASSES

GO
CLASSES

GO CLASSES

GO CLASSES

GO CLASSES

GO CLASSES

GO CLASSES

GO CLASSES

GO CLASSES

GO
CLASSES

GO CLASSES

GO
CLASSES

GO CLASSES

GO CLASSES

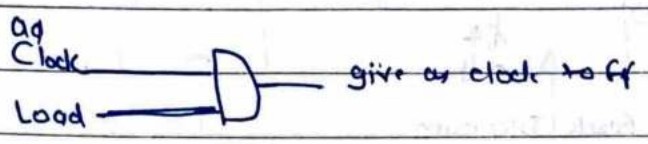
GO CLASSES

GO CLASSES

All registers are synchronous seq. ckt.

PAGE NO.:
DATE: / /

→ We could also do AND of clock and load and give that as clock, but not a good sol^n as can cause some timing problems.

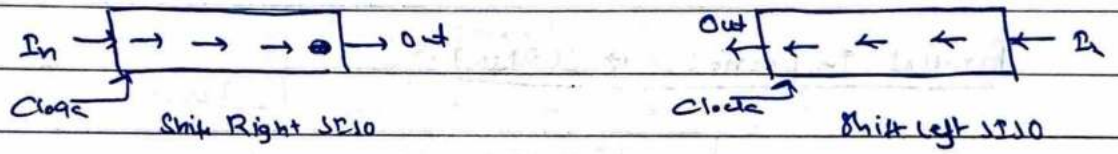


→ This PIPO was the only Parallel access register. Now the remaining registers we'll see are shift registers.

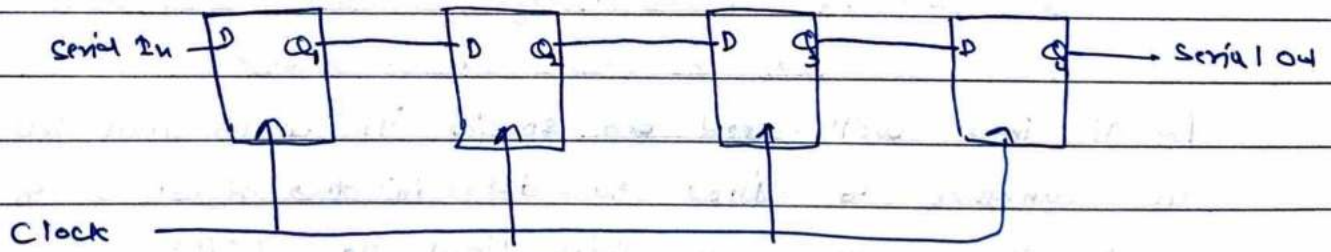
It is a register in which binary data can be stored and this data can be shifted to the left or right when a shift signal is applied. Bits shifted out may be lost, or can move to other end if shift register is cyclic.

GO CLASSES

B) Serial In | Serial Out: This is the simplest shift register.



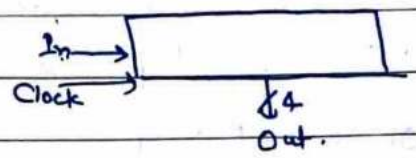
Imp. using D ff:



	Q_1	Q_2	Q_3	Q_4	
Initially	0	0	0	0	1 clock edge
In = 1	1	0	0	0	
In = 0	0	1	0	0	
In = 0	0	0	1	0	
In = 1	1	0	0	1	Output finally.
In = 1	1	1	0	0	

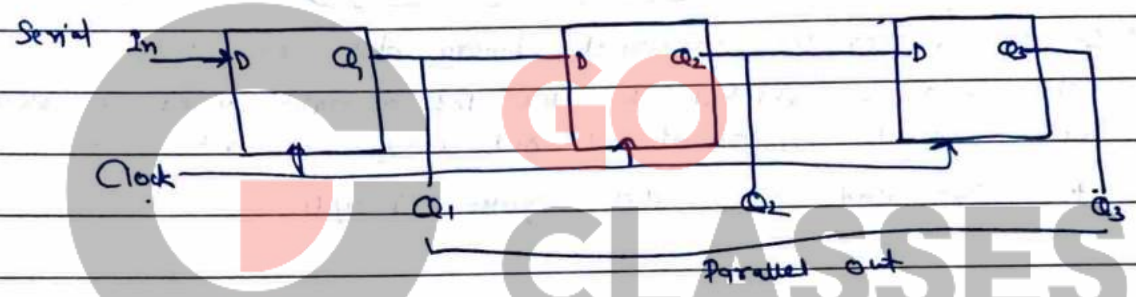
PAGE NO :
DATE : / /

C). Serial In | Parallel Out (SIPO):



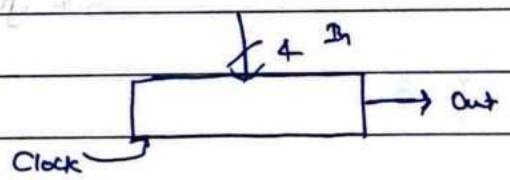
Block Diagram.

Implementation using D-FF:



3bit SIPO.

* D) Parallel In | Serial out (PISO):



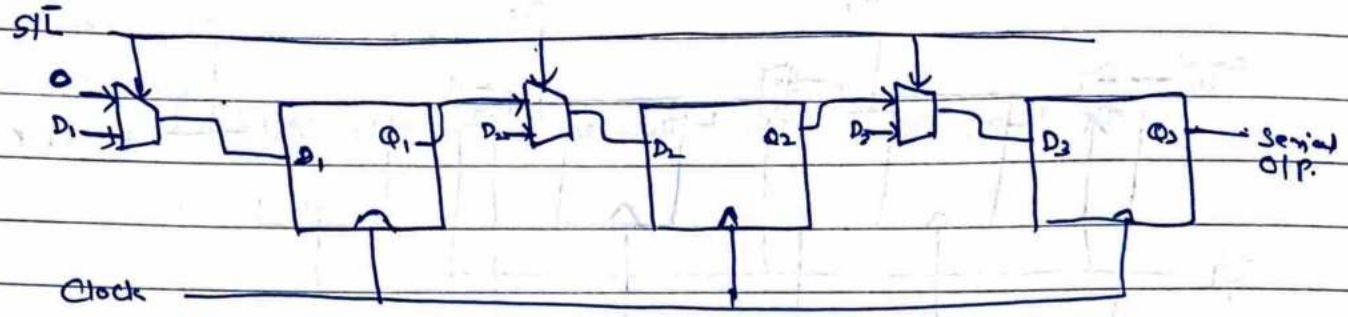
for its imp. we'll need an special i/p which will tell us whether to input the data in this signal or to shift it. This is shift /load i/p (S/L)

when $\bar{S}/L = 0$ → shift the data

$\bar{S}/L = 1$ → load the data.

PAGE NO. :
DATE : / /

→ Again we can use mux for it:

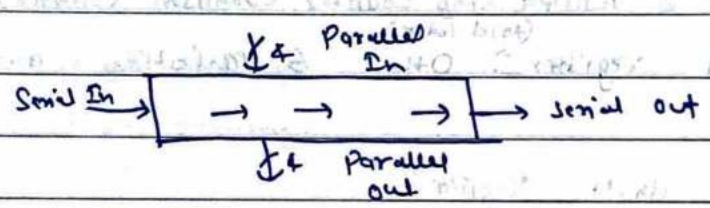


3 bit PISO.

	D_1	D_2	D_3	$\bar{S}/L$	O/P.	
Now initially, →	0	0	0	0	-	
	1	0	1	1	0	(Load)
	0	1	0	0	1	
	0	0	1	0	0	
	1	0	0	1	1	(Load).

GO CLASSES

E) Parallel-Access Shift Register: (PASR)



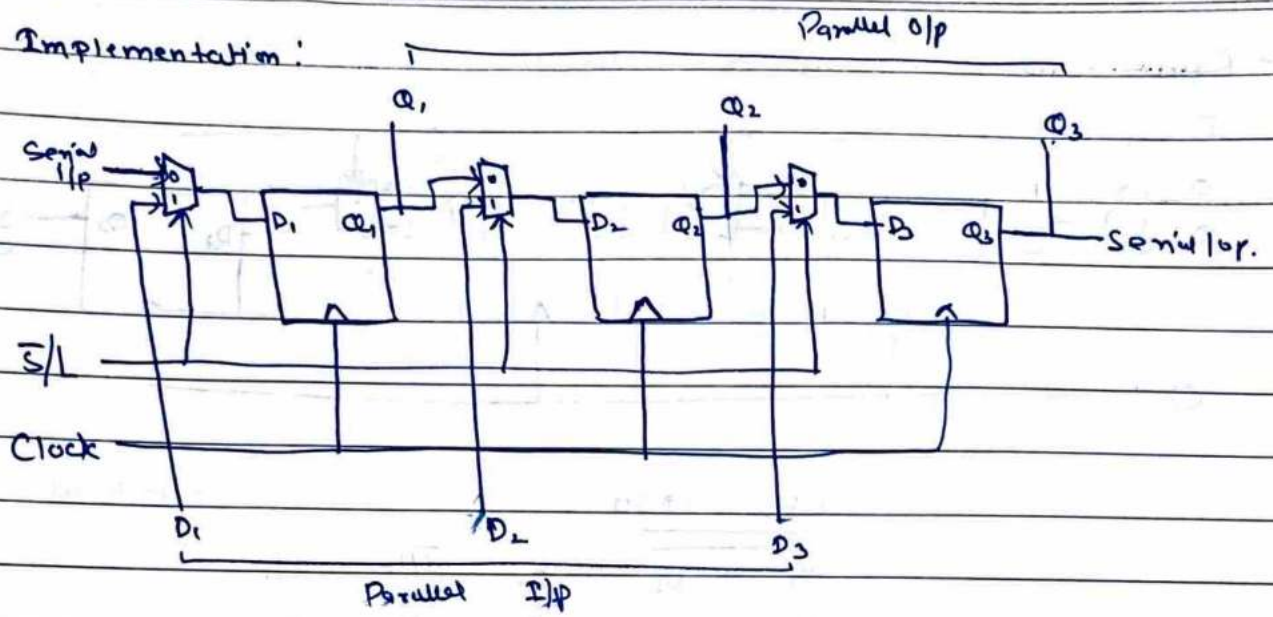
It is a special register which we can use for parallel access or for shift register

If $\bar{S}/L = 1$ then behave like PIPO

If $\bar{S}/L = 0$ then behave like SISO

PAGE NO :
DATE : / /

Implementation :



4 bit PISR

when S/L = 0 it acts as shift register &
when S/L = 1 , it acts as parallel register.

NOTE Ring Counter & Twisted Ring Counter (Johnson Counter) are variations of shift register. Other 3 variations are :
(Fold later)

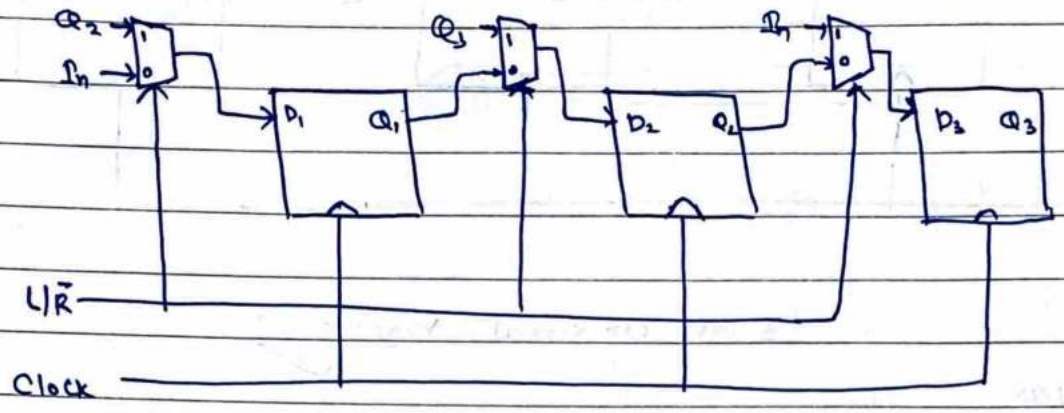
A) **Bidirectional shift Register :**

Using i/p L/$\bar{R}$ we ~~choose~~ control register as unidirectional shift left or unidirectional shift right register.

When L/$\bar{R}$ = 0 , then shift Right,
When L/$\bar{R}$ = 1 , then shift left

for cyclic shift register give last o/p

Implementation:

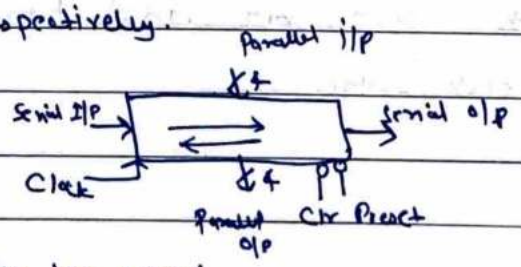


3-bit bidirectional shift Register.

Q) Universal shift Register: It is most general shift register having following capabilities:

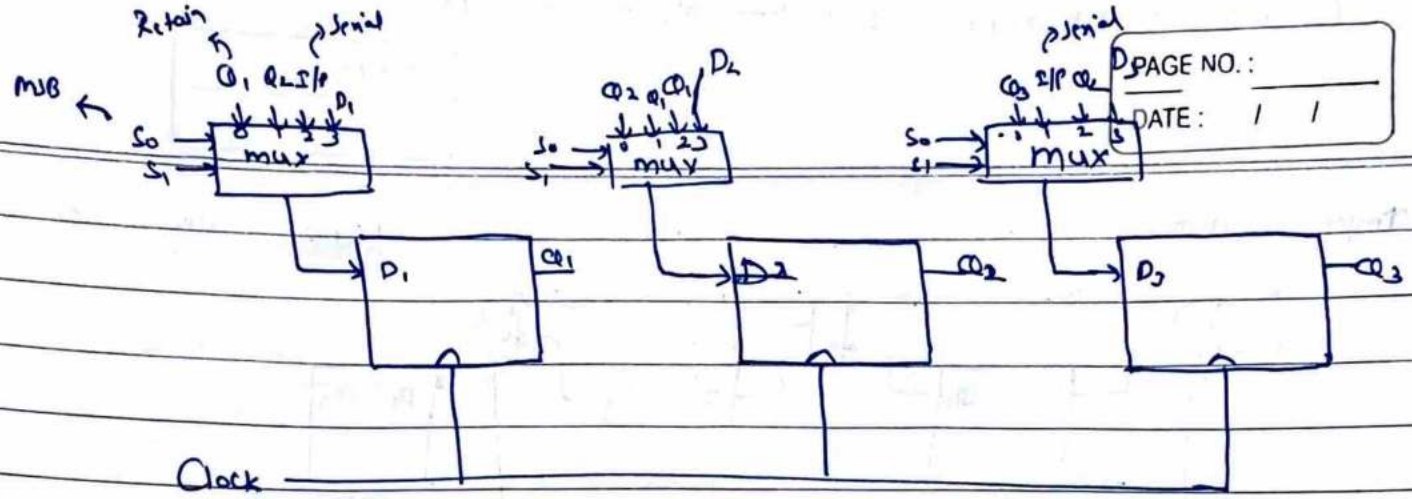
- → Clear to Preset (Actively low)
- → Clock i/p to synchronise.
- → Shift Right
- → Shift left
- → Parallel i/p | Parallel o/p.
- → State that retains.

for such a register we will use 4:1 mux and do the combination respectively.



Two control i/p for mux:

S_0	S_1	Action.
0	0	Retain
0	1	Shift Left
1	0	Shift Right
1	1	P LOAD

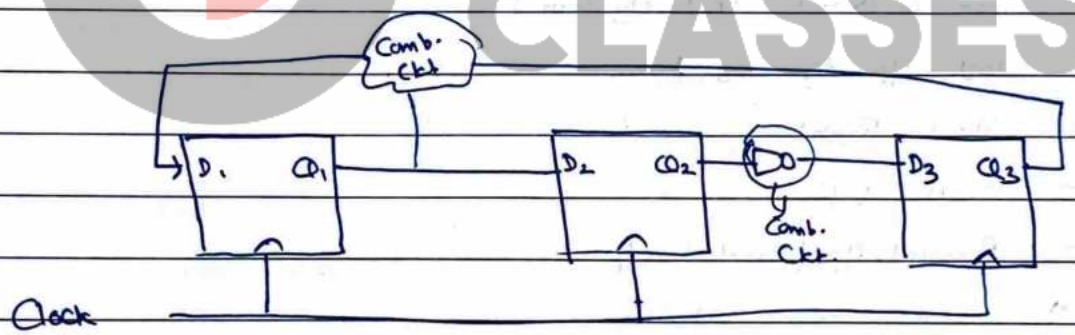


3 bit universal register

gr% on this

** HJ. **Linear feedback shift Register (LFSR)** : It is basically a SISO right shift register where the serial i/p is generated as a linear combination (XOR) of some of the FF o/p.

Ex:



We initialise with some bits and then calculate next state after each clock edge.

Applications:

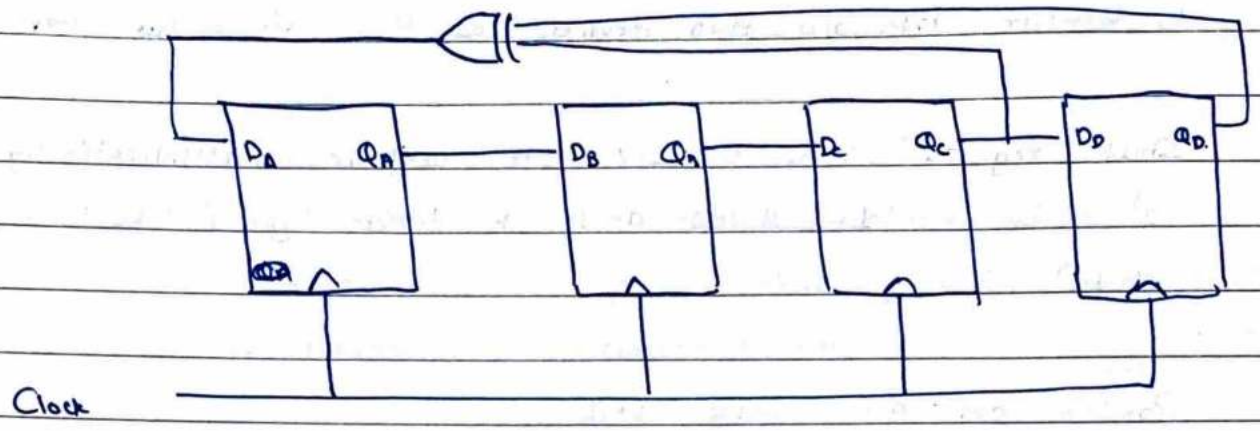
→ Random no. generation.

→ Comb. ckt gen. is XOR of o/ps, so 0...0 is not gen. generated and also can't go to any other state from 0...0.

PAGE NO. :
DATE : / /

CJ-87.

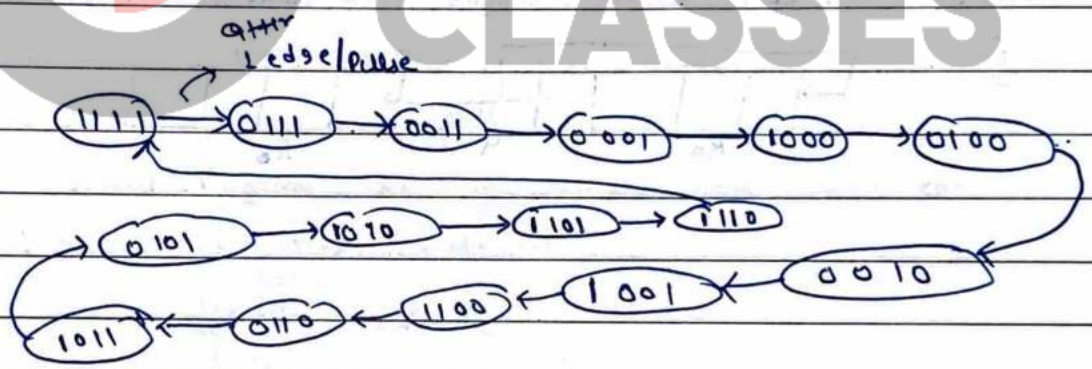
Que: If initial state is 1111 for following circuit (MSB is Q_D) then state after 4th pulse will be:



⇒ Let's analyze it fully:

$$Q_A, Q_B, Q_C, Q_D \longrightarrow Q_C \oplus Q_D, Q_A, Q_B, Q_C$$

So.



After 4th pulse it will be 1000

It is gen. 15 distinct states (all except 0000)

GO CLASSES

PAGE NO. :
DATE : / /

★ Some applications of registers are:

→ Parallel to serial and serial to parallel conversion. (Req. to transfer data b/w two devices as there is no bus there).

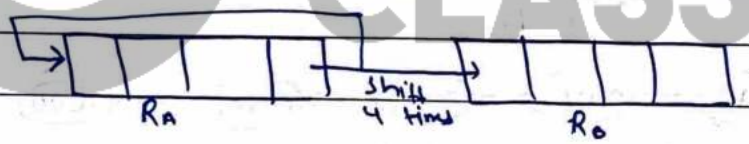
→ Shift registers can be used to perform multiplication by 2^k (by k left shifts) and to divide by 2^k (by k right shifts).

→ Random seq. gen. using LFSR.

→ Delaying a pulse using SISO shift register.
(as initial pulses are not useful to delayed).

→ Serial Transfer i.e. Copying RA to RB using SISO.

Ex:



GO CLASSES

PAGE NO. :
DATE : / /

* Miscellaneous of Seq. Ckts.

→ In seq. circuits the states of the circuit is crucial for determining the o/p. values. for a given i/p, a seq. circuit can produce diff. o/p based on prev. state.

→ A seq. circuit has following representation:

- Excitation Table
- State Table / Behaviour Table / Char. Table
- State Diagram
- Next state eqn [f(i/p, current state)]
  ↳ from state table or state diagram
- Output equation [f(i/p, current state)]
  ↳ from state diagram
    ↳ In FSM

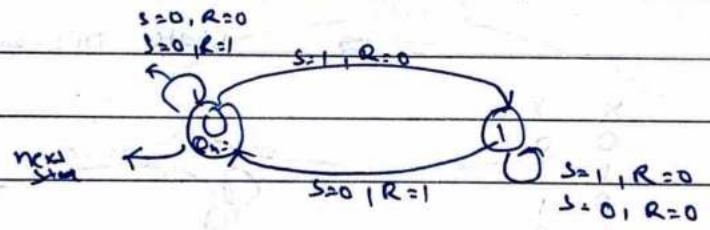
→ for a ff o/p is Q i.e. bit stored in FF. (also called state of FF).

→ State of register ≡ ffi o/p combination ≡ register content

→ Present states is state before next clock triggered and Next state is state after next clock triggers.

→ from state diagram we can write next state eqn for those variables when next state is 1.

Ex:



Now

$$Q_n = (S=1, R=Q_n, Q=1) \text{ OR } (S=0, R=0, Q=1), \text{ OR } (S=1, R=0, Q=0)$$

$$Q_n = S\bar{R}Q + \bar{S}\bar{R}Q + S\bar{R}\bar{Q}$$

↳ Not min. but correct.

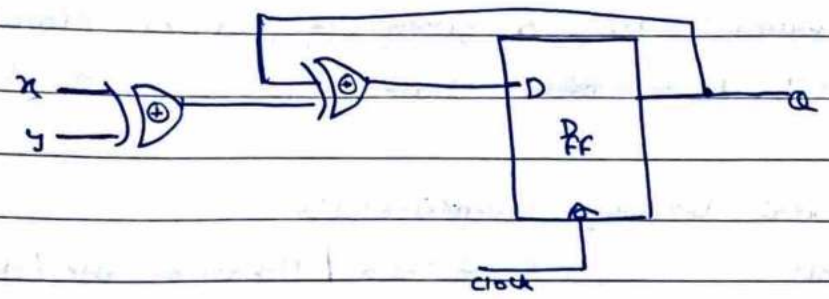
Similarly can get easily from state table, using k-map.

→ No o/p eqn is true for flipflops, its for FSM.

PAGE NO. :
DATE : / /

Qu: Analyse following sequential circuit :

A)



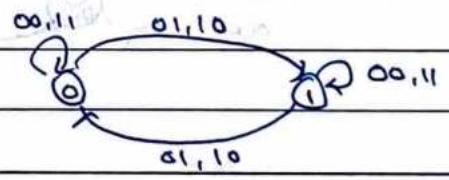
⇒ State Table :

Q	x	y	Qn
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Excitation Table :

Q	Qn	x	y
0	0	0	0
		1	1
0	1	0	1
		1	0
1	0	0	1
		1	0
1	1	0	0
		1	1

⇒ State Diagram :



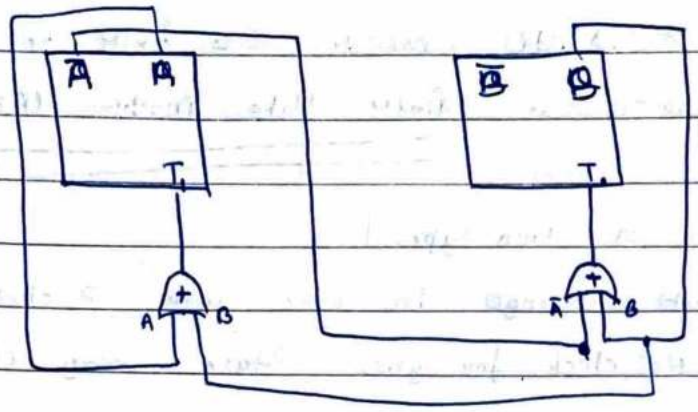
$$Q_n = \underbrace{\bar{x}y\bar{Q} + x\bar{y}\bar{Q} + \bar{x}\bar{y}Q + xyQ}$$

↳ next state equation from state diagram

GO CLASSES

PAGE NO. :
DATE : / /

B)

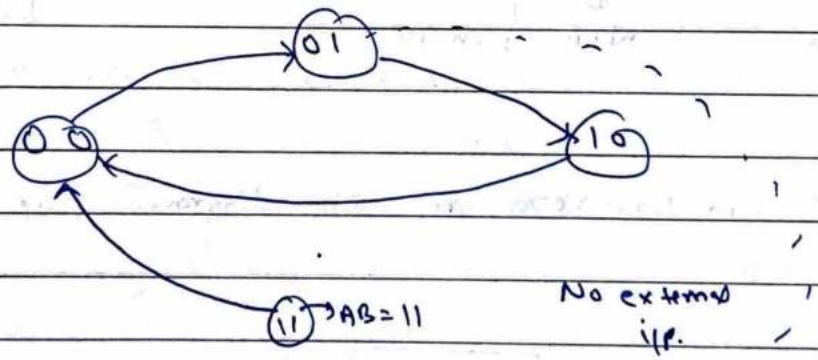


$\Rightarrow$ $T_1 = A + B$, $T_0 = \bar{A} + B$

State Table: (No external i/ps here)

A	B	A_n (from T_0 & A,B)	B_n	T_1	T_0 (from A&B)
0	0	0	1	0	1
0	1	1	0	1	1
1	0	0	0	1	0
1	1	0	0	1	1

$A_n = \bar{A}B$
$B_n = \bar{A}\bar{B}$ } Next state equation



No external i/p.

Can get eqn as $A_n = 1$ when $A=0, B=1 = \bar{A}B$
$B_n = 1$ when $A=0, B=0 = \bar{A}\bar{B}$ } next state eqn from state diagram

→ In seq. ckts we use flipflops to store internal states & using k ffs we can have atmost 2^k internal states.

GO CLASSES

PAGE NO :
DATE : / /

Now as the no. of states possible are finite i.e. 2^k, seq. circuit is also known as finite state machine (FSM).

→ Seq. circuits can be of two types:

a). Synchronous: state changes in sync. with a clock pulse.

b). Asynchronous: No clock for sync. State may change in b/w.

more detail on this in counters.

★★ Finite State Machine (FSM): Synchronous Sequential Machine.

FSM.

- Automaton → Accepts / Rejects → So o/p is 1 or 0
  - DFA
  - NFA. } In TOC
- Transducer → o/p is f(i/p)
  - any sync. seq. ckt
  - Mealy
  - Moore
  - ! } Not in GATE.

→ A FSM can be repr. as state diagram (we did in TOC) and state table.

$w \rightarrow \boxed{\text{Automaton}} \rightarrow$ Yes / No

$w \rightarrow \boxed{\text{Transducer}} \rightarrow f(w)$

→ Here we also have outputs.

→ For this chapter FSM = Finite state Transducers.

GO CLASSES

PAGE NO. :
DATE : / /

Ex: FSM circuit to detect 3 or more 1's in a serial bit stream. O/P is 1 when it detects 3 or more 1's else 0. With every clock one i/p is given.

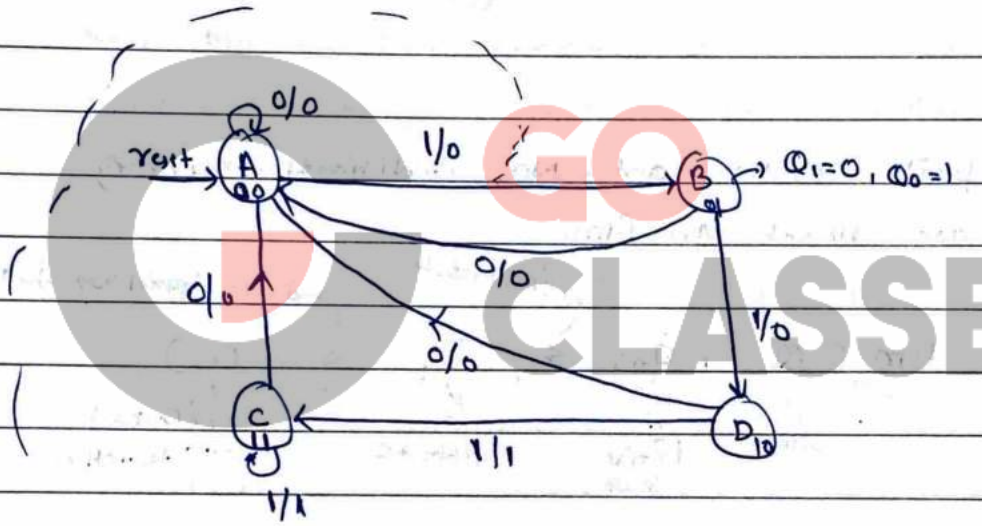
like

I/P O/P

0111010 → FSM → 01100000 (Read Right to left)

w f(w)

State diagram: x/y means x i/p and y o/p



From state diagram, next state equation can be:

$Q_1n = \bar{Q}_1 Q_0 x + Q_1 \bar{Q}_0 x + Q_1 Q_0 x$ } when Q_1 is 1

$Q_0n = \bar{Q}_1 \bar{Q}_0 x + Q_1 \bar{Q}_0 x + Q_1 Q_0 x$ } when Q_0 is 1

this transition — Q_0 is 1 when $Q_1 = 0, Q_0 = 0$ and $x = 1$

Similarly next o/p equation can be:

$z =$ o/p is 1 when $(Q_1 = 1, Q_0 = 0, x = 1)$ or $(Q_1 = 1, Q_0 = 1, x = 1)$

$z = Q_1 \bar{Q}_0 x + Q_1 Q_0 x$

$= Q_1 x$ → output equation.

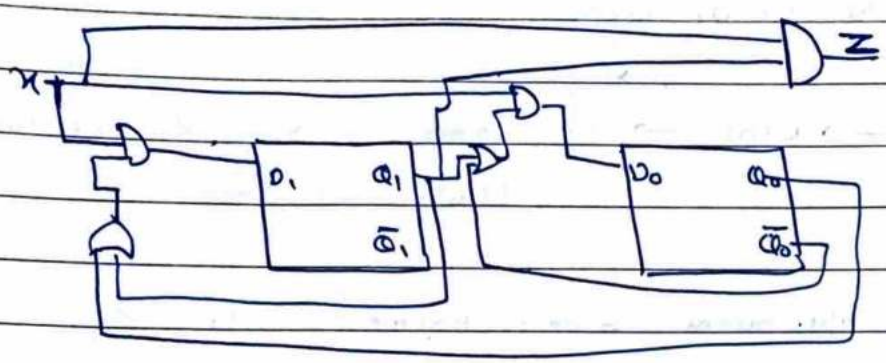
PAGE NO. :
DATE : / /

Now the seq. circuit can be: (using DFF)

O/P Eqn: $z = Q_1 x$

next state $Q_{0n} = x(Q_1 + \overline{Q_0}) = D_0$

$Q_{1n} = x(Q_0 + Q_1) = D_1$



→ FSM has finite states and NO additional memory.

→ Automaton are subset of FSMs

FSM: $FSM(Q, q_0, \Sigma, \Gamma, \delta, \omega)$

Q → states, q_0 → initial state, Σ → i/p alphabets, Γ → o/p alphabets, δ → state transition function, ω → output function

→ O/P of automaton was Yes/No so we had final states, here o/p is function so we have o/p function.

→ Two types of FSM in syllabus that differ only in O/P function
- ↳ Mealy m/c (GH Mealy in 1955)
- ↳ Moore m/chine (EF Moore in 1956)

$\delta : Q \times \Sigma \to Q$

$\omega : Q \times \Sigma \to \Gamma$ - In mealy m/c (o/p depends on i/p + state)

$Q \to \Gamma$ In moore m/ch (o/p depends on state)

PAGE NO. :
DATE : / /

So O/P = fun.
- f(present state, i/p) → Mealy m/c
- f(present state) → Moore m/c

State trans. function } same for both moore & mealy. $\delta : Q \times \Sigma \to Q$.

→ Mealy m/c's state diagram will have x/y notation on state transition showing i/p & prev. state giving o/p & next state

→ Moore m/c's state diagram will have x notation on transition showing i/p & prev. state giving next state. Each state will have o/p. ~~letting~~

w: $\underbrace{(q_0, 0) \to 1}_{\text{Mealy m/c}}$ w: $\underbrace{q_0 \to 1}_{\text{Moore m/c}}$

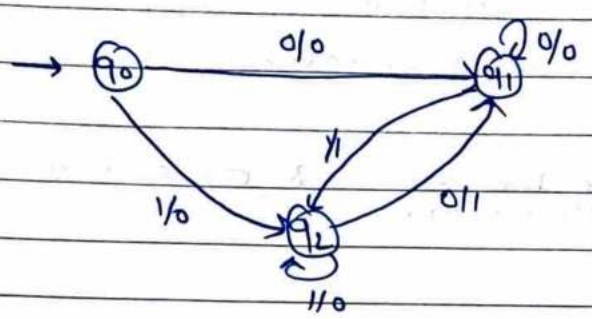
→ n length i/p n length o/p in mealy machine.

→ In moore m/c n length i/p
Author dep. o/p
- n length o/p
- n+1 length o/p
  - It consider initial state o/p also. (pre-output)

GO CLASSES

PAGE NO. :
DATE : / /

Ex: 1



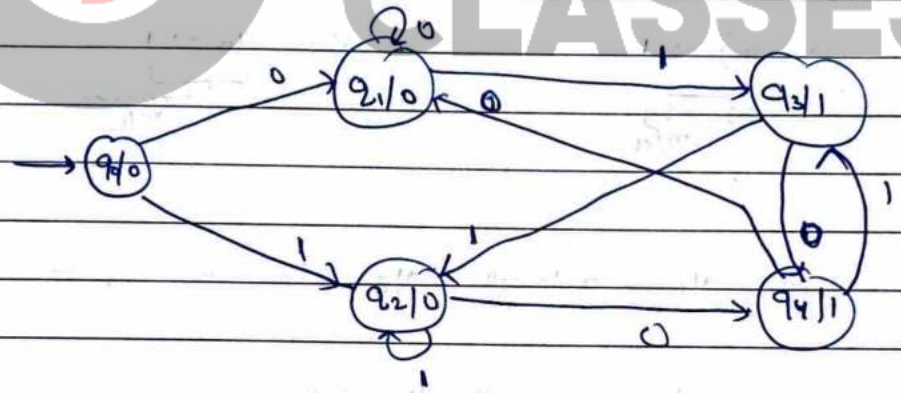
i/p = 0 0 1 0 1 1

q_0 q_1 q_1 q_1 q_1 q_2 q_2 → At last in this state.

o/p = 0 0 1 1 1 0 ✓

Mealy m/ch.

Ex: 2



i/p : 0 1 1 0 0

q_0 q_1 q_3 q_2 q_4 q_1

o/p : 0 1 0 1 0

Pre o/p ← 0 ← Another dep.

GO CLASSES

PAGE NO. :
DATE : / /

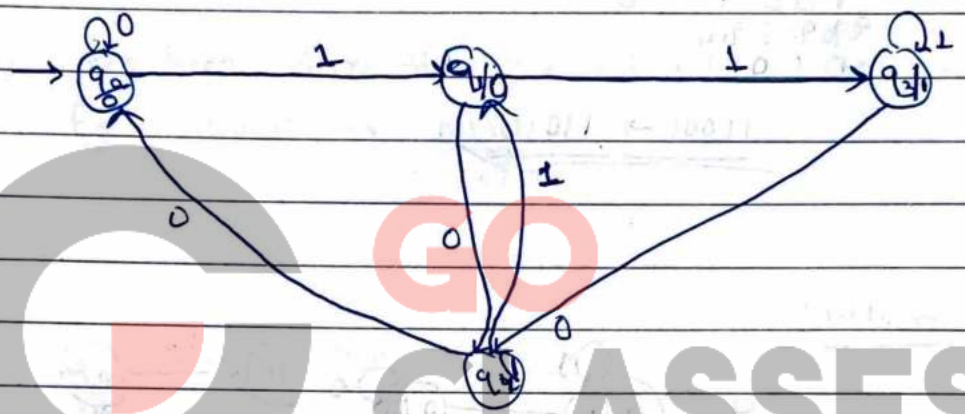
⇒ Some Mealy and Moore machine designs:

A Divide an i/p into half.

Ex: 100 → 010

101 → 010

Moore Machine: (Assuming i/p read from ~~Right to LSB~~ MSB to LSB)



half of 1100 is 110

1100
$q_0\ q_1\ q_2\ q_3\ q_4\ q_5$
0 0 1 1 0

Just shifting 1 bit ahead. (towards right)

B. Increment the input no by 1. (mag.)

W → FSM → |w+1|

Idea: If LSB is 0, then make it 1.

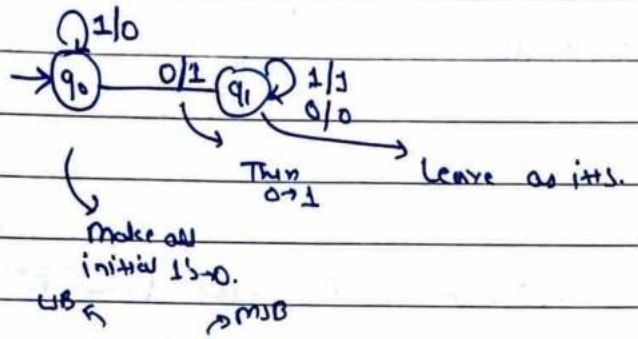
Ex: 1001001011100<u>0</u> +1 = 1001001011101<u>1</u>

If LSB is 1, then make all bits 0 till we don't find 0, make this 0 as 1

Ex: 110011 +1 = 110100

PAGE NO. :
DATE : / /

Now, assuming mealy reads from R to L (LSB to MSB)

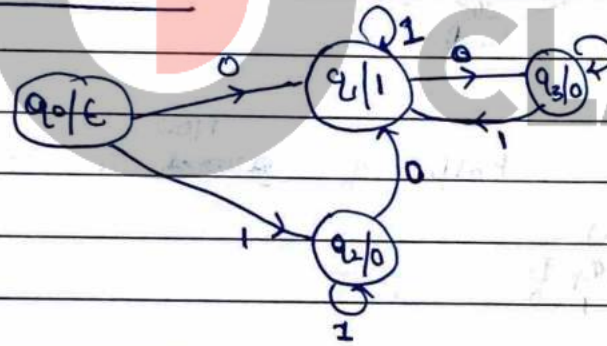


Ex: LSB ← → MSB

1 0 0 1 1
q_4 q_3 q_2 q_1 q_0
0 1 0 1 1

11001 → 11010.

Moore machine:



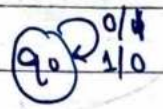
Idea is imp., machines are easily drawable.

C Decrement mag. by 1

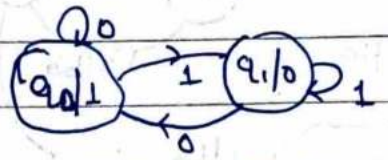
⇒ Idea is if LSB is 1 make it 0
If LSB is 0, make it 1 till you find 1st 1, make this 1→0.

PAGE NO. :
DATE : / /

D. 1's complement:



Mealy m/c

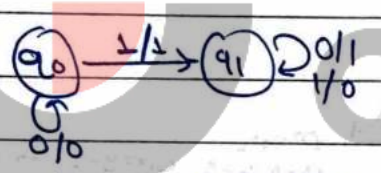


Moore machine

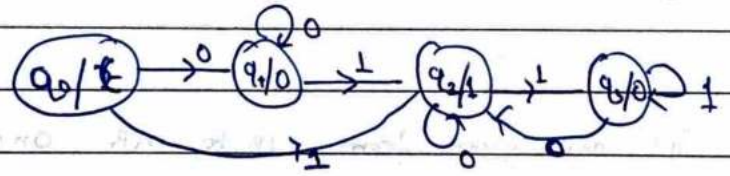
E. 2's Complement (Assuming i/p given from LSB to MSB)

→ Idea: keep same till you find first 1, and then flip.

Ex: 10010 → 01110.
2's



Mealy m/c.



Moore m/c.

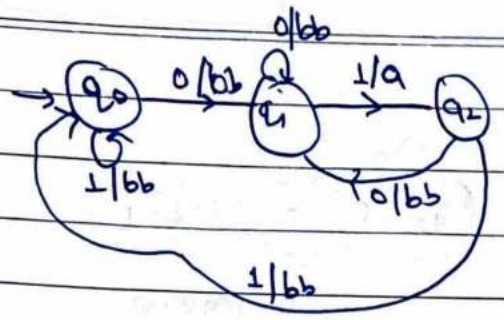
F. Print 'a' when you find "01" else print "bb"

Ex: 1100100001110

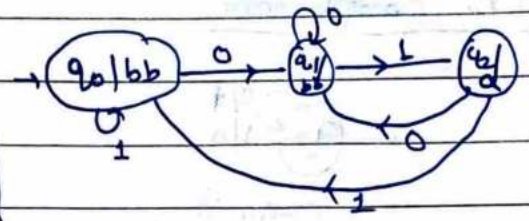
bb bb bb bb a bb bb bb bb a bb bb bb a bb ✓

GO CLASSES

PAGE NO. :
DATE : / /



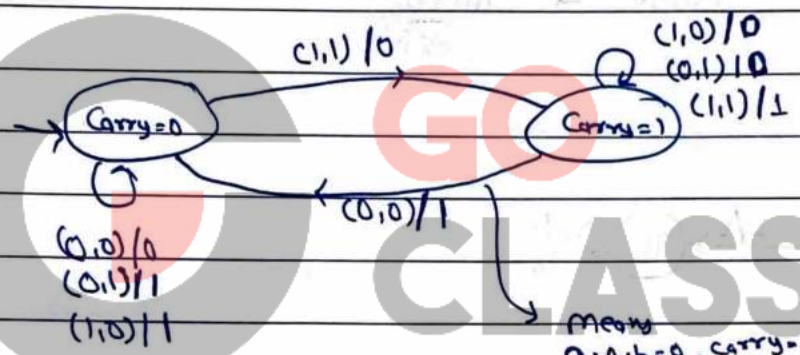
Mealy m/c



Moore m/c

Gate 1994

Q Add two int in binary of any length.



Mealy
a=0, b=0, carry=1 → o/p = 1 and carry=0

Mealy m/c.

Assuming i/p are given from LSB to MSB and pad zeros to make length = max(len(A), len(B)) + 1

↳ Extra if only carry remains.

Ex: A = 1110, B = 11

A+B = ?

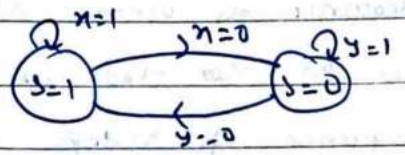
```
 01110
 00011
 -----
 10001
```

→ If que. asked in exam is what this machine does then take 2-3 arbitrary examples and eliminate options.

PAGE NO. :
DATE : / /

Gate 2106

Que: find $s^+ = f(s, x, y)$



∴ Directly from state diagram

$s^+ = \bar{s}\bar{y} + sx$

can also do using state table & then k-map.

→ Many easy PYQs for FSM design and analysis ←

GO CLASSES

PAGE NO :
DATE : / /

→ Last topic of DLD.

Counter: A counter is essentially a register that goes through a pre determined sequence of binary states.

The gates in the counter are connected in such a way to produce the prescribed sequence of states. Although counters are special type of register, it's common to diff them by giving them a diff. name.

→ These are very widely used in almost all computers and other digital electronic systems.

→ A counter that follows the binary sequence is called a binary counter.

Ex: $\overset{0}{00} \to \overset{1}{01} \to \overset{2}{10} \to \overset{3}{11} \to \overset{0}{00} \to \overset{1}{01} - - - -$

2 bit binary up counter.

$\overset{1}{01} \to \overset{0}{00} \to \overset{3}{11} \to \overset{2}{10} \to \overset{1}{01} \to \overset{0}{00} \to \overset{3}{11} \to - - -$

2 bit binary down counter.

→ In general counter can have any random sequence.

→ n modulo counter has n diff (unique) states in its sequence.

$x_1 \to x_2 \to x_3 \to x_4 - - \to x_n \to x_1 \to x_2 \to - - -$

modulo n counter.

→ n bit binary up counter has 2^n states i.e. modulo 2^n and goes 0 to 2^n-1 if it's up counter. ✓

→ In sync. seq. ckt. state changes with a clock pulse only. And in async. seq. ckt. state/o/p changes w/o clock pulse.

GO CLASSES

PAGE NO. :
DATE : / /

→ Decimal counter goes from 0 to 9. and is mod-10 counter.

→ Counter can be broadly divided as: (infact any seq. ckt)

a) Synchronous Counter: All flip-flops clocked at the same time

b) Asynchronous Counter: Clock of a FF can be o/p of prev. FF. So o/p gen. at random and may incur delay → glitch. This is why also known as Ripple Counter.
↳ int. undesired states

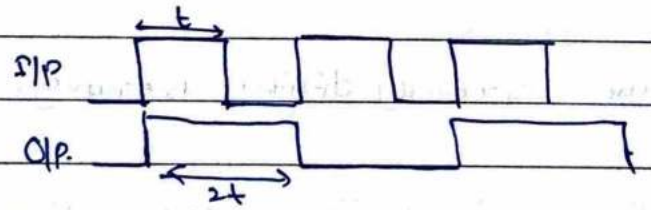
Synchronous Counter Async. Counter

GO CLASSES

☆☆ Asynchronous / Ripple Counters:

☆ Frequency Divider: Given a clock of freq. f Hz, how to get another clock of freq. f/2 Hz.

→



Dividing the frequency means, multiplying the time period of pulse obviously.

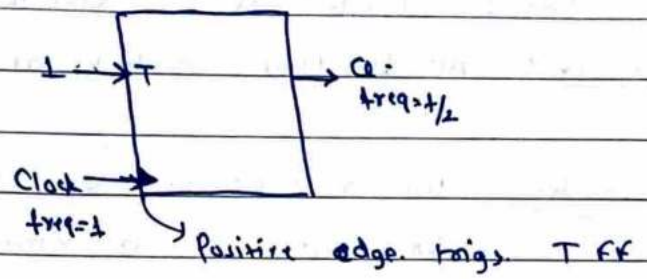
i/p freq = $\frac{1}{t}$ Hz

o/p freq = $\frac{1}{2t} = \frac{1}{2}$ i/p freq. Hz

Perfect square wave has duty cycle = 50%
i.e. On period = off period

PAGE NO :
DATE : / /

This can be done by T- flip flop.

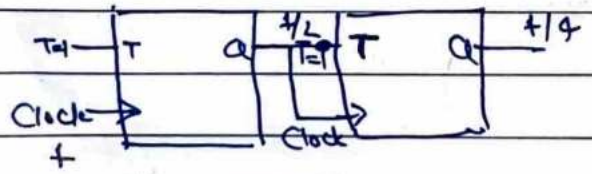


Timing Diagram:

$$\text{freq. of } Q = \frac{\text{freq. of clock}}{2}$$

So, using T- FF we successfully divided frequency.

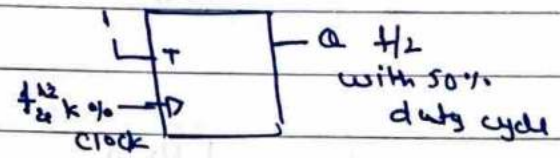
→ We can get any div. of frequency in power of 2 like. for f/4 i.e. $f/2^2$ we 2 - T-FF.



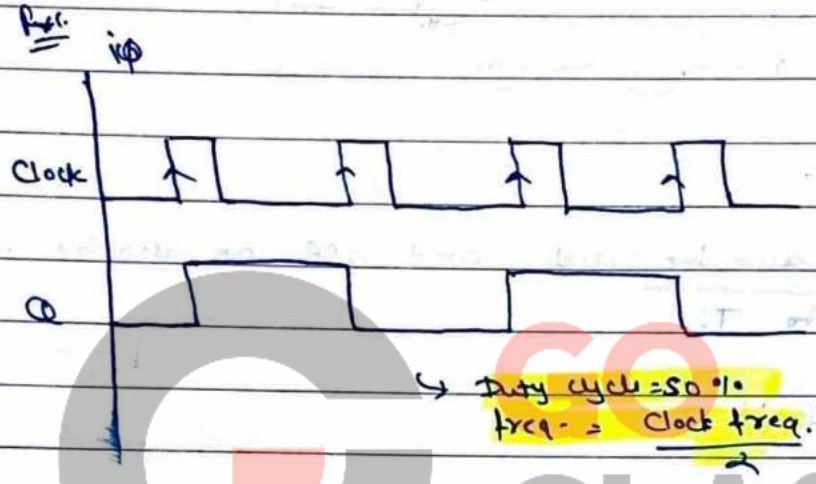
So using K T flip flop we get freq. as $f/2^k$.

PAGE NO. :
DATE : / /

→ Also T FF can be used to get perfect square wave.



Ref.

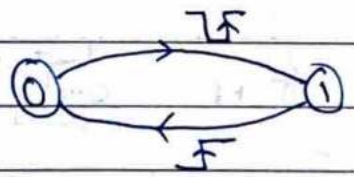


↳ Duty cycle = 50 %
freq. = $\frac{\text{Clock freq.}}{2}$

CO CLASSES

NOTE ↔ O/P of T FF divides the clock freq. by 2 (and makes 50 % duty cycle), so T-FF is also called divide by 2 ~~device~~ counter.
Why?

It is also mod-2 counter because:

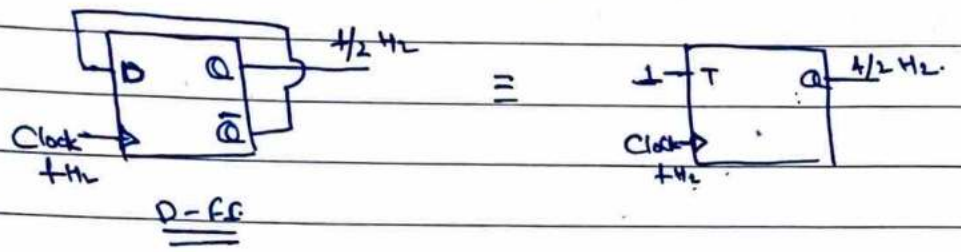


with each trigg. edge 0→1→0→1 occurs.

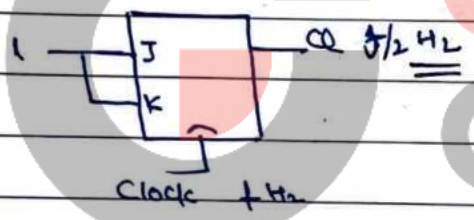
PAGE NO. :
DATE : / /

Que: Can we use D-ff to divide clock freq. by 2?

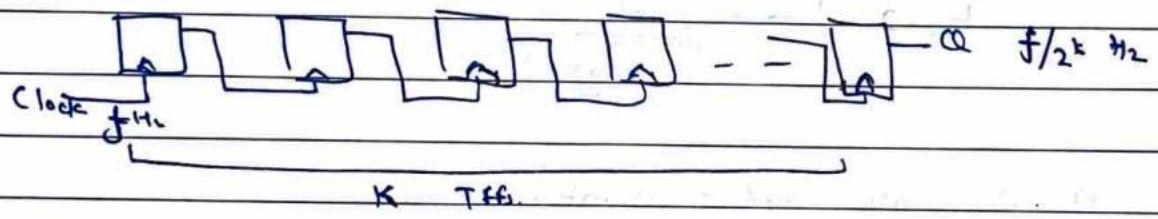
→ Yes, (Convert DFF to TFF)



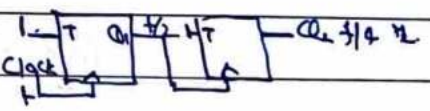
Similarly JK ff can also be used and SR can also be used, just convert them to T.

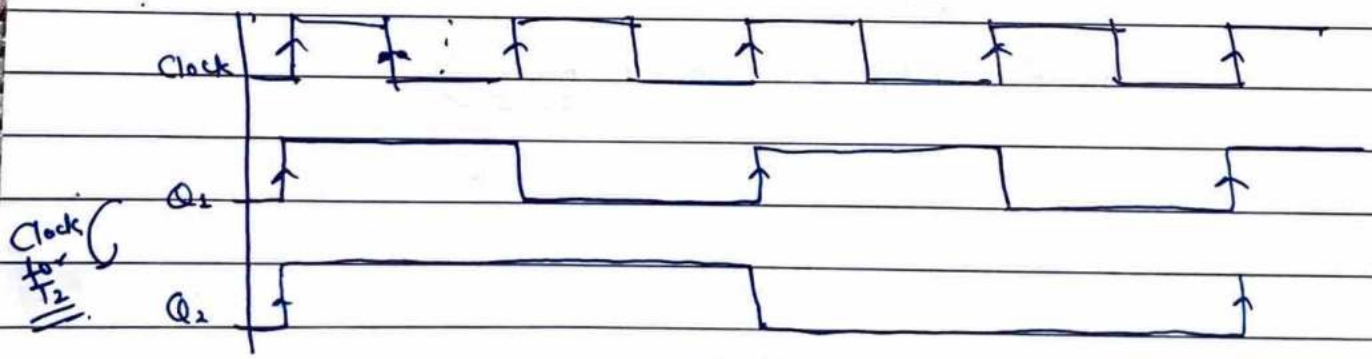


→



→ **Ex:** Timing diagram for 2 T ff.





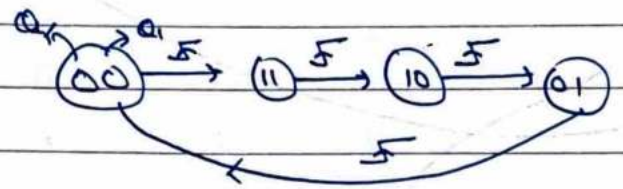
f/4 freq.

GO CLASSES

PAGE NO.:
DATE: / /

Now clearly it's dividing the frequency, but how to interprete this as counting?

The prev. ckt was mod 4 counter.
If we take Q_1 as LSB and Q_2 as MSB then:

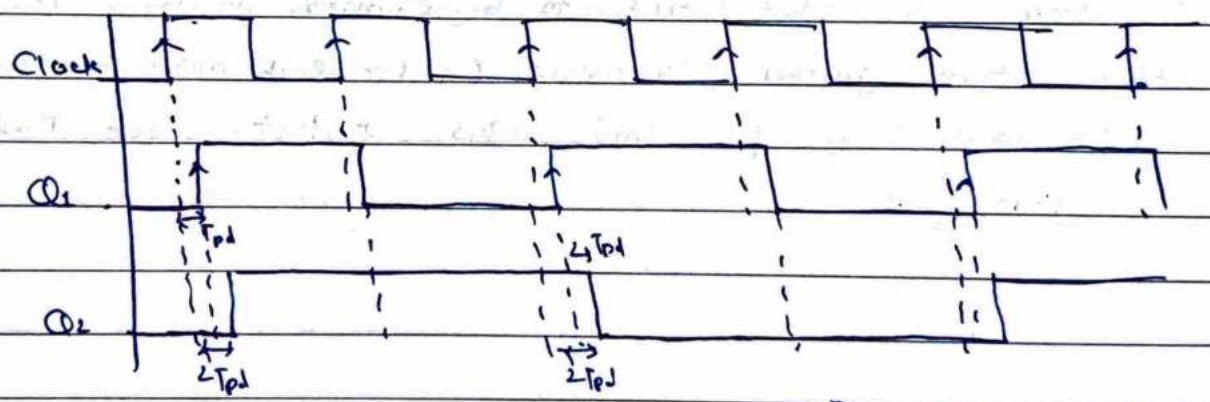


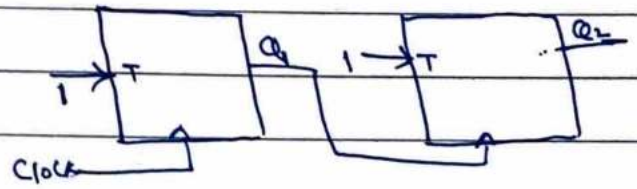
So, it is actually a mod 4 binary down counter (0→3→2→1)

The above state diagram can be verified from prev. timing diagram.

GO CLASSES

*→ Practically every FF has same prop. delay, let's assume our T FF has prop. delay t_{pd}.
Then, timing diagram will be



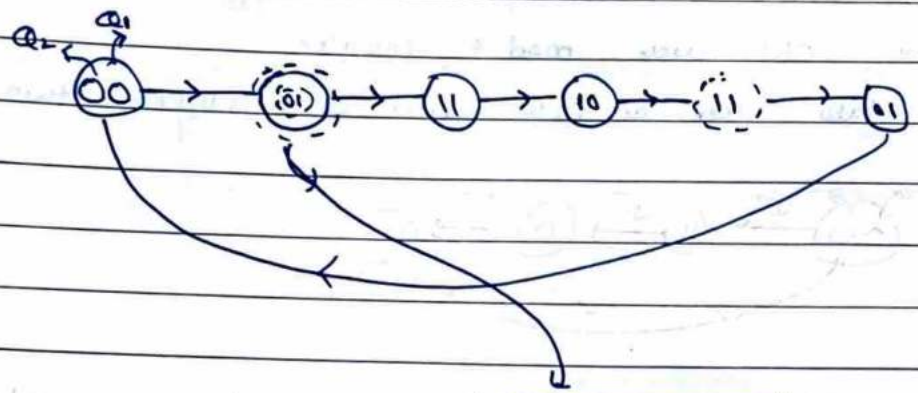


Circuit is

PAGE NO. :
DATE : / /

→ At each level overall prop. delay increases.

→ State diagram now will be: (from Timing diagram)



Transient state / Glitch
- only for Tpd time
- Not even noticed
- Due to asynchronous nature.
- first LSB changes then after Tpd MSB also changes.

→ So, due to prop. delay and async. nature we get Glitches / Transient states in asynchronous counter.

Conv →

If more than 1 bit needs to be changed in async. counter state then glitches will occure (1 for each extra bit change) LSB will chang first and MSB at last. see Timing diagram. ✓

PAGE NO. :
DATE : / /

* Binary Down Ripple Counters:

↳ 0→1→2-- ↳ 3→2→ --. ↳ Asynchronously.

We have already seen it i.e. i/p of last FF as clock of next FF and FF must be +ive edge triggered.

Idea behind working:

Clock	A (MSB)	B	C (LSB)
Initial, 0	1	1	1
1	1	1	0
2	1	0	1
3	1	0	0
4	0	1	1
5	0	1	0
6	0	0	1
7	0	0	0

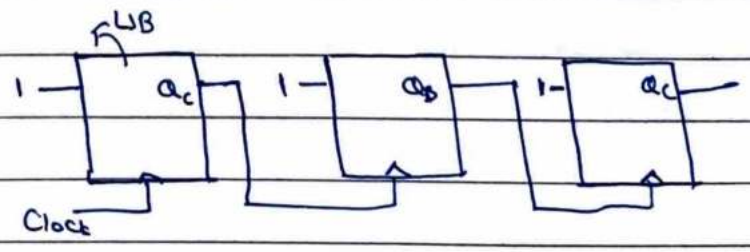
↳ our req. for binary down counter

Observations:

→ LSB changes (toggles) with every clock, so give clock to it.

→ B (middle bit) toggles when LSB goes from 0 to 1 i.e. rising edge.

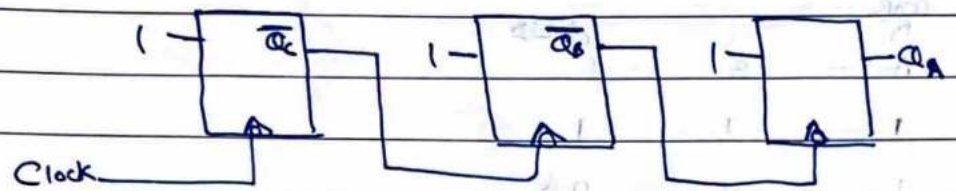
→ MSB changes / toggles when B goes from 0 to 1.
So give B as clock in MSB with +ive edge trigg.
OR give $\overline{B}$ as clock in MSB with -ive edge trigg.



Modulo 8 binary down ripple counter
(goes from 7 to 0).

PAGE NO. :
DATE : / /

→ In binary down counter n^{th} bit (from LSB) toggles when $(n-1)^{th}$ bit (from LSB) goes from 0 to 1. So we use o/p of $(n-1)^{th}$ bit as positive edged clock of nth bit from LSB or comp. of o/p of $(n-1)^{th}$ bit as -ive edge clock of nth bit from LSB.



→ for msb in n-bit ripple counter (any), the prop. delay becomes $n * t_{pd}$, where T_{pd} is prop. delay of 1 ff.

Clearly, this MSB's prop. delay should be less than or equal to time period of clock.

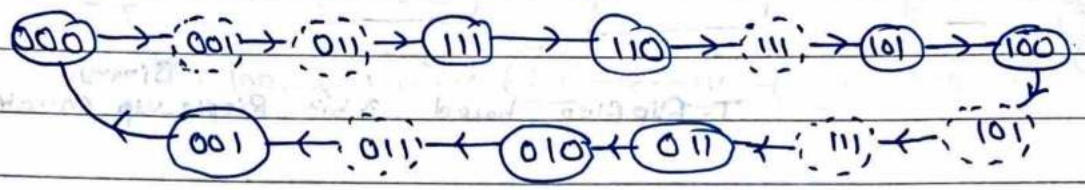
So, $\boxed{\text{min. clock period req.} = n\,t_{pd}}$ → for n bit ripple counter

$$\boxed{\text{max. freq.} = \frac{1}{n\,t_{pd}}}$$

GO CLASSES

PAGE NO. :
DATE : / /

State diagram of 3-bit binary down ripple counter with $t_{pd} \neq 0$



* Binary up ripple counter:

Clock	A (MSB)	B	C (LSB)
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Observation:

→ LSB always toggles

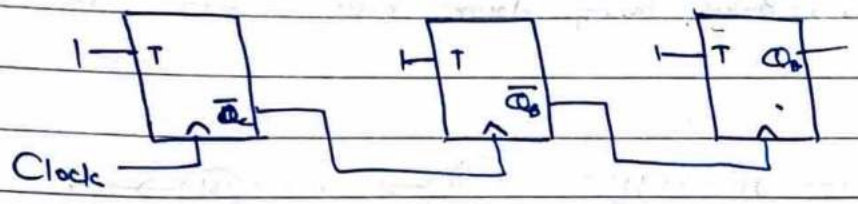
→ Middle bit toggles when LSB goes from 1 to 0 that is falling edge.

→ MSB toggles when middle bit goes from 1 to 0.

So we give B as clock to -ive edge trigg. FF of C
or B as clock to +ive edge trigg. FF of C

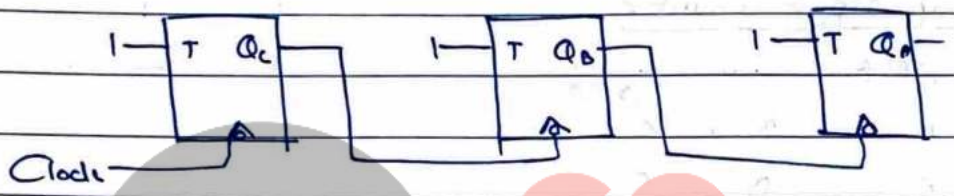
GO CLASSES

PAGE NO :
DATE : / /

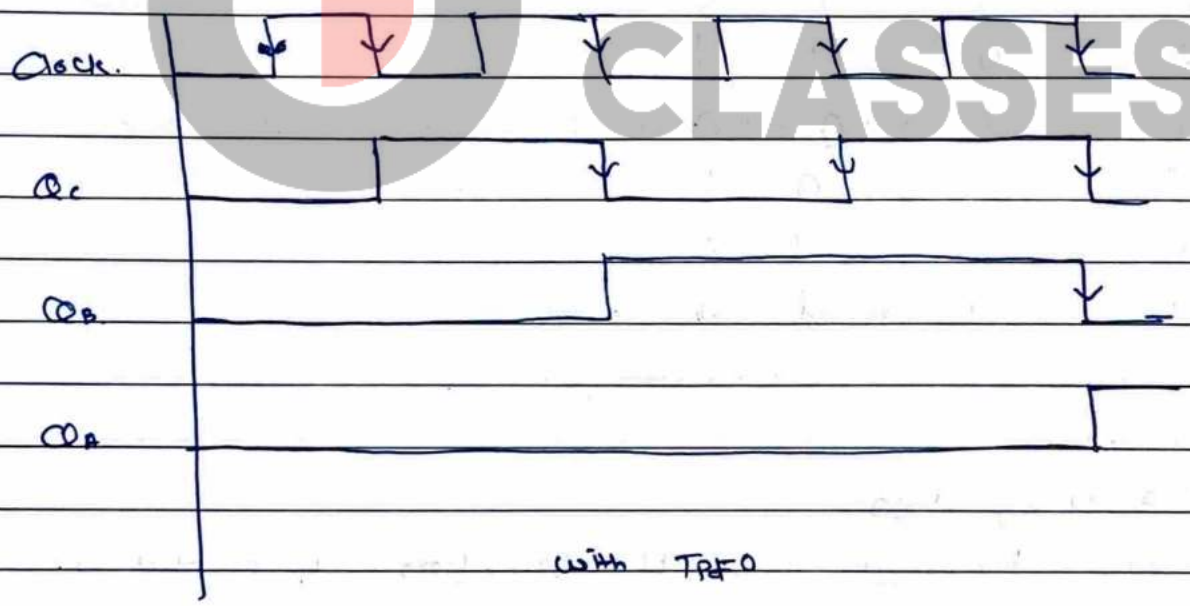


T- flip flop based 3 bit Ripple Binary up counter

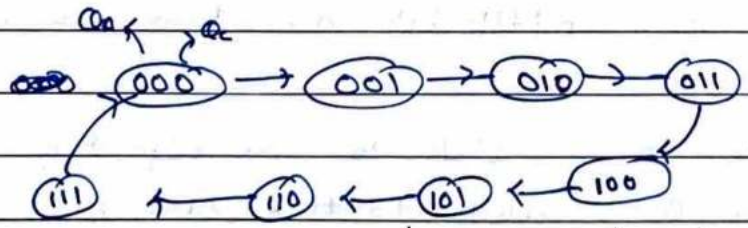
|||| Eq. to



GO CLASSES



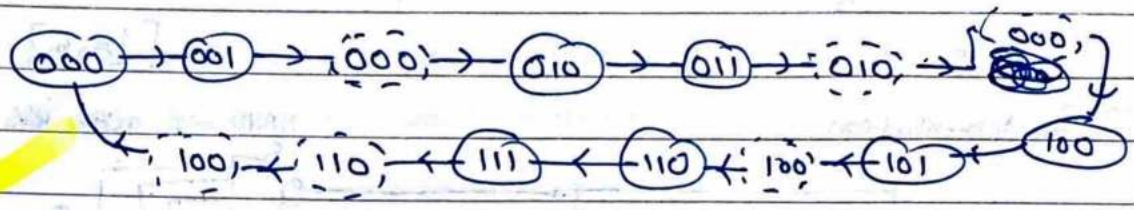
with TPFO



State diagram for 3 bit up binary counter without glitch

PAGE NO. :
DATE : / /

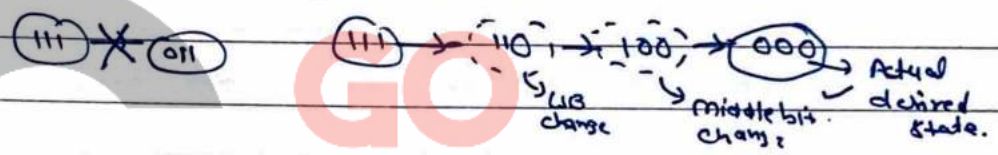
If $T_{pd} \neq 0$, then state diagram



→ It is called async. bcz not the same clock for all flipflops.

→ It is called ripple counter due to glitches/transient states bcz. of prop. delay which change bit by bit.

Ex:



Now we are able to create 0 to 2^n-1 counter but what if we want 0 to 6 or 0 to 10, etc.?

⇒ We can use preset and clear i/p.

For Ex: For 0 to 10, as soon as 11 (i.e 1011) comes reset all FF to 0000. So, a transient state of 11 will come but then it will go back to 0.

→ Preset & Clear are active low & async. i/p i.e. do not wait for the clock edge.

* Modulo m Counter (0 to m-1) and $m \neq 2^k$.

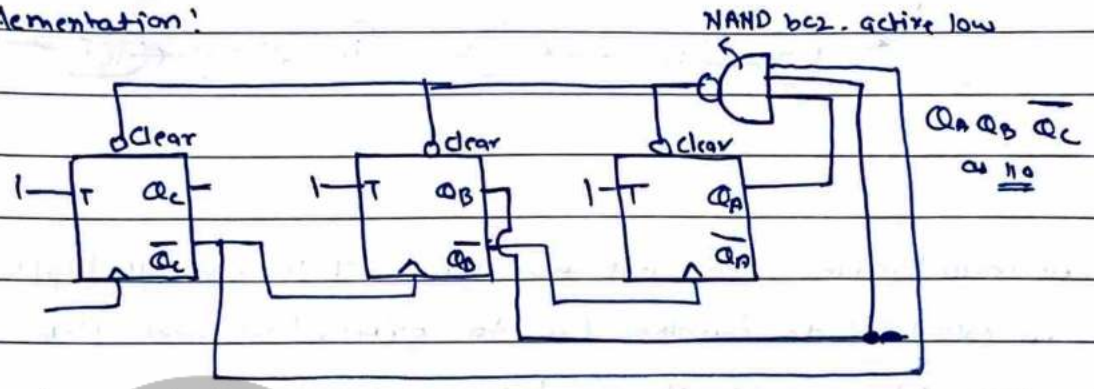
Use Clear i/p for this.

PAGE NO :
DATE : / /

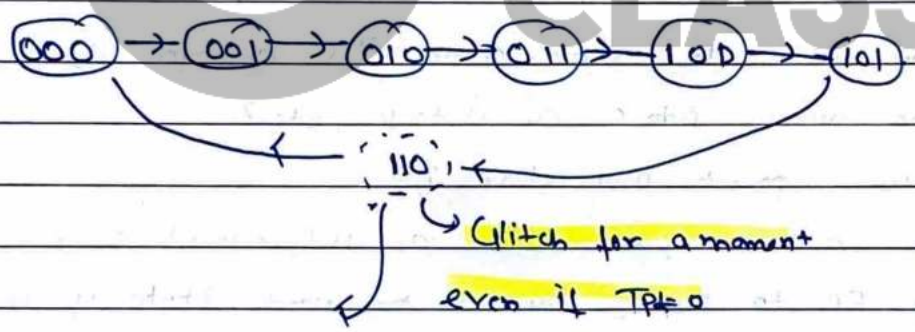
Ex: 0 → 1 → 2 → 3 → 4 → 5 → 0 modulo 6 counter.

$\lceil \log_2 6 \rceil$ ff needed.

Implementation:



GO CLASSES



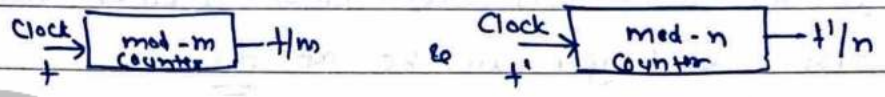
NOTE → Do not clear on 5 as we want 5 (101) to stay, clear on 6 as it is undesired.

(6) (7)
we don't want 110 & 111 so even if we give $Q_A Q_B$ as NAND i/p it will work i.e. as soon as $Q_A = 1$ & $Q_B = 1$, reset it.

→ If we want 0 to 13 i.e. mod 14 counter then we will reset when value is 14 i.e. 1110 so $Q_A Q_B Q_C \overline{Q_D}$, but 1111 & 1110 both are unwanted so $Q_A Q_B Q_C$ will also work well. (No. 14 will be available for a moment but then 0 will come).

→ Similarly we can create decade counter (0 to 9). | BCD counter.

* Cascading Counter:

Given: 

then mod mn counter can be made as:

mod mn counter.

So we can cascade counters.

Que: How many decade counters req. for mod 1000 counter?

→ Ans: 3.

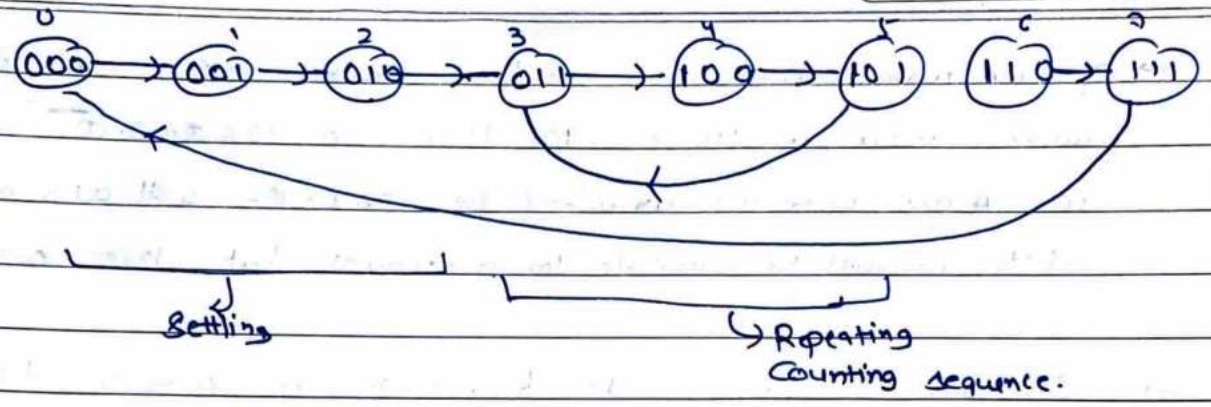
* Anywhere to Anywhere Counting: (Que)

use preset + clear to get back to initial state.

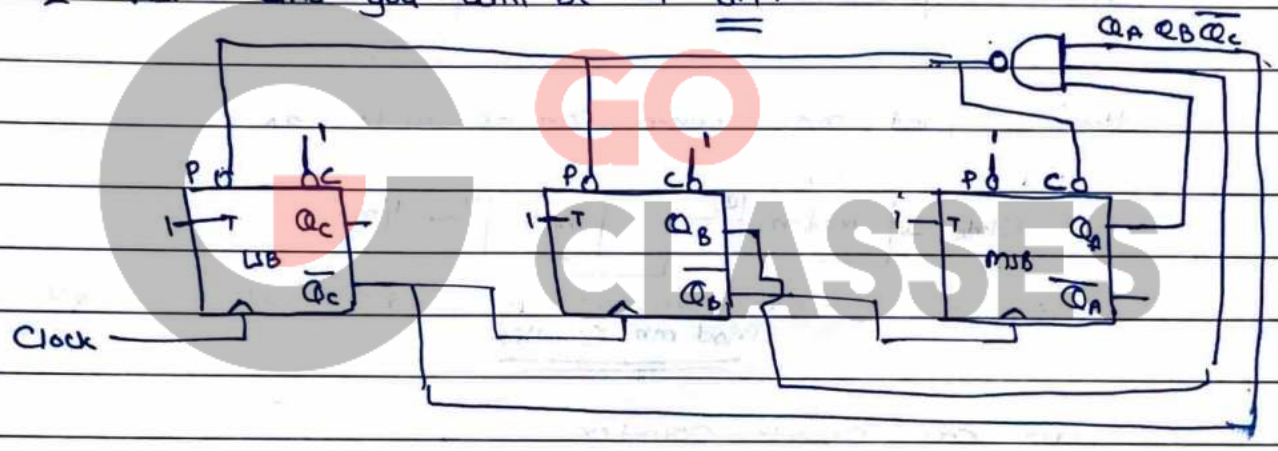
for Ex: 3 → 4 → 5

PAGE NO. :
DATE : / /

Req.



Idea is as soon as you arrive as 6 i.e. 110, then we need to go to 3 i.e 011 so clear MSB FF and preset remaining 2 FFs. and you will be at 011.



$110 = Q_A Q_B \overline{Q_C}$

$\Downarrow$

011 = Clear Q_A & Preset Q_B & Preset Q_C

$3 \rightarrow 4 \rightarrow 5 \rightarrow 3 \rightarrow 4 \rightarrow 5.$

1990

Que: A 4-bit mod 16 ripple counter uses JK FFs. If T_{pd} of each FF is 50 ns, then max. freq. is:

$\Rightarrow \frac{1}{50 \times 4}$ GHz $\Rightarrow$ 5 MHz.

That's all about async. counter.

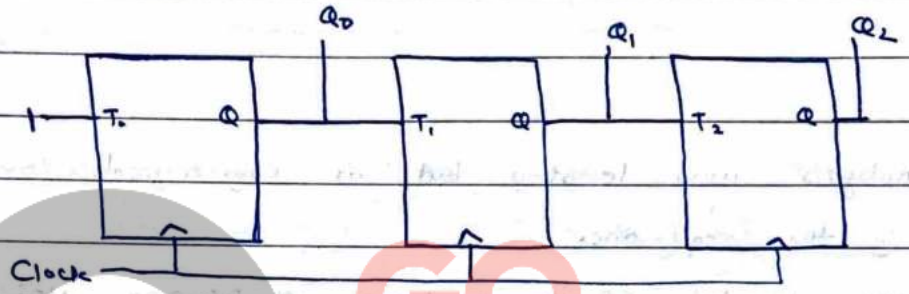
PAGE NO. :
DATE : / /

★★ **Synchronous Counter:** Till now we have studied Async. counter, Now we'll learn to analyse & then create the Synchronous Counters. We'll also see special counters used in computers i.e. Ring & Johnson Counter.

Very Easy and intuitive.

⇒ **Analysing synchronous sequential circuit:**

Ex:1



Initially $Q_2Q_1Q_0 = 000$.

→ now after 1 Clock. Q_0 will toggle as T_0 is 1 &
Q_1 will be same as T_1 is $Q_0 = 0$
Q_2 " " " " T_2 is $Q_1 = 0$

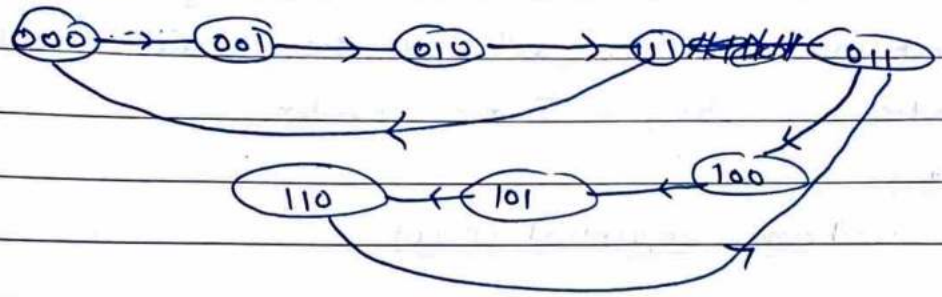
It can be analysed as:

Q_2	Q_1	Q_0	T_2 (→ Q_1)	T_1 (→ Q_0)	T_0 (→ 1)	Q_2^+ (from Q_2 & T_2)	Q_1^+	Q_0^+
0	0	0	0	0	1	0	0	1
0	0	1	0	1	1	0	1	0
0	1	0	1	0	1	1	1	1
0	1	1	1	1	1	1	0	0
1	0	0	0	0	1	1	0	1
1	0	1	0	1	1	1	1	0
1	1	0	1	0	1	0	1	1
1	1	1	1	1	1	0	0	0

GO CLASSES

PAGE NO. :
DATE : / /

State diagram: (No Glitch here)



This analysis was lengthy but is easy & good. can also find only the req. ones.

* (Here we founded FF i/p in terms of present state & then using present state & i/p we found next state) *

State after 4 clocks if initial is 110 = 110
" " 100 " " " " 110 = 110
" " 206 " " " " " = 206 % 4 = 2 ⇒ 100.

It is a mod-4 counter.

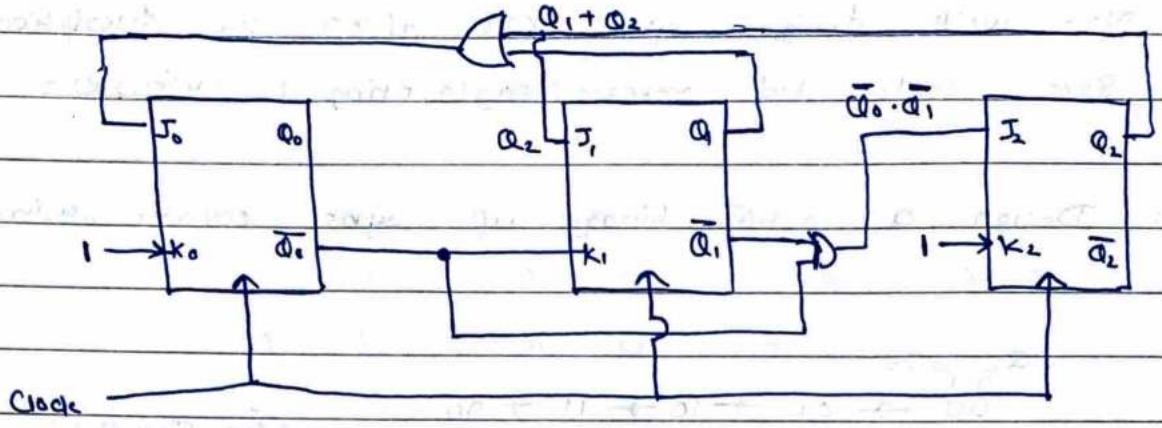
NOTE ⟷ Gen. mod of counter is taken as max. # states of repeating seq. (like 4 in above), but if a counter has fix purpose then that can be mod also, not need to be maximum.

GO CLASSES

PAGE NO. :
DATE : / /

CS-90
Ex:2

Initially $Q_0 = Q_1 = Q_2 = 0$, determine counting sequence.



⇒ Here short analysis will not work. — Toggle —

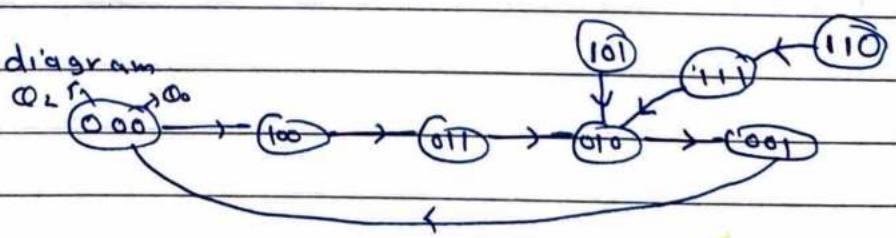
Ex: 000 → 001 ($J_0 = 0, K_0 = 1$), ($J_1 = 0, K_1 = 1$), ($J_2 = 1, K_2 = 1$)

Reset Reset

and keep on doing this.

Q_2	Q_1	Q_0	$\bar{Q}_0 \cdot \bar{Q}_1$ J_2	K_2	$Q_2 \bar{Q}_0$ J_1	K_1	$Q_1 + Q_2$ J_0	K_0	Q_2^+	Q_1^+	Q_0^+
0	0	0	1	1	0	1	0	1	1	0	0
0	0	1	0	1	0	0	0	1	0	0	0
0	1	0	0	1	0	1	1	1	0	0	1
0	1	1	0	1	0	0	1	1	0	1	0
1	0	0	1	1	1	1	1	1	0	1	1
1	0	1	0	1	1	0	1	1	0	1	0
1	1	0	0	1	1	1	1	1	0	0	1
1	1	1	0	1	1	0	1	1	0	1	0

State diagram



Mod-5 Counter.

Ans

PAGE NO. :
DATE : / /

Now we know analysing sync. circuit. Do its practice. Now we'll design sync. ckt given the description using same table but reverse engineering i.e. v'q. exc. table.

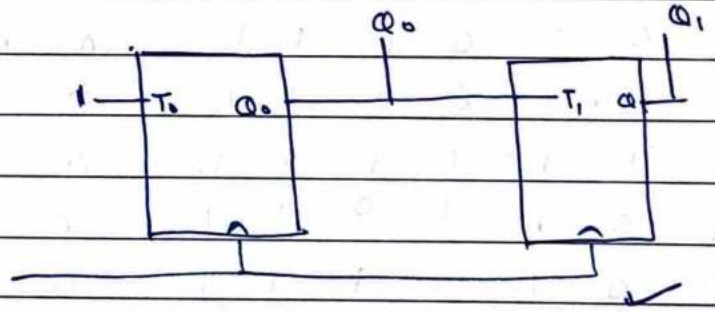
Ex:
Que: Design a 2 bit binary up. sync. counter using two T ff.

$Q_1 \nearrow Q_0$

⇒ 00 → 01 → 10 → 11 → 00

Q_1	Q_0	Q_1^+	Q_0^+	T_1	T_0 (from Q_0 to Q_0^+)
0	0	0	1	0	1 → As toggles
0	1	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	1

$T_0 = 1$
$T_1 = Q_0$ } with ff i/p as f(Present, states)



2 bit binary up. sync. counter.

GO CLASSES

PAGE NO. :
DATE : / /

Ex:2 2 bit binary down using JK ff.

⇒ $00 \to 11 \to 10 \to 01 \to 00$

Q_1	Q_0	Q_1^+	Q_0^+	J_1	K_1	J_0	K_0
0	0	1	1	1	X	1	X
1	1	1	0	X	0	X	1
1	0	0	1	X	1	1	X
0	1	0	0	0	X	X	1

Using k-map:

J_1

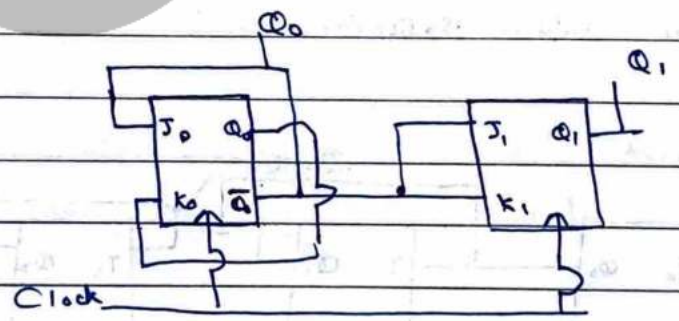
$Q_1 \backslash Q_0$	0	1
0	1	0
1	X	X

$J_1 = \overline{Q_0}$

Similarly, $K_1 = \overline{Q_0}$

$J_0 = \overline{Q_0}$

$K_0 = Q_0$



Similarly, we can design any # bits binary (up/down) synchronous counter using any ff.

Just find ff i/p as f (present, states)

PAGE NO :
DATE : / /

Ex Design 4 bit binary up counter (sync.) using 4 TFF & use only logic and analysis. (no table).

→

$2xy0 \rightarrow 2xy1$

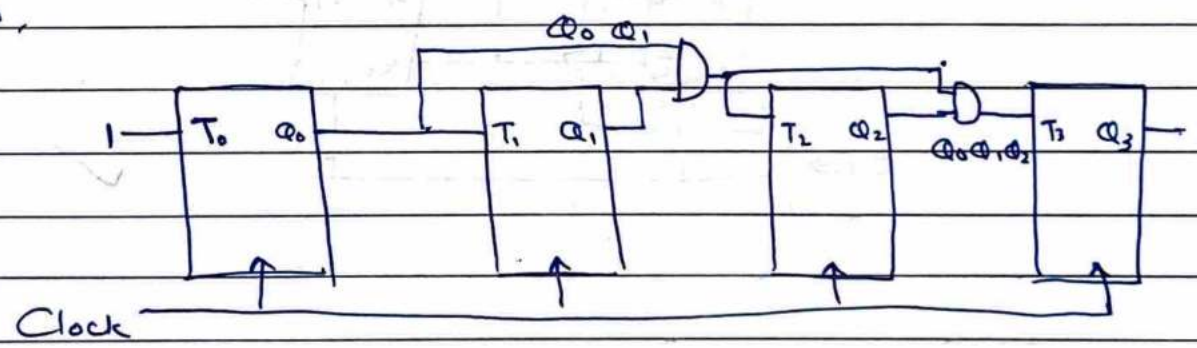
Obs:

Used in Async.
- → LSB changes every time.
- → Middle bit changes when LSB goes from 1 to 0
- → Any bit changes (except LSB) when next bit goes from 1 to 0.

Obs: (Sync) $Q_3 Q_2 Q_1 Q_0$

Req
- → LSB always toggles (Q_0)
- → Middle bit Q_1 toggles when $Q_0 = 1$
- → Q_2 toggles when Q_1 & $Q_0 = 1$. i.e. $Q_1 Q_0 = 1$
- → Q_3 toggles when $Q_2 Q_0 Q_1 = 1$.

So,

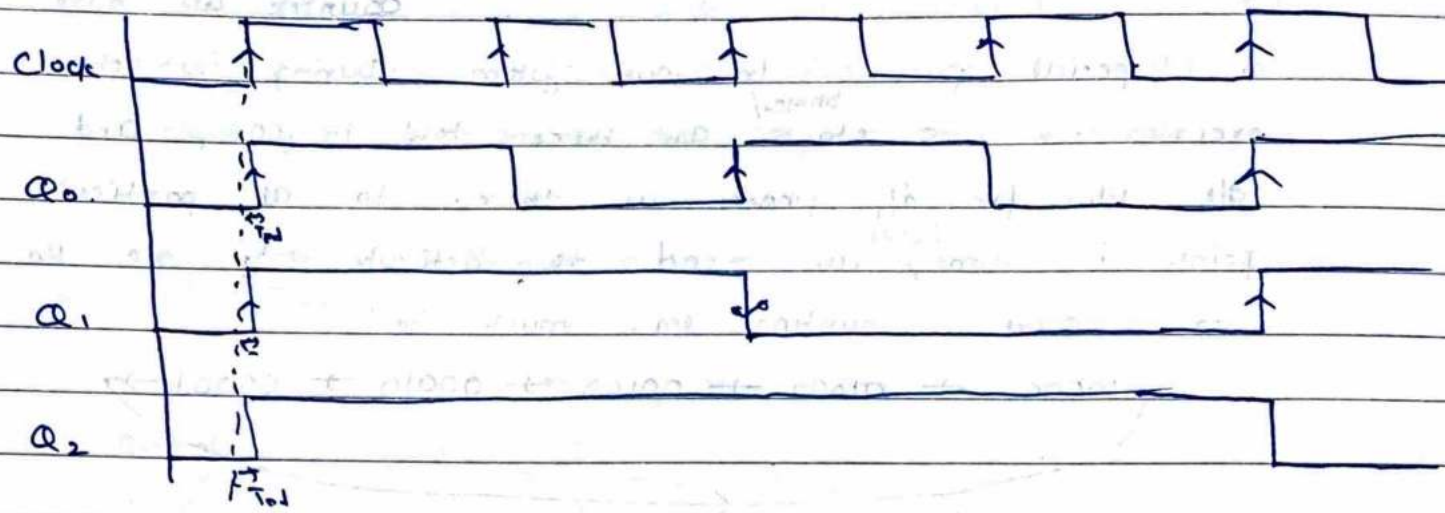


If it was -ive edge trigg. then also works same. So,

→ In sync. circuit clock triggering does not change the behaviour.

PAGE NO. :
DATE : / /

Also, no glitches here bcz, prop. delay at each ff will be same



Prop. delay is same at each level. i.e. Tpd. It do not depend on #ff.

So, min clock period = Tpd.

$$\text{freq.} \leq \frac{1}{T_{pd}}$$

$$\text{Clock Time} \geq T_{pd}$$

} In synchronous circuit.

If setup time also given T_{sup}, then

$$\text{freq} \leq \frac{1}{T_{pd} + T_{sup} + T_{comb}} \quad \text{for sync. counter}$$

$$\text{freq} \leq \frac{1}{n\,T_{pd} + T_{sup} + T_{comb}} \quad \text{for async. counter}$$

GO CLASSES

PAGE NO :
DATE : / /

Ring, Twisted Ring & Johnson Counter: To design this counter we have a special purpose in our system. During instruction exec. gen. 5 stages (phases) are there told in COA, and diff h/w for diff phases is there. So at particular point in time (clock) we need to activate only one h/w. So our counting seq. must be:

$$10000 \rightarrow 01000 \rightarrow 00100 \rightarrow 00010 \rightarrow 00001 \rightarrow$$

State diagram Req.

Now, it's classification is slightly ambiguous, but let me be clear.

Ring Counter: Those counters which fulfills the above purpose (Gen. timing signals that control seq. of op^n)

- There are:

a) **Straight Ring Counter:** Sometimes just called ring counter.

– **Twisted Ring Counter:** Slightly modified ring counter. Some authors also call it johnson counter but it is not. Also it doesn't fulfil our purpose.

b) **Johnson Counter:** Using AND gates with twisted ring counter to fulfil our purpose. (Sometimes also called twisted ring counter).

GO CLASSES

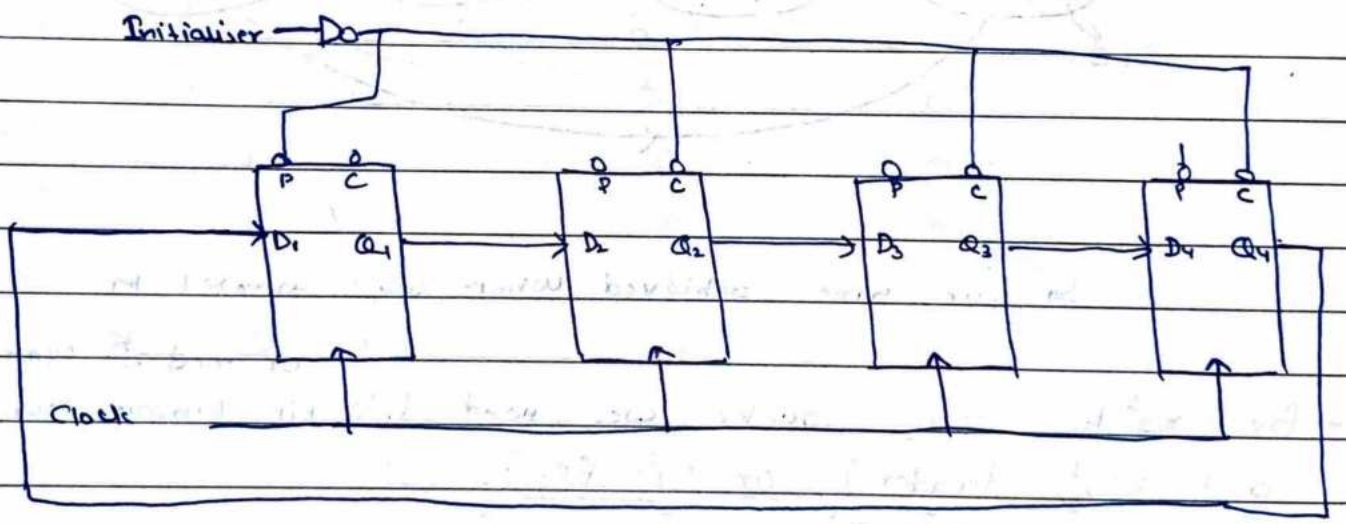
PAGE NO. :
DATE : / /

★ **Straight Ring Counter:** It is a circular shift register such that exactly 1 ff is set at a particular time. Single bit is shifted to next FF in clock cycle to produce the seq. of timing signals.

Implementation: Using shift register: (DFF)

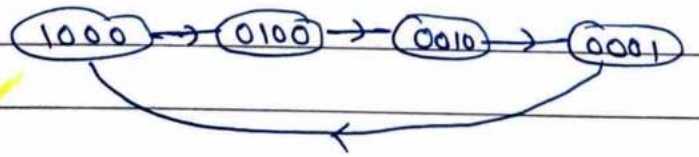
	D_1	D_2	D_3	D_4
Initial	1	0	0	0
	0	1	0	0
	0	0	1	0
	0	0	0	1
	1	0	0	0

Can be easily done as:



To initialise preset Q_1 & clear Others.

State diagram:



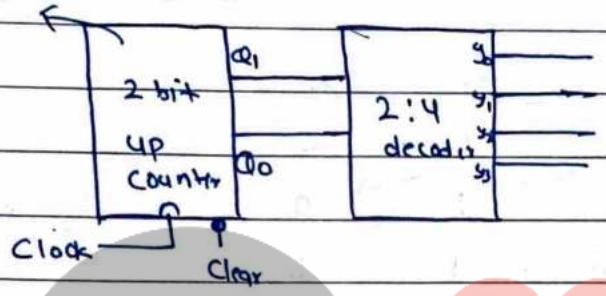
→ Similarly n-bit ring counter can be implemented using n ffs.

PAGE NO. :
DATE : / /

→ **Implementation 2:** Using a counter and a decoder

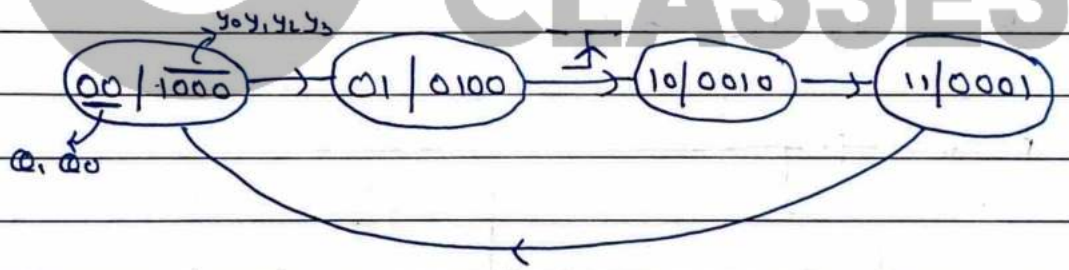
→ we can't only use decoder because it's a comb. circuit so no clock but we need synchronisation so we use counter with decoder.

It's a counter's block diagram



Now:

State diagram:



So we have achieved what we wanted to.

→ for 2^k bit ring counter we need (k bit binary counter or mod 2^k counter and $k:2^k$ decoder.) or (2^k ff.).

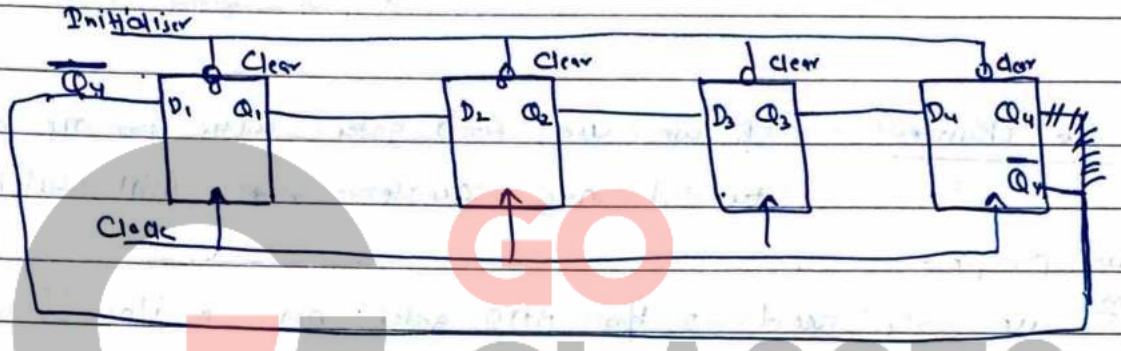
→ This can be initialised with clear.

PAGE NO. :
DATE : / /

→ It is called ring counter as in imp. with D ff it is forming ring like structure.

★ **Twisted Ring Counter:** Also called "Switch Tail Ring Counter" or "Mobius counter".

Its implementation is:

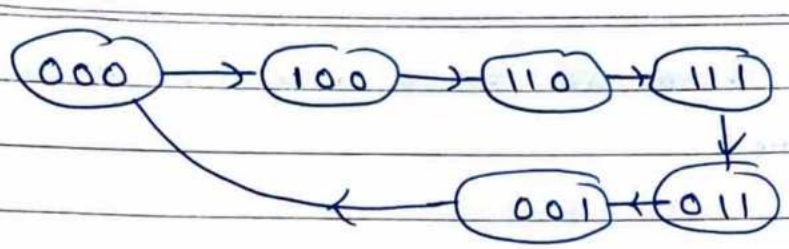


Here we initialise with 0000

	Q_1	Q_2	Q_3	Q_4	
Initially	0	0	0	0	
↑	1	0	0	0	$Q_2^+ = Q_1$
↑	1	1	0	0	$Q_3^+ = Q_2$
↑	1	1	1	0	$Q_4^+ = Q_3$
↑	1	1	1	1	$Q_1^+ = \overline{Q_4}$
↑	0	1	1	1	
↑	0	0	1	1	
↑	0	0	0	1	

→ n bit ~~johnson~~ Twisted ring counter has 2n states unlike ring counter which has n states.

PAGE NO. :
DATE : / /



3 bit twisted ring counter's state diagram.

Clearly it's not fulfilling our purpose of timing signals.

★ **Johnson Counter:** If we use AND gates with the o/p of twisted ring counter, it will fulfil our purpose.

Imp. We only need 2 i/p AND gates not n i/p as n bits are there.

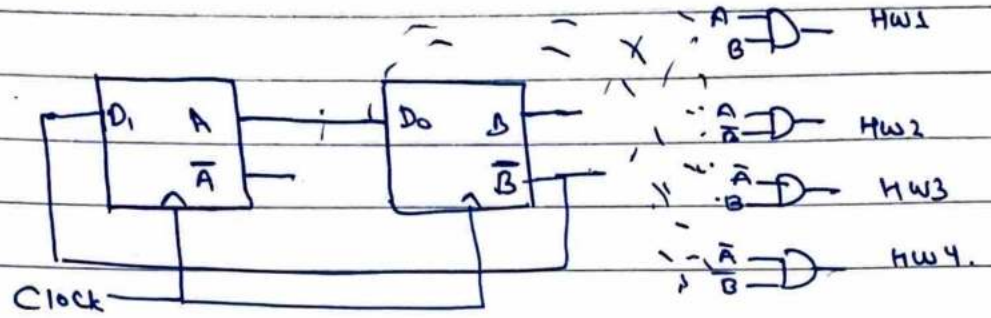
$$\underbrace{\text{n-bit twisted ring counter} + \text{2n two i/p AND Gates}}_{\text{Johnson Counter with 2n timing signals.}}$$

Ex: In 2 bit twisted ring counter add 4 AND gates as:

Counter op.	#1 $\bar{A}\bar{B}$	#2 $A\bar{B}$	#3 $\bar{A}B$	#4 AB
00	1	0	0	0
10	0	1	0	0
01	0	0	1	0
11	0	0	0	1

So we got 4 timing signals as per our requirement

PAGE NO.:
DATE: / /



In this only 1 AND gate is 1 at any point of time.

Qu: How this 2n two i/p AND gate setup will work for 4 bit twisted ring based johnson counter?

→ There is a rule to take first 01 or 10, if not present then take extreme i/ps. (for 0000 or 1111)

Twisted ring O/P.				AND Gate which will be 1.	
A	B	C	D		
0	0	0	0	$\bar{A}\bar{D}$	(extreme as no 10 & 01)
1	0	0	0	$A\bar{B}$	(10 at start)
1	1	0	0	$B\bar{C}$	
1	1	1	0	$C\bar{D}$	
1	1	1	1	AD	
0	1	1	1	$\bar{A}B$	
0	0	1	1	$\bar{B}C$	
0	0	0	1	$\bar{C}D$.	

8 - 2 i/p AND gate only.

That's all.

These notes are the property of GOClasses. Please do not share them without their consent.

For any questions or issues regarding my handwritten notes, feel free to reach out via email at karan757527@gmail.com or DM me on telegram at @karan757527 (or by clicking here).

I would love to know if you liked my notes. Your feedback and appreciation are invaluable, so don't hesitate to share your thoughts. Click here to connect with me on LinkedIn or you can find me as @agrawalkaran.

Wishing you success on your journey ahead!

- Karan Agrawal

GO CLASSES