BOOLEAN ALGEBRA

* Boolean Algebra is a 2-valued system with operations like AND, OR, XOR, etc.

* Boolean	Binary
• 0 & 1 are used as two states of digital circuit. Ex: 0 can be 0v, closed state, False etc.	• Used to represent the numerical values 0 & 1.
• Meaning of $x+y$ is OR operation.	• $x+y$ represents binary addition

• Binary is just like decimal, a number system, with two digits.

* Basic Operations

Boolen algebra consists of:

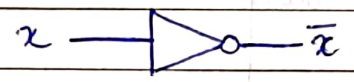
1. States: 0 & 1
2. Boolean Variables: $x, y, z, \ldots$
3. Operations: $+, \cdot, \bar{x}$, etc.

* ~~Negate~~

* Negation

($\bar{x}$ or $\sim x$)

x	$\bar{x}$
0	1
1	0

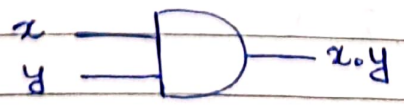


NOT Gate

* And

($x.y$)

x	y	$x.y$
0	0	0
0	1	0
1	0	0
1	1	1

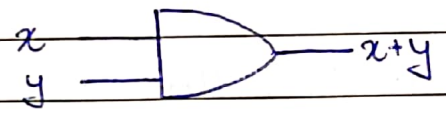


AND Gate

* Or

$(x+y)$

x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1



OR Gate

* Boolean Expressions

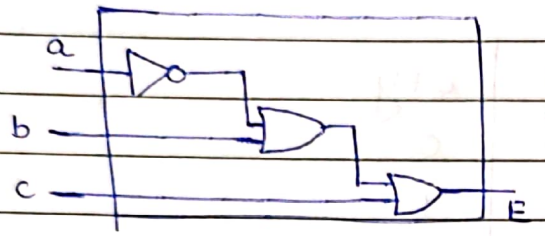
- Boolean state and/or boolean variables and/or their combinations using different boolean operator is called boolean expression.

  Ex. 1, 0, x, y, z, $x+y$, $x.0$, $\bar{y}$, $\overline{x+y+z}$, etc.

- If E_1 & E_2 are boolean expressions then E_1+E_2, $E_1.E_2$, $\bar{E_1}$, $\bar{E_2}$, etc are also boolean expressions.

- Every boolean expression can be represented using digital ckt of logic gates. The input to the circuit are boolean variables, & the output is the expression.

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



* Operator Precedence: $\bar{x} > x.y > x+y$

* Binary Number System

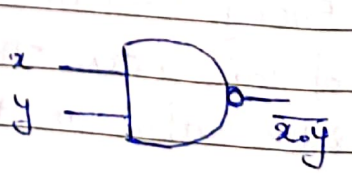
- Base of a number system is the no. of digits in it. Binary has two digits: 0 & 1.

* Other Boolean Operations

* NAND

$(a \uparrow b$ or $\overline{a.b})$

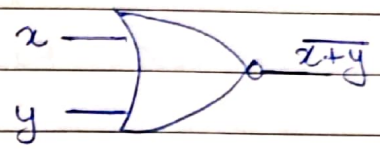
x	y	$\overline{x.y}$
0	0	1
0	1	1
1	0	1
1	1	0



* NOR

($x \downarrow y$ or $\overline{x+y}$)

x	y	$\overline{x+y}$
0	0	1
0	1	0
1	0	0
1	1	0

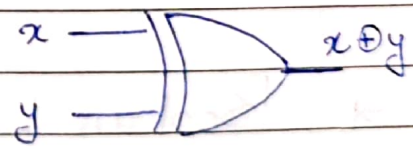


- NAND & NOR are called Universal gates

* XOR

($x \oplus y$)

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

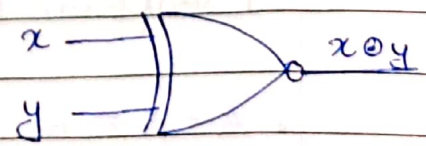


$$x \oplus y = x\bar{y} + \bar{x}y$$

* XNOR

($x \odot y$ or $x \Leftrightarrow y$)

x	y	$x \odot y$
0	0	1
0	1	0
1	0	0
1	1	1



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

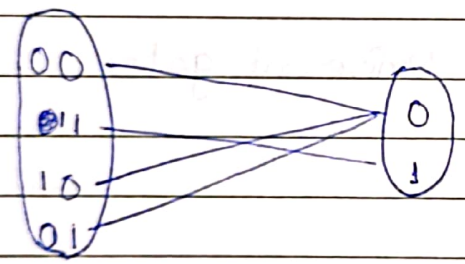
classmate
Date
Page

* **Literal:** In an expression, each appearance of a variable or its compliment counts as a literal.

		# vars	# literals
Ex.	$a+b$	2	2
Ex.	$a\bar{b}+b\bar{a}$	2	4
Ex.	$a+a+a$	1	3

* Bodean function == Boolean expressions.

Ex. $f = ab$



Domain Codomain

* Boolean Laws

1. Domination law

- $A + 1 = 1$
- $A \cdot 0 = 0$

2. Identity Law

- $A + 0 = A$
- $A \cdot 1 = A$

3. Idempotent Law

- $A + A = A$
- $A \cdot A = A$

4. Complement Law

- $A + \bar{A} = 1$
- $A \cdot \bar{A} = 0$

5. Double Complement

- $\overline{\overline{A}} = A$

6. Commutative Law

- $A + B = B + A$
- $A \cdot B = B \cdot A$
- $A \oplus B = B \oplus A$
- $A \odot B = B \odot A$
- $A \uparrow B = B \uparrow A$
- $A \downarrow B = B \downarrow A$

7. Associative Law

- $A + (B + C) = (A + B) + C$
- $A \cdot (B \cdot C) = (A \cdot B) \cdot C$
- $A \oplus (B \oplus C) = (A \oplus B) \oplus C$
- $A \odot (B \odot C) = (A \odot B) \odot C$

8. DeMorgan Law

- $\overline{(A + B)} = \overline{A} \cdot \overline{B}$
- $\overline{(A \cdot B)} = \overline{A} + \overline{B}$

9. Distributive Law

- $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$
- $A + (B \cdot C) = (A + B) \cdot (A + C)$

* Complement of Expressions

If a boolean expression only contains literals, +, . & ~, 1 & 0, then its complement can be found using DeMorgan's laws.

$$E(\text{literal } l, +, \cdot, 0, 1)$$
$$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow$$
$$\bar{E}(\bar{l}, \cdot, +, 1, 0)$$

Ex. $E = \overline{ab} + c$ $\qquad \bar{E} = \overline{(\bar{a}+\bar{b}) + \bar{c}}$

$\Rightarrow (a \cdot b) \cdot \bar{c}$

Ex. $E = (\overline{a+b}) \cdot \bar{c}$ $\qquad \bar{E} = \overline{(\overline{a+b}) \cdot \bar{c}}$

$\Rightarrow (a+b) + c$

Ex. $E = \bar{A}B + A\bar{B}$ $\qquad \bar{E} = \overline{(\bar{A}B)} \cdot \overline{(A\bar{B})}$

$\Rightarrow (A+\bar{B}) \cdot (\bar{A}+B)$

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

* Some useful properties

1. Absorption Laws $\qquad$ 2.

- $a + ab = a$
- $a \cdot (a+b) = a$

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

3. Consensus Law

- $ab + \bar{a}c + bc = ab + \bar{a}c$

* Dual of Boolean Functions.

If a boolean expression only consists of literals; $\cdot$, $+$ & $\sim$ operations; 1 & 0, then its dual f^D can be found as:

$$f(a, \cdot, +, \sim, 1, 0)$$
$$\downarrow \downarrow \downarrow \downarrow \downarrow \downarrow$$
$$f^d(a, +, \cdot, \sim, 0, 1)$$

So, only $(1 \Leftrightarrow 0)$ & $(+ \Leftrightarrow \cdot)$.

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

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

Ex. Is it possible $f^d = f$?

Yes. $f = a$, $f^d = a$.

Ex. Is it possible $f^d = \bar{f}$?

Yes. $f = 1$, or $f = a\bar{b} + \bar{a}b$, etc.

- For an equation, $(E = F)$, its dual is $(E^d = F^d)$.
  If eq^n $(E = F)$ is valid then its ~~dd~~ dual $(E^d = F^d)$ is also valid.

Ex. ~~a+b~~ $a.(b+c) = (a.b) + (a.c)$

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

Both are valid.

* Algebraic Simplification

Ex. $A\bar{B}D + A\bar{B}\bar{D}$

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

$\Rightarrow A\bar{B}$

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

$\Rightarrow \bar{A}C + AB + BC$

$\Rightarrow AB + \bar{A}C$

Ex. $\bar{B}\bar{C}D + (\overline{B+C+D}) + \bar{B}\bar{C}\bar{D}E$

$\Rightarrow \bar{B}\bar{C}D + \bar{B}C\bar{D} + \bar{B}\bar{C}\bar{D}E$

$\Rightarrow \bar{B}\bar{C} + \bar{B}\bar{C}\bar{D}E$

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

Ex. $BCD[BC+\bar{D}(CD+BD)]$

$\Rightarrow BCD$

Ex. $\bar{X}YZ+\bar{X}Y\bar{Z}+XZ$

$\Rightarrow \bar{X}Y+XZ$

- The more simplified a Boolean Expression is, the simpler the digital circuit corresponding to the expression will be.

Ex. Prove $\bar{A}B+\bar{B}\bar{C}+AB+\bar{B}C=1$

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

$\Rightarrow B+\bar{B}$

$\Rightarrow 1$

So, original expression's literal count = 8

now, l.c. = 0.

Ex. Prove $(A+B)(A+\bar{A}\bar{B})C+\overline{(\bar{A}.(B+\bar{C}))}+\bar{A}B+ABC$

$=A+B+C.$

$\Rightarrow AC+ABC+(\overline{\bar{A}B+\bar{A}\bar{C}})+B(\bar{A}+AC)$

$\Rightarrow AC+A+\bar{B}C+\bar{A}B+BC.$

$\Rightarrow A+\bar{B}C+\bar{A}B+BC$

$\Rightarrow A+B+C.$

- If $a \oplus b = a \oplus c \Rightarrow b = c$.
  $a \odot b = a \odot c \Rightarrow b = c$.

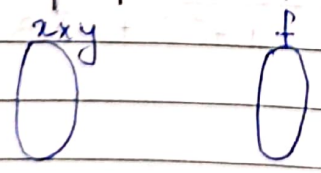
Ex. Prove: $A \odot (B \odot C) = (A \odot B) \odot C$

C1: $A = 0$	C2: $A = 1$
$LHS = \overline{(B \odot C)} = B \oplus C$	$LHS = B \odot C$
$RHS = \bar{B} \odot C = B \oplus C$	$RHS = B \odot C$

Ex. Prove $X(Y \oplus Z) = XY \oplus XZ$.

C1: $X = 0$.	C2: $X = 1$
$LHS = 0.(Y \oplus Z) = 0$	$LHS = Y \oplus Z$
$RHS = 0.Y \oplus 0.Z = 0 \oplus 0 = 0$	$RHS = Y \oplus Z$

Ex. No. of functions using 2 variables.



No. of input combinations $= 2 \times 2 = 4$.
for each input comb. we have either 0 or 1.

$\therefore 2^4 = 16$.

$\therefore 2^{2^n}$ where n is no. of input vars.

- On n input variables, number of ~~inp~~ boolean functions is

$$2^{2^n}$$

* Standard Forms of Boolean Expressions

- Literal: A variable or a variable's complement.

  Ex. a ✓, $\bar{x}$ ✓, $a\bar{x}y$ ✗, $a+\bar{x}+y$ ✗

- Product Term: A literal or AND of literals.

  Ex. a ✓, $\bar{x}$ ✓, $a\bar{x}y$ ✓, $a+\bar{x}+y$ ✗

- Sum Term: A literal or OR of literals.

  Ex. a ✓, $\bar{x}$ ✓, $a\bar{x}y$ ✗, $a+\bar{x}+y$ ✓

- Sum of Products (SOP):

  Ex. $a+b$, $\underbrace{ax}_{\text{Prod. Term}}+\underbrace{\bar{y}z}_{\text{Prod. Term}}$, $a\bar{b}$

- Product of Sums (POS):

  Ex. $(a+b)$, $(a+x)$, $(\bar{y}+z)$, $(a)(\bar{b})$

- **Minterm:** Minterm of n variables is a product term which contains each of the n terms exactly once in either complemented or uncomplemented form, but not both.

  Ex. $\bar{a}, ab, \bar{a}b, a\bar{b}, \bar{a}\bar{b}, \bar{a}bc, a\bar{b}c$, etc.

- **Maxterm:** Maxterm of n variables is a sum term which contains each of the n terms exactly once in either complemented or uncomplemented form, but not both.

  Ex. $\bar{a}$, $(a+b)$, $(\bar{a}+b)$, $(a+\bar{b})$, $(\bar{a}+\bar{b})$, $(\bar{a}+b+c)$, etc.

Ex. No. of minterms & maxterms possible for:

n	#minterms	#maxterms
1	2	2
2	$2 \times 2 = 4$	4
3	$2^3 = 8$	8
4	$2^4 = 16$	16

- For n variables the minterms & maxterms can be designated a representation.

x	y	z	minterm	repr	maxterm	repr
0	0	0	$\bar{x}\bar{y}\bar{z}$	m_0	$(x+y+z)$	M_0
0	0	1	$\bar{x}\bar{y}z$	m_1	$(x+y+\bar{z})$	M_1
0	1	0	$\bar{x}y\bar{z}$	m_2	$(x+\bar{y}+z)$	M_2
0	1	1	$\bar{x}yz$	m_3	$(x+\bar{y}+\bar{z})$	M_3

- Let's consider 2 var. minterms as func.

x	y	$\bar{x}\bar{y}$	$\bar{x}y$	$x\bar{y}$	xy
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Every minterm is only 1 for a single row, that's why each minterm is associated with the row for which it is 1.

- Let's consider 2 var. maxterms as func.

x	y	$\bar{x}+\bar{y}$	$x+\bar{y}$	$\bar{x}+y$	$x+y$
0	0	1	1	1	0
0	1	1	0	1	1
1	0	1	1	0	1
1	1	0	1	1	1

Every maxterm is only 0 for a single row. Therefore each maxterm is associated with the row for which it is 0.

- Every minterm m_i is complement of maxterm M_i.

i.e. $m_i = \overline{M_i}$

Ex. 3 vars. $M_3 = (x+\bar{y}+\bar{z})$

$\overline{M_3} = \overline{x+\bar{y}+\bar{z}}$

$\Rightarrow \bar{x}yz = m_3$

True / False Minterms / Maxterms

For n variables, out of 2^n minterms, the minterms for which ~~th~~ a func. f is equal to 1 are called as True minterms or 1-minterms of the func. f.
Rest are known as false minterms.

Similary for n vars, out of 2^n maxterms, the maxterms for which a func. f is equal to 1 are called as True maxterms or 1-maxterm of the func. f.
Rest are known as false maxterms.

Ex. $f = a+b$.

a	b	$f=a+b$	minterm	maxterm
0	0	0	$\bar{a}\bar{b}$	$(a+b)$
0	1	1	$\bar{a}b$	$(a+\bar{b})$
1	0	1	$a\bar{b}$	$(\bar{a}+b)$
1	1	1	ab	$(\bar{a}+\bar{b})$

1 minterms

1 maxterm.

Expressing functions using minterms & maxterms

A func. f can be represented as sum of minterms for which f is 1. (true minterms)

Ex.

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

So, $f = \bar{a}b + ab$

Similarly f can be represented as product of maxterms for which f is 0. (false maxterms)

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

Ex.

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

So,

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

$\Rightarrow \sum_m 2, 3, 5, 6$

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

$\Rightarrow \prod_M 0, 1, 4, 7$

The 1-minterm repr. of f is called Cannonical SOP.
The 0-maxterm repr. of f is called Cannonical POS.

Ex. $f(a,b) = a+b$
- SOP ✓
- Cann. SOP ✗
- POS ✓
- Cann. POS ✓

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

$f = \bar{a}b + a\bar{b} + ab = \Sigma_m(1,2,3)$

Cann. SOP

$f = (a+b) = \Pi_M(0)$

Cann. POS

$f = \Sigma(1,2,3) = \Pi(0)$

$\bar{f} = \Sigma(0) = \Pi(1,2,3)$

Ex. No. of minterms in the minimized expression of:

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

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

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

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

$\Rightarrow (\overline{ABCD})'$

$\Rightarrow ABCD = \Sigma_m(15)$

$\therefore$ # of minterms : 1.

Ex. $f(a,b) = a+b$. No. of minterms?

{Note: The question is implicitly asking for all minterm.}

So, answer would be 4. (not 3).

- Conjunctive Normal Form (CNF) = POS
  Disjunctive Normal Form (DNF) = SOP

Cannonical CNF = Cannonical POS
Cannonical DNF = Cannonical SOP

Ex. Express $F = A + \bar{B}C$ as sum of minterms.
{Already SOP}

Method 1:

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

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

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

$\Rightarrow \Sigma_m(7,5,6,4,1)$

Method 2:

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

A			$\bar{B}C$	
1	_	_	_	0 1
0	0	m_4	0	m_1
0	1	m_5	1	m_5
1	0	m_6		
1	1	m_7		

$\therefore F = \Sigma_m(1,4,5,6,7)$

Ex. Express $F = xy + \bar{x}z$ as product of maxterms.

Method 1:

$$\cancel{F = xy + \bar{x}z}$$
$$\cancel{\Rightarrow xy + z\bar{z} + \bar{x}z + y\bar{y}}$$
$$\cancel{\Rightarrow xy + -}$$

$$F = xy + \bar{x}z$$
$$\Rightarrow (xy + \bar{x}).(xy + z)$$
$$\Rightarrow (y + \bar{x}).(x + z).(y + z) \quad \} \text{ POS}$$

$$\Rightarrow (y + \bar{x} + z\bar{z})(x + z + y\bar{y})(y + z + x\bar{x})$$
$$\Rightarrow (\bar{x} + y + z)(\bar{x} + y + \bar{z})(x + y + z)(x + \bar{y} + z)(x + y + z)(\bar{x} + y + z)$$

$$\Rightarrow (\bar{x} + y + z)(x + y + z)(\bar{x} + y + \bar{z})(x + \bar{y} + z).$$

$$\therefore F = \Pi_M(4, 0, 5, 2)$$

Method 2: (Find Minterms → Convert to maxterms)

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

$$\Rightarrow \Sigma m(6, 7, 3, 1)$$
$$\Rightarrow \Pi_M(0, 2, 4, 5)$$

Ex. $F = xy + \bar{x}z$

	F	$\bar{F}$
SOP	$\Sigma m(1, 3, 6, 7)$	$\Sigma m(0, 2, 4, 5)$
POS	$\Pi_M(0, 2, 4, 5)$	$\Pi_M(1, 3, 6, 7)$

- If $f_1 = \Sigma m(1, 4, 5, \ldots)$ & $f_2 = \Sigma m(1, 2, 5, 7, \ldots)$ then,

(1) $f_1 \cdot f_2 = \Sigma m$ (common minterms) {Intersection}

(2) $f_1 + f_2 = \Sigma m$ (all minterms) {Union}

Ex. Find complement of $F = (a+c)(a+\bar{b})(a+\bar{b}+\bar{c})$

$$F = (a+\bar{b}+c)(a+b+c)(a+\bar{b}+\bar{c})(a+\bar{b}+c)(a+b+\bar{c})$$
$$\Rightarrow \Pi M(2, 0, 3, 1)$$

$$\therefore \bar{F} = \Pi M(4, 5, 6, 7)$$
$$\Rightarrow (\bar{a}+b+c)(\bar{a}+b+\bar{c})(\bar{a}+\bar{b}+c)(\bar{a}+\bar{b}+\bar{c})$$

$$\text{or } \bar{F} = \Sigma m(0, 1, 2, 3)$$
$$\Rightarrow \bar{a}\bar{b}\bar{c} + \bar{a}\bar{b}c + \bar{a}b\bar{c} + \bar{a}bc$$

* Expression Minimization & K-Map

- The minimum expression of a given expression, in SOP / POS form is the expression with min. number of terms possible, with each term having min. number of literals.

- Cannonical SOP & POS are unique for a func., but minimized expression may or maynot be unique.

- K-map is a systematic way of minimizing expressions.

K-map is a special representation of Truth table where each subsequent cell differs to previous by only 1 digit.

Ex. 100 → 000 101 → 111

Hence K-Map's cells are represented using gray code.

a \ b	0	1
0	m_0	m_1
1	m_2	m_3

a \ bc	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

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

The idea of K-map is to start with cannonical SOP, then taking 2 terms at a time we combine them if only one variable changes.

Since K-map is systematic, it always results in the minimized expression.

~~Adjecent~~

- Adjecent Cells : Two cells in a K-map are adjacent iff they differ by a single variable.

Ex. $ab\bar{c}d$ & $ab\bar{c}\bar{d}$ ✓ (13 & 12)

Ex. abc & $a\bar{b}\bar{c}$ ✗ (7 & 4)

Ex. $a\bar{b}\bar{c}$ & $ab\bar{c}$ ✓ (4 & 6)

Ex. $\bar{a}\bar{b}\bar{c}d$ & $a\bar{b}\bar{c}d$ ✓ (1 & 9)

Cube in K-Map : A group of 2^m cells $(m \geq 0)$ such that every cell is adjecent to m cells, in the group.

Ex. In 4 variable K-map, $(m_0, m_4, m_{12}, m_8, m_2, m_6, m_{14}, m_{10})$ form a cube of $2^3 = 8$ cells.

m_0 is adjacent to (m_2, m_4, m_8)

m_{12} is adjacent to (m_4, m_8, m_{14})

The reason to group 2^m cells $(m \geq 0)$ is to get min. SOP.

Each product term covers ~~a~~ 2^K ~~term~~ minterms where $K = n - |P.T.|$.

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

$|a\bar{b}| = 2$ $K = 3 - 2 = 1$ $\therefore 2^K = 2$ $(a\bar{b}c, a\bar{b}\bar{c})$

$|\bar{b}| = 1$ $K = 3 - 1 = 2$ $\therefore 2^K = 2^2 = 4$

$(\bar{a}\bar{b}\bar{c}, \bar{a}\bar{b}c, a\bar{b}\bar{c}, a\bar{b}c)$.

So, when we make cube of size 2^m $(m \geq 0)$, we get a product term.

Ex. $a\bar{b}c + abc + a\bar{b}\bar{c} + ab\bar{c}$

$ac \qquad a\bar{c}$

a { Prod. Term }

So, this prod. term represents $2^2 = 4$ minterms.

Hence only 2^K or 2^m cell groups can be reduced to a prod. term of size $n-K$, where n is no. of inputs.

Ex. $f(a,b)$

a \ b	0	1
0	0	1
1	0	1

$f(a,b) = b$

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

a \ bc	00	01	11	10
0	0	1	1	0
1	0	0	1	1

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

Ex. $f(a,b)$

a \ b	0	1
0	1	1
1	1	0

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

Ex. $f(a,b)$

a \ b	0	1
0	1	1
1	1	1

$f(a,b) = 1$

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

a \ bc	00	01	11	10
0	1	1	0	1
1	1	0	0	1

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

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

a \ bc	00	01	11	10
0	1	0	0	1
1	1	1	1	0

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

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

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

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

Ex. $f(a,b,c,d) = \sum(0,3,5,8,9,11,12)$

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

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

Ex.

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

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

Ex.

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

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

{ Middle cube of size 4 is redundant }.

Ex.

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

• SOP

Same

POS

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

Ex.

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

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

Ex. For a func. if mSOP is not unique then min. POS is also not unique?

False. Ex.

a \ bc	00	01	11	10
0	0	1	0	1
1	0	1	1	1

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

a \ bc	00	01	11	10
0	0	1	0	1
1	0	1	1	1

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

a \ bc	00	01	11	10
0	0	1	0	1
1	0	1	1	1

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

Functions with Don't Care Combinations
(Incompletely specified func.)

For some circuits certain input combinations never occurs, or for the circuit the input combination isn't significant. These input combinations, for which the output doesn't matter, are called Don't Care combinations.

Ex.

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

$f = \Sigma m(1) + d(0, 2)$

A partial func. with K don't care combinations results in 2^K total functions.

Since every D.C. can be 0 or 1, while finding MSOP, each D.C can be included or excluded in a cube. Similarly for MPOS.

Ex.

a \ b	0	1
0	X	1
1	X	0

$f = \bar{a}$, if D.C is considered in cube
$= \bar{a}b$, else

Ex.

a \ bc	00	01	11	10
0		1		X
1		1	1	X

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

Ex.

a \ bc	00	01	11	10
0	0		0	X
1	0			X

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

Ex. $f(w,x,y,z) = \sum m(0,2,4,5,8,14,15) + d(7,10,13)$

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

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

Ex. The largest cube is always present in MSOP ?
No. Above example.

- Covering function

A func. f covers func. g iff

(when g is 1) $\longrightarrow$ (f is 1)

g	f
1 $\rightarrow$	1
0 $\rightarrow$	0/1.

Ex. $f = a+b$ $\quad g = a.b.$
f covers g.

Ex. $f = a \oplus b$ $\quad g = a+b$
g covers f.

Ex. $f = a \oplus b$ $\quad g = ab$.

f doesn't cover g.

g doesn't cover f.

When f covers g & g covers f, $f \equiv g$.

Ex. $f = x\bar{y}\bar{z}$ $\quad g = x\bar{z}$

g covers f.

Since if $f = 1$, $x = 1, \bar{z} = 1 \Rightarrow x\bar{z} = 1 \Rightarrow g = 1$.

Ex. $f = x + \bar{y} + \bar{z}$ $\quad g = x + \bar{z}$

$g = 1 \Rightarrow (x = 1) \vee (\bar{z} = 1) \Rightarrow (x + \bar{y} + \bar{z}) = 1 \Rightarrow f = 1$

$\therefore$ ~~g~~ ~~cov~~ f covers g.

Ex. $f = (x+y)(x+y+z)$ $\quad g = xy + xyz$.

if $g = 1 \Rightarrow (xy = 1) \vee (xyz = 1) \Rightarrow (x = 1) \wedge (y = 1) \Rightarrow (x+y = 1) \wedge (x+y+z = 1)$

$\Rightarrow f = 1$.

$\therefore$ ~~g~~ f covers g.

if $f = 1 \Rightarrow ((x+y) = 1) \wedge ((x+y+z) = 1) \Rightarrow (x+y) = 1 \not\Rightarrow xy = 1 \Rightarrow$

~~g~~.

~~g covers f~~ $\therefore$ g doesn't cover f.

* Implicant, Prime Implicant & Essential Prime Implicant.

- **Implicant:** An implicant is just any cube of size 2^m $(m \geq 0)$ made of all 1's.

Ex.

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

This has 14 implicant.

$2^0 = 1$: 7

$2^1 = 2$: 6

$2^2 = 4$: 1

14.

Implicant is a product term for which func. is 1.

Ex. In above example:

size 1: $\bar{a}\bar{b}\bar{c}\bar{d}$, $\bar{a}\bar{b}cd$, etc.

size 2: $ab\bar{c}$, $\bar{a}cd$, etc.

size 4: ab

Ex. $f^{(a,b,c,d)} = a + cd$. Which is/are implicant?

(A) a ✓

(B) $a\bar{b}$ ✓

(C) $a+c$ ✗ (Not prod term)

(D) $\bar{c}d$ ✗

(E) $ab\bar{c}$ ✓

- **Prime Implicant:** A PI is an implicant which isn't fully contained within any other implicant.

  (or for a cell the largest possible cube covering it.)

Ex.

1		1	
		1	
1	1	1	1

Implicants = 14

PI = 4

- **Essential Prime Implicants:** EPI is a PI which includes at least one such "1" cell which isn't covered by any other PI.

Ex.

1		1	
		1	
1	1	1	1

Implicants = 14

PI = 4

EPI = 3

$f = \bar{w}\bar{x}\bar{y}\bar{z} + wx + \bar{w}yz$

Ex.

a \ bc	00	01	11	10
0	1		1	
1	1	1		

Is = 6

PIs = 3

EPIs = 3

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

Ex.

a \ bc	00	01	11	10
0		1	1	1
1	1		1	

Is = 8
PIs = 4
EPIs = 4

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

Ex.

a \ bc	00	01	11	10
0	1	1	1	1
1		1	1	1

Is = 7 + 9 + 3 = 19
PIs = 0 + 0 + 3 = 3
EPIs = 0 + 0 + 3 = 3

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

Ex.

a \ bc	00	01	11	10
0	1	1	1	
1		1	1	1

Is = 6 + 6 + 1 = 13
PIs = 0 + 2 + 1 = 3
EPIs = 0 + 2 + 1 = 3

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

Note: If all EPIs cover all PIs then we get unique mSOP.

Ex.

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

Is = 5 + 5 + 1 = 11
PIs = 0 + 1 + 1 = 2
EPIs = 0 + 1 + 1 = 2.

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

Ex.

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

Is = 8 + 8 + 1 = 17

PIs = 0 + 4 + 1 = 5

EPIs = 0 + 4 + 0 = 4

$$f = w\bar{x}\bar{y} + wyz + \bar{w}\bar{x}y + \bar{w}\bar{y}z$$

Ex.

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

Is = 6 + 6 + 1 = 13

PIs = 0 + 2 + 1 = 3

EPIs = 0 + 2 + 1 = 3

$$f = bd + \overset{a\bar{c}d}{\cancel{abc}} + abc$$

Ex.

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

Is = 9 + 10 + 1 = 20

PIs = 0 + 6 + 1 = 7

EPIs = 0 + 0 + 1 = 1

Ex.

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

↳ EPI

Is = 10 + 13 + 4 = 27

PIs = 0 + 0 + 4 = 4

EPIs = 0 + 0 + 2 = 2

EPI

$$f = \underbrace{\bar{a}\bar{d} + ad}_{EPI} + cd$$

$$f = \underbrace{\bar{a}\bar{d} + ad}_{EPIs} + \bar{a}c.$$

Note: To get mSOP, take all EPIs & if there are 1-cell remaining, cover them with min. possible PIs.

Note: If in the KMap, we have don't cares, we consider them in ~~P/I~~ Implicants as 1s. Similarly for PIs, consider DCs as 1s. For EPIs, may/maynot consider DCs as 1s.

Ex.

a \ bc	00	01	11	10
0	1	X	1	X
1	X			X

Is = 6 + 7 + 2 = 15

PIs = 0 + 0 + 2 = 2

EPIs = 0 + 0 + 1 = 1

- Cyclic K-Map: A K-map in which no PI is EPI, all PIs have same size & every 1-cell is covered by atleast 2 ~~exactly one~~ PI.

  - Cyclic KMaps have 2 mSOPs.

- Alternate defn of PI:
  - From an implicant, if one literal is removed, it may or maynot be implicant.
  - PIs are product terms, as small as possible, for which f is 1.

- If a literal is removed from PI, it will definetly not be a implicant.

Ex.

a \ bc	00	01	11	10
0	1			1
1	1	1		1

$\bar{a}\bar{b}\bar{c}$ is Implicant.

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

Here $\bar{b}\bar{c}$ & $\bar{a}\bar{c}$ are still implicant, but $\bar{a}\bar{b}$ isn't.

$a\bar{b}$ is Prime Implicant.

a $\quad$ $\bar{b}$

Neither a nor $\bar{b}$ is an implicant anymore.

- Note: Is, PIs & EPIs are all product terms (not sum term), i.e. they cover 1-cells (not 0-cells).

NUMBER THEORY

* Octal System

Digits : 0, 1, 2, 3, 4, 5, 6, 7.

Powers : $\ldots\ldots, 8^3, 8^2, 8^1, 8^0$

Ex. $(7)_{10}$: $(7)_8$

$(8)_{10}$: $(10)_8$

$(16)_{10}$: $(20)_8$

$(42)_{10}$: $(52)_8$

$(79)_{10}$: $(117)_8$

* Hexadecimal System

Digits : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f.

Powers : $\ldots\ldots\ldots, 16^3, 16^2, 16^1, 16^0$

Ex. $(7)_{10}$: $(7)_{16}$

$(16)_{10}$: $(10)_{16}$

$(32)_{10}$: $(20)_{16}$

$(55)_{10}$: $(37)_{16}$

$(79)_{10}$: $(4f)_{16}$

* Base-r to Base-10 Conversion

Say $(abcde)_r$.

$$\Rightarrow (abcde)_r = (e \times r^0 + d \times r^1 + c \times r^2 + b \times r^3 + a \times r^4)_{10}$$

So, $(\ldots edcba)_r = (a \times r^0 + b \times r^1 + c \times r^2 + d \times r^3 + e \times r^4 + \ldots)_{10}$.

Ex. $(2af)_{16} = 15 \times 16^0 + 10 \times 16 + 2 \times 16^2 = 15 + 160 + 512 = 687$

$(327)_8 = 7 \times 8^0 + 2 \times 8 + 3 \times 8^2 = 7 + 16 + 192 = 215$

Similarly, $(abc.de)_r$

$$\Rightarrow [(c \times r^0 + b \times r^1 + a \times r^2) + (d \times r^{-1} + e \times r^{-2})]_{10}$$

Ex. $(101.01)_2 = (1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2) + (0 \times 2^{-1} + 1 \times 2^{-2})$

$\Rightarrow 5.25$

* Base-10 to Base-r Conversion.

(1) Repeated division. (for integer part)

Say, $(abcde)_{10}$. such that

$$(abcde)_{10} = (x \times r^2 + y \times r + z)_{10}$$

S1: $(x \times r^2 + y \times r + z) \% r = z$

$/r = (x \times r + y)$

S2: $(x \times r + y) \% r = y$

$/r = (x)$

S3: $(x) \% r = x$

$\therefore (abcde)_{10} = (xyz)_x$

Ex. $(259)_{10}$ to Base-2, Base 8, Base 16.

(1)

2	259	1
2	129	1
2	64	0
2	32	0
2	16	0
2	8	0
2	4	0
2	2	0
2	1	1
	0	

$\therefore (259)_{10} = (100000011)_2$

(2)

8	259	3
8	32	0
8	4	4
	0	

$\therefore (259)_{10} = (403)_8$.

(3)

16	259	3
16	16	0
16	1	1
	0	

$\therefore (259)_{10} = (103)_{16}$

(2) Repeated Multiplication (for fractional part)

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

S1: $\left(\frac{x}{r} + \frac{y}{r^2} + \frac{z}{r^3}\right) \times r = \underline{\underline{x}} + \left(\frac{y}{r} + \frac{z}{r^2}\right)$

S2: $\left(\frac{y}{r} + \frac{z}{r^2}\right) \times r = \underline{\underline{y}} + \left(\frac{z}{r}\right)$

S3: $\left(\frac{z}{r}\right) \times r = \underline{\underline{z}} + (0)$.

Ex. $(0.152)_{10}$ to B-2, ~~[illegible]~~

(1)

$0.152 \times 2 =$	0.304	0
$0.304 \times 2 =$	0.608	0
$0.608 \times 2 =$	1.216	1
$0.216 \times 2 =$	0.432	0
$0.432 \times 2 =$	0.864	0
$0.864 \times 2 =$	1.728	1
$0.728 \times 2 =$	1.456	1
$0.456 \times 2 =$	0.912	0
$0.912 \times 2 =$	1.824	1
$0.824 \times 2 =$	1.648	1
$0.648 \times 2 =$	1.296	1
⋮		

$(0.152)_{10} = (0.00100110111..)_2$

Ex. $(25.625)_{10}$ to B2.

2	25	1
2	12	0
2	6	0
2	3	1
2	1	1
	0	

$(25)_{10} = (11001)_2$

$0.625 \times 2 = 1.250 \quad 1$

$0.256 \times 2 = 0.500 \quad 0$

$0.5 \times 2 = 1.0 \quad 1$

0.0

$(0.625)_{10} = (0.1001)_2$

$\therefore (25.625)_{10} = (11001.101)_2$

Ex. $(23.31)_4$ to B7.

$B4 \longrightarrow B10 \longrightarrow B7.$

$(23.31)_4 = (2 \times 4 + 3) + (3/4 + 1/4^2)$

$\Rightarrow (11.8125)_{10}$

$(11)_{10} = (14)_7$

$0.8125 \times 7 = 5.6875 \quad 5$

$0.6875 \times 7 = 4.8125 \quad 4$

0.8125 (Repeat)

$\therefore (11.8125)_{10}$

$\Rightarrow (14.54..)_7$

$\therefore (0.8125)_{10} = (0.54...)_7$

* Binary, Octal & Hexadecimal

(1) Binary to Octal

Every 3 Binary digits in a number make one Octal digit. (Starting from LSB)

Ex. $(1010110100)_2$ •

$\cancel{1}\,\cancel{010}\,\cancel{110}\,\cancel{100}$

$(\underline{001}\ \underline{010}\ \underline{110}\ \underline{100})_2 = (1264)_8$

Ex. $(547)_8$

$(\underline{101}\ \underline{100}\ \underline{111})_2$

(2) Binary to Hexadecimal

Every 4 Binary digits in a number make one Hex digit. (Starting from LSB).

Ex. $(10101101110)_2$

$(\underline{0101}\ \underline{0110}\ \underline{1110})_2 = (56d)_{16}$

Ex. $(2af7)_{16}$

$(0010\ 1010\ 1111\ 0111)_2$

* Binary Arithmetic

* Binary Addition

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Ex.
```
  101101
+ 100001
  1001110
```

* Binary Subtraction

A	B	A-B=D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Ex.
```
  101000
- 100111
  000001
```

Ex.
```
  11101
- 10011
  01010
```

* Complements.

When we subtract b from a, the operation becomes complicated if there is borrow.

In Base-10 if we subtract x from all 9s there won't be any borrow.

Ex. $x = 543$

$$\begin{array}{r} 999 \\ 543 \\ \hline 456 \end{array}$$

$x = 12345$

$$\begin{array}{r} 99999 \\ 12345 \\ \hline 87654 \end{array}$$

We can use this information.

Ex. 82 - 64.

$\Rightarrow 82 - 64 + 100 - 100$

$\Rightarrow 82 + (99 - 64) + 1 - 100$

$\Rightarrow (82 + 35 + 1) - 100$

$\Rightarrow \underline{\underline{1}}18 - 100$

end carry

$\Rightarrow 18$ { Discarded the end carry }

- 9's Complement of x = (All 9s − x)

* Complement of a number in base r

For any number in base r there are two complements:

(1) (r-1)'s Complement (2) r's complement.

Say, $r = 10$, then 9's & 10's complement.
Say, $r = 2$, then 1's & 2's complement.

for any number N in base r with n digits :

$(r-1)$'s comp.	r's comp.
$\Rightarrow (r^n - 1) - N$	$\Rightarrow (r^n) - N$

Ex. Base 10 $N = 536$

9's compl.	10's Compl.
$(10^3 - 1) - 536$	$(10^3) - 536$
$\Rightarrow 463$	$\Rightarrow 464$.

- r's compl. of x = $(r-1)$'s comp. of $x + 1$.

Ex. Base 2. $N = 10110$

1's comp.	2's Compl.
$(11111 - 10110)$	$(100000 - 10110)$
$\Rightarrow 01001$	$\Rightarrow 01011$

or $01001 + 1 = 01011$

- 2's complement :
  (1) Starting from right copy bits as it is till first 1.
  (2) After that copy each bits complement.

Ex. 1010011000

↑

0101101000

- Subtraction using r's complement

There can be two cases when doing
M - N

C1: M >= N

M - N = M + (r's compl. of N)

⇒ c x y z

↓

end carry.
(discard)

∴ M-N = xyz.

C2: M < N

M - N ⇒ M + (r's compl. of N)

⇒ xyz

∴ M - N = - (r's compl. of xyz)

Ellipse

Ex. 98 - 34

$\Rightarrow 98 + 66$

$\Rightarrow \underset{\times}{1}64$

$\Rightarrow 64.$

34 - 98.

$\Rightarrow 34 + 2.$

$\Rightarrow 36.$

$\Rightarrow -$ (10's comp. of 36)

$\Rightarrow -64$

Ex. $(\underset{(22)}{10110} - \underset{(17)}{10001})$

$\Rightarrow 10110 + 01111$

$\Rightarrow \underset{\times}{1}00101$

$\Rightarrow (00101)$

$(10001 - 10110)$

$\Rightarrow 10001 + 01010$

$\Rightarrow 11011$

$\Rightarrow -$ (2's compl. of 11011)

$\Rightarrow -00101$

Ex. $(1111010 - 1101011)$

$\Rightarrow 1111010 + 0010101$

$\Rightarrow \underset{\times}{1}0001111$

$\Rightarrow (0001111)$

* Subtraction using (r-1)'s complement

C1: $M \geq N$

$M - N$

S1: $M +$ [(r-1)'s compl. of N]

S2: Discard End Carry

S3: Add 1.

C2: $M < N$

$M - N$

S1: $M +$ [(r-1)'s compl. of N]

S2: Take (r-1)'s compl. of result.

S3: $-$ ((r-1)'s compl. of result)

Ex. 82 - 24

S1 ⇒ 82 + 75 = 157

S2: 157
$\overline{\overline{X}}$

S3: 57+1 = 58.

24 - 82

S1: 24 + 17 = 41

S2: 9's Compl. ⇒ 58

S3: ⇒ -58.

Ex. 1111010 - 1010110

S1: 1111010 + (0101001)

⇒ 10100011

S2: 10100011
$\overline{\overline{X}}$

S3: 01000111 + 1 =

⇒ 0100100

1010110 - 1111010

S1: 1010110 + (0000101)

⇒ 1011011

S2: $-$(0100100)

* Representation of Signed Numbers

There are three representations:
(1) Signed Representation
(2) 1's Compl. Representation
(3) 2's Compl. Representation.

• Signed Magnitude

For every number, an additional bit is used to represent sign.

0 : +ve

1 : -ve.

Ex. $(-1)_{10}$: (11) or (101) or (1001)

Ex: $(1)_{10}$: (01) or (001) (1) X {no sign bit}

Ex: $(-65)_{10}$: $\underbrace{1}_{-}\underbrace{1000001}_{65}$

0 has 2 repr: 00 : 0

10 : -0.

Range : $-(2^{n-1}-1)$ to $(2^{n-1}-1)$

• 1's Complement System

Every +ve value is appended with 0 at MSB side.

To get negative counterpart, take 1's complement.

Ex. -51

51 = 0110011 -51 = 1001100.

Ex. -8

8 = 01000 -8 = 10111

Ex. 0

0 = 00 also -0 = 11

So, 1's Compl. Representation also has 2 repr. for 0.

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

- 2's Complement System

Every positive value is appened with a 0 bit at MSB side.
To get the negative counterpart, take 1's complement.

Ex. -51

51 = 0110011 -51 = 1001101

Ex. -8

8 = 01000 -8 = 11000

In 2's Complement System, value can be found out as:

C1: ~~MSB is 1~~ MSb is 1.

Interpret MSb bit as neg. power of 2, rest as +ve power.

Ex. 1001101

$-2^6 + 2^3 + 2^2 + 2^0$

$\Rightarrow -64 + 8 + 4 + 1$

$\Rightarrow -51$

C2: MSb is 0.

Interpret MSb bit & all other bit as +ve power of 2.

Ex. 0001101

$2^3 + 2^2 + 2^0$

$\Rightarrow 8 + 4 + 1$

$\Rightarrow 13.$

There is only one representation of 0, since (1000..0) can be interpreted as -ve number.

Ex. 10000

-2^4

$\Rightarrow -16$

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

* Addition in Signed Magnitude.

(1) $(+M) + (+N) = +(M+N)$

(2) $(-M) + (-N) = -(M+N)$

(3) $(+M) + (-N) = \begin{cases} +(M-N) & M > N \\ -(N-M) & N > M \end{cases}$

Steps for addition :

(1) Comparision of sign bit

(2) Comparision of magnitude.

(3) Perform one of four above operation.

Ex. 1011 + 0100

Sign: -ve +ve.

Mag: 3 < 4.

$\Rightarrow +(001)$

$\Rightarrow 0001$

Subtraction of Signed Magnitude.

$$M - N = M + (-N) = \begin{cases} +(M-N) & M > N \\ -(N-M) & N > M \end{cases}$$

Ultimately subtraction requires a separate circuit.

* Addition of 2's Complement

Ex. (+5) + (+6)

00101 00110

```
(+5)      00101
+(+6)     00110
+11       01011
```

Ex. (+5) + (-4)

0101 1100

```
(+5)      0101
+(-4)   + 1100
+1      10001
          ≡
          X
```

Ex. (+5) + (-6)

0101 1010

```
(+5)      0101
+(-6)   + 1010
-1        1111
```

⇒ -(0001)

Ex. (-5) + (-6)

~~00101~~ ~~1010~~

11011 11010

```
(-5)        11011
+(-6)       11010
-11       1 10101
            ≡
            X
```

-(01011)

Since both the inputs are in 2's complement, output would also be in 2's complement.

- Subtraction in 2's Complement

$M - N = M + (-N)$

$\Rightarrow [2's(M)] + [2's(N)]$

$\Rightarrow 2's(M-N)$

Ex. 5-4

$\Rightarrow 5 + (-4)$

$\Rightarrow 0101 + 1100$

$\Rightarrow \underset{\cancel{X}}{1}0001$

$\Rightarrow (0001)$

$\Rightarrow (1)_{10}$

Ex. 5-(-4)

$\Rightarrow 5 + (-(-4))$

$\Rightarrow 5+4$

$\Rightarrow 00101 + 00100$

$\Rightarrow (01001)$

$\Rightarrow (9)_{10}$

Ex. -5 +4

$\Rightarrow (-5) + (4)$

$\Rightarrow 1011 + 0100$

$\Rightarrow 1111$

$\Rightarrow (-1)_{10}$

2's complement subtraction doesn't require additional circuits.

* Addition in 1's Complement

Ex. (+5) + (+6)

00101 00110

⇒ 01011

⇒ $(+11)_{10}$

Ex. (+5) + (-4)

00101 101011

~~01110~~ 00101

+ 111011

1 00000

~~1~~ +1

00001

∴ $(+1)_{10}$

Ex. 4 + (-9)

00100 10110

00100

10110

11010

∴ - (00101)

⇒ $(-5)_{10}$

Ex. (-4) + (-9)

11011 10110

11011

10110

1 10001

~~1~~ + 1

10001

So, in 1's Complement form, extra addition operation in case end carry is there.

* Hence 2's complement is by default used to represent signed numbers in computers.

Computer always assumes that signed numbers are in 2's complement form.

For Addition of signed nos. A & B, i.e., (A+B), it just uses adder circuit, and the result is in 2's comp. form.

For subtraction of A & B, i.e. (A-B) it first takes 2's complement of B & then uses adder circuit to add A & 2's (B), i.e., A + 2's (B), & result is in 2's compl. form.

* Overflow

When a n-bit data is added to another n-bit data, sometimes the result might need (n+1) bits, this situation is known as Overflow.

Ex. 4 bit Addition of 4 & 6.

```
  0100
  0110        No overflow
  1010
```

Ex. 4 bit addition of 10 + 9.

```
10     1010
+9   + 1001        Overflow
19     10011 ⇒ 19
```

Overflow depends on the operation, i.e. addition, subtraction, multiplication.

It also depends on the representation of data.

Ex. $R_1 = 10$ $R_2 = 2$ (4 bit Unsigned)

① Addition: 1010 + 0010 = 1100
② Subtraction: 1010 - 0010 = 1000
③ Multiplication: 1010 × 0010 = 10100

Only mult. results in overflow.

Def^n: When an operation # is performed on two values A & B, represented using n bits, and the result can't be represented using n bits, it is known as overflow.

Ex. $R_1 = 4$ $R_2 = 5$ (4 bits) (Addition)

① Unsigned: 0100 + 0101 = (1001) = $(9)_{10}$

② 2's compl.: 0100 + 0101 = (1001) = $(-7)_{10}$
↳ Overflow!!

• Unsigned Addition

In Unsigned Addition, the end carry represents overflow.

• end carry $\Longleftrightarrow$ Overflow.

Ex. 1 bit

$$\begin{array}{r} 0 \\ +0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 0 \\ +1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +1 \\ \hline \underline{\underline{10}} \end{array}$$

end carry = Overflow

Ex. $\begin{array}{r} 01 \\ +10 \\ \hline 11 \end{array} \qquad \begin{array}{r} 01 \\ 11 \\ \hline \underline{\underline{100}} \end{array}$

overflow

So,

$$\begin{array}{rl} A & a_n \ldots a_3\ a_2\ a_1\ a_0 \\ +B & b_n \ldots b_3\ b_2\ b_1\ b_0 \\ \hline & r_{n+1}\ r_n \ldots r_3\ r_2\ r_1\ r_0 \end{array}$$

Overflow iff $(r_{n+1} == 1)$

• For any signed representation, if one operand is -ve and another is +ve & addition is performed, overflow will never occur.

When sign of both operands is same, then overflow might occur.

• Signed Magnitude Addition

A	S_A	Mag_A
+ B	S_B	Mag_B

C1: $S_A \neq S_B$

No overflow

C2: $S_A = S_B$

If end carry when adding ($Mag_A + Mag_B$) then overflow.

Ex. 4 bit.

$(-6) + (-3)$

$$\begin{array}{r} 1\,1\,1\,0 \\ 1\,0\,1\,1 \\ \hline 1\,1\,0\,0\,1 \end{array}$$

↳ overflow

$-6 + 3$

No overflow

$6 + 3$

$$\begin{array}{r} 0\,1\,1\,0 \\ 0\,0\,1\,1 \\ \hline 0\,1\,0\,0\,1 \end{array}$$

↳ overflow.

A	S_A	$a_m \ldots a_2 a_1$
+ B	S_B	$b_m \ldots b_2 b_1$

C1: $a_m = b_m = 1$ Overflow

C2: $a_m = b_m = 0$ No Overflow

C3: $a_m \neq b_n$ May / Maynot be overflow

2's Complement Addition

Overflow iff both operands have same sign & the result has different sign.

i.e.

$(+A) + (+B) = -R$

or $(-B) + (-A) = +R$

then overflow.

Ex. 3 bits (Range: 8-4 to 3)

011	010	110
+010	+001	+110
101	011	1100
Overflow	No overflow	Discard. No Overflow.

Suppose we have 2's Complement nos.

$A = a_m \ldots a_2 a_1$ & $R = r_m \ldots r_2 r_1$

$B = b_m \ldots b_2 b_1$

then overflow occurs iff:

(1) $(a_m \odot b_m) \cdot (a_m \oplus r_m)$ or $a_m b_m \bar{r}_m + \bar{a}_m \bar{b}_m r_m$

(2) If carry to MSb is C_{in} & carry out of MSB is C_{out} then overflow iff $C_{in} \oplus C_{out} = 1$

- 1's Complement Addition

Overflow iff both operand have same sign & the result has a different sign.

Ex. 3 bit

3	011	2	010
+2	010	+1	+ 001
5	101		011
	↳ Overflow.		No overflow.

-2	101	-2	101
-2	101	-1	1010
	1010		1011
	+1		+1
	011		100
	Overflow.		No Overflow.

* Sign Extension

1. Unsigned

While going from n to m bits (m > n), copy the n bits to least significant position, then append (m-n) 0s on left.

Ex. 3 bit to 5 bit

(101) ⇒ (00101)

(110) ⇒ (00110)

(001) ⇒ (00001)

2. Signed Magnitude.

(m > n)

① Copy (n-1) bits.

② Fill (m-n) bits with 0s

③ Copy n^{th} bit to m^{th} position.

Ex. 3 bit to 6 bit

(101) ⇒ (100001)

(110) ⇒ (100010)

(001) ⇒ (000001)

$a_n a_{n-1} \ldots a_2 a_1 \Rightarrow a_n \underbrace{00\ldots0}_{(m-n)\,0s} a_{n-1} \ldots a_2 a_1$

3. 1's Complement

$m > n$

① Copy $n-1$ bits.

② Extend sign bit to $(m-(n-1))$ positions.

Ex. 3 bit to 6 bit.

(110) ⇒ (111110)

(101) ⇒ (111101)

(001) ⇒ (000001)

4. 2's Complement

$m > n$.

① Copy $n-1$ bits.

② Extend sign bit to $(m-(n-1))$ positions.

Ex. 3 bit to 6 bit.

(110) ⇒ (111110)

(101) ⇒ (111101)

(001) ⇒ (000001)

* Binary Codes

Any pattern of 0s & 1s is called Binary Codes.

To represent m values using binary code, we require $\lceil \log_2 m \rceil$ bits, i.e.

$$2^n \geq m$$
$$\Rightarrow n \geq \log_2 m$$
$$\Rightarrow n = \lceil \log_2 m \rceil$$

Ex. $\{a, b, c\}$ $m = 3$

$$n = \lceil \log_2 m \rceil = \lceil \log_2 3 \rceil = 2$$

* BCD Code

A binary code for decimal.
BCD: Binary Coded Decimal.

Each digit from 0 to 9 is represented as 4 digit binary.

0	0000	6	0110
1	0001	7	0111
2	0010	8	1000
3	0011	9	1001
4	0100		
5	0101		

Ex.

Data	BCD
23	0010 0011
483	0100 1000 0011
321	0011 0010 0001

There are $2^4 - 9 = 6$ unused BCD codes.
They are called invalid BCD codes.

BCD code is also called 8421 code, since each digit is represented using the weights. 8, 4, 2 & 1.

* 2421 Code.

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

When two alternatives, lower binary value is chosen.

like for 3 : (0011) = 3 ✓
(1001) = 9

for 6 : (0110) = 6 ✓
(1100) = 12

etc.

* Self Complementary Code.

A code is called Self Complementary code if the 9's complement of its digit in decimal is equivalent to ~~the~~ 1's complement of its digit in binary.

Like

9's: 2 → 7

1's: 0010 → 1101

Ex. 2421

0 ↔ 9	0000 ↔ 1111
1 ↔ 8	0001 ↔ 1110
2 ↔ 7	0010 ↔ 1101
3 ↔ 6	0011 ↔ 1100
4 ↔ 5	0100 ↔ 1011

* Excess-3 Codes.

Every digit is represented as +3 in binary.

Dec.	E-3	Dec.	E-3
0	0011	5	1000
1	0100	6	1001
2	0101	7	1010
3	0110	8	1011
4	0111	9	1100

E-3 codes are similar to BCD, but they are self complementary in addition. And its unweighted unlike BCD.

* 84-2-1 Codes.

Dec.		Dec.	
0	0000	5	1011
1	0111	6	1010
2	0110	7	1001
3	0101	8	1000
4	0100	9	1111

84-2-1 is also self-complementary code.

* Gray Code.

Gray Code is a unit distance code. Every adjoining digit differs by only one bit. It is also unweighted.

Dec		Dec.	
0	0000	5	0111
1	0001	6	0101
2	0011	7	0100
3	0010	8	1100
4	0110	9	1101

Dec.	
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000.

- Applications :
  1. K-map
  2. Analogue to Digital conversion.

- Gray code is beneficial since ~~from~~ going from n to n+1 only one bit has to change.

- Finding Gray Code of decimal nos.

G_n: Gray code with n bits

Say. G_1

Dec.	Gray.
0	0
1	1

G_2

0	0	0
1	0	1
2	1	1
3	1	0

mirror

G_3

Dec.	Gray	
0	0	00
1	0	01
2	0	11
3	0	10
4	1	10
5	1	11
6	1	01
7	1	00

Hence Gray Codes are also called Reflexive Codes.

G4.

0	0 000
1	0 001
2	0 011
3	0 010
4	0 110
5	0 111
6	0 101
7	0 100
8	1 100
9	1 101
10	1 111
11	1 110
12	1 010
13	1 011
14	1 001
15	1 000

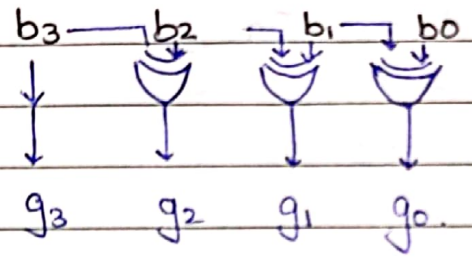
This recursive method is time consuming.

- Method 2:
  1. find decimal to binary form.
  2. Starting from lhs apply this algo.

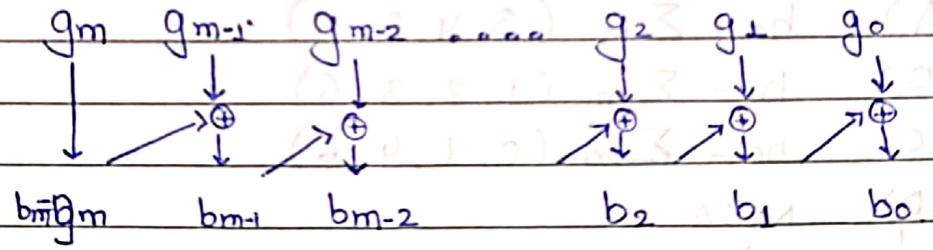
Binary: $b_n \quad b_{n-1} \quad b_{n-2} \ldots\ldots \quad b_1 \quad b_0$

Gray: $b_n \quad (b_n \oplus b_{n-1}) \quad (b_{n-1} \oplus b_{n-2}) \quad (b_2 \oplus b_1) \quad (b_1 \oplus b_0)$

Ex. 29 : $(11101)_2$: $(10011)_{Gray}$
32 : $(100000)_2$: $(110000)_{Gray}$
15 : $(1111)_2$: $(1000)_G$
13 : $(1101)_2$: $(1011)_G$



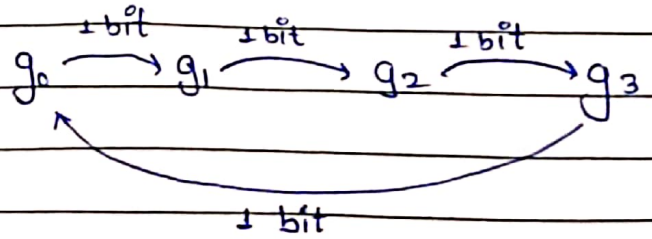
- Gray to Binary



Ex. $(1101)_G$: $(1001)_2$: $(9)_{10}$
$(1111)_G$: $(1010)_2$: $(10)_{10}$
$(1000)_G$: $(1111)_2$: $(15)_{10}$
$(101011)_G$: $(110010)_2$: $(50)_{10}$

- Properties of G_n

① Gray Codes are cyclic.



② G_n is just a permutation of n-bit bit strings.

③

$$G_n: \quad a_0 \quad a_1 \quad a_2 \ldots\ldots \quad a_{2^n-1}$$
$$\downarrow \quad \downarrow \quad \downarrow \qquad\qquad \downarrow$$
$$G_{n+1}: \quad 0a_0 \quad 0a_1 \quad 0a_2 \qquad\qquad 0a_{2^n-1}$$

Ex. Consider $g_2g_1g_0$ as 3 bit Gray code repr. & $h_2h_1h_0$ as $(g_2g_1g_0+1) \bmod 8$. Which is correct?

(A) $h_2 = \Sigma m\ (3, 4, 5, 6)$

(B) $h_1 = \Sigma m\ (1, 2, 3, 6)$ ✓

(C) $h_0 = \Sigma m\ (0, 1, 4, 6)$

(D) NOTA

Dec		$g_2g_1g_0$	$h_2h_1h_0$
0	m_0	000	001
1	m_1	001	011
2	m_3	011	010
3	m_2	010	110
4	m_6	110	111
5	m_7	111	101
6	m_5	101	100
7	m_4	100	000

$h_2 = \Sigma m\ (2, 6, 7, 5)$

$h_1 = \Sigma m\ (1, 3, 2, 6)$

$h_0 = \Sigma m\ (0, 1, 6, 7)$

- Another method to find Gray code.

  n is decimal.

  $$\text{Gray}(n) = \text{Bin}(n) \oplus \text{Bin}\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$$

Ex. $n = 29$.

$$\text{Gray}(29) = \text{Bin}(29) \oplus \text{Bin}(14)$$

$$\Rightarrow 11101 \oplus 01110$$

$$\Rightarrow 10011$$

also.

$$(29)_{10} \Rightarrow \begin{array}{ccccc} 1 & 1 & 1 & 0 & 1 \\ \downarrow & \searrow & \searrow & \searrow & \searrow \\ 1 & 0 & 0 & 1 & 1 \end{array}$$

FUNCTIONAL COMPLETNESS

A set of operations which can be used to express any boolean expression possible is called "functional complete" set of operations.

Ex. {AND, OR, NOT} is functionally complete.
Ex. {NAND, NOR}
Ex. {NAND}

* {AND, OR, NOT}

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

Proof: Any boolean func. can be represented using Truth table.
We can express T.T as SOP or POS, i.e., only use $\{\cdot, +, \sim\}$.

* {AND, NOT} , {OR, NOT}

$A + B = \overline{\bar{A} \cdot \bar{B}}$ $\qquad$ $A \cdot B = \overline{\bar{A} + \bar{B}}$

$\Downarrow$ $\qquad$ $\Downarrow$

$\{\cdot, \sim\}$ is func. complete $\qquad$ $\{+, \sim\}$ is func. complete

Ex. $x(\underbrace{\bar{y}+\bar{z}}_{x \quad B}) + \bar{x}y$

$\Rightarrow \underbrace{x(\overline{yz})}_{\alpha} + \underbrace{\overline{xy}}_{b}$

$\Rightarrow \overline{\overline{x\overline{yz}} \cdot \overline{\overline{xy}}}$

$\Rightarrow \overline{\overline{x\overline{yz}} \cdot xy}$

Ex. $\underbrace{\bar{x}}_{A}\underbrace{y}_{B} + \underbrace{x}_{A}\underbrace{\bar{y}}_{B}$

$\Rightarrow \overline{(\bar{\bar{x}}+\bar{y})} + \overline{(\bar{x}+\bar{\bar{y}})}$

$\Rightarrow \overline{(x+\bar{y})} + \overline{(\bar{x}+y)}$

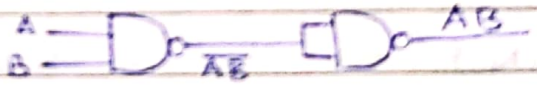
* {NAND}

NOT:

$\bar{A} = \overline{A.A}$

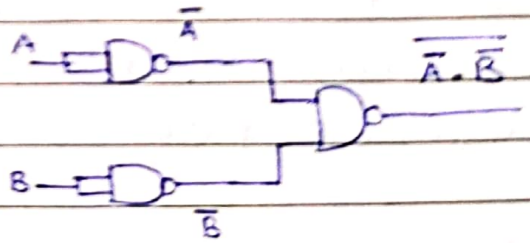
AND

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



OR

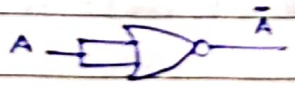
$A+B = \overline{\overline{A.A} \cdot \overline{B.B}}$



* {NOR}

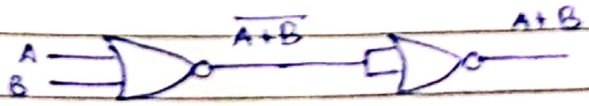
NOR:

$\bar{A} = \overline{A+A}$



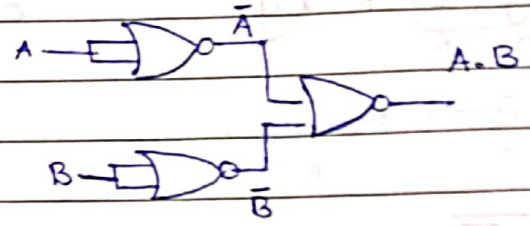
OR

$A+B = \overline{\overline{A+B}}$



AND:

$A \cdot B = \overline{\bar{A} + \bar{B}}$



- NOR & NAND are universal gates since a single type of gate can be used to realise any boolean expression.

- To prove that a set A of operations is functionally complete, derive a set B of operations such that B is functionally complete.
  Ex. {NOR} → {NOT, OR, AND}

- For a set of func. $\{F_1, F_2, F_3, \ldots\}$
  1. If all func. $F_i = F_i^D$ {self dual} then not func. complete.

  2. If all func. $F_i(a=0, b=0, \ldots) = 0$ then not func. complete.

  3. If all func. $F_i(a=1, b=1, \ldots) = 1$, then not func. complete.

Ex. {AND, OR}

① $a+b \xRightarrow{\text{Dual}} a.b$ Not self dual

② f_1: AND $\Rightarrow f_1(0,0) = 0.0 = 0$
f_2: OR $\Rightarrow f_2(0,0) = 0+0 = 0$.

all func. $f_i(0,0,0,\ldots) = 0$

$\therefore$ Not func. complete.

Ex. {NOT}

① $\bar{a} \xRightarrow{\text{Dual}} \bar{a}$ self dual.

$\because$ all operations in set are self dual

$\therefore$ Not func. complete.

Ex. {XOR, OR}

① $f_1: a \oplus b \Rightarrow f_1(0,0) = 0$
$f_2: a+b \Rightarrow f_2(0,0) = 0$

$\because$ all func. in set
$f_i(0,0,\ldots) = 0$

$\therefore$ Not func. complete.

Ex. Prove $\{\rightarrow, \sim\}$ is functionally complete.

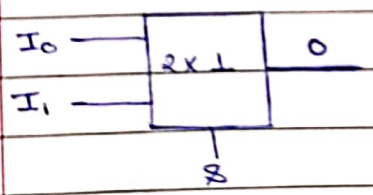
w.k.t. $\{+, \sim\}$ is func. complete.

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

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

$\therefore \{\rightarrow, \sim\}$ is func. complete.

Ex. Check if $\{2\times1$ MUX, 1, 0$\}$ is func. complete



$f = \bar{S} I_0 + S I_1$

we can get NOT by setting:

$I_0 = 1 \quad I_1 = 0 \quad S = A \qquad f = \bar{A}.1 + A.0 = \bar{A}$

AND:

$I_0 = 0 \quad S = A \quad I_1 = B$

$f = 0\bar{S} + AB$

$\Rightarrow AB$.

OR

$I_0 = A \quad S = B \quad I_1 = 1$

$f = \bar{B}A + B.1$

$\Rightarrow A+B$.

Ex. Check $\{\rightarrow, 0\}$.

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

$\{\bar{A} \rightarrow B = \overline{\bar{A} + B}\}$

NOT: Set $b = 0$. $f = a \rightarrow 0$

OR: Set $a = \bar{A}$, $b = B$. $f = \bar{A} \rightarrow B \Rightarrow (\bar{A} \rightarrow 0) \rightarrow B$.

AND: Set $a = A$, $b = \bar{B}$ $f = \overline{A \rightarrow \bar{B}} \Rightarrow \overline{A \rightarrow (B \rightarrow 0)}$

$\Rightarrow (A \rightarrow (B \rightarrow 0)) \rightarrow 0$.

$\{\overline{A \rightarrow \bar{B}} = \overline{\bar{A} + \bar{B}} = A.B\}$

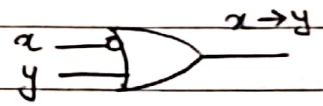
Ex. Check $\{\rightarrow, 1\}$

① $a \rightarrow b = \bar{a} + b \xrightarrow{\text{Dual}} \bar{a}.b$ ✗

② $f_1(0,0) = 0 \rightarrow 0 = 1$ ✗

③ $f_1(1,1) = 1 \rightarrow 1 = 1$
$f_2(1) = 1$ $\Big\}$ $\therefore$ Not func. complete.

Ex. Realise $f = \bar{x}y + x\bar{y}$ using "$\rightarrow$" func., with at most 4 implication gates.



$f = x \rightarrow y \equiv \bar{x} + y$

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

$\Rightarrow \overline{(x + \bar{y})} + \overline{(\bar{x} + y)}$

$\Rightarrow (x + \bar{y}) \rightarrow ((x \rightarrow y) \rightarrow 0)$

$\Rightarrow (y \rightarrow x) \rightarrow ((x \rightarrow y) \rightarrow 0)$

* Linear func. : A boolean func. f is linear if one of the following two statements holds for f :

① For every 1-value of f, the no. of 1's in the corresponding input is odd, & for every 0-value of f, the number of 1's in the corresponding input is even.

② For every 1-value of f, the no. of 1's in the corr. input is even & for every 0-value of f, the no. of 1's in corr. input is odd.

- For a given set S of funcs., if every func. in the set is linear, then S is not functionally complete.

Ex. NOT

A	$\bar{A}$	
0	1	} even 1's
1	0	} odd 1's

∴ Linear.

Ex. OR.

A B	A+B	
0 0	0	} even 1's
0 1	1	} odd
1 0	1	} odd
1 1	1	} even ✗

∴ not linear.

Ex. AND

A	B	A.B	
0	0	0	} ev
0	1	0	} odd. ‖
1	0	0	
1	1	1	

∴ Not linear.

Ex. →

A	B	A→B	
0	0	1	} even
0	1	1	} odd ‖
1	0	0	
1	1	1.	

∴ Not linear

Ex. XOR

A	B	A⊕B	
0	0	0	even
0	1	1	Odd
1	0	1	odd.
1	1	0	even

∴ Linear.

Note: With a digital ckt., like 2x1 MUX, OR gate, etc. it can be assumed that 0 & 1 signals are available.

With func., like, $f = \bar{a}b + ac$, $f = ab + c$, etc. 0 & 1 signals aren't available.

COMBINATIONAL CIRCUITS

* Compact Truth Table.

We can write the following expression in another way as well:

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

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

Same

$f = 0.\bar{a}\bar{b} + 1.\bar{a}b + 1a\bar{b} + 0.ab.$

Similarly.

a	b	f
0	0	w
0	1	x
1	0	y
1	1	z

$f = w.\bar{a}\bar{b} + x.\bar{a}b + ya\bar{b} + zab$

Ex.

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

$\Rightarrow$

a	b	c	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0.

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

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

So, the table above is compact representation of the normal truth table.

Ex. Given the following standard T.T. find the Compact T.T.

a	b	c	f
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	1
1	1	1	0

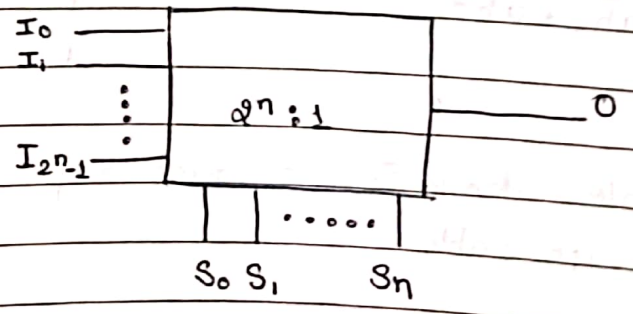
a	b	f
0	0	$\bar{c}$
0	1	1
1	0	$\bar{c}$
1	1	$\bar{c}$

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

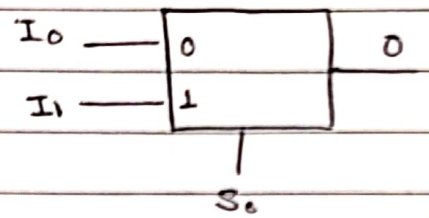
a	f
0	$b+\bar{c}$
1	$\bar{c}$

* Multiplexer

A Mux is a $2^n:1$ combination circuit, with 2^n inputs & 1 output. Depending on the n select lines one of the 2^n inputs is selected as output.

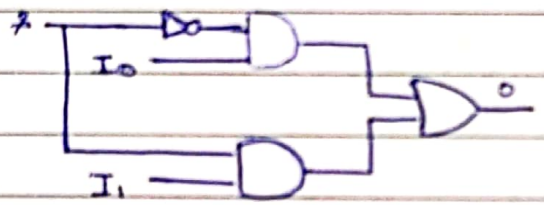


- 2 x 1 MUX

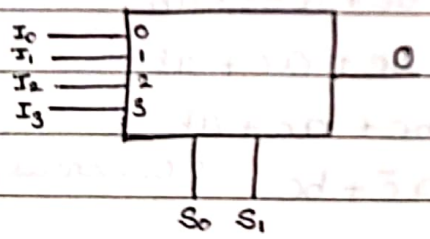


S	O
0	I_0
1	I_1

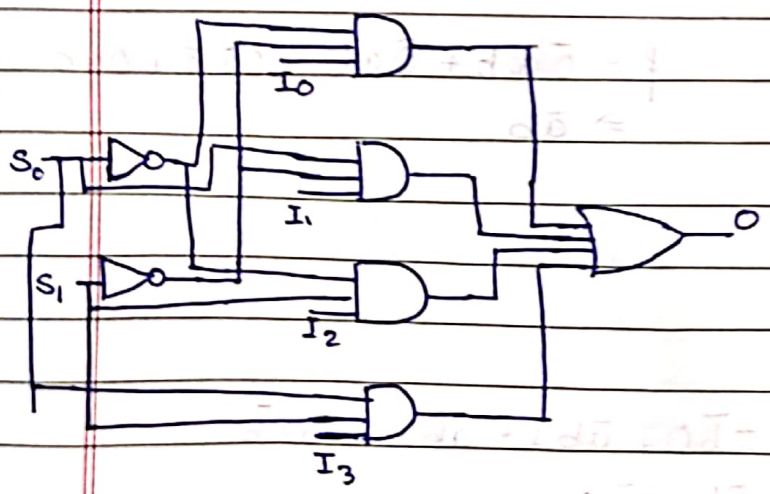
$$O = \bar{S}_0 I_0 + S_0 I_1$$

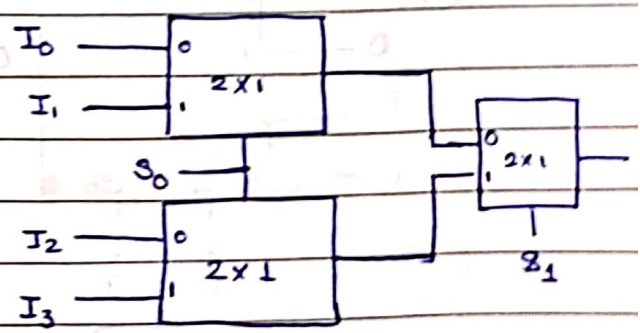


- 4 x 1 MUX



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





$$O = \bar{S}_0 \bar{S}_1 I_0 + S_0 \bar{S}_1 I_1 + \bar{S}_0 S_1 I_2 + S_0 S_1 I_3$$

- $2^K : 1$ Mux expression can be expressed by forming minterms of select lines. (K)

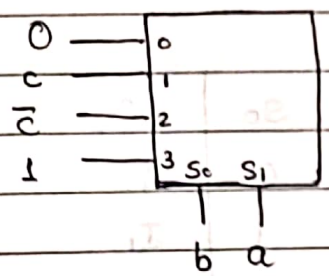
$$\prod_{i=0}^{K} \bar{S}_i \Rightarrow m_0$$

$$S_0 \times \left(\prod_{\substack{i=0 \\ (i \neq 0)}}^{K} \bar{S}_i \right) \Rightarrow m_1$$

$$O = \sum_{i=0}^{2^K - 1} m_i \times I_i$$

$$S_1 \times \left(\prod_{\substack{i=0 \\ (i \neq 1)}}^{K} \bar{S}_i \right) \Rightarrow m_2$$

Ex.



$f = \bar{a}\bar{b}0 + \bar{a}bc + a\bar{b}\bar{c} + ab1$

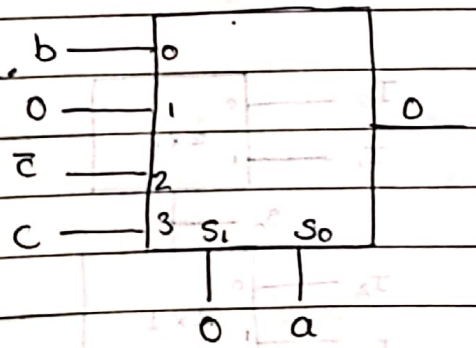
$\Rightarrow \bar{a}bc + a\bar{b}\bar{c} + ab$

$\Rightarrow \bar{a}bc + a\bar{c} + ab$

$\Rightarrow bc + a\bar{c} + ab$

$\Rightarrow a\bar{c} + bc$ {Consensus property}

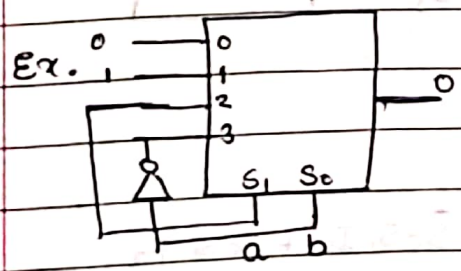
Ex.



$f = \bar{0}\bar{a}b + \bar{0}a0 + 0\bar{a}\bar{c} + 0ac$

$\Rightarrow \bar{a}b$.

Ex.

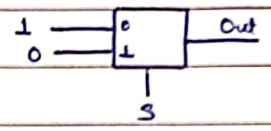


$f = \bar{a}\bar{b}0 + \bar{a}b1 + a\bar{b}a + ab\bar{b}$

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

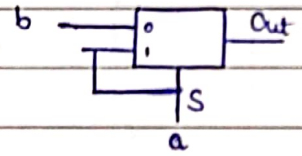
Ex. Using 2x1 MUX implement

① NOT.



S	Out
0	1
1	0.

② OR.



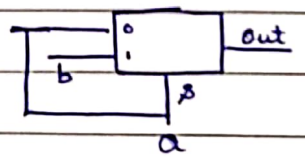
~~S~~	~~Out~~
~~$\bar{a}$~~	

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

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

$\Rightarrow a+b.$

③ AND

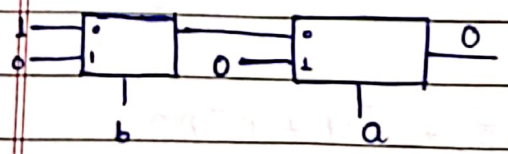


$Out = \bar{a}a + ab$

$\Rightarrow ab.$

④ NOR

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



$Out = \bar{a}I_0 + aI_1$

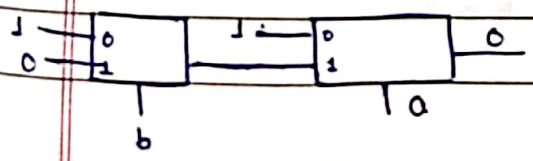
$\Rightarrow \bar{a}[\bar{b}.1 + b.0] + a.0$

$\Rightarrow \bar{a}\bar{b}$

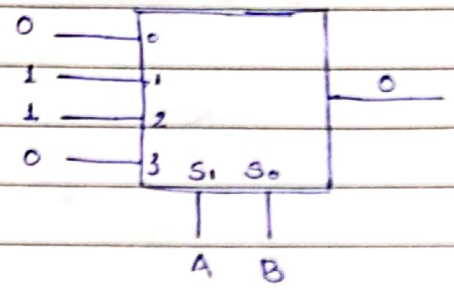
⑤ NAND

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

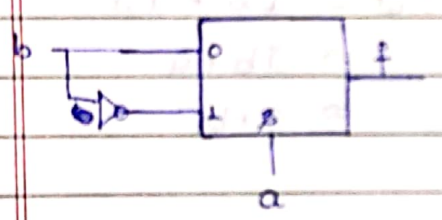
$\Rightarrow \bar{a} + a\bar{b}$



Ex. XOR using 4x1 MUX.

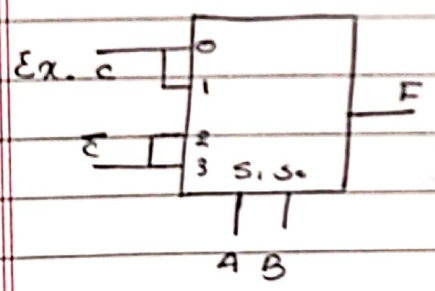


Ex. XOR using 2x1 MUX



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

Ex.



(a) $F = AC$

(b) $F = A \oplus C$

(c) $F = BC$

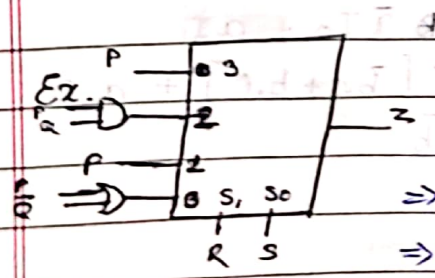
(d) $F = B \oplus C$.

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

$\Rightarrow \bar{A}C + A\bar{C}$

(B)

Ex.



$Z = \bar{R}\bar{S}(P+\bar{Q}) + \bar{R}SP + R\bar{S}PQ + RSP$

$\Rightarrow P\bar{R}\bar{S} + \bar{Q}\bar{R}\bar{S} + P\bar{R}S + PQR\bar{S} + PRS$

$\Rightarrow P\bar{R} + \bar{Q}\bar{R}\bar{S} + PQR\bar{S} + PRS$

$\Rightarrow P\bar{R} + \bar{Q}\bar{R}\bar{S} + PQR\bar{S} + PR$

$\Rightarrow P\bar{R} + PQ + PR + \bar{Q}\bar{R}\bar{S}$

$$Z = \bar{R}\bar{S}(P+\bar{Q}) + \bar{R}S(P) + R\bar{S}PQ + RSP$$

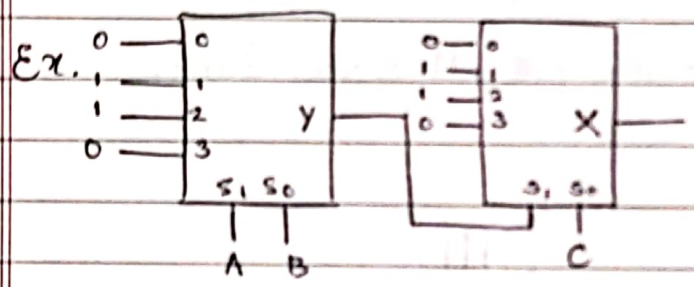
$$\Rightarrow P\bar{R}\bar{S} + \bar{Q}\bar{R}\bar{S} + P\bar{R}S + PQR\bar{S} + PRS$$

$$\Rightarrow P\bar{R} + \bar{Q}\bar{R}\bar{S} + PQR\bar{S} + PRS$$

$$\Rightarrow P\bar{R} + PS + PQR\bar{S} + \bar{Q}\bar{R}\bar{S}$$

$$\Rightarrow P\bar{R} + PS + PQ + \bar{Q}\bar{R}\bar{S}$$

Ex.



$Y = A \oplus B$ $\qquad X = \overline{(A \oplus B)}C + \bar{C}\,\overline{(A \oplus B)}$

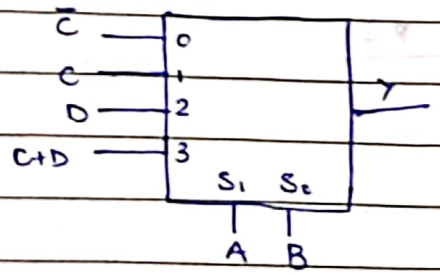
$$\Rightarrow (A \oplus B \oplus C)$$

Ex. $F = AD + BC + \overline{A+B+C}$ using 4:1 MUX, where $(S_1, S_0) = (A, B)$

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

$$\Rightarrow ABD + A\bar{B}D + \bar{A}BC + ABC + \bar{A}\bar{B}\bar{C}$$

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



* MUX is not functionally complete. Any $2^n:1$ MUX. The set {2×1 MUX, 0, 1} is func. complete.

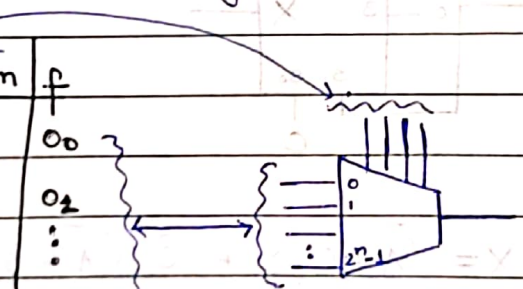
* $2^n \times 1$ MUX is enough to implement every n-variable func.

like 2:1 MUX is enough to implement any 1 var. func.

$2^2:1$ (4×1) MUX is enough for all 2 var func.

Ex.

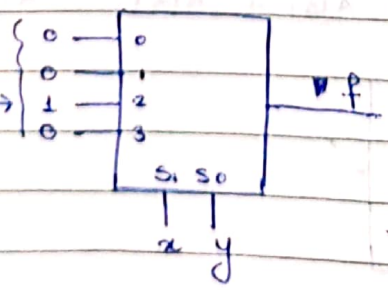
$a_1 a_2 \ldots a_n$	f
	O_0
	O_1
	⋮
	O_{2^n-1}



Put n inputs on n select lines of $2^n:1$ mux.
Put 2^n outs to 2^n input lines of $2^n:1$ mux.

Ex. $f = x\bar{y}$

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



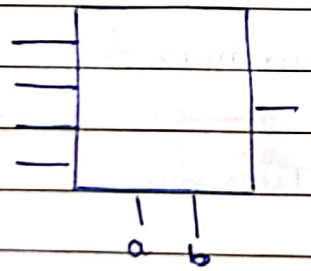
Ex. Is 4:1 MUX + 1 invertor enough to implement all three variable func?

Yes.

Ex. $f = a \oplus b \oplus c$.

a	b	f	
0	0	c	$0 \oplus 0 \oplus c = 0 \oplus c = c$
0	1	$\bar{c}$	$0 \oplus 1 \oplus c = 1 \oplus c = \bar{c}$
1	0	$\bar{c}$	$1 \oplus 0 \oplus c = 1 \oplus c = \bar{c}$
1	1	c	$1 \oplus 1 \oplus c = 0 \oplus c = c$.

So,



inputs can be $0/1/c/\bar{c}$.

So for 3 var funcs:

1. single 8:1 MUX
2. single 4:1 MUX & one inverter
3. Three 2:1 MUX & one inverter {3 2:1 MUX ⇒ 1 4:1 MUX}
4. single 2:1 MUX & additional ckt (OR, AND, etc)

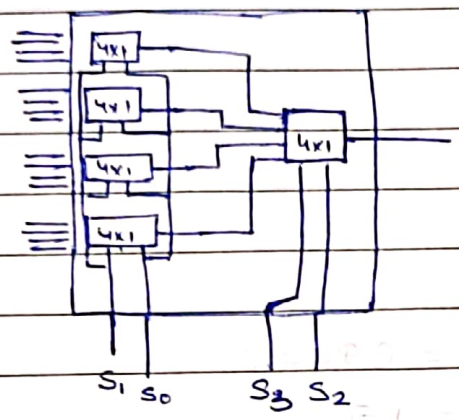
is enough.

For n-var funcs:

1. $2^n : 1$ MUX
2. $2^{n-1} : 1$ & one invertor

is enough.

* Creating 16:1 MUX using 4:1 MUX.



like $S_3 S_2 S_1 S_0$

0 0 1 0.

$S_1 S_0 \Rightarrow 10$ get I_2 from all four mux of level one,

$S_3 S_2 \Rightarrow 00$ get I_0 from level two mux.

Ex. No. of 4x1 mux for 64:1 mux.

$\frac{64 \text{ inp}}{4} = 16$ MUX

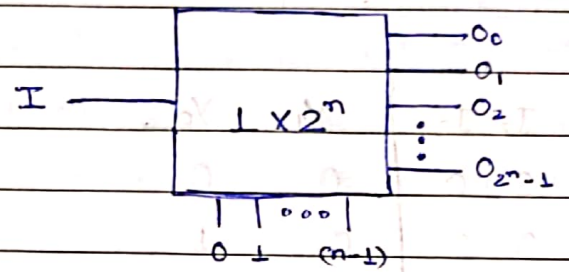
$\frac{16 \text{ mux out}}{4} = 4$ MUX.

$\frac{4 \text{ mux out}}{4} = 1$ MUX

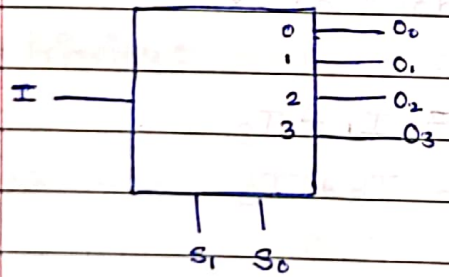
2:1 MUX.

* DEMUX

DEMUX is a $1:2^n$ circuit, which based on n select inputs directs the input to one of the 2^n output lines.



Ex. 1 : 4 Demux.



$O_0 = \bar{S}_1 \bar{S}_0 I$

$O_1 = \bar{S}_1 S_0 I$

$O_2 = S_1 \bar{S}_0 I$

$O_3 = S_1 S_0 I$

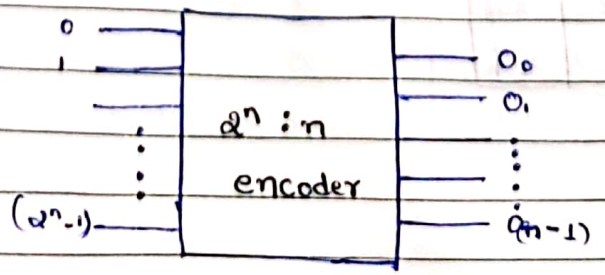
S_1	S_0	O_3	O_2	O_1	O_0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

* Encoder.

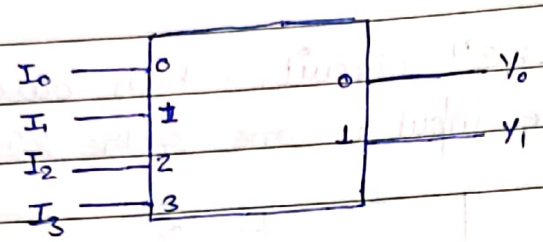
Encoder is a circuit which gives n elements encoding using $\log n$ bits or 2^n elements using n bits.

$2^n : n$ circuit.

(Think of indexing /numbering)



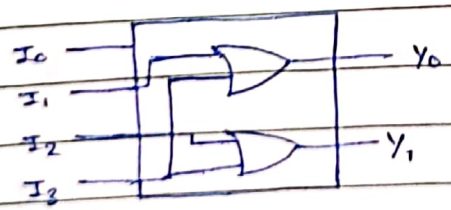
- 4:2 Encoder



I_0	I_1	I_2	I_3	Y_1	Y_0
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 input comb. are don't cares.

$$Y_0 = I_1\bar{I}_0\bar{I}_2\bar{I}_3 + \bar{I}_0\bar{I}_1\bar{I}_2 I_3 = I_1 + I_3$$
$$Y_1 = \bar{I}_0\bar{I}_1 I_2\bar{I}_3 + \bar{I}_0\bar{I}_1\bar{I}_2 I_3 = I_2 + I_3$$



Y_0 ; I_1I_0 \ I_3I_2	00	01	11	10
00	X	0	1	X
01	0	X	X	X
11	1	X	X	X
10	X	X	X	X

$Y_0 = I_1 + I_3$

• Priority Encoder

Even when multiple inputs are on, only one is decoded at output. The decoded input is called priority input.

Say an encoder is used to select which of the four: Power supply, Keyboard, Mouse, Printer gets CPU time.
Even if both Power & Printer request CPU time, only ~~E~~ Power gets it.

• 4:2 Encoder

Priority: $I_2 > I_0 > I_1 > I_3$

I_3	I_2	I_1	I_0	Y_1	Y_0	Z → is any input on.
0	0	0	0	x	x	0
0	0	0	1	0	0	1
0	0	1	0	0	1	1
0	0	1	1	0	0	1
0	1	0	0	1	0	1
0	1	0	1	1	0	1
0	1	1	0	1	0	1
0	1	1	1	1	0	1
1	0	0	0	1	1	1
1	0	0	1	0	0	1
1	0	1	0	0	1	1
1	0	1	1	0	0	1
1	1	0	0	1	0	1
1	1	0	1	1	0	1
1	1	1	0	1	0	1
1	1	1	1	1	0	1

Y_0:

I_1I_0 \ I_3I_2	00	01	11	10
00	X			1
01				
11				
10	1			1

Y_1:

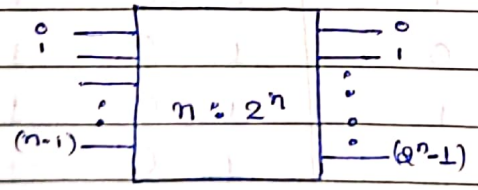
I_1I_0 \ I_3I_2	00	01	11	10
00	X	1	1	1
01		1	1	
11		1	1	
10		1	1	

$$Y_0 = \overline{I_0}\,\overline{I_2}$$

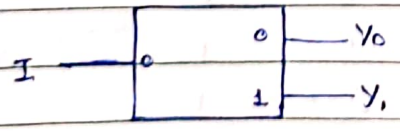
$$Y_1 = I_2 + \overline{I_1}\,\overline{I_0}$$

* Decoder

Given a n-bit binary code as input, activates one of 2^n outputs

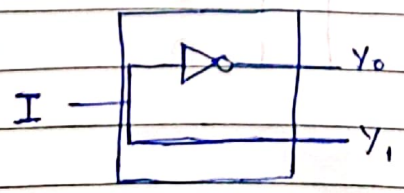


Ex. 1 : 2^1 Decoder.

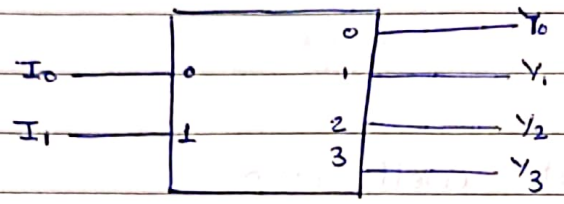


I	Y_1	Y_0
0	0	1
1	1	0.

$$Y_0 = \overline{I} \qquad Y_1 = I$$



Ex. 2 : 2^2 Decoder.



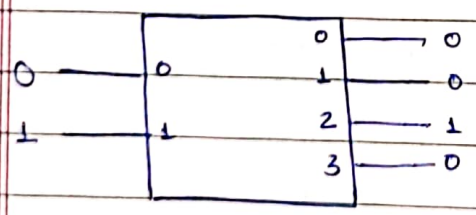
I_1	I_0	Y_3 Y_2 Y_1 Y_0
0	0	0 0 0 1
0	1	0 0 1 0
1	0	0 1 0 0
1	1	1 0 0 0

$Y_0 = \bar{I}_1 \bar{I}_0 \quad Y_1 = \bar{I}_1 I_0 \quad Y_2 = I_1 \bar{I}_0 \quad Y_3 = I_1 I_0$

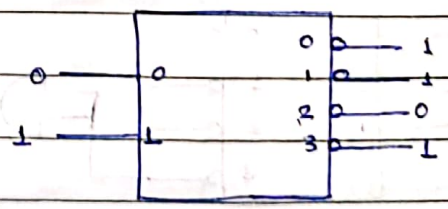
- Additionally there can be an active high or active low Enable input.

- Active Low Decoder (NAND Decoder)

In Active high decoder one output is 1 & rest are 0.
In an Active low decoder one output is 0 & rest are 1.



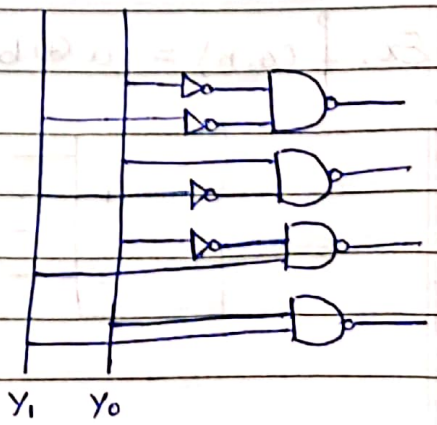
Act. High



Act. Low

Ex. 2×2^2 Act. Low Decoder:

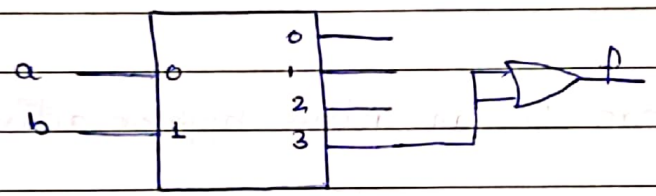
I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1.



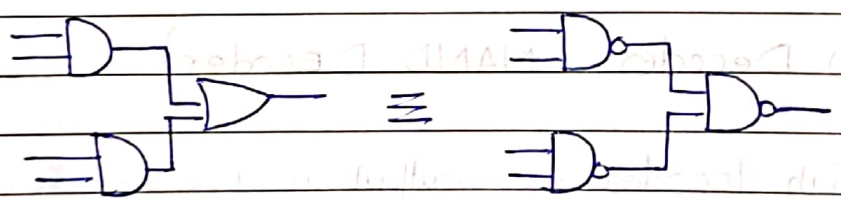
• Functions Using Decoder

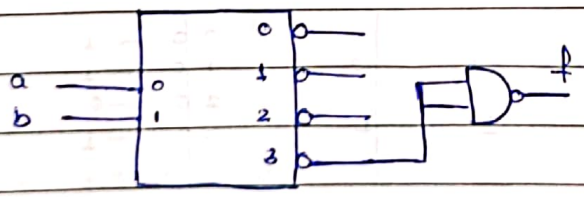
Ex. $f(a,b) = ab$.

Since there are 2 inputs, we'll have to use at least $2:2^2$ Decoder.

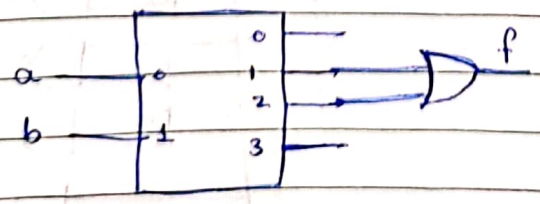


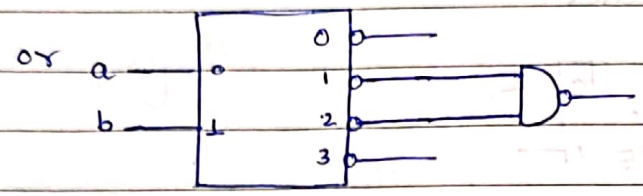
☆



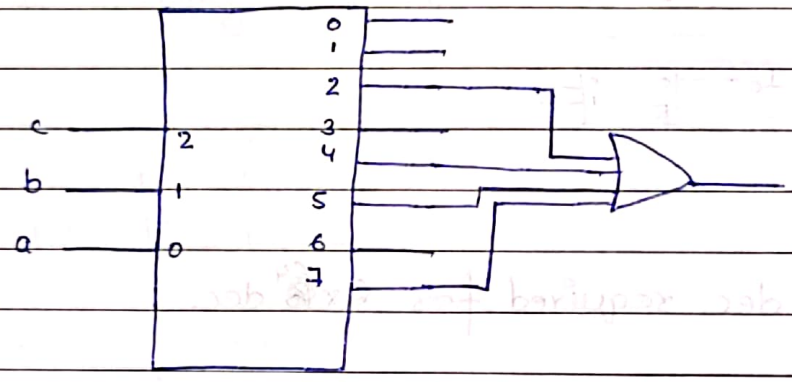


Ex. $f(a,b) = a \oplus b = a\bar{b} + \bar{a}b$.



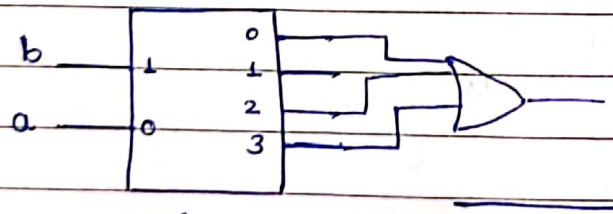


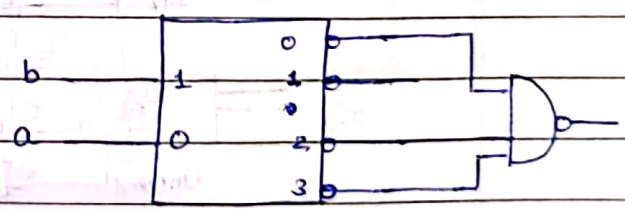
Ex. $f(a,b,c) = \Sigma(2,4,5,7)$



Ex. $f(a,b) = a \rightarrow b$.

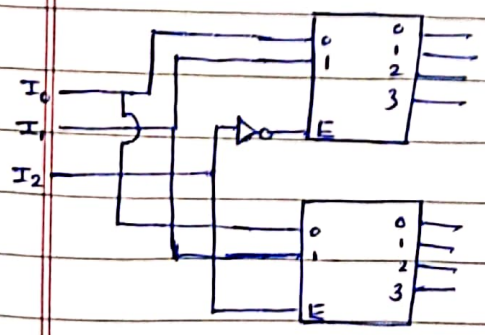
$\Rightarrow \bar{a}\bar{b} + \bar{a}b + ab$



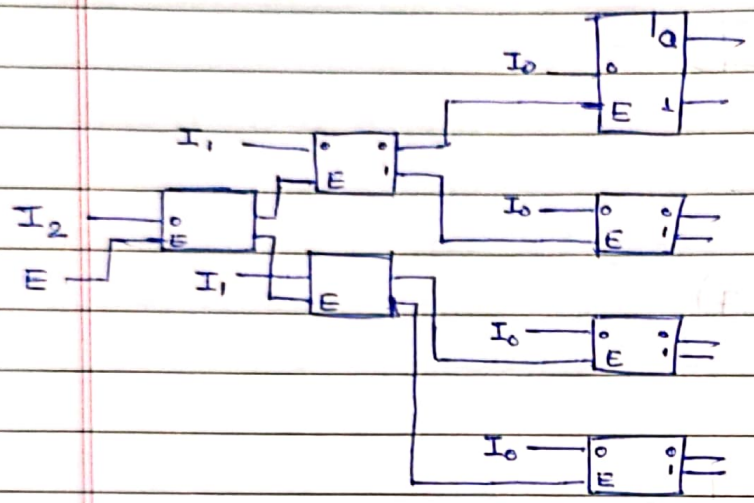


$\Sigma(0,2,3) \equiv \overline{\Pi(0,2,3)} = \Pi(1)$

Ex. 3:8 decoder using 2:4 decoder.

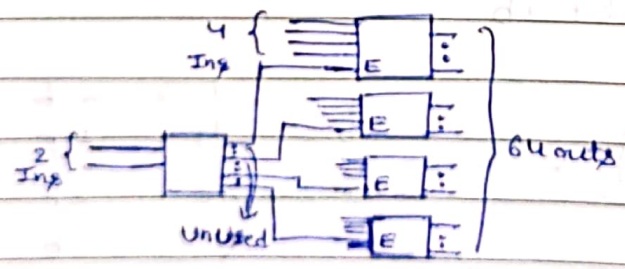


Ex. 3x8 using 1x2 Decoder.



Ex. No. of 4x16 dec. required for 6x64 dec

64 outs will req. [4] 16 out decoders
4 decoders req. [1] 4x16 dec. to select enable input.



Ex. # of 2x4 Decs for 7x128 Dec.

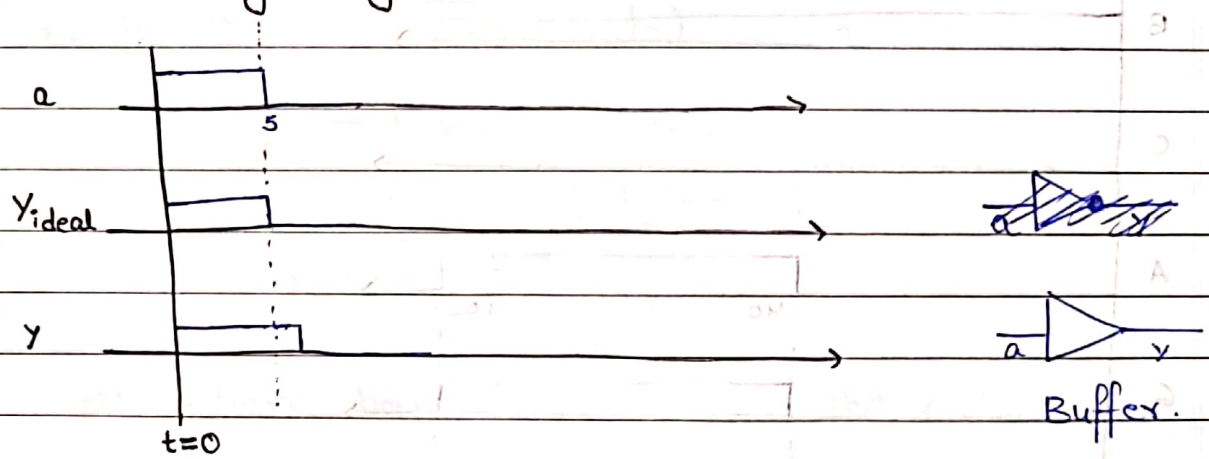
$\frac{128}{4} = 32$ $\frac{32}{4} = 8$ $\frac{8}{4} = 2$ 1

$(32+8+2+1) = 43$

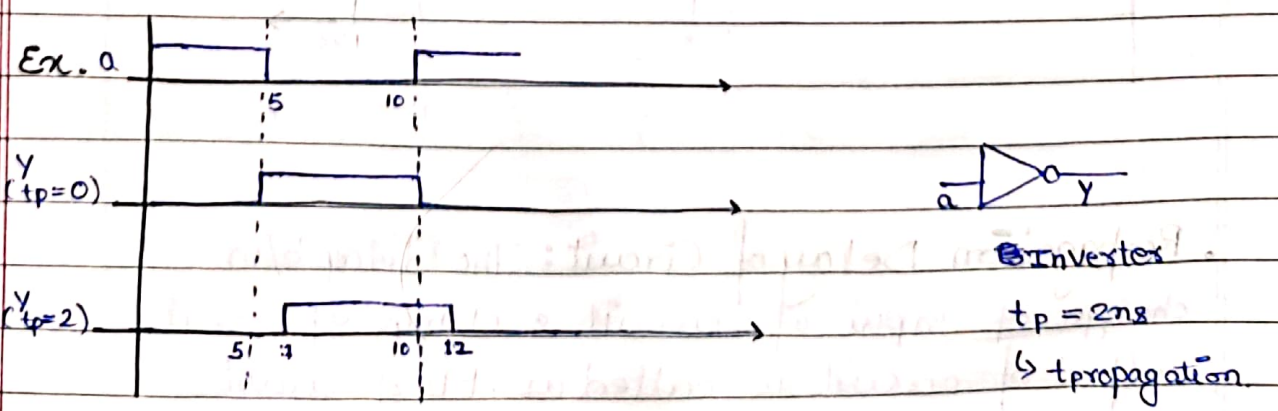
* Propagation Delay

Every logic gate is made of physical connections, each of which takes some time to process the input to output, which is known as "Delay".

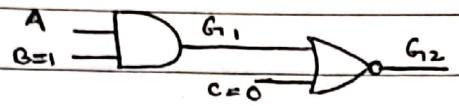
Ex. (Timing Diagram)



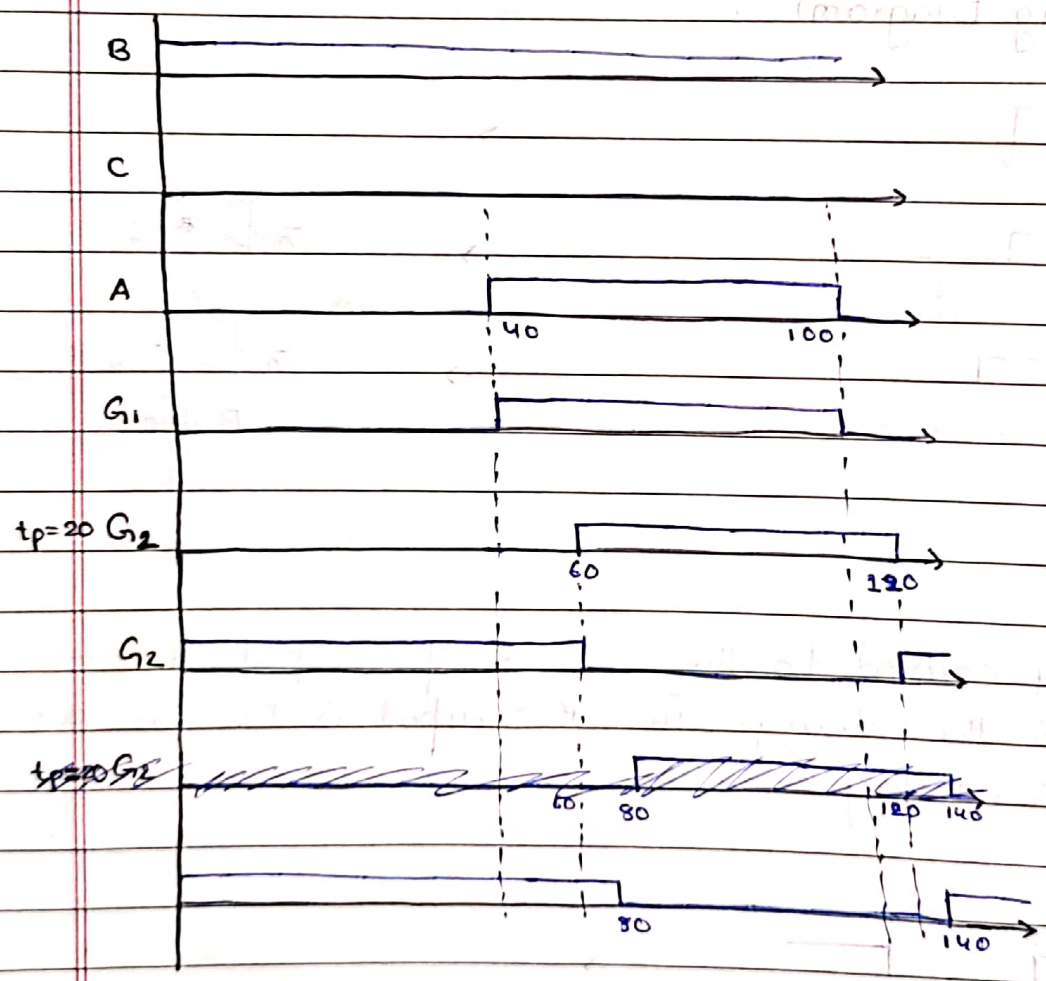
This delay caused to the change of output with respect to the change in the input is known as "Propogation Delay".



So, we can think of Prop. delay as shifting of ideal waveform by t_p.

Ex. 

tp of both gates is 20ns.
Draw the timing diag. when B & C resp. remain 1 & 0 constantly, & A starts with 0, then changes to 1 at t = 40ns then back to 0 at t = 100ns.



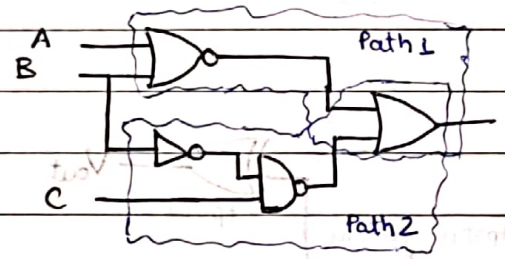
- Propagation Delay of Circuit: The delay b/w change of input of circuit & change of final output of circuit is called as PD of circuit.

In above example, G2 is final output.
∴ Tp(circuit) = 40ns.

i.e. the maximum propagation delay possible b/w input & final output of the circuit is known as "Propagation Delay of circuit".

The path that causes this delay is called as the "Critical Path".

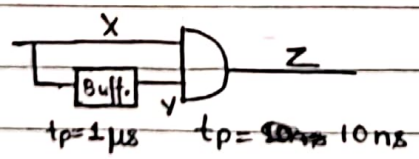
Ex. $t_p = 20ns$. (for every gate)

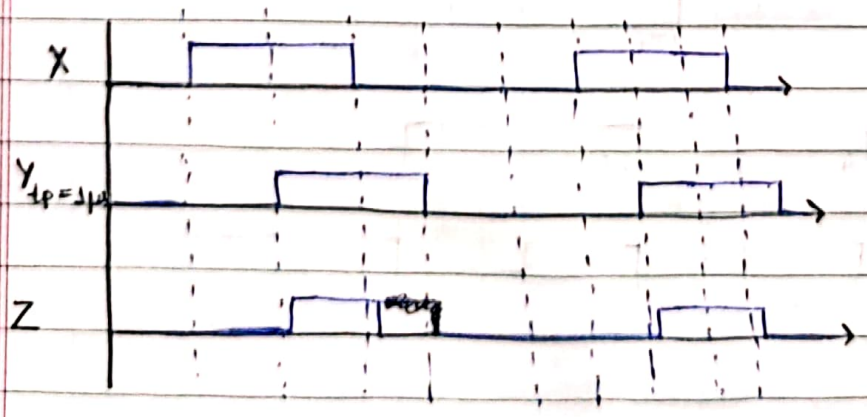


Path 1 here has $t_{p(ckt)} = 2t_p$, Path 2 has $t_{p(ckt)} = 3t_p$.

∴ Path 2 is the critical path here.

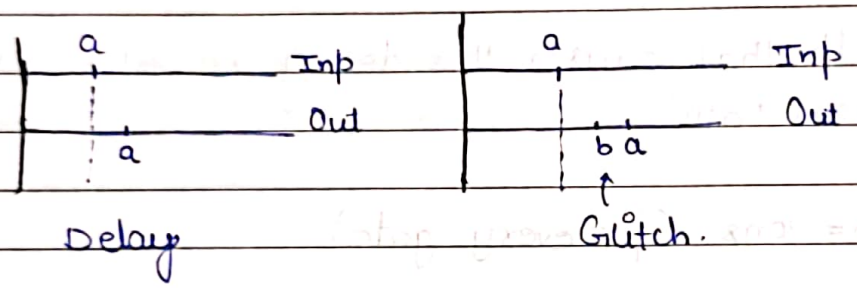
Ex.



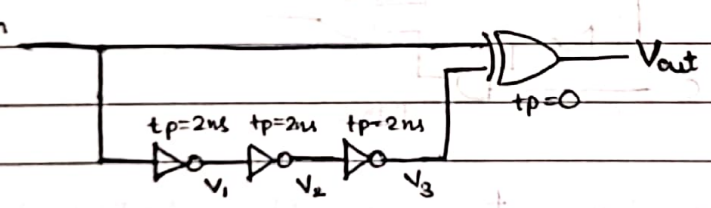


- Glitches: With change in the input if there is a period of incorrect output before getting correct output it is called a glitch.

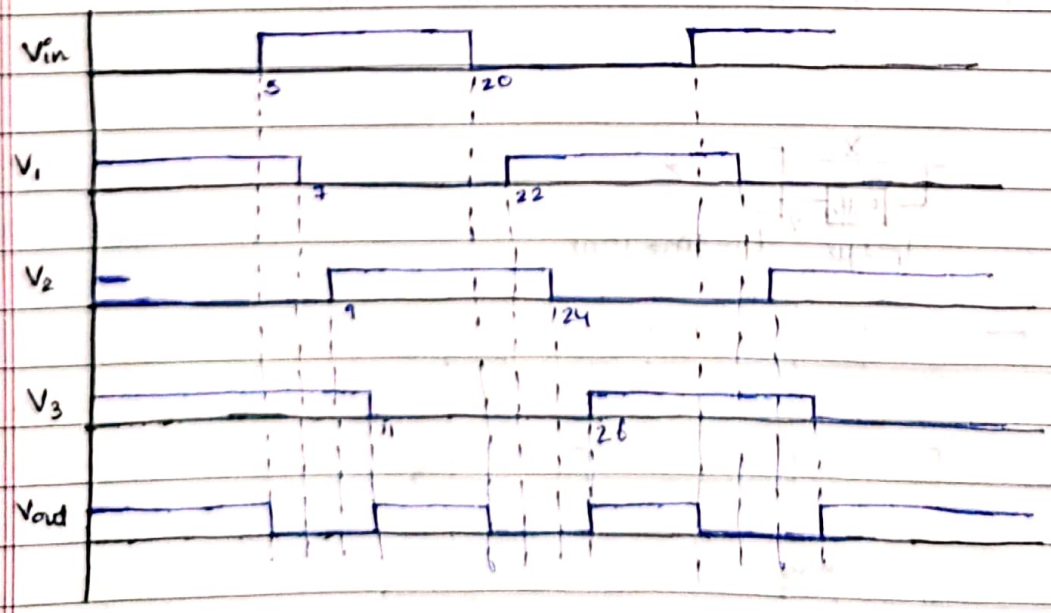
Ex.



Ex.

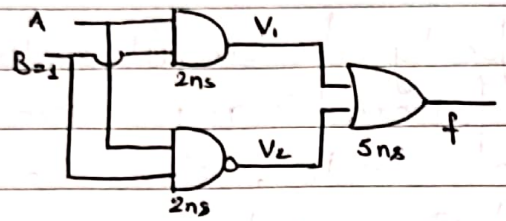


$V_{out} = V_{in} \oplus \overline{V_{in}} = 1$ (This should be ideal result).

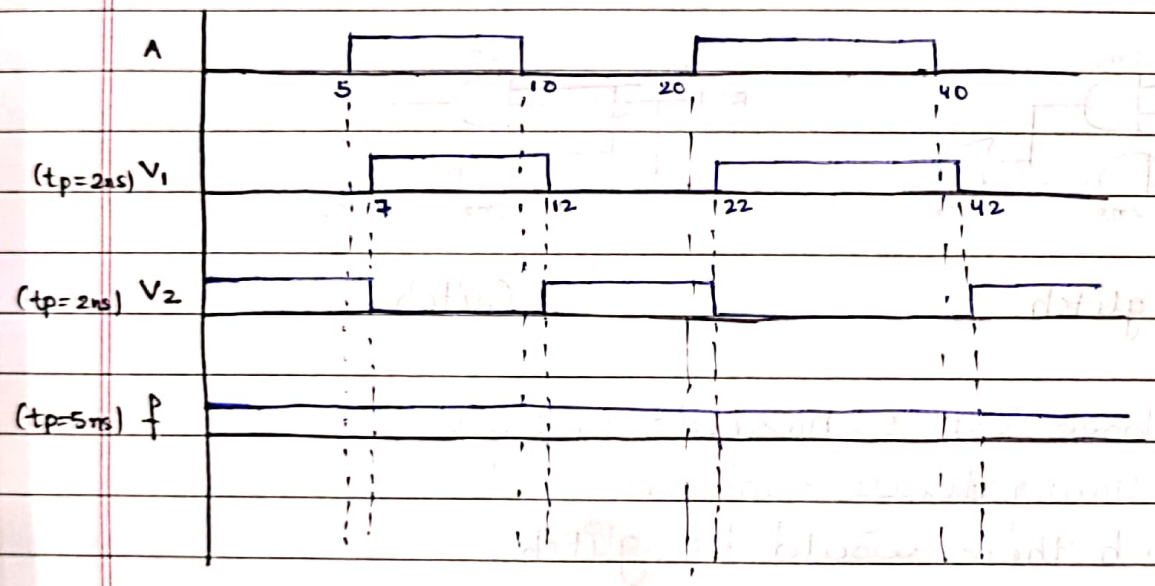


So, we expected Vout to be 1 constantly, but we get 0 in-between before going to 1.

Ex.

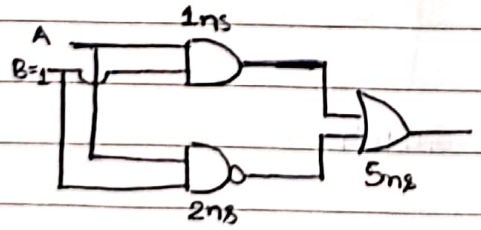


$V_1 = A \quad V_2 = \bar{A} \quad f = A + \bar{A} = 1$

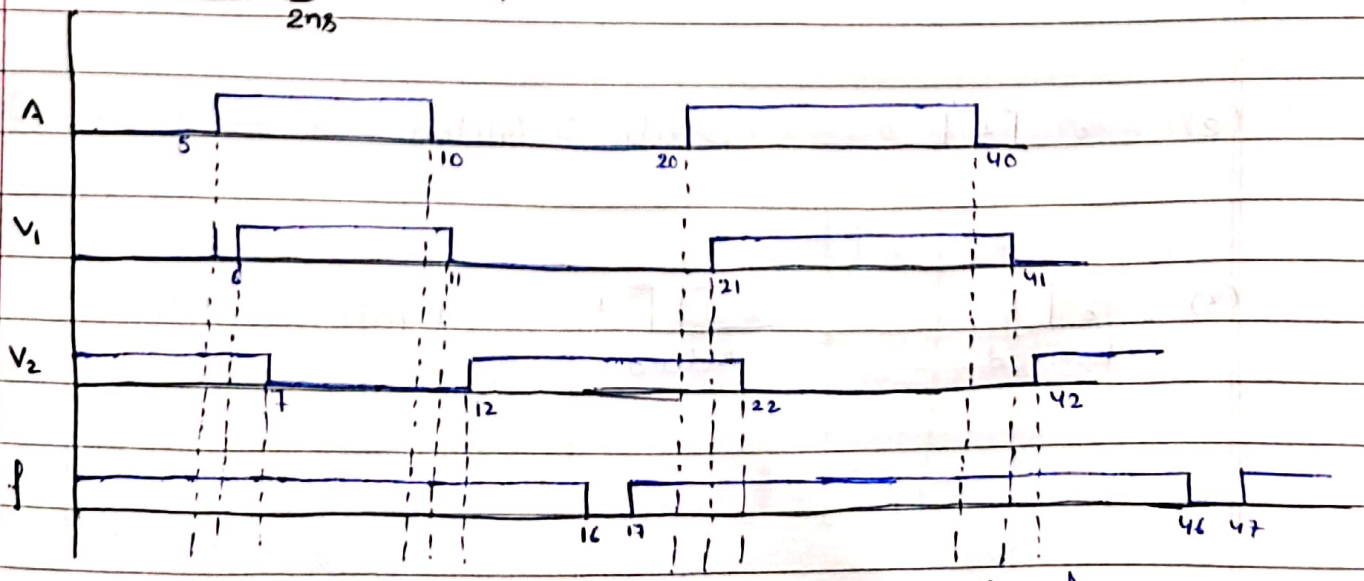


So, this circuit has no glitch.

Ex.



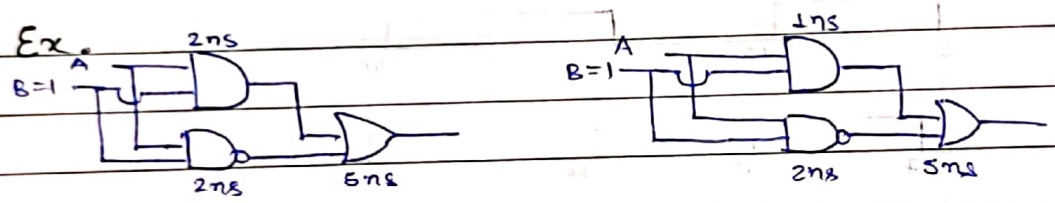
$V_1 = A \quad V_2 = \bar{A} \quad f = A + \bar{A} = 1.$



So, there is a glitch. We get 0 for few moments.

- **Hazard:** Glitch is property of circuit & timing delays. A circuit is hazardous if there is any chance at all that it can have glitches, i.e for some arbitrary combination of timing delay there is glitch.

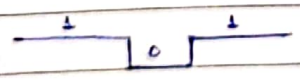
Ex.



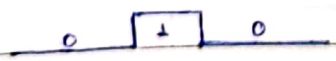
No glitch Glitch

The above ckt is hazardous, since there is one timing delay combination possible for which there would be glitch.
The glitch is caused by uneven propagation delay in parallel circuits.

- **Types of Glitches**

(1) 

Static 1 Glitch

(2) 

Static 0 Glitch.

(3)

Dynamic Glitch.

~~Ex. st~~

Static glitch happens when the result has to be constant.

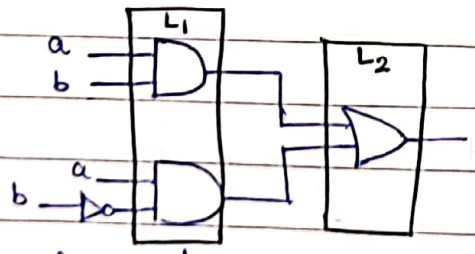
Dynamic glitch happens when the result has to transition.

* Static Hazards

GATE syllabus has only two level circuits with one input varying.

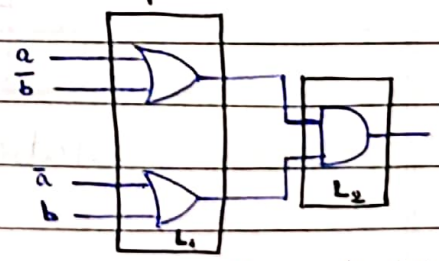
(NOT gate by itself ~~dont~~ doesn't create a level).

Ex. $f = ab + a\bar{b}$



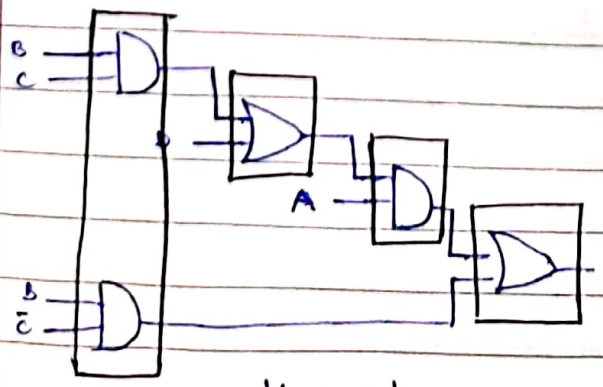
2-Levels. AND-OR ckt.

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

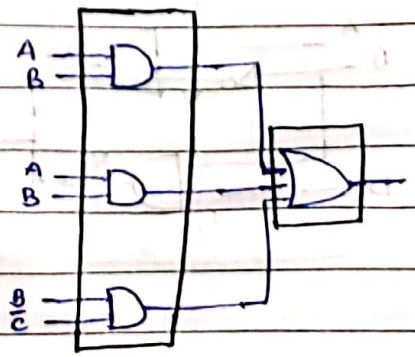


2-level OR-AND ckt.

Ex. $f = A(BC+D) + B\bar{C} \equiv ABC + AD + B\bar{C} \equiv AB + AD + B\bar{C}$

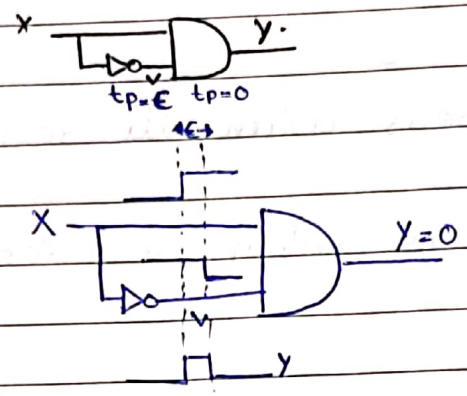


4-level.



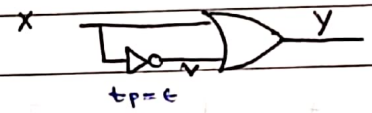
2-level.

Ex.

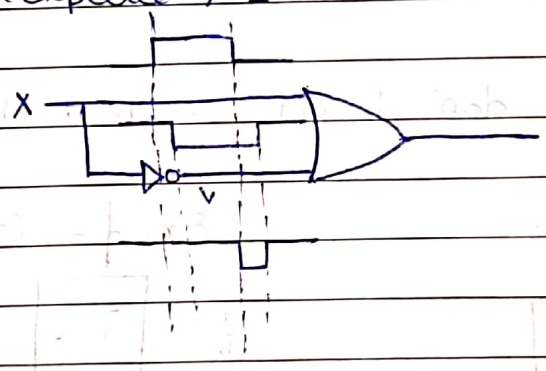


∴ Static 0 hazard.

Ex.



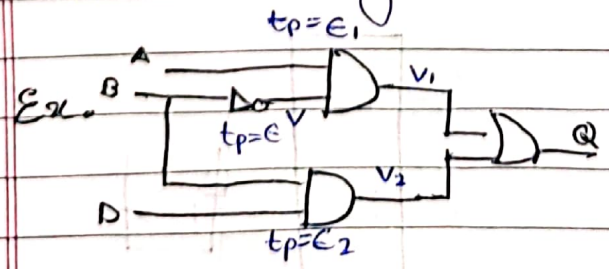
(expected Y=1 (constant))



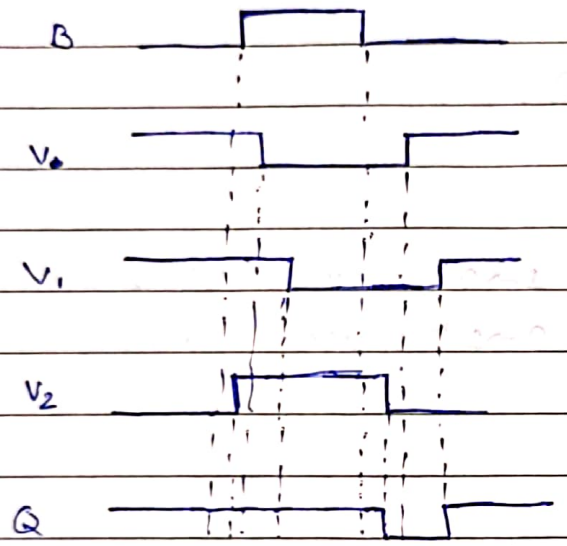
∴ Static 1 hazard.

- K-map can be used to resolve static hazards from 2-level circuits with single variable change.

Ex.



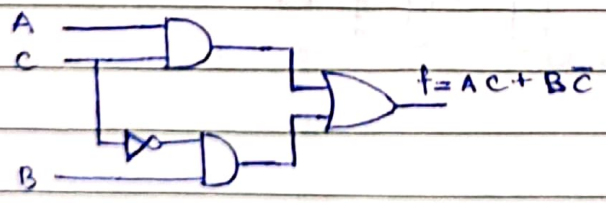
Say A & D = 1, then irrespective of B, Q = 1.



∴ Static 1 hazard

• Note: Standard template for Static-1 hazard in 2-level circuits is AND-OR, for static-0 hazard is OR-AND.

Ex,



This is template for static-1 hazard.
This gives glitch for $\{A = B = 1 \ \& \ C : 0 \to 1\}$. We can find this configuration using K-map.

C \ AB	00	01	11	10
0	0	1	1	2
1	4	5	7	6
		1	1	1

$1 \to 3$: $\bar{A}B\bar{C} \to AB\bar{C}$ No glitch. $B\bar{C}$ will handle it.

$7 \to 6$: $ABC \to A\bar{B}C$ No glitch. AC will handle it.

$3 \to 7$: $AB\bar{C} \to ABC$ Glitch. Neither expression handles it.

To remove the hazard, we put another PI in the expression : AB.

$$f = AC + \bar{B}\bar{C} + AB.$$

Note: In SOP expression $1 \to 0$ produces glitch.
In POS expression $0 \to 1$ produces glitch.

- SOP Expression

Static 1 hazard for single-variable change is possible. No Static 0.

Hazard is caused by some pair of adjacent 1-cells not covered by any PI..

Ex. $f = ab + bc$.

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

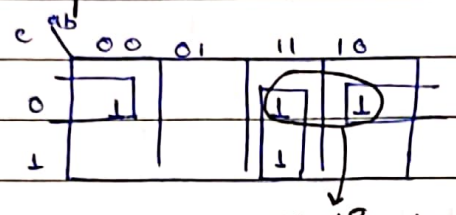
∴ No Static-1 hazard.

Ex. $f = ab + \bar{b}\bar{a}$

a \ b	0	1
0	1	
1		1

No hazard
(∵ No adjacent cells)

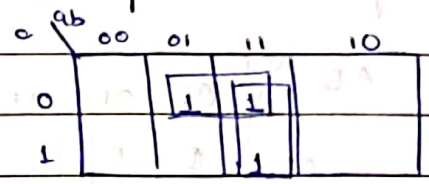
Ex. $f = ab + \bar{b}\bar{c}$



static hazard.

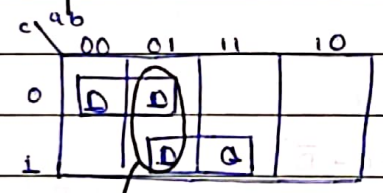
$f = ab + \bar{b}\bar{c} + a\bar{c}$

Ex. $f = ab + b\bar{c}$



No hazard.

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



static 0 Hazard.

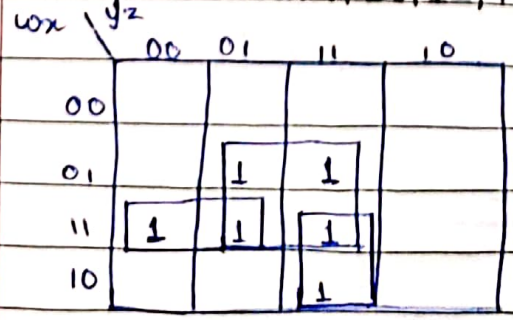
$f = (a+c)(\bar{b}+\bar{c})(a+\bar{b})$

- POS Expression

Only static 0 hazard exist because of single variable change. No static hazard.

Hazard caused by consecutive 0-cells not being covered by any Prime Implicate.

Ex. $f = \Sigma(5, 7, 11, 12, 13, 15)$



$f = wx\bar{y} + xz + wyz$

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

AB \ CD	00	01	11	10
00	0	0		
01	0	0		0
11		0	0	0
10		0	0	

Expr. has static 0 hazard.

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

- If multiple inputs are changing at same time nearly all circuits have hazard, & can't be removed using K-map.

* Binary Adder

• Half Adder

Adds only two bits.

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

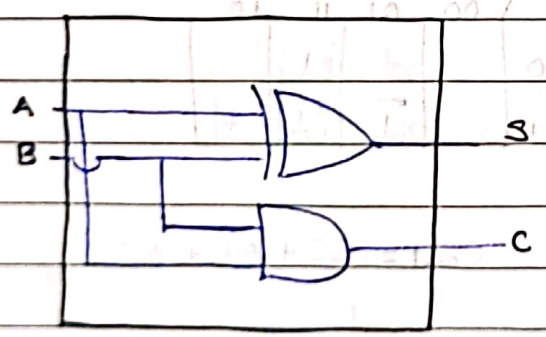
A \ B	0	1
0		$\boxed{1}$
1	$\boxed{1}$	

A \ B	0	1
0		
1		$\boxed{1}$

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

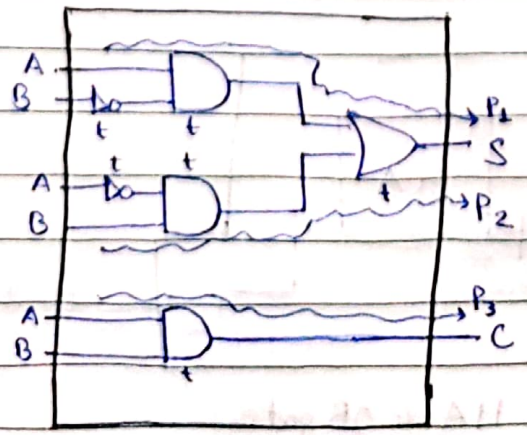
$\Rightarrow A \oplus B$

$C = A.B$



Half Adder.

• Ex. If 't' is tp of AND, OR, NOT, XOR gate, what is the delay of 2-level (AND-OR) Half adder.



Ckt delay =

$\max(\{delay_{P_1}, d_{P_2}, d_{P_3}\})$

$\Rightarrow \max(\{3t, 3t, t\})$

$\Rightarrow 3t.$

Full Adder

A	B	Cin	S	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

Cin \ AB	00	01	11	10
0		1		1
1	1		1	

Cin \ AB	00	01	11	10
0			1	
1		1	1	1

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

$\Rightarrow A \oplus B \oplus C.$

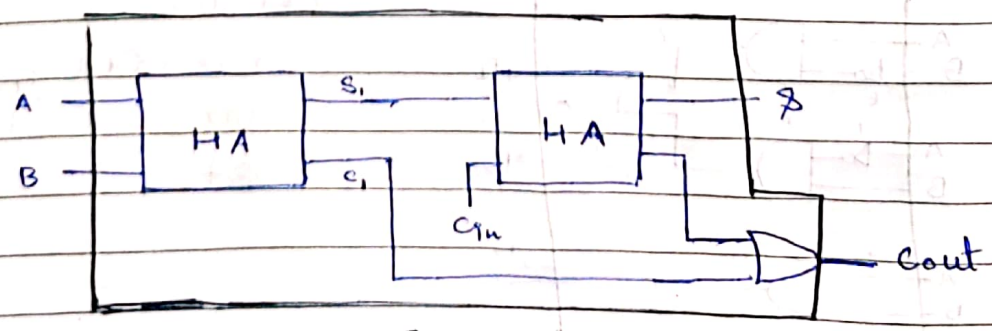
$Cout = BC + AB + AC.$

$\Rightarrow (A+B)C + AB$

$\Rightarrow (A \oplus B)C + AB$

(Since A & B are never simultaneously 1).

$\Rightarrow (A \oplus B)C \oplus AB$



FA using HA & OR gate.

- Note:

(1) If α, β are never simultaneously one, then

$$\alpha \oplus \beta = \alpha + \beta$$

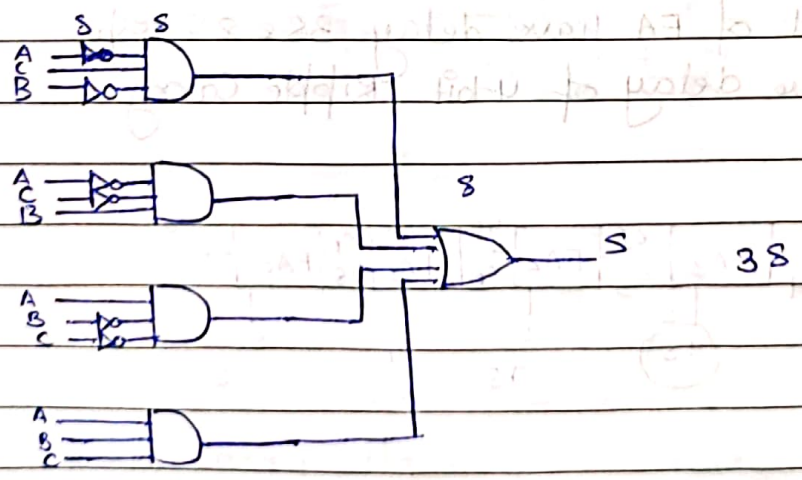
(2) If α, β, γ: no two are pairwise 1, then

$$\alpha \oplus \beta \oplus \gamma = \alpha + \beta + \gamma$$

(i.e. all are 0 or only one of the three is 1).

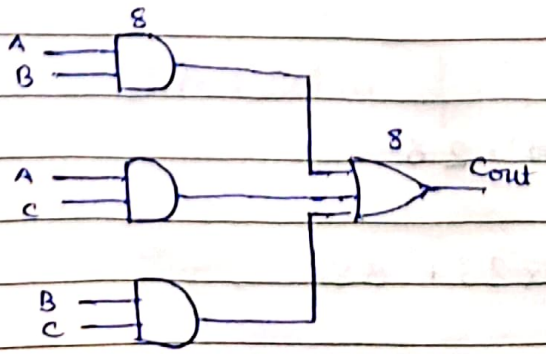
(3) Any func. can be represented as XOR of minterms, since only one of the minterms can be 1 at once.

Ex. Assume tp of all basic gates = 8. Find delay of Sum & Cout of full adder using (AND, OR, NOT).



3S

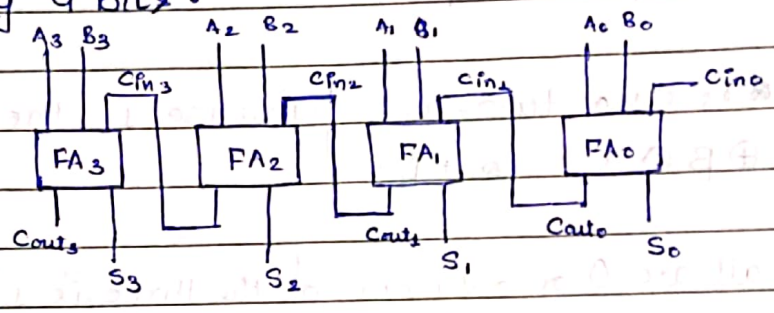
∴ Ckt delay of Full Adder ⇒ 3S



2S

Ripple Carry Adder.

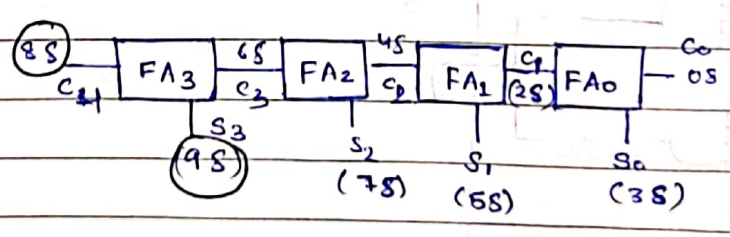
Say 4 bits :



Each subsequent FA from right to left depends on the carry from previous FA. before processing result.

Only when the result of FA_0 is ready can FA_1 start processing its result.

Ex. If S & Cout of FA have delay 3S & 2S resp., what is the delay of 4-bit Ripple Carry adder.



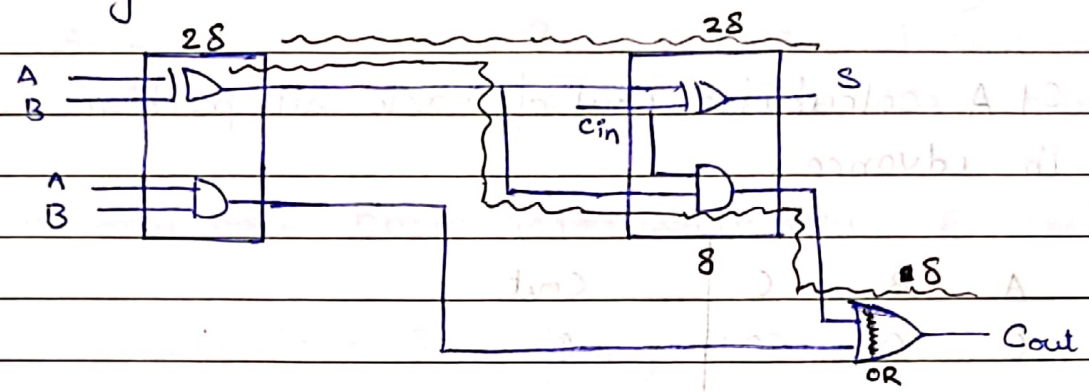
∴ CKt delay = 9S.

For $C_{n_{out}}$, delay = $(n-1) \times 2S$

For S_n, delay = $n \times 2S + 3S$

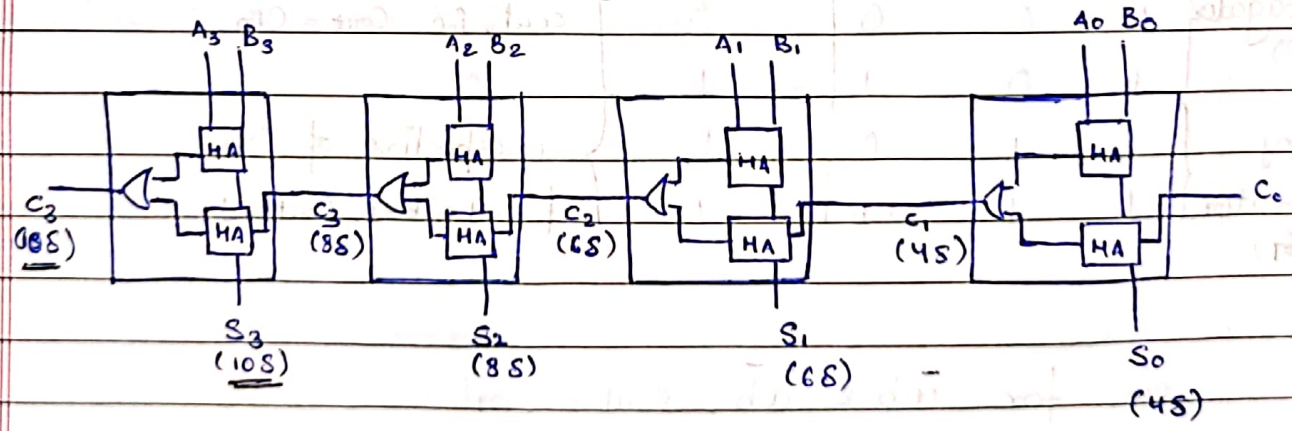
Ex. Let AND, ~~XOR~~ & OR gate have delay of δ, XOR has 2δ, what is the circuit delay of 4bit ripple carry adder.

FA using 2HA & XOR



~~Cout~~ Ckt delay, Cout = $2\delta + \delta + \delta = 4\delta$

Ckt delay, S = $2\delta + 2\delta = 4\delta$.



Since the first HA of each FA is independent of previous carry, its output is ready after 2δ.

∴ Circuit delay of Ripple Carry adder = 10δ

Carry Lookahead Adder

RCA is slow since it depends of $(n-1)^{th}$ bit's carry out to calculate n^{th} sum.

CLA calculates Cout of each bit position in advance.

	A	B	C	Cout	
	0	0	0	0	
	0	0	1	0	
Carry Propagator (P)	0	1	0	0	if A,B = (1,0) or (0,1) then whatever is Cin, is also cout, i.e. Cout = Cin.
	0	1	1	1	
	1	0	0	0	
	1	0	1	1	
Carry Generator (G)	1	1	0	1	irrespective of Cin, if A=1 & B=1, Cout = 1.
	1	1	1	1	

So, for $\bar{a}b$ & $a\bar{b}$, cout = Cin
for ab, cout = 1.
for $\bar{a}\bar{b}$, Cout = 0

Cout/≠/Tx.

A	B	Cout
0	0	0
0	1	Cin
1	0	Cin.
1	1	1

$$Cout = (\bar{a}b + a\bar{b})\,Cin + ab.1 + \bar{a}\bar{b}.0$$

$$\Rightarrow Cout = (a \oplus b)\,Cin + ab$$

$$\Rightarrow P\,Cin + G$$

In 4-bit ripple carry adder

$G_0 = A_0 B_0$ $\quad P_0 = A_0 \oplus B_0$ $\quad C_1 = P_0 C_0 + G_0$

$G_1 = A_1 B_1$ $\quad P_1 = A_1 \oplus B_1$ $\quad C_2 = P_1 C_1 + G_1$

$G_2 = A_2 B_2$ $\quad P_2 = A_2 \oplus B_2$ $\quad C_3 = P_2 C_2 + G_2$

$G_3 = A_3 B_3$ $\quad P_3 = A_3 \oplus B_3$ $\quad C_4 = P_3 C_3 + G_3$

If we only have basic gates with delay (δ), then:

delay for any $G_i = \delta$ {only AND}
delay for any $P_i = 3\delta$ {NOT → AND → OR}.

The idea now is to be able to calculate each carry out with just initial data, i.e.
$(A_3 A_2 A_1 A_0,\ B_3 B_2 B_1 B_0,\ C_0)$.

$C_1 = G_0 + P_0 C_0$ {only init. data}

$C_2 = G_1 + P_1 C_1$

$\Rightarrow G_1 + P_1 (G_0 + P_0 C_0)$

$\Rightarrow G_1 + P_1 G_0 + P_1 P_0 C_0$

$C_3 = G_2 + P_2 C_2$

$\Rightarrow G_2 + P_2 (G_1 + P_1 G_0 + P_1 P_0 C_0)$

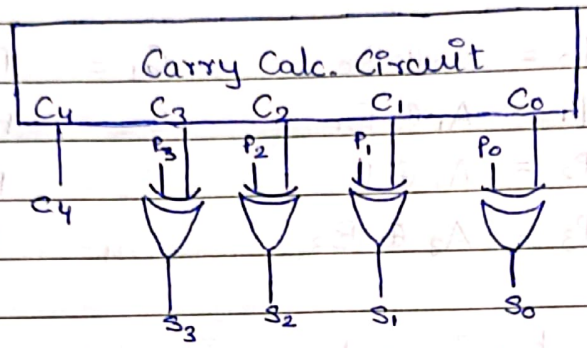
$\Rightarrow G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$

$C_4 = G_3 + P_3 C_3$

$\Rightarrow G_3 + P_3 (G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0)$

$\Rightarrow G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$

So, now the adder is like

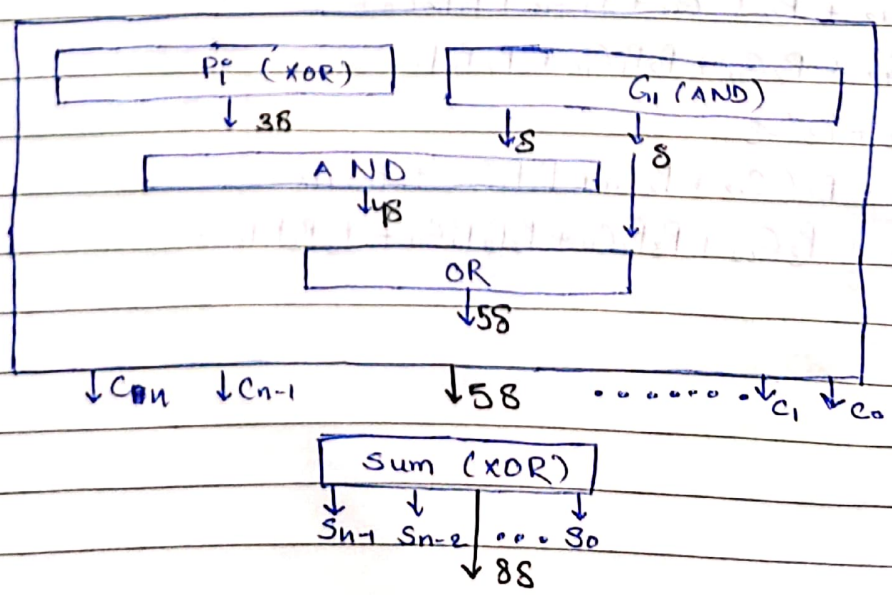


- Each C_i has $(i+1)$ product terms. So fan-in of OR gate for C_i is $(i+1)$.

- Fan-in$_{max}$ for AND gate of C_i is $(i+1)$.

- For C_i, one OR gate is required & i AND gates, the $(i+1)^{th}$ input to OR gate is G_{i-1}.

Ex. Assume that the fan-in of all 3 gates: AND, OR, NOT gates is unlimited, each with δ delay, then what is the delay of n-bit CLA.



∴ 8 δ for Sum.

Ex. Same question as above, but compliments are available

Each XOR will be of 2δ, rather than 3δ.

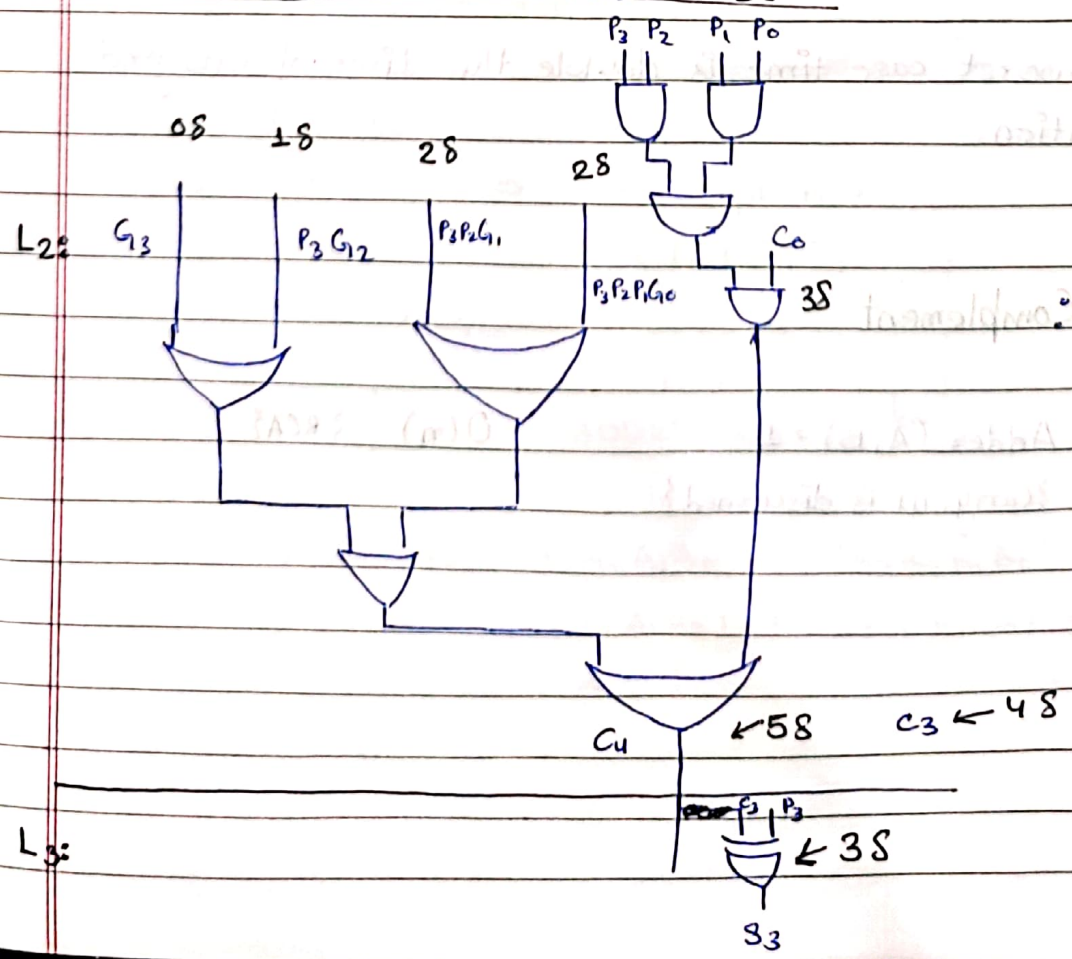
∴ 6 δ for Sum
& 4 δ for C_i.

Ex. Same as above. 4 bit CLA. Fan-in is 2.
(# inputs)

Since C_4 has largest expression it will ~~take~~ have max. delay:

$C_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$.

L1: $P_3 = a_3 \oplus b_3$ $G_3 = a_3b_3$ 3δ



∴ C_4 in 5δ
S_3 in 7δ.

- When fan-in is limited or const., then carry gen. ckt takes $\Theta(\log n)$ levels.

- Signed Addition

  - Both numbers are assumed to be in the same representation.
  - The output will be in the same repr. as the inputs.

(1) 1's Complement

S1: Adder (A, B) = I $O(n)$ {RCA}
S2: Adder (I, Cout) = R $O(n)$ {RCA}

In worst case time is double the time of unsigned addition.

(2) 2's Complement

S1: Adder (A, B) = I $O(n)$ {RCA}
{CarryOut is discarded}

• Subtraction

(1) 1's Complement

$$A - B$$

{A, B are unsigned}

$$\Rightarrow 1'\text{s}(A) + 1'\text{s}(-B)$$

S1: 1's (A)

S2: 1's (B)

S3: 1's (A) + 1's (-B)

(2) 2's Complement

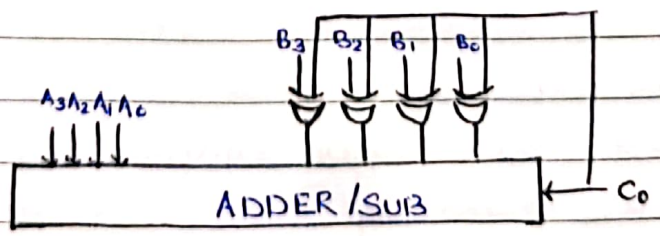
$$A - B$$

{A, B are unsigned}

$$\Rightarrow A + 2'\text{s}(B)$$

S1: 2's (B)

S2: A + 2's (B). $\Rightarrow$ A + Compl (B) + 1



if $C_0 = 0$: $(A_3 A_2 A_1 A_0) + (B_3 B_2 B_1 B_0)$

if $C_0 = 1$: $(A_3 A_2 A_1 A_0) + (\bar{B}_3 \bar{B}_2 \bar{B}_1 \bar{B}_0) + 1$

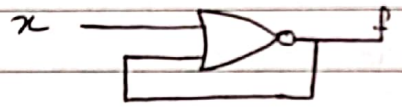
$$\Rightarrow A - B.$$

SEQUENTIAL CIRCUIT

- Combinational circuit's output depended on only the current input.

In <u>Sequential circuits</u> output(s) depend on current input, as well as previous output(s).

Ex.



t	x	f
0	1	0
1	0	1
2	0	0

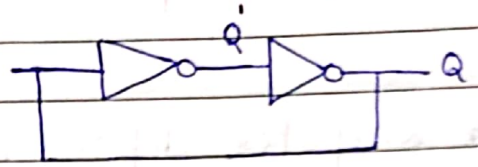
$\therefore f_t = \overline{x + f_{t-1}}$

In seq. ckt. current output may depend on just the output of prev. timeframe, or may depend on multiple timeframe's output.

To remember the previous output we need some kind of memory. The most basic info that needs to be stored is a "single bit". If one bit can be stored, then everything can be stored.

* Basic Memory Elements

Ex.

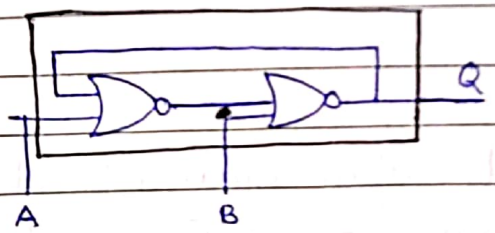


Say initial input is 0. then $\bar{Q}$ is 1 & Q is 0 again & this value loops in the circuit.

This is a single bit storage device.
We get both Q & Q'.

The problem with this ckt is that the input is fixed & can't be changed.

Ex.



This has two user inputs: A & B.

① $A = 0,\ B = 1$

Since $B = 1$, $\overline{1 + x} = 0$

$\therefore\ Q = 0$

Now, $A = 0$, $\overline{0 + Q} = \bar{0} = 1$

so, $Q = 0$ & $\bar{Q} = 1$

Say, user then removes signal, i.e.

$A = B = 0$.

The state is retained as $Q=0$ & $\bar{Q}=1$.

② $A=1$, $B=0$.

Since $A=1$, $\overline{1+x}=0$.
Since $B=0$, $\overline{0+0}=1=Q$.
$\therefore$ $Q=1$, $\bar{Q}=0$

Now, if signal is removed, the state is retained.

t	A	B	Q
0	0	1	0
1	0	0	0
2	1	0	1
3	0	0	1

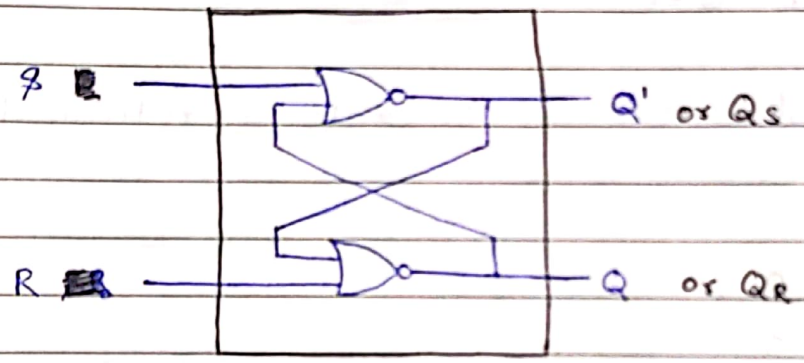
③ $A=1$, $B=1$

Since $A=1$, $\overline{1+x}=0$
Since $B=1$, $\overline{1+x}=0$.

Say if signal is removed, i.e. $A=B=0$.
Now, $Q=1$, $\bar{Q}=1$ at t
$Q=0$, $\bar{Q}=0$ at $t+1$
⋮
if both gates work at same speed.

So, output is uncertain after we switch to $A=B=0$ config.

A	B	Q_n
0	0	Q_{n-1} (retain)
0	1	0 (reset)
1	0	1 (set)
1	1	0 (forbidden)



This circuit is called SR latch. Set-Reset latch.

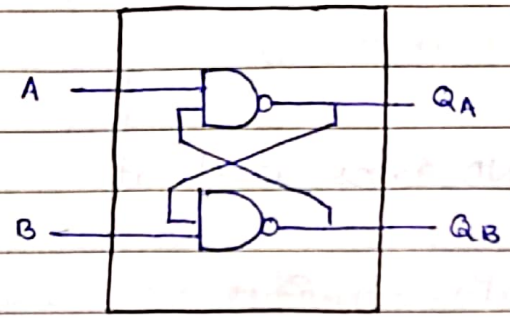
- Why is A=B=1 forbidden in SR latch.

  When we apply S=R=1, we get certain output as Q=0, and the result remains stable as long as S=R=1, but as soon as S=R=0, the output becomes unstable. ∴ S=R=1 is forbidden.

- for all allowed combinations of S & R, Qs & QR are complemenent of each other.

- $Q = 1$ for $S = 1$ $R = 0$
  $Q = 0$ for $S = 0$ $R = 1$
  $Q^t = Q^{t-1}$ for $S = 0$ $R = 0$

- Similarly SR latch can be made using NAND gates.



① $A=0$, $B=1$

Since $A=0$, $\overline{0.Q_B} = 1 = Q_A$

Since $B=1$ $\overline{1.Q_A} = \overline{1.1} = 0 = Q_B$

$\therefore Q_A = 1$, $Q_B = 0$

Now When $A=B=0$.

Since $A=0$, $\overline{0.Q_B} = 1 = Q_A$

& $B=0$, $\overline{0.Q_A} = \overline{0} = 1 = Q_B$.

So, this doesn't retain.

When $A=B=1$.

Since $A=1$, $\overline{1.Q_B} = \overline{Q_B} = Q_A = \overline{0} = 1$

& $B=1$, $\overline{1.Q_A} = \overline{Q_A} = Q_B = \overline{1} = 0$.

② $A=1$, $B=0$.

Since, $B=0$, $\overline{0.Q_A} = 1 = Q_B$

& $A=1$, $\overline{1.Q_B} = \overline{Q_B} = \overline{1} = 0 = Q_A$.

$\therefore Q_A = 0$, $Q_B = 1$.

When, $A=B=1$, data is retained.

③ $A=B=0$.

Since, $A=0$, $\overline{0.Q_B} = 1$

$B=0$, $\overline{0.Q_A} = 1$

$\therefore Q_A = Q_B = 1$.

But when $A = B = 1$

$$Q_A^{t+1} = \overline{Q_B^t}$$

$$Q_B^{t+1} = \overline{Q_A^t}$$

i.e. Oscillation

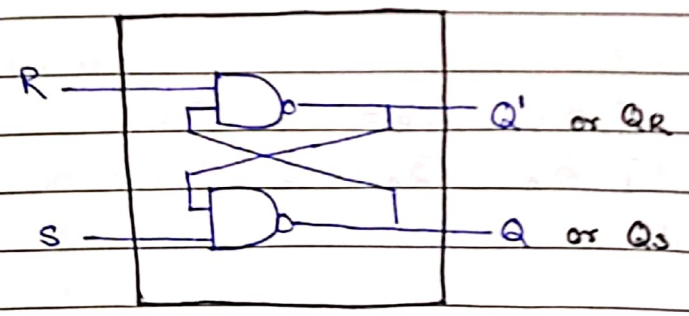
if both NAND gates work at same speed.

else uncertain behaviour.

A	B	Q_A^t	Q_B^t	
0	0	1	1	(forbidden)
0	1	1	0	
1	0	0	1	
1	1	Q_A^{t-1}	Q_B^{t-1}	

$\therefore$ A = Active low reset $Q_A = Q'$

B = Active low set $Q_B = Q$



So, Q is 1 for $S = 0$ $R = 1$

& Q is 0 for $S = 1$ $R = 0$

& Q^t is Q^{t-1} for $S = R = 1$.

Hence it is also called $\overline{S}\overline{R}$ latch.

		SR		$\bar{S}\bar{R}$	
S	R	$Q = QR$	$Q' = Q\bar{R}$	$Q = Q\bar{S}$	$Q' = QR$
0	0	Q^{t-1} (Retain)	Q'^{t-1}	1 (forbidden)	1
0	1	0 (Reset)	1	1 (Set)	0
1	0	1 (Set)	0	0 (Reset)	1
1	1	0 (forbidden)	0	Q^{t-1} (Retain)	$\bar{Q}^{t-1}$

The above is not Truth Table, rather its Characteristic Table, since Q^{t+1} is func. of S, R & Q^t.

S	R	Q^t	Q^{t+1}	
0	0	0	0	Retain
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	X	Forbidden.
1	1	1	X	

Q^t \ SR	00	01	11	10
0			X	1
1	1		X	1

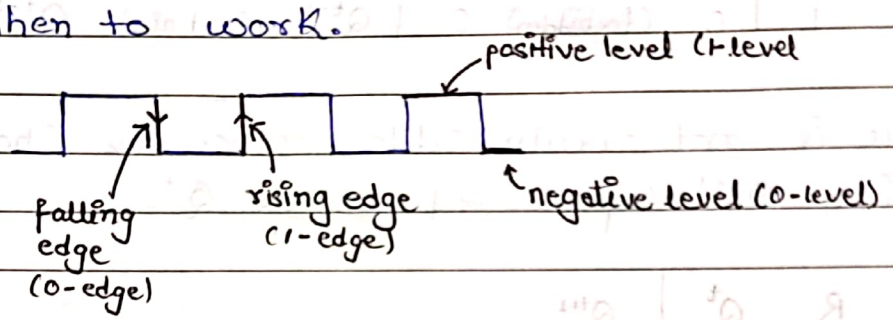
$$Q^{t+1} = S + Q^t\bar{R}$$

The problem with SR latch is that there is no control input such as En. If the input changes & we don't want to affect the output, that is not possible.

Like Enable signal in combinational circuit, Sequential circuits have clock which control the usage of the circuit.

* Clock: It is just an oscillating signal which toggles b/w 0 & 1 continuously.

Different circuits use the clock signal to decide when to work.



The duration b/w two consecutive rising or falling edges is called Time Period (t).

Frequency is defined as no. of oscillations within a time period.

$$\text{freq} \propto \frac{1}{\text{time period}}$$

$$\Rightarrow f = \frac{1}{t}$$

if time period is in seconds, frequency is in hertz.

* Flip-Flops

FFs are latches with clock-signal. There are four standard flip-flops:

(1) SR FF
(2) JK FF
(3) D FF
(4) T FF

* SR flip-flop.

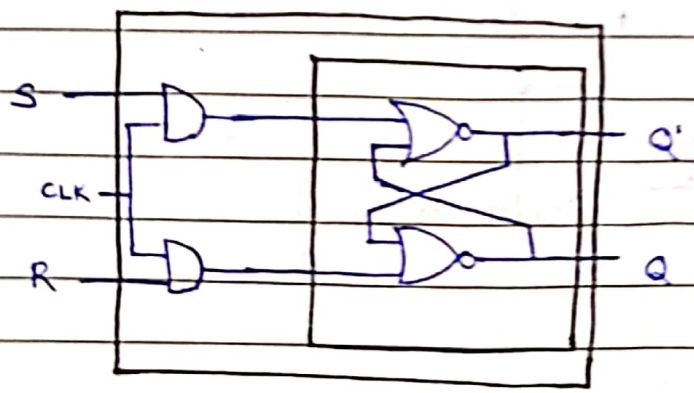
(We first define the characteristic of a FF, the we implement the behaviour in different ways)

SR f-f is just SR latch with clock. The behaviour isn't affected by the clock.

Clock	S	R	Q^{t+1}	
0	x	x	Q_t	(Circuit doesn't respond since CLK=0)
1	0	0	Q_t	(retain)
1	0	1	0	(reset)
1	1	0	1	(set)
1	1	1	x	(forbidden)

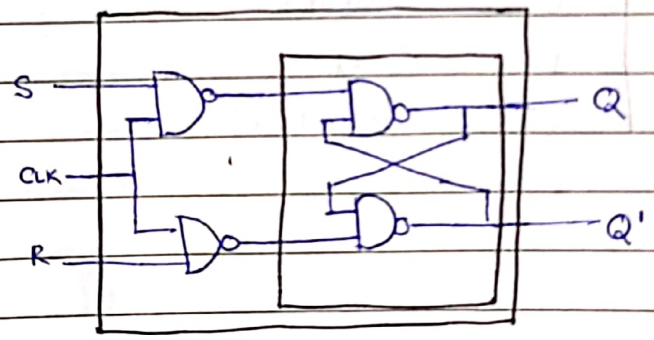
- Implementation

(1) SR latch ⇒ SR ff

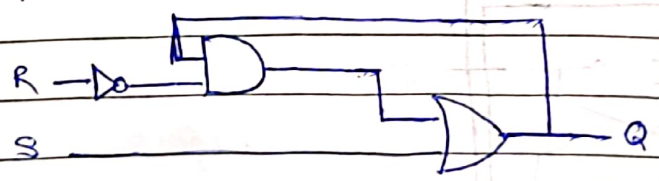


Ex.

(2)



(3) from $Q^{t+1} = Q^t \bar{R} + S$



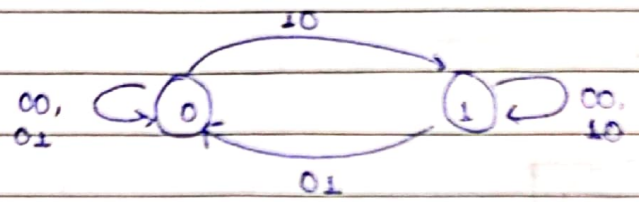
- Excitation Table.

Q_t	Q_{t+1}	S	R
0	0	0	1
0	0	0	0
0	1	1	0
1	0	0	1

Q_t	Q_{t+1}	S	R
1	1	1	0
1	1	0	0

- State Diagram

<SR>



* D Flip-Flop.

Data storage ff.

- Characteristic Table

CLK	D	Q_t
0	X	Q_{t-1}
1	0	0
1	1	1

- Characteristic Eqn

$$Q_t = D$$

- Truth Table

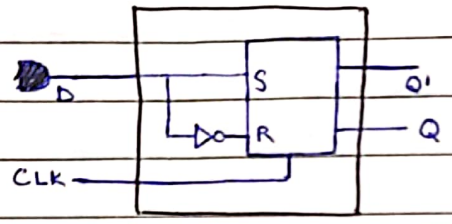
D	Q_{t-1}	Q_t
0	0	0
0	1	0
1	0	1
1	1	1

- Excitation Table.

Q_t	Q_{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

- Implementation

(1) Using SR ff



* Clock Triggering

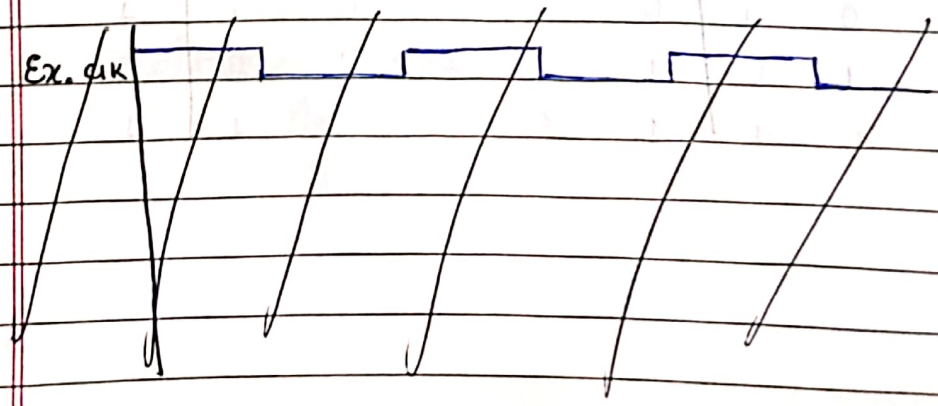
There are two ways in which sequential circuits can respond to Clock Signal.

(1) Level Triggered: Ckt is active when either the clock is on +ve level or -ve level (not both).

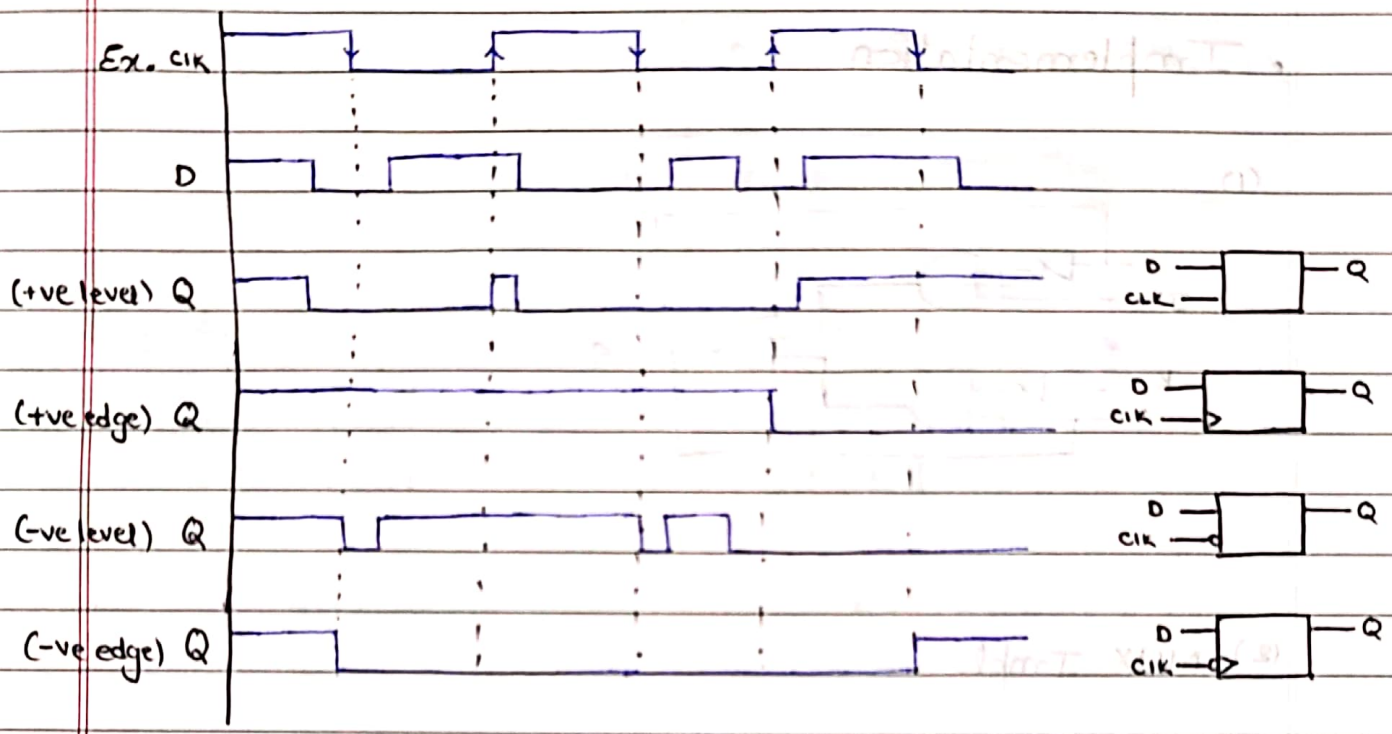
(2) Edge Triggered: Ckt is active only on rising or falling edge of clock (not both).

So there can be four variations of the same flip-flop.

Ex. clk



Note: By default seq. ckts used edge triggered clock, since the state changes only once in a clock cycle (time period).



* JK Flip Flop.

Same as SR ff. but 11 combination is also usable.

- Characteristic Table

CLK	J	K	Q_{t+1}
0	X	X	Q_t
1	0	0	Q_t
1	0	1	0
1	1	0	1
1	1	1	$\overline{Q_t}$

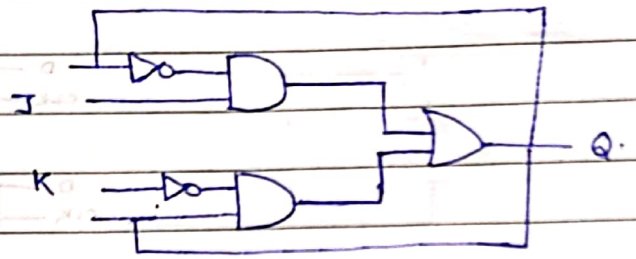
- Characteristic Eqn

Q_t \ JK	00	01	11	10
0			1	1
1	1			1

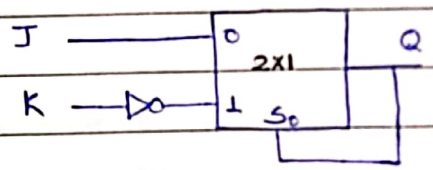
$$Q_{t+1} = J\overline{Q_t} + \overline{K}Q_t$$

- Implementation.

(1)



(2) MUX Impl.



- Excitation Table.

Q_t	Q_{t+1}	J	K
0	0	0	0
0	0	0	1
0	1	1	1
0	1	1	0
1	0	1	1
1	0	0	1
1	1	0	0
1	1	1	0

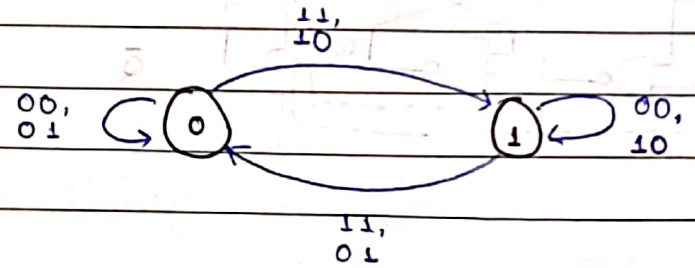
Q_t	Q_{t+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Q_t \ Q_{t+1}	0	1
0		1
1	X	X

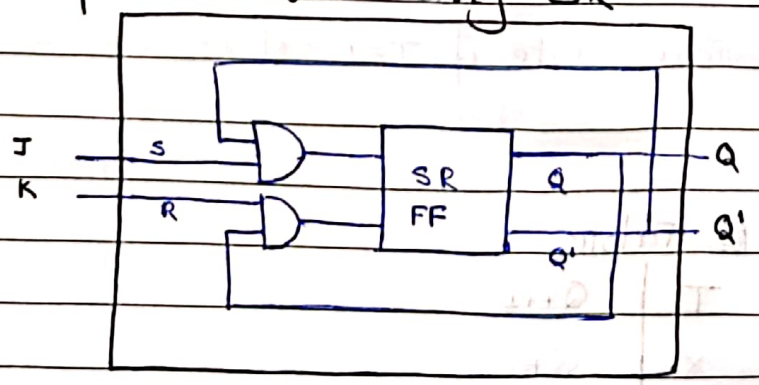
Q_t \ Q_{t+1}	0	1
0	X	X
1	1	

$J = Q_{t+1}$ $K = \overline{Q_{t+1}}$

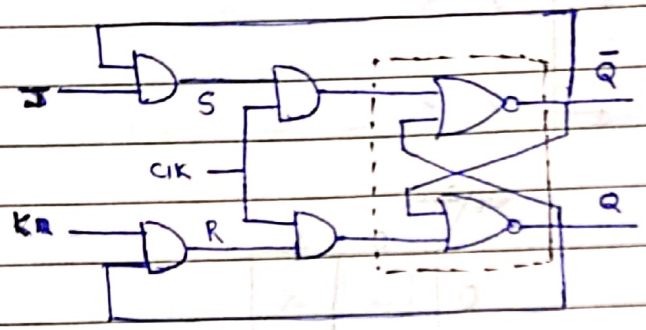
- State Diagram



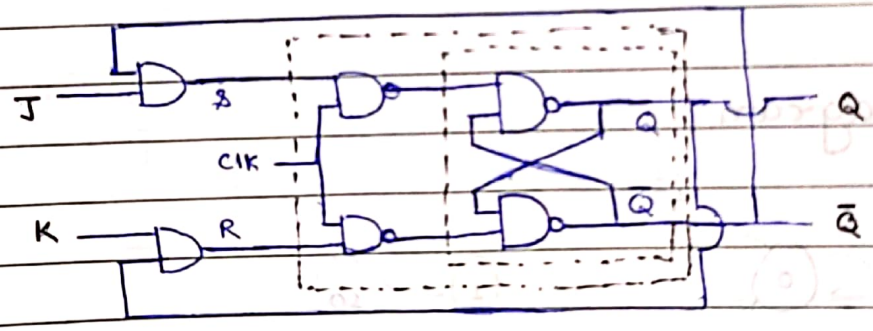
- Implementation using SR



(1) Using SR latch. (NOR impl. of SR ff)



(2) Using $\bar{S}\bar{R}$ latch (NAND impl. of SR FF)



* T FlipFlop.

Toggles from previous state if T=1, else retains.

- Characteristic Table.

dK	T	Q_{t+1}
0	X	Q_t
1	0	Q_t
1	1	$\bar{Q}_t$

- Characteristic Eq^n

$Q_t \backslash T$	0	1
0		$\boxed{1}$
1	$\boxed{1}$	

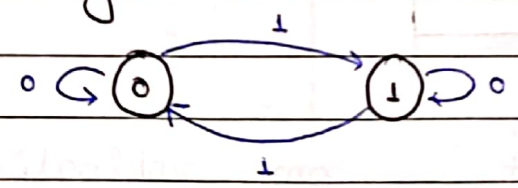
$$Q_{t+1} = \overline{T}\,Q_t + T\,\overline{Q_t}$$
$$\Rightarrow Q_t \oplus T$$

- Excitation Table

Q_t	Q_{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

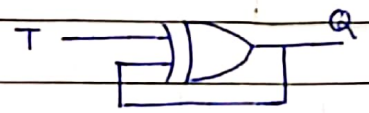
$$\therefore T = Q_t \oplus Q_{t+1}$$

- State Diagram

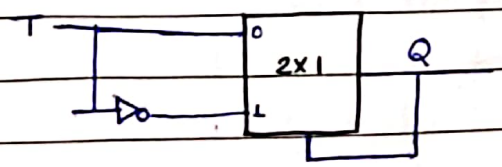


- Implementation

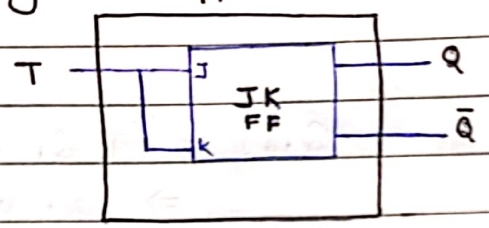
(1) from Char. Eq^n



(2) Using MUX



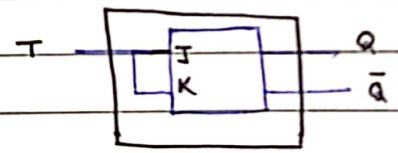
(3) Using JK ff



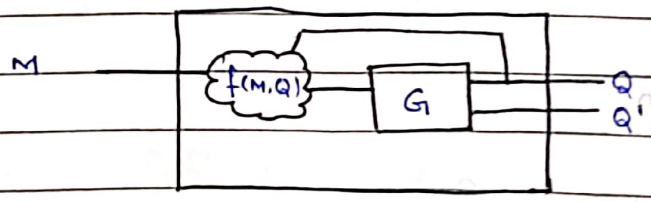
* Flip Flop Conversion

Given a flip-flop G, we want to create the desired flip-flop D.

Ex. T ff using JK ff.



So, Using given ff, G & some combinational circuit, we create ff D.



The combinational ckt. is func. of M & Q, output of which is fed to G.

- Steps:

(1) Write truth table of ff M. as $Q_{t+1} = f(M, Q_t)$

(2) From Excitation Table of $Q_t \to Q_{t+1}$, find G as

$G = f(Q_t, Q_{t+1})$

$\Rightarrow f(Q_t, M)$

Ex. D ff to T ff.

Given: D ff

Desired: T ff.

S1:

T	Q	Q_{t+1}
0	0	0
0	1	1
1	0	1
1	1	0

S2:

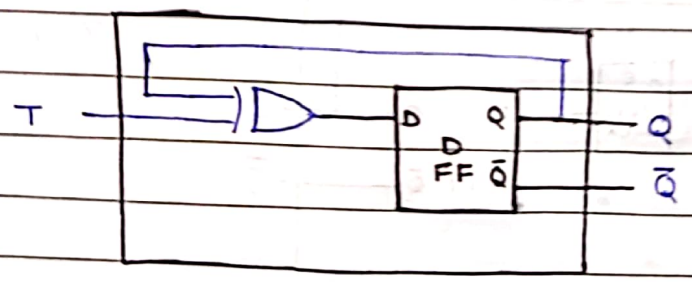
Q_t	Q_{t+1}	D
0	0	0
1	1	1
0	1	1
1	0	0

S3:

T \ Q	0	1
0		$\boxed{1}$
1	$\boxed{1}$	

$\therefore D = Q_{t+1}$

$= T \oplus Q$.



Ex. T ff to DFF. Given: T Desired: D.

D	Q_t	Q_{t+1}	T
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

D \ Q^t	0	1
0		$\boxed{1}$
1	$\boxed{1}$	

$\therefore T = D \oplus Q_T$

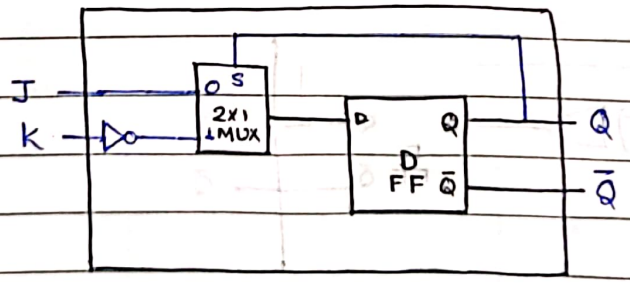
Ex. JK ff. from Dff.

Given: D Desired: J

J	K	Q_t	Q_{t+1}	D
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_t \ JK	00	01	11	10
0			1	1
1	1			1

$\therefore D = J\overline{Q_t} + Q_t\overline{K}$



Ex. PN FF

P	N	Q_{t+1}
0	0	0
0	1	Q_t
1	0	$\overline{Q}_t$
1	1	1

Write Excitation Table & Create D FF using PNff.

(1)

Q_t	Q_{t+1}	P	N
0	0	0	X
0	1	1	X
1	0	X	0
1	1	X	1

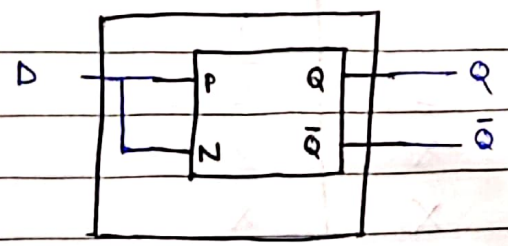
(2)

D	Q_t	Q_{t+1}	P	N
0	0	0	0	X
0	1	0	X	0
1	0	1	1	X
1	1	1	X	1

D \ Q_t	0	1
0		X
1	$\boxed{1}$	$\boxed{X}$

D \ Q_t	0	1
0	X	
1	$\boxed{X}$	$\boxed{1}$

$\therefore P = D$ $\therefore N = D$



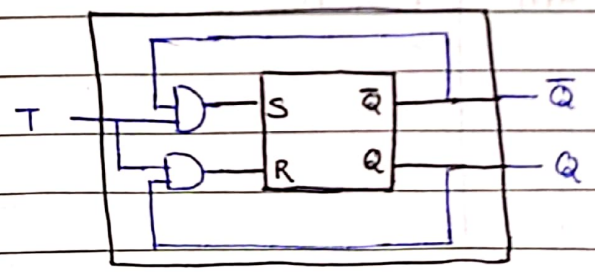
Ex. SR ff to T ff.

T	Q_t	Q_{t+1}	S	R
0	0	0	0	X
0	1	1	X	0
1	0	1	1	0
1	1	0	0	1

T \ Q_t	0	1
0		X
1	$\boxed{1}$	

T \ Q_t	0	1
0	X	
1		$\boxed{1}$

$S = T\overline{Q_t}$ $\qquad$ $R = TQ_T$

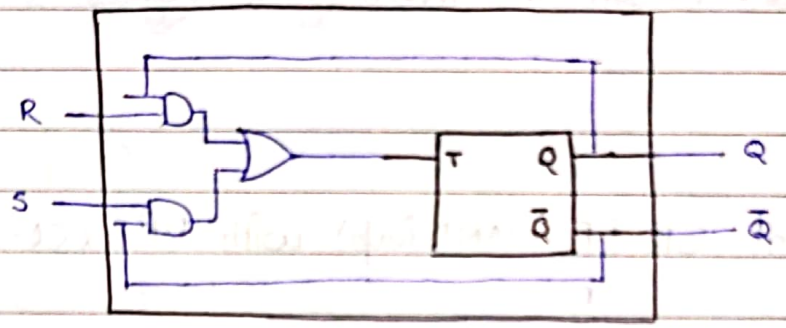


Ex. T ff to SR ff.

S	R	Q_t	Q_{t+1}	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	X
1	1	1	X	X

Q_t \ SR	00	01	11	10
0			X	1
1		1	X	

$\therefore \; T = S\bar{Q}_t + Q_t\bar{R}$



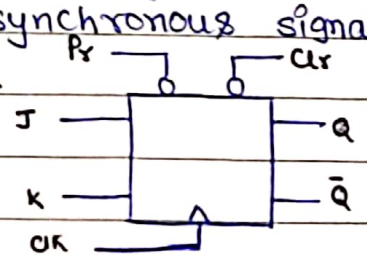
(we can also use Mux in-place of basic gates).

* Special Inputs in FlipFlop.

When a digital circuit is started, the values in flip-flops are garbage. We generally want to either set all FFs to 0s or to 1s.
Also in counters we generally count in ascending order from all 0s or descending from all 1s.

For setting all the Flipflops to 0s we have Clear input & for setting all FFs to 1s we have Preset input. Both are Asynchronous signals (i.e. don't care about clock).

Ex.



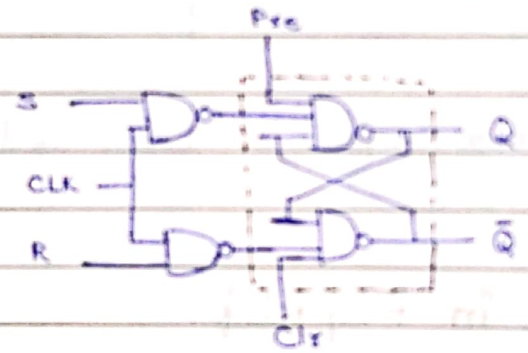
Preset	Clear	CLK	J K	Q_{t+1}
0	1	X	XX	1
1	0	X	XX	0
1	1	– Normal Operation –		
0	0	– Forbidden – (Uncertain)		

* In Sequential circuit

(1) Synchronous: Works according to clock trigger.

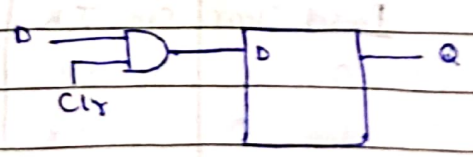
(2) Asynchronous: Doesn't care about clock triggers.

Ex. Implement SR FF (NAND imp) with PRE & Clr.



Ex. Impl. D FF with Sync. Preset & Clear
(i.e. works on clock triggers)

Clr	Pre	Q_{t+1}	D
0	1	0	0
1	0	1	1
1	1	D	D
0	0	X	X



Sync. Clear

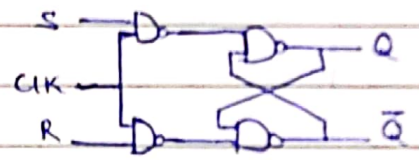
* Master Slave Flip Flop.

Any ff. ~~ed~~ can be designed in one of four ways :

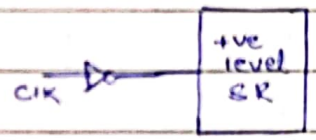
(1) +ve Edge Trigg.
(2) -ve Edge Trigg.
(3) +ve Level Trigg.
(4) -ve Level Trigg.

Ex. SR ff.

① +ve level trigerred.



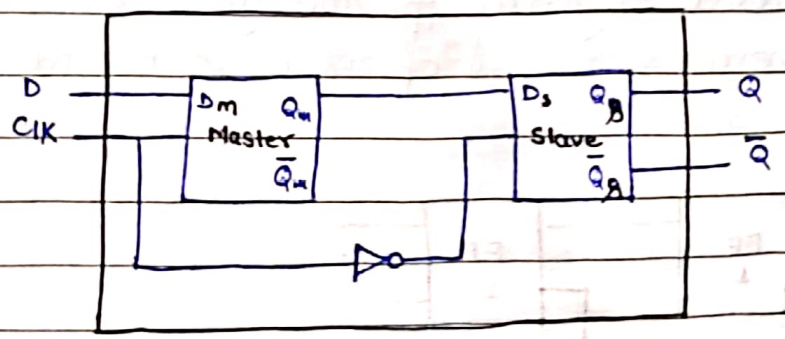
② -ve level trigerred



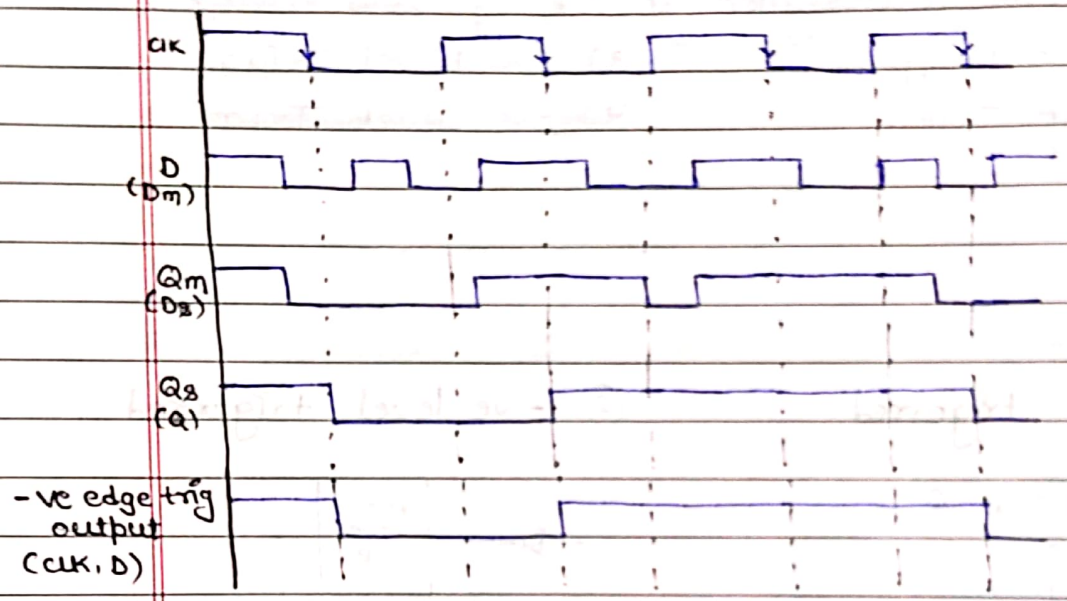
There are several ways of creating edge trigerred ff. but we only study Master-Slave ff.

M-S ff ≡ Edge Trigerred ff.

Ex. Master-Slave D flip-flop.
(we use +ve level trigg. DFF)



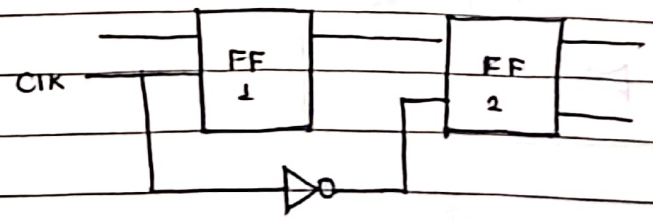
Here, Master acts as +ve level trig & Slave as -ve level trig.



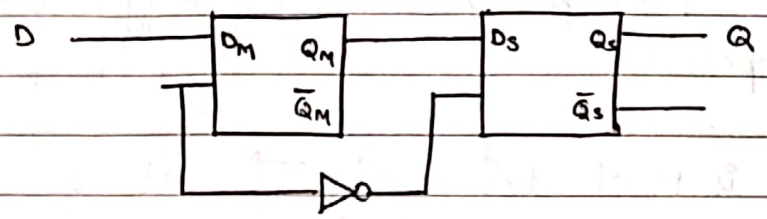
So, the output of Q_s in MS config. is same as the expected output of -ve edge trig. ff.

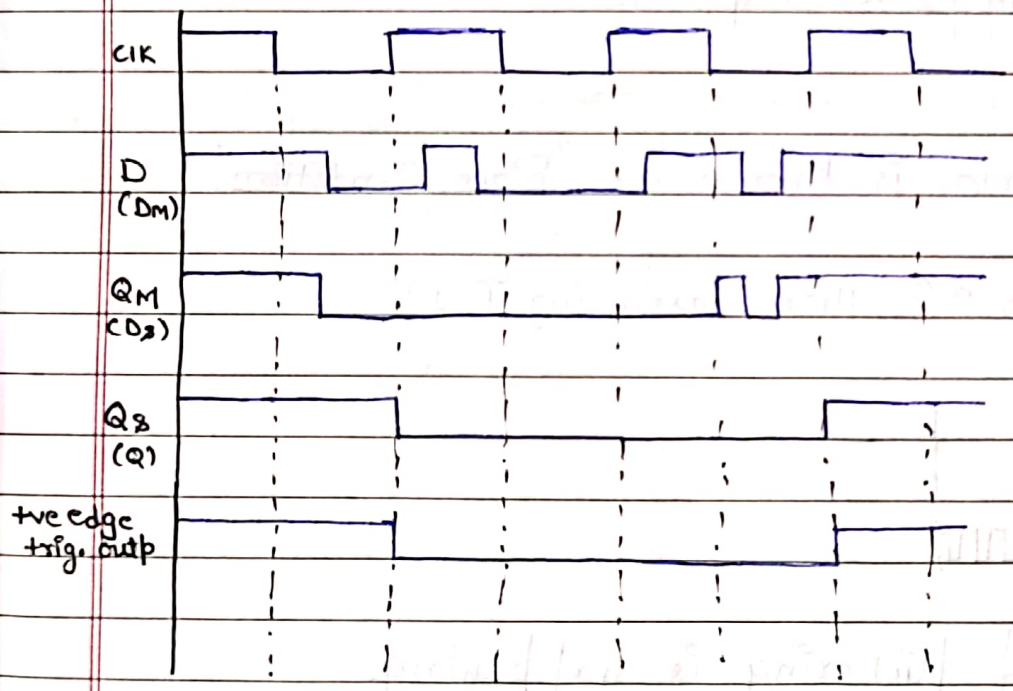
- To create +ve edge trigg. give -ve level clock to master & +ve level clock to Slave.

- General idea of Master Slave flipflop:
  Take two level trigerred ff. and give opposite clocks to them, & output of one is connected to input of other.



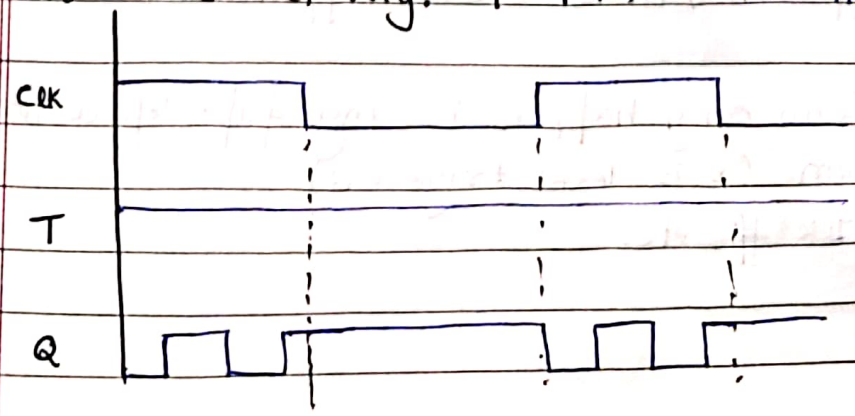
Ex. Verify the behaviour of -ve level trigerred MS config.





So, when both ffs are -ve level trigerred, then the above MS config. acts as +ve edge trig. ff.

Ex. (Level (+ve) trig. T FF) FF has prop. delay t_{PD}



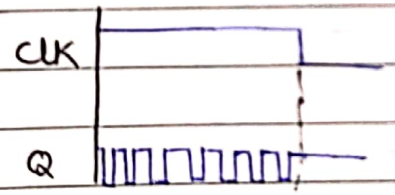
toggling happens after every t_{PD} delay when ckt is on (i.e CLK is high)

- The reason for the unnecessary toggle ~~s~~ are:
  1. On time > t_{PD}. If the on period was smaller than prop. delay time, the toggle would only happen once.

  2. CKt is level triggered. If ckt was edge trig. then on time would be very small for any flickering to happen.

- This scenario is known as "Race Condition".

Say if $t_{PD} \sim 0$, then (assuming T-1)



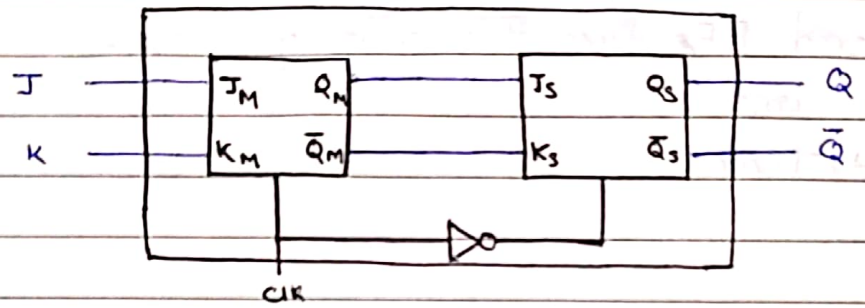
So, a lot of flickering is happening.

- Since T_{PD} can change over time, making on time less than T_{PD} is not a viable solution.

Using Edge trigerred / Master-Slave ff is the solution.

- Race condition can only happen in any flip-flop which has Toggle option (& is level trigerred)
  Ex. T ff, JK ff, etc.

Ex. M-S JK flip-flop

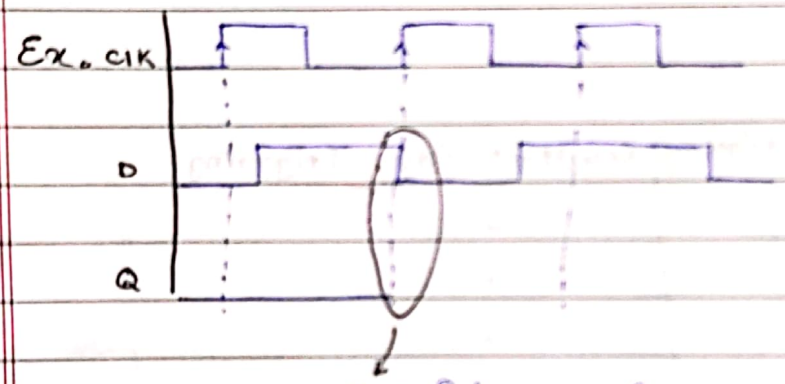


*

Clock to Master	Level triggering	Resultant Edge Triggering
-ve	-ve	-ve.
-ve	+ve	+ve
+ve	-ve	+ve
+ve	+ve	-ve

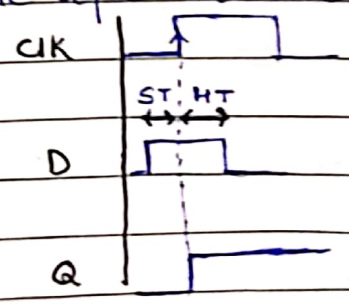
* Timing Issues in FlipFlops

Edge triggered FFs face two timing issues:
(1) Setup Time
(2) Hold Time.



at this point there is ambiguity, since at +ve edge of clock, D signal is also at edge. Is it +ve or -ve?

So, if we have to make Q equal to 1 at this point, we'll have to make D=1 a while before rising edge & will have to keep it 1, till a little while after the +ve edge.

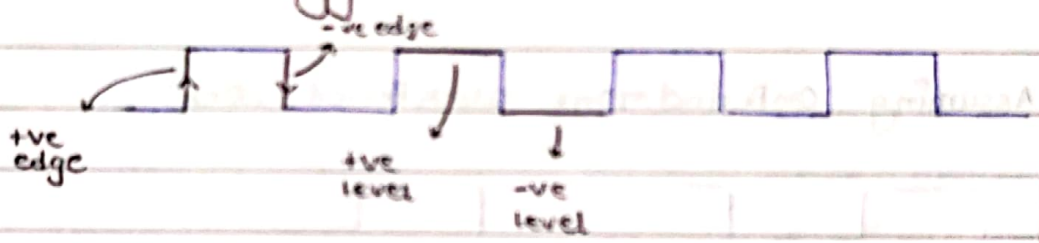


The time for which D needs to change before rising edge is called Setup Time.

The time after the rising edge for which D needs to remain stedy is called <u>Hold Time</u>.

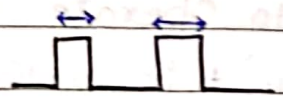
During the hold & setup time, D <u>mustn't change & remain</u> stable at a single value.

* Clock Terminology

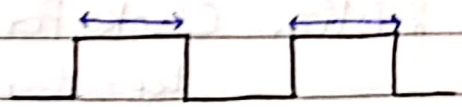


- **Clock Period :** Duration b/w two rising edges or b/w two falling edges is called Clock Period.
- **Frequency :** 1/clock Period
- **Clock Width / Pulse Width :** On period or length of +ve level is called Pulse width.

Ex.



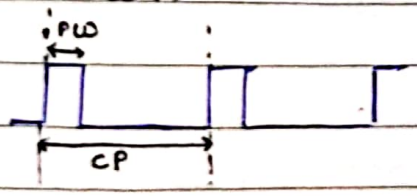
Ex.



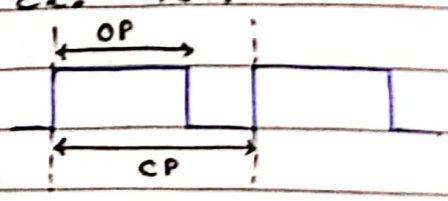
- **Duty Cycle :** Percentage of time for which clock is on.

$$\frac{\text{On period}}{\text{Clock Period}} \times 100\%$$

Ex. 20%



Ex. 70%

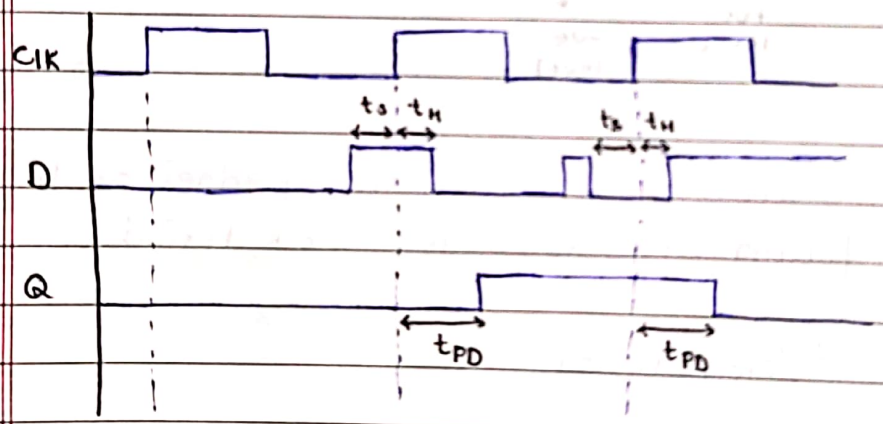


- Hold Time vs Setup Time vs Propagation Delay

Propagation delay in Flip-Flop is the time b/w the arrival of trigering edge & the availability of stable output.

Ex. $t_H = 1ns$ $t_s = 5ns$ $t_{PD} = 3ns$

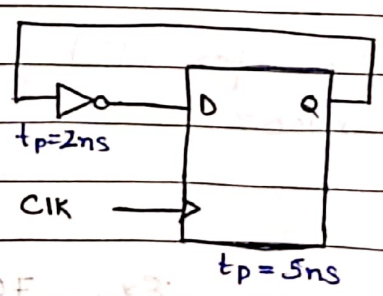
(Assuming On Period = 10ns Clock Period = 20ns)



- Min. Clock Period for CKt to Operate

$$t_{cp} = t_{setup} + max(t_{PD}, t_{hold}) + t_{combinational}$$

Ex.



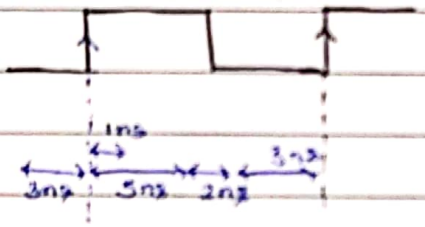
$t_{setup} = 3ns$

$t_{hold} = 1ns$

$\because$ $t_{setup} = 3ns$, so D must be availabe 3ns before rising edge.

& $\because$ $t_{Hold} = 1ns$, so D must be available 1ns after rising edge.

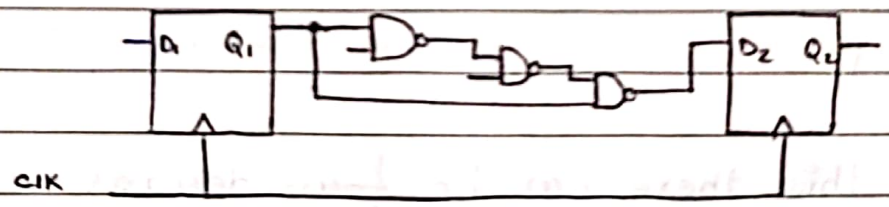
Q would be stable after 5ns after rising edge.
Output of inv. would be stable after 2ns.
So next rising edge must be at least 3ns (t_{setup}) after this.



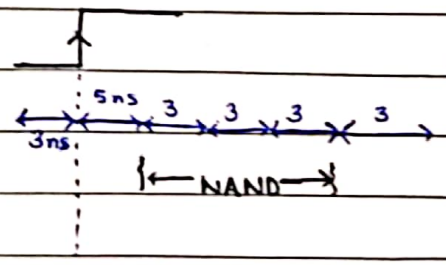
$\max(t_{pd}, t_{hold})$ setup NOT ($t_{comb.}$)

$\therefore$ Clock period$_{min}$ = 5ns + 3ns + 2ns
= 10ns.

Ex.



$t_{p(NAND)} = 3ns$
$t_{p(DFF)} = 5ns$
$t_{setup} = 3ns$



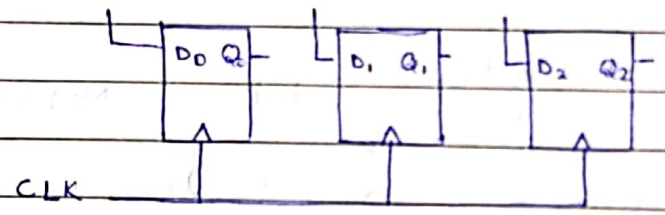
$\therefore$ $t_{cp_{min}} = \underbrace{5}_{t_{p(DFF)}} + 3 \times \overbrace{3}^{t_{p(NAND)}} + \underbrace{3}_{t_{setup}} = 17ns$

$\therefore$ Max freq $= \frac{1}{17ns} = \frac{1}{17}$ GHz.

* Registers

Registers are n-bit storage device, comprising of n flip-flops.

Ex. 3bit Register using D FF



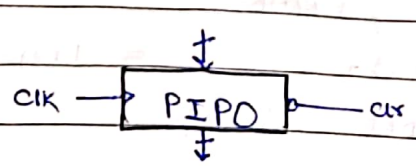
The input to the register & output from register can be retreived in two ways

① Serial
② Parallel.

Based on this there can be four designs of flip-flop:

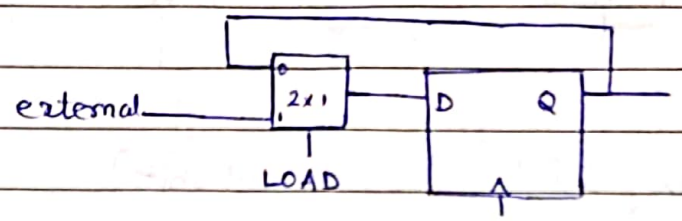
① PIPO
② SIPO
③ PISO
④ SISO.

- PIPO



- All ffs have common clock.

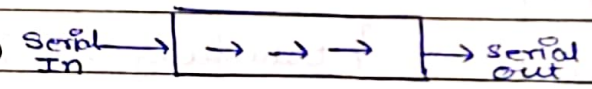
- There is an additional LOAD input, which when active, allows the data to be stored on the register otherwise not.



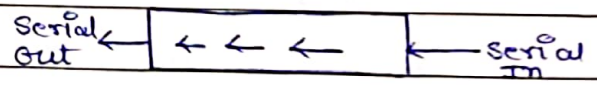
LOAD $= 1$ $\quad Q_t = Ext$

$= 0$ $\quad Q_t = Q_{t-1}$

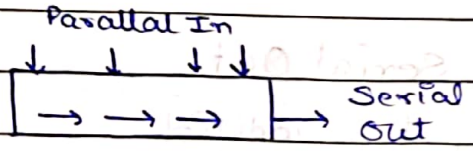
- Shift Registers are used to store data, & allows for data to be left or right shifted when shift signal is applied. Bits shifted out can be wasted or input again in cyclic fashion on other end.

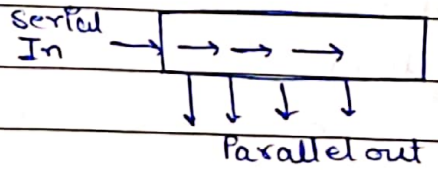
① 

Shift Right

② 

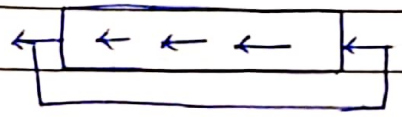
Shift Left.

③ 

④ 

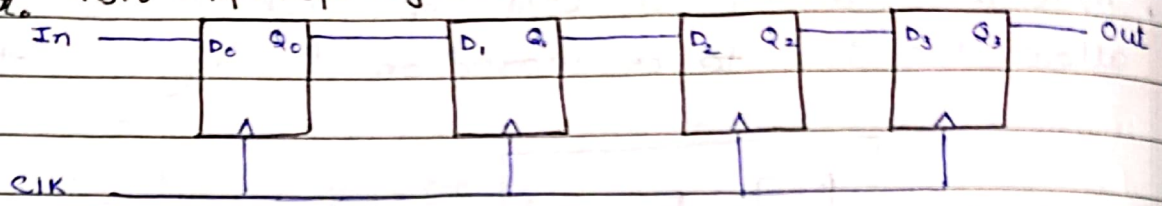
⑤

Rotate Right

⑥ 

Rotate Left

Ex. 4bit Shift Left Register



(Shift left means Shift towards MSbit)

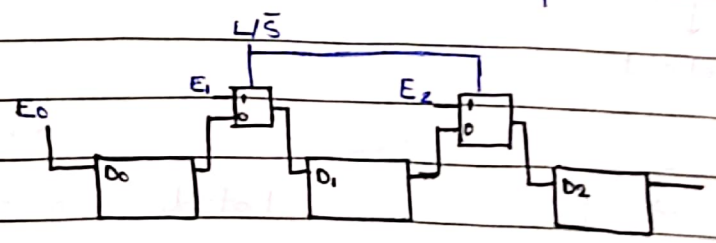
t	In	Q_0	Q_1	Q_2	Q_3	Out	
0	1	0	0	0	0	0	} Initial
1	0	1	0	0	0	0	
2	1	0	1	0	0	0	
3	1	1	0	1	0	0	
4	0	1	1	0	1	0	} Data fully loaded
5	0	0	1	1	0	1	
6	0	0	0	1	1	0	
7	0	0	0	0	1	1	
8	0	0	0	0	0	1	} Data fully retrieved.

Ex. 3 bit Parallel In Serial Out.

We use $\overline{\text{Shift}}$/Load input in addition

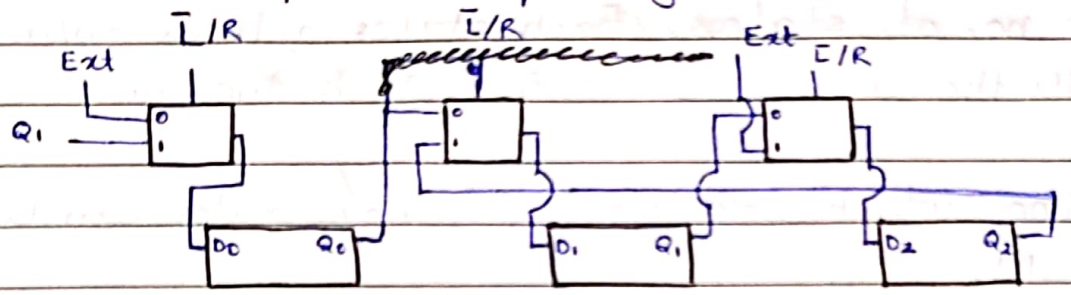
$\overline{S}/L = 0$ Shift operation

$= 1$ Load Operation.



• Bidirectional shift Register

Allows for both left & right shift.



When $\bar{L}/R = 0$ left shift enabled

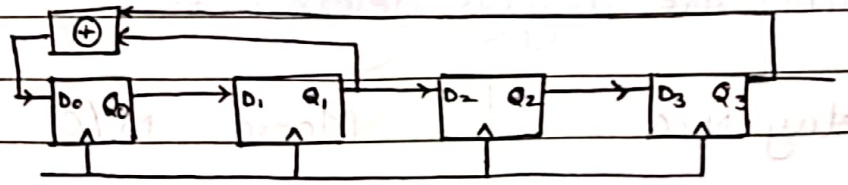
$= 1$ Right shift enabled.

• Universal Shift Register

Shift registers having all functionality:

1. PIPO
2. SISO
3. Clr & Preset
4. Bidirectional Shift.

Ex.



Assume you have $Q_0Q_1Q_2Q_3$ 0101 initially.

t	D_0	D_1	D_2	D_3	Q_0	Q_1	Q_2	Q_3
0	-	-	-	-	0	1	0	1
1	0	0	1	0	0	0	1	0
2	0	0	0	1	0	0	0	1
3	1	0	0	0	1	0	0	0
4	0	1	0	0	0	1	0	0

t	Q_0	Q_1	Q_2	Q_3
5	1	0	1	0
6	0	1	0	1

Back to initial state.

* Finite State Machine

Finite no. of states. Each states acts as m/m as it tells the current state in which the machine is.

Anything which concurs with above rules can be called as FSM.

FS Automata acts as language acceptor i.e. output is Yes or No.

FS Transducer produces output as a func. of the input.

Transducer is represented using:

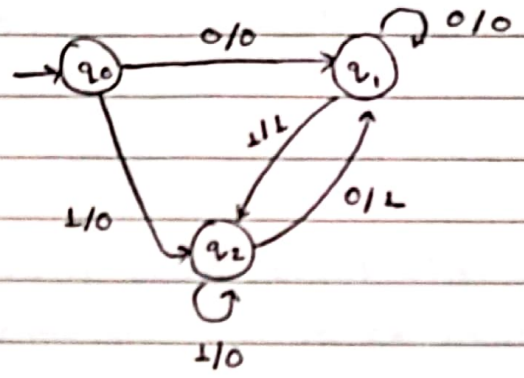
$$(Q, \Sigma, \Gamma, \delta, \omega, q_0)$$

- Q → states
- Σ → inp alphabet
- Γ → output alphabet
- δ → next state func.
- ω → output func.
- q_0 → initial state

Transducer don't have final state(s).
Transducers are always deterministic.

Melay M/C	Moore M/C
oup. = f(present state, input)	output = f (present state).
$\omega: Q \times \Sigma \rightarrow \Gamma$	$\omega: Q \rightarrow \Gamma$

Ex. Check if Melay or Moore m/c.

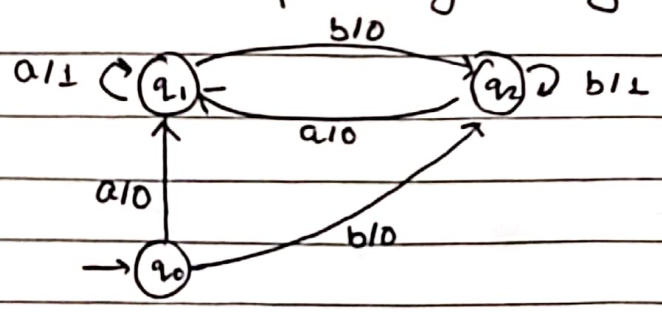


from q_0 output is always 0
from q_1 output is 0 when inp. is 0
1 when inp. is 1.

$\therefore \omega : Q \times \Sigma \rightarrow \Gamma$

$\therefore$ Melay m/c.

Ex. What is the following Melay m/c doing?

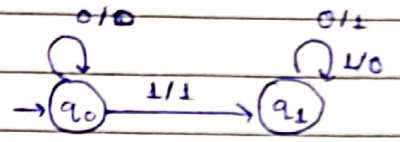


$\Sigma = \{a, b\}$
$\Gamma = \{0, 1\}$.

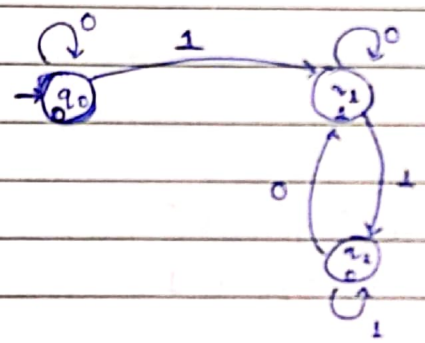
Taking example: aaa babbb.
out: 01100011

$\therefore$ The m/c marks consecutive occurance of characters with 1.

Ex. Design Melay & Moore m/c for 2's comp.

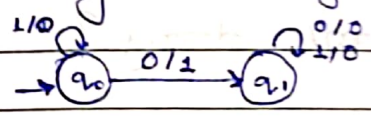


Melay

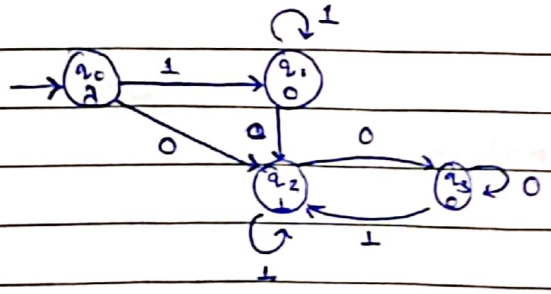


Ex. Design Melay & Moore m/c for incrementing.
The first 0 from RtoL should be turned 1.

Melly m/c: (Right to left reading)



Moore m/c:



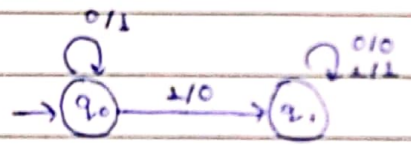
Ex Decrement a binary number.

① 101101 − 1 = 101100

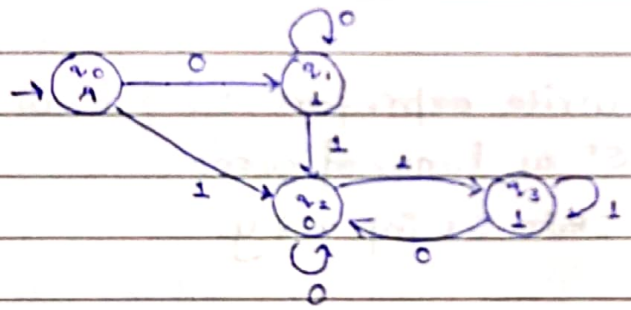
② 10010 − 1 = 10001

③ 1000 − 1 = 0111

So, starting from right first 1 should be turned 0 & all 0s before that have to be made 1. Else remains same.



Mealy m/c

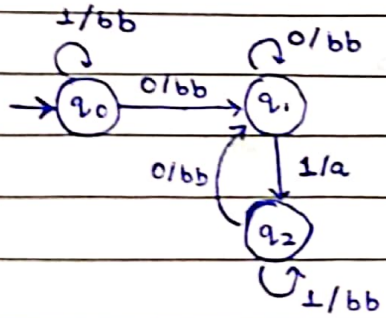


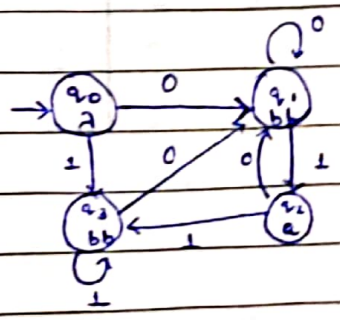
Moore m/c.

Ex. Detect "01" & output 'a', else output "bb".

① 0 1 1 0 → bb a bb bb

② 0 1 0 1 0 → bb a bb a bb

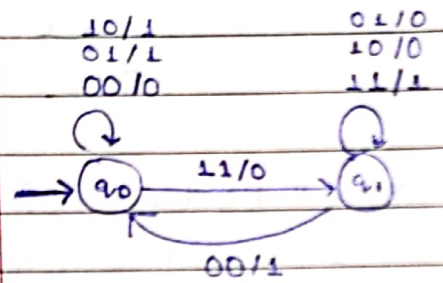




Ex. Design an FSM to add two binary nos.

$\Sigma = \{00, 01, 10, 11\}$

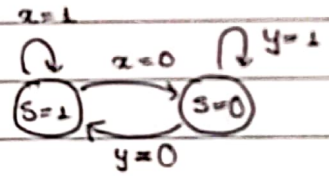
$\Gamma = \{0, 1\}$.



q_0 : $c = 0$

q_1 : $c = 1$

Ex.



write expr. for S^+ as func. of curr. state S & inp x & y.

S	x	y	S^+
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

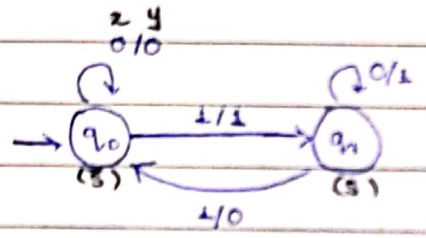
S \ xy	00	01	11	10
0	1			1
1			1	1

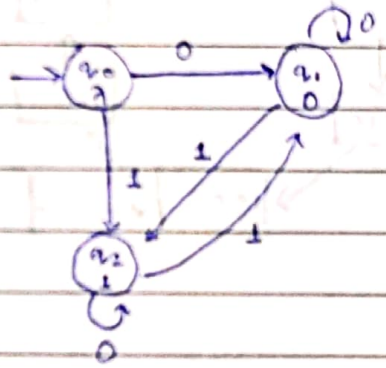
$S^+ = \bar{S}\bar{y} + Sx$

Ex. Construct FSM. which takes binary string as inp. It outputs 0s untill first 1, then it prints 1 untill 1 is encountered again & switches output again & so on.

Inp: 0001001101

Out: 0001110110





Output func:

$\omega = f(s, x)$

state, inp

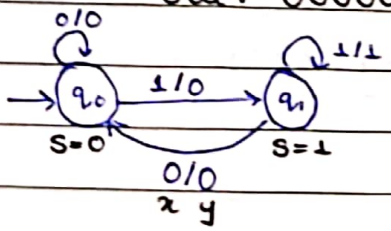
$y = \bar{s}x + s\bar{x}$

Next state eqn:

$s^+ = \bar{s}x + s\bar{x}$

Ex. Design FSM which replaces the first 1 in a subsequence with 0. Ex. Inp: 0010001100001100

Out: 0000000100000100



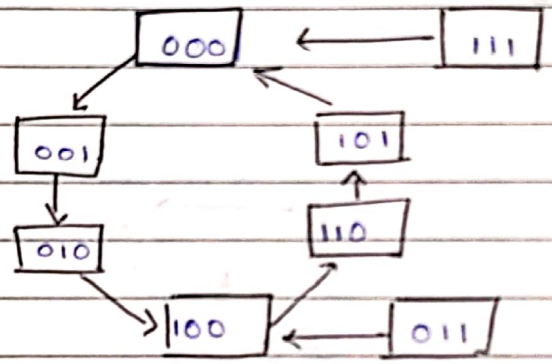
$s^+ = \bar{s}x + sx = x$

$y = sx$

* Counters

A register which goes through a predetermined sequence of states.

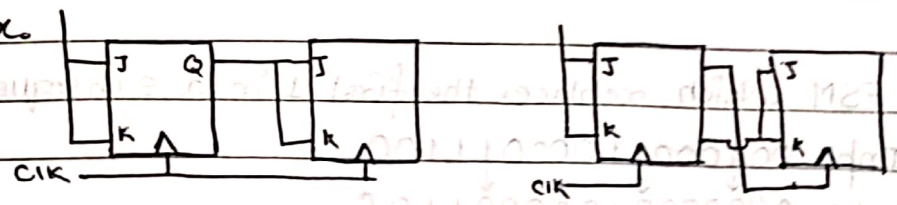
Ex.



- Synchronous Counter: All FFs in ckt share a common clock.

- Asynchronous Counter: The output of a FF can act as clock of another FF.

Ex.



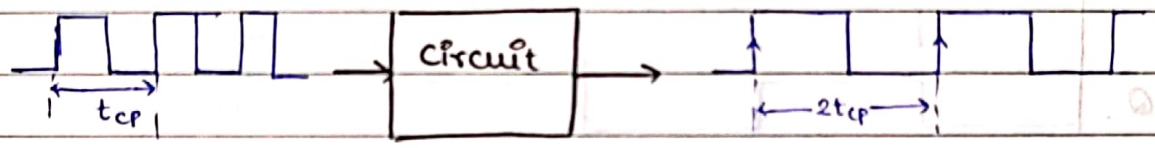
Synchronous Asynchronous

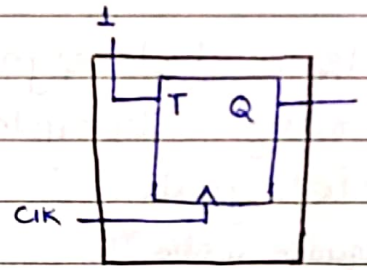
- ~~Asynchronous Counter~~

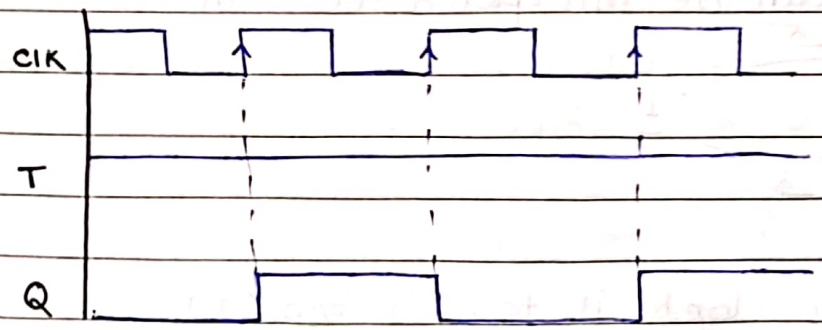
- T FF Counter (& other Application)

Ex. Given a clock signal of freq. f Hz, generate another clock pulse with freq. f/2 Hz.

$f = \frac{1}{CP}$ So, $\frac{f}{2} \Rightarrow 2CP$



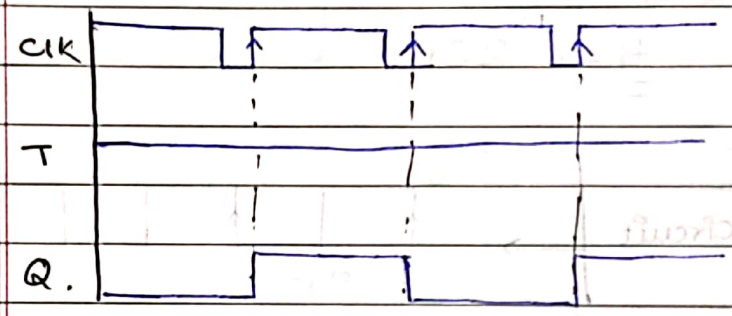




∴ Q is $2t_{CP}$ width.

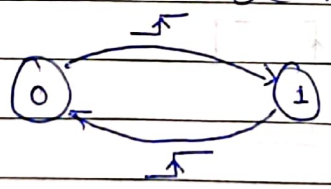
Ex. T FF converts any clock signal of arbitrary duty cycle to clock signal of 50% duty cycle.

Say duty cycle is 80%.



∴ T ff with T converts a clock signal of freq. f Hz & arbitrary duty cycle to f/2 Hz & 50% duty cycle signal.
(perfect square wave)

The same ckt can be interpreted as counter



Since after every clock it toggles b/w 0 & 1.
Hence the ckt acts as Modulo-2 counter.

- D FF Application

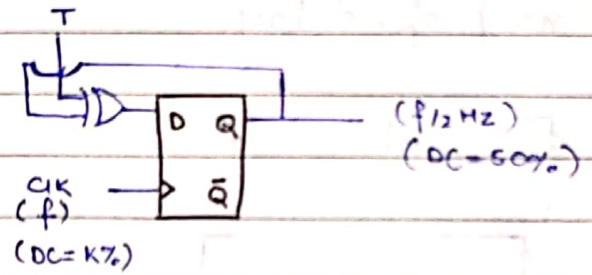
Ex. freq. division.

Just impl. T ff logic with D FF.

T	Q	Q^+	D
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0.

$\therefore D = T \oplus Q$

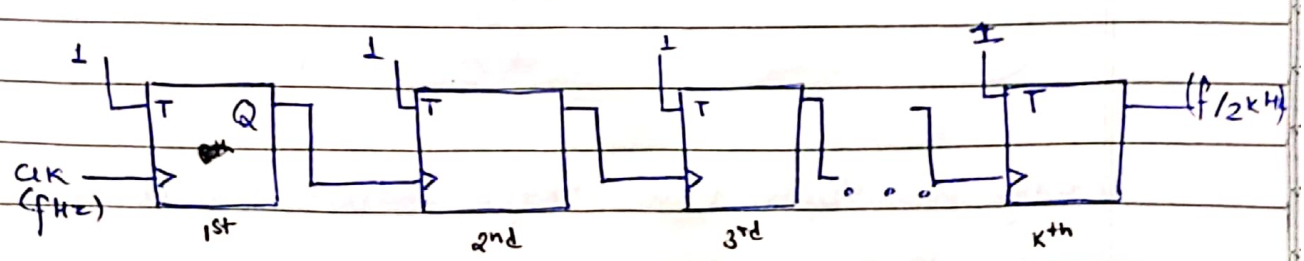
Rest ckt remains same.



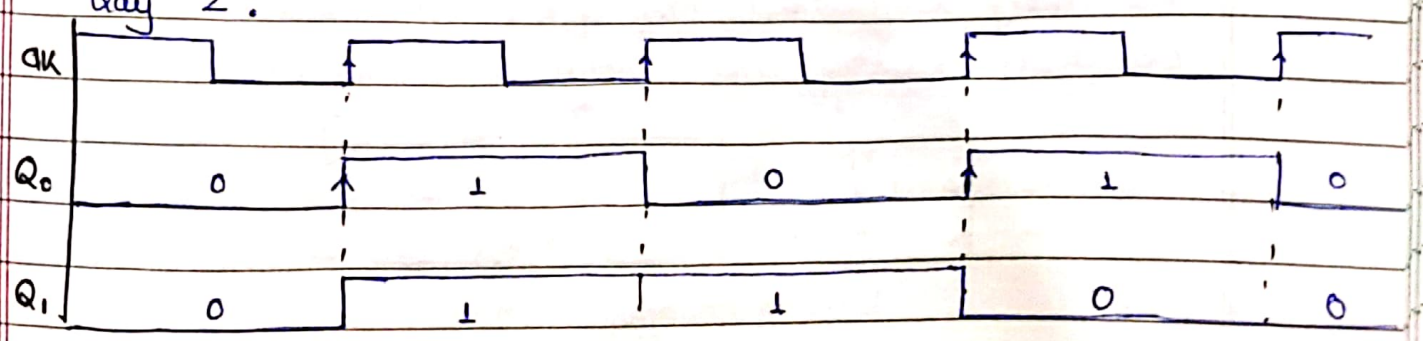
Note: A Modulo-n counter goes through n unique states.

Ex. Design a ckt which takes clock of f Hz & produces signal of $f/2^K$ Hz.

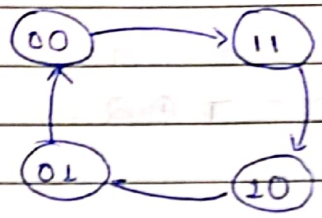
W.K.t 1 Tff divides f by 2.
So, K Tff will divide f by 2^K.



Say $f/2^2$.



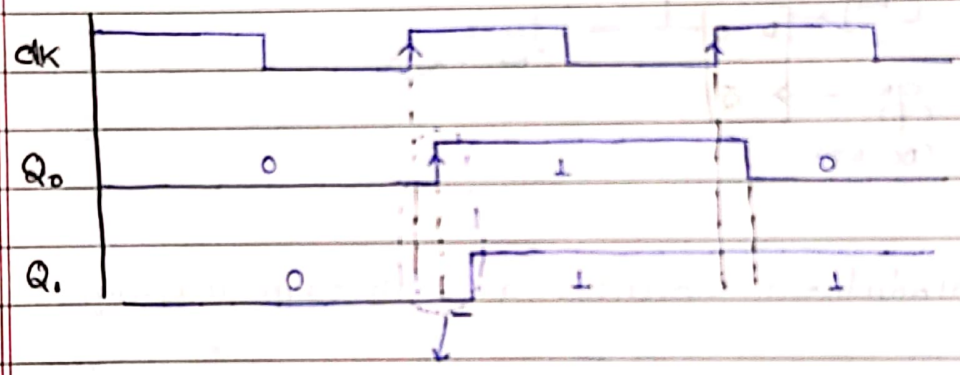
$Q_1 Q_0$



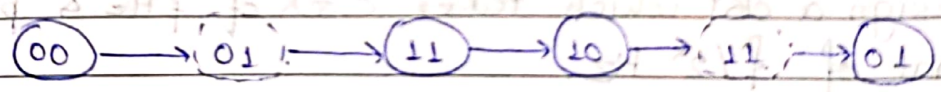
$3 \rightarrow 2 \rightarrow 1 \rightarrow 0$

∴ This is a modulo-4 counter as well.

- Assuming some prop. delay:



$Q_1 Q_0 = 01$



This glitch is called **transient state**.
Here 2 transient states are possible.

Binary Counter & Binary Ripple Counter

A n-bit Binary counter counts $0, 1, 2, \ldots, 2^n - 1$ either in ascending or descending order.

It can be implemented as both Async & Sync counters.
(Ripple)

Ripple Counter is just another name of Async counters.

① Binary Ripple Down Counter.

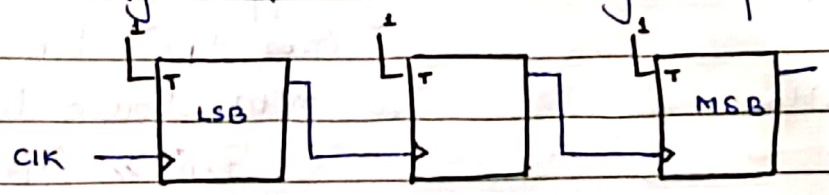
Say 3 bit, then sequence is:

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

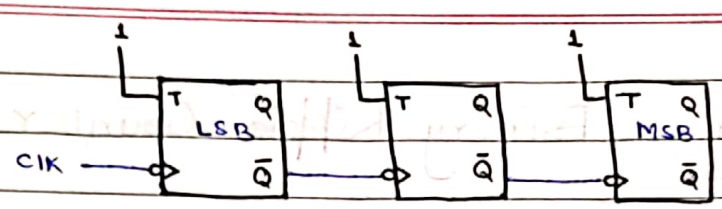
① LSB changes fastest, every clock cycle. So connect LSB to clock.

② Middle bit changes every time LSB goes from 0 to 1. ↑

③ MSB changes when middle bit goes from 0 to 1. ↑

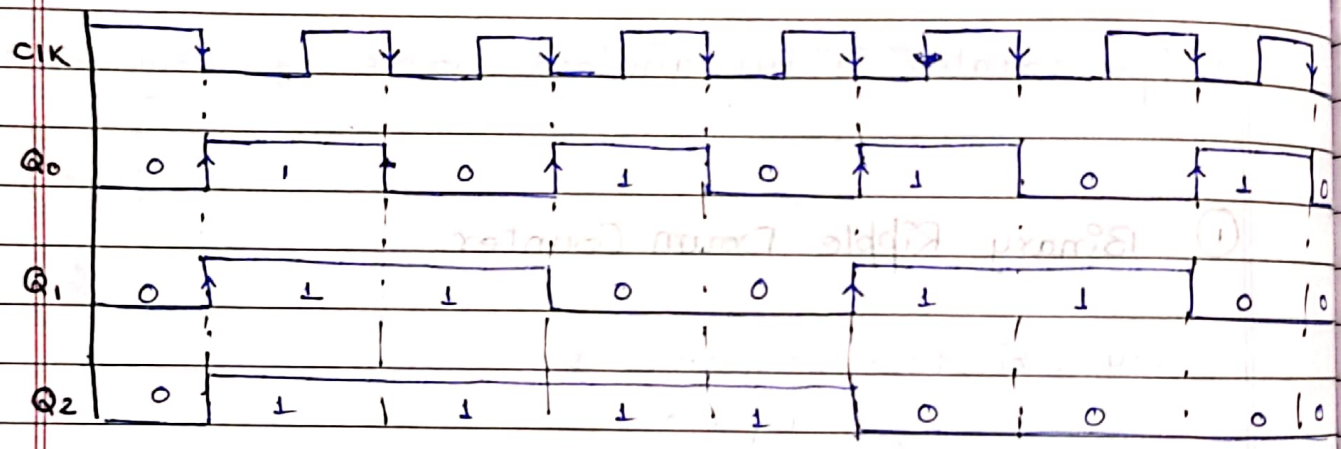


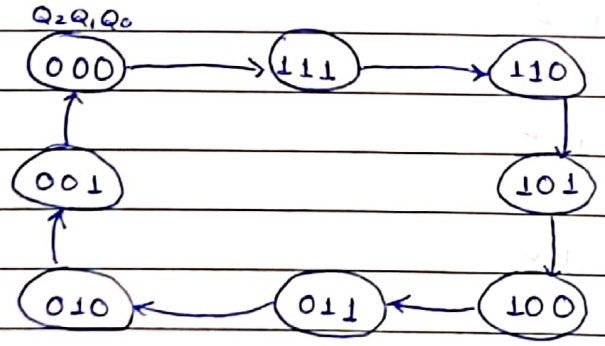
+ve edge trigerred.



-ve edge trigerred.

{ when LSB goes from 0 to 1, $\overline{LSB}$ goes from 1 to 0, hence trigerring middle bit }





∴ 3 bit binary down ripple counter.

Say if each ff had prop. delay of t_p, then p.d. for

Q_0 : t_p

Q_1 : $2t_p$

Q_2 : $3t_p$

⋮

Q_{n-1} : $(n)t_p$.

If n is sufficiently large, then it may happen so:

$(n)t_{PD} > t_{CP}$

∴ n should be such that

$t_{CP} \geq (n)t_{PD}$

∴ for any ripple counter with n ffs (0, 1, ... n-1), it should be such that

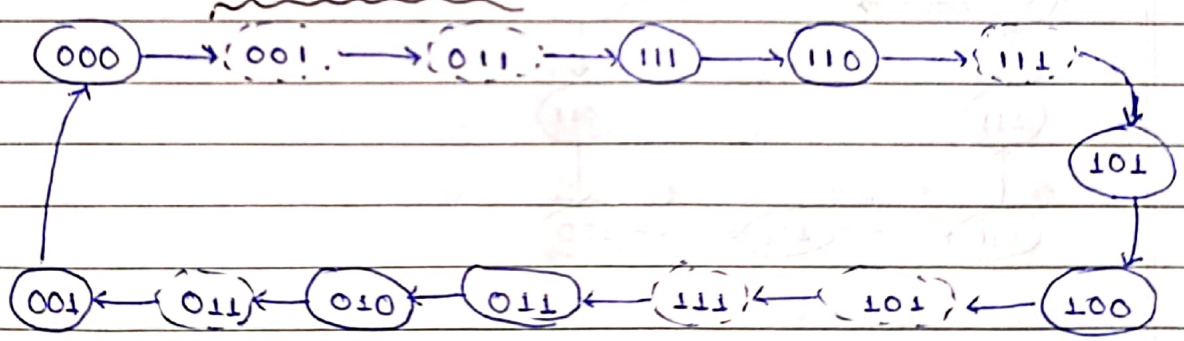
$$t_{CP} \geq n\, t_{PD}$$

$$\Rightarrow \frac{1}{n t_{PD}} \geq \frac{1}{t_{CP}}$$

$$\Rightarrow \frac{1}{n\, t_{PD}} \geq \text{freq.}$$

Transient states (~~LSB chan~~ Q_0 changes faster than Q_1 than Q_2).

ripple effect



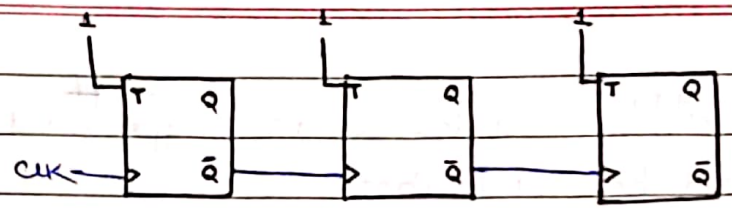
∴ total 6 transient states.
& at max 2 trans. states b/w two states.

② Binary Ripple Up Counter

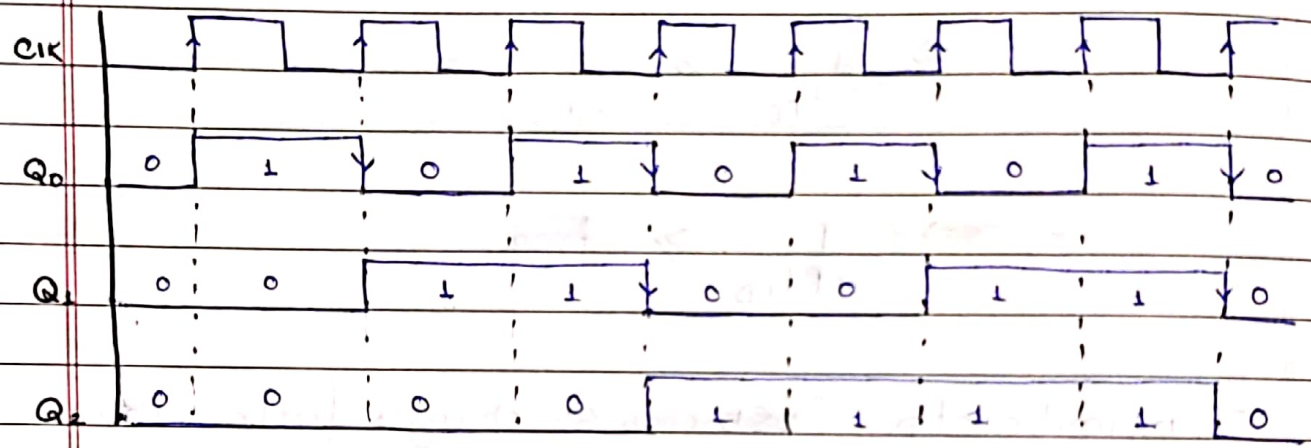
$Q_2 Q_1 Q_0$
000
001
010
011
100
101
110
111

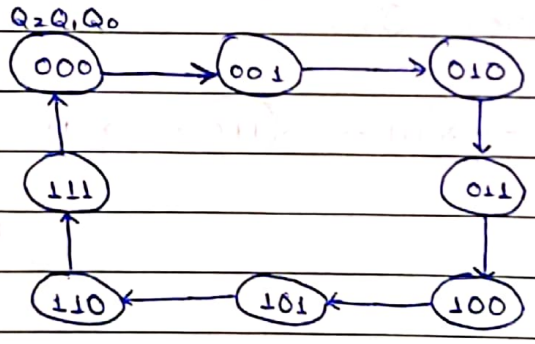
① Q_0 changes every clock.
② Q_1 changes when Q_0 goes from 1 to 0. ↓
③ Q_2 changes when Q_1 goes from 1 to 0. ↓

Impl1:

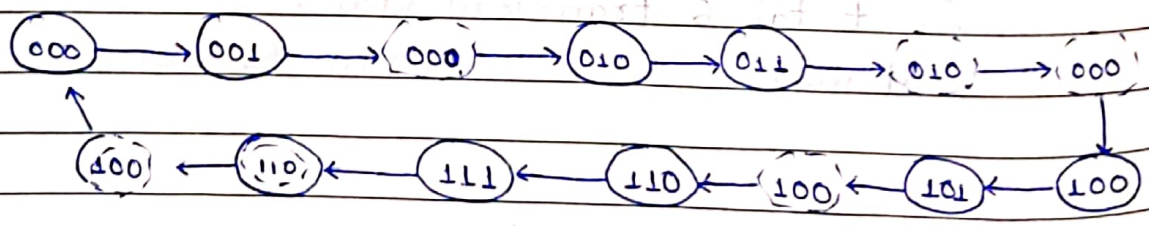


When Q_0 goes 1 to 0, $\bar{Q}$ goes from 0 to 1 & hence Q_1 is activated



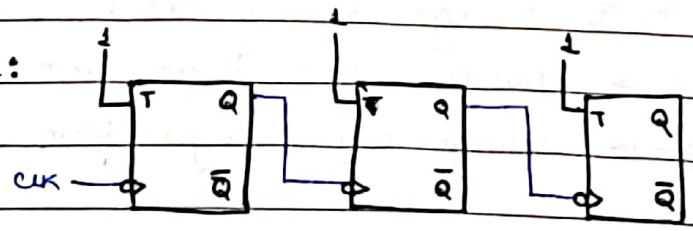


Say there is some prop. delay:



∴ 6 transient states.

Impl2:

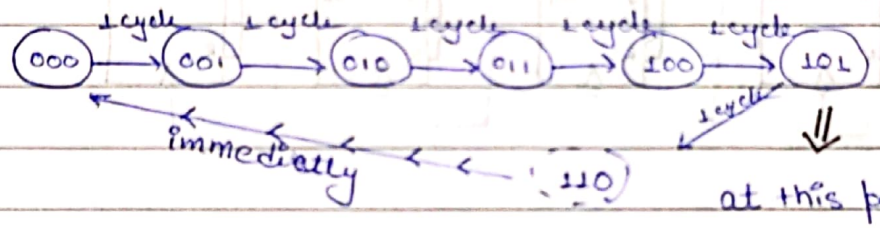


• Mod-M Async Counter.

Every ff has two ~~async~~ input $\overline{PRE}$ & $\overline{CLR}$ (active low).
(don't wait for clock)

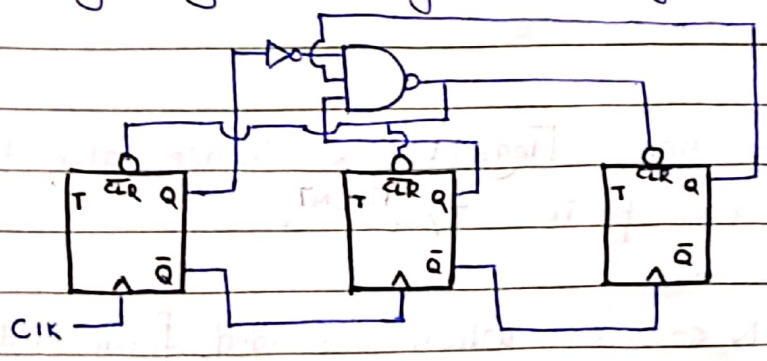
Ex. Sequence : $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$

(i.e. Mod-6 counter)



at this point we want the value to change back to 000 after 1 clock cycle.

Since $\overline{PRE}$ & $\overline{CLR}$ both don't care about clock they can immediately change from 101 to 000, but of course we want 101 for 1 full clock cycle.
So we introduce 110 state. As soon as state goes from 101 to 110 ~~with~~ $\overline{CLR}$ can be made active so that with a very very small glitch it goes to 000 state.



110 → clear

11X (since 111 never occurs)
↓
clear.

This is basically Binary ripple up counter, which as soon as state $(Q_2Q_1Q_0)$ becomes 110 immediately goes to 000.

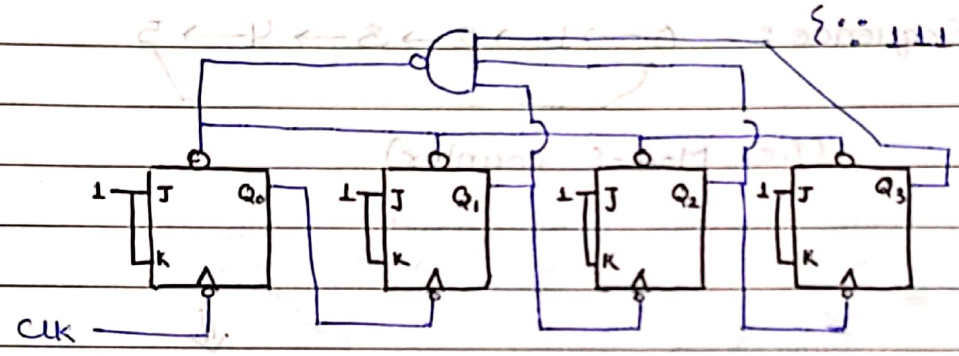
Ex. $0 \rightarrow 1 \rightarrow 2 \ldots \rightarrow 13.$

ffs $= \lceil \log_2 13 \rceil = 4.$

Comb. to connect to clear $Q_3 Q_2 Q_1 Q_0 = 1110$ {i.e. 14}

(or 111X)

{∵ 1111 doesn't occur}



Ex. Decade Counter. $\{0 \rightarrow 1 \rightarrow \ldots \rightarrow 9 \rightarrow 0\}$

ffs $= \lceil \log_2 10 \rceil = 4.$

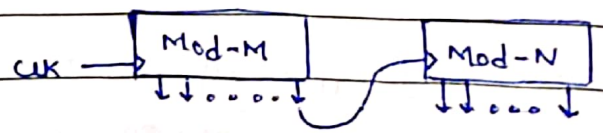
Comb. to connect to clear $Q_3 Q_2 Q_1 Q_0 = 1010.$

$\Rightarrow 101X$ {1011 don't

$\Rightarrow 1X1X$ {∵ 1110 & 1111 don't occur}

- Mod-M counter has $\lceil \log_2 M \rceil$ ffs, hence the freq of clock out of last ff is $f / 2^{\lceil \log_2 M \rceil}$.

- Mod-M & Mod-N counter when cascaded, form Mod-MN counter.



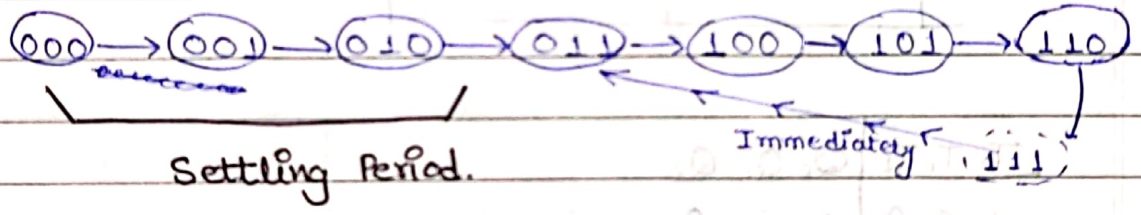
Mod-MN counter.

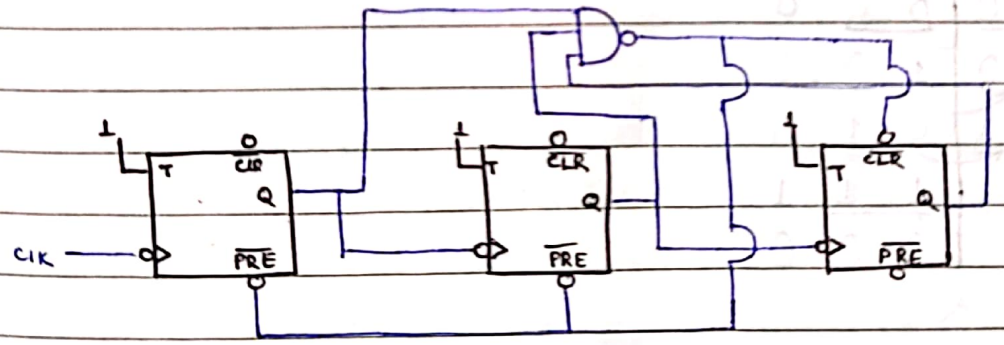
* Counting from anywhere to anywhere.

Ex. 3 → 4 → 5 → 6 → 3

We can use $\overline{PRE}$ & $\overline{CLR}$ inps.

Initially when ckt starts we are always at 000 state, so for first cycle we'll go from 000 to 110, then rather than going from 110 to 000 we'll go to 011.





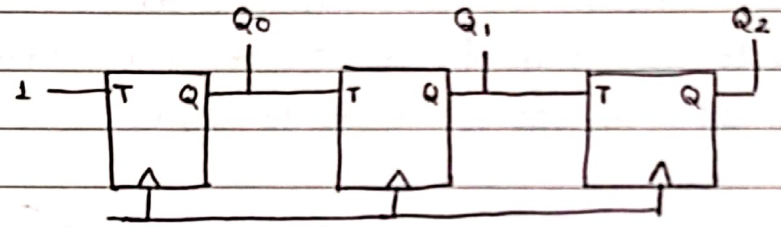
at state (transient) 111, $\overline{CLR}$ Q_2 & $\overline{PRE}$ Q_1 & Q_0.

* Synchronous Counters

All ffs are operated using the same clock.

Ex. Find output after 4 clock cycle.

① Initial state: $Q_2Q_1Q_0$: 000.

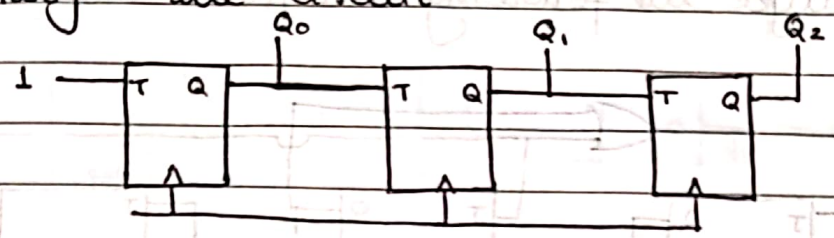


T_0	T_1	T_2	Q_0	Q_1	Q_2
1	0	0	0	0	0
1	1	0	1	0	0
1	0	1	0	1	0
1	1	1	1	1	1
			0	0	0

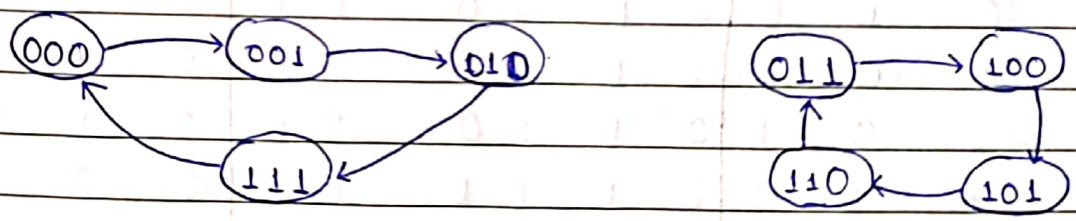
② Initial state: $Q_2Q_1Q_0$ = 101

T_0	T_1	T_2	Q_0	Q_1	Q_2
1	1	0	1	0	1
1	0	1	0	1	1
1	1	1	1	1	0
1	0	0	0	0	1
			1	0	1

Ex. Analyze the circuit



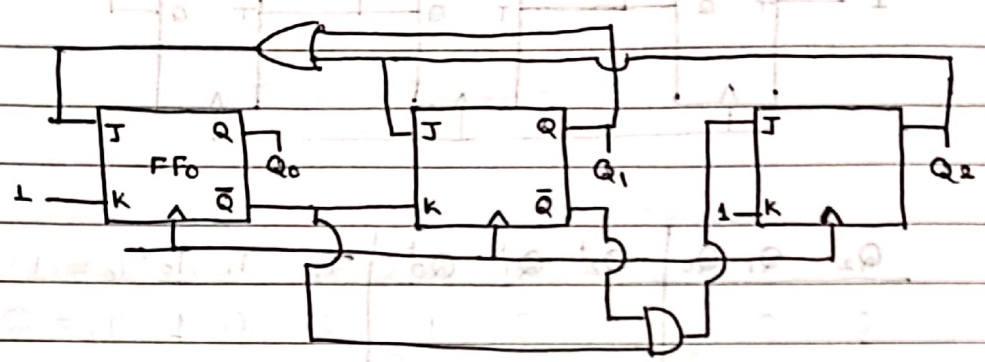
Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+	T_2	T_1	T_0	
0	0	0	0	0	1	0	0	1	$T_0 = 1$
0	0	1	0	1	0	0	1	1	$T_1 = Q_0$
0	1	0	1	1	1	1	0	1	$T_2 = Q_1$
0	1	1	1	0	0	1	1	1	
1	0	0	1	0	1	0	0	1	
1	0	1	1	1	0	0	1	1	
1	1	0	0	1	1	1	0	1	
1	1	1	0	0	0	1	1	1	



Since in a particular sequence maximum four states are there, therefore the counter is Mod-4 counter.

Note: Unless some sequence is specified by default the sequence with max. states is taken for Mod.

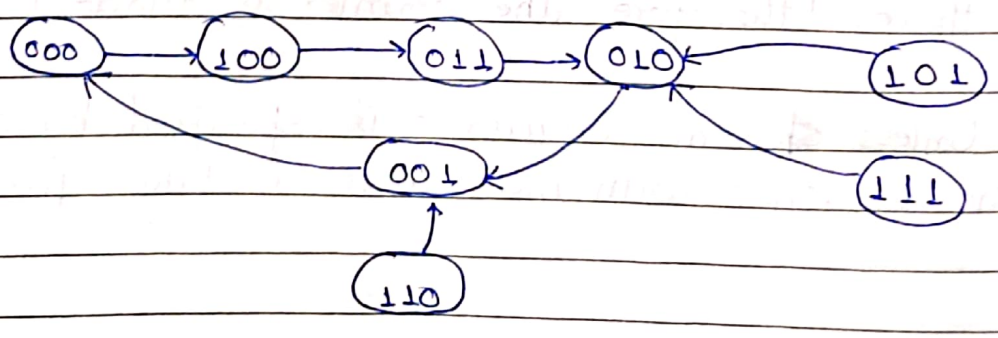
Ex. Analyze the following counter.



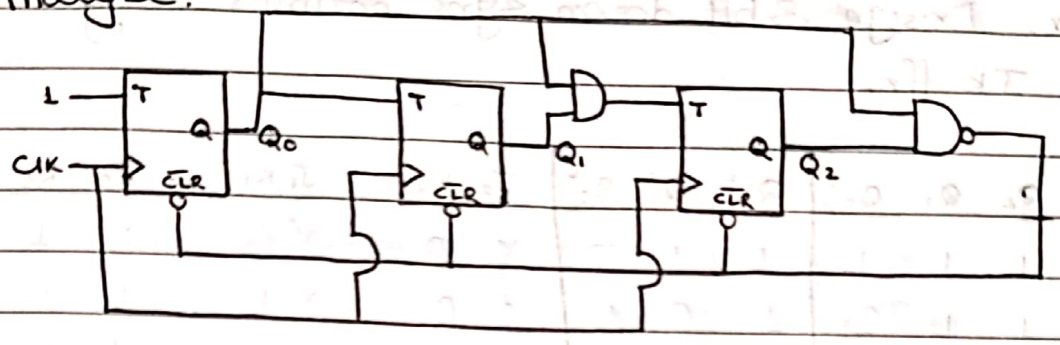
$J_0 = Q_1 + Q_2$ $\quad$ $J_1 = Q_2$ $\quad$ $J_2 = \bar{Q}_0 . \bar{Q}_1 = \overline{Q_0 + Q_1}$

$K_0 = 1$ $\quad$ $K_1 = \bar{Q}_0$ $\quad$ $K_2 = 1$

Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	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



Ex Analyze.



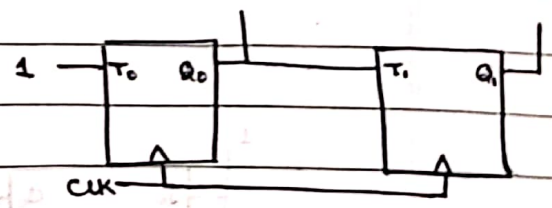
$T_0 = 1$ $\quad T_1 = Q_0$ $\quad T_2 = Q_1 . Q_0$

$$\overline{CLR_0} = \overline{CLR_1} = \overline{CLR_2} = \overline{Q_0 . Q_2}$$

Q_2	Q_1	Q_0	T_2	T_1	T_0	$\overline{CLR}$	Q_2^+	Q_1^+	Q_0^+
0	0	0	0	0	1	1	0	0	1
0	0	1	0	1	1	1	0	1	0
0	1	0	0	0	1	1	0	1	1
0	1	1	1	1	1	1	1	0	0
1	0	0	0	0	1	1	1	0	1
1	0	1	0	1	1	0	0	0	0
1	1	0	0	0	1	1	1	1	1
1	1	1	1	1	1	0	0	0	0

Ex. Design 2 bit binary up sync. counter using T ff.

Q_1	Q_0	Q_1^+	Q_0^+	T_1	T_0
0	0	0	1	0	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	1



$\therefore T_0 = 1 \quad T_1 = Q_0\overline{Q}_1 + Q_0Q_1$

$= Q_0$

Ex. Design 3 bit down sync counter using JK ffs.

Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+	J_2	K_2	J_1	K_1	J_0	K_0
1	1	1	1	1	0	X	0	X	0	X	1
1	1	0	1	0	1	X	0	X	1	1	X
1	0	1	1	0	0	X	0	X	0	X	1
1	0	0	0	1	1	X	1	1	X	1	X
0	1	1	0	1	0	0	X	X	0	X	1
0	1	0	0	0	1	0	X	X	1	1	X
0	0	1	0	0	0	0	X	0	X	X	1
0	0	0	1	1	1	1	X	1	X	1	X

J_2

Q_2 \ Q_1Q_0	00	01	11	10
0	1			
1	X	X	X	X

$J_2 = \bar{Q}_1\bar{Q}_0$

K_2

Q_2 \ Q_1Q_0	00	01	11	10
0	X	X	X	X
1	1			

$K_2 = \bar{Q}_1\bar{Q}_0$

J_1

Q_2 \ Q_1Q_0	00	01	11	10
0	1		X	X
1	1	X	X	X

$J_1 = \bar{Q}_0$

K_1

Q_2 \ Q_1Q_0	00	01	11	10
0	X	X		1
1	X			1

$K_1 = \bar{Q}_0$

J_0

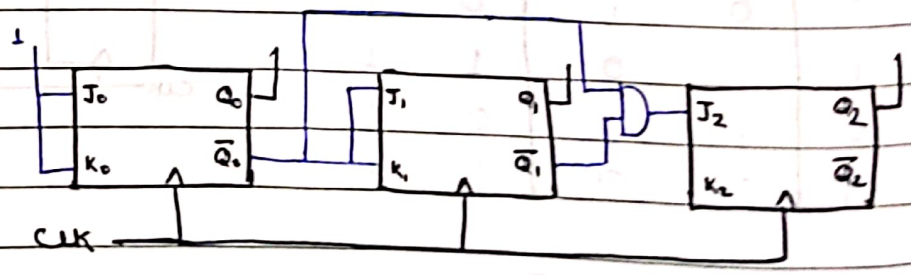
Q_2 \ Q_1Q_0	00	01	11	10
0	1	X	X	1
1	1	X	X	1

$J_0 = 1$

K_0

Q_2 \ Q_1Q_0	00	01	11	10
0	X	1	1	X
1	X	1	1	X

$K_0 = 1$



Sync Counters	Async Counters
① All ffs have same clock.	① Output of one ff acts as clock of other.
② Even with PD, since all ff change ~~at~~ state at same time, no transient states.	② When PD, since ffs change state at diff. ~~tim~~ delays after clock, so transient states.
③ Since t_{PD} is same for all ffs: $t_{CP} \geq t_{PD}$	③ t_{PD} of furthest (MSB) ff depends on previous ff, so n^{th} ff has $n \times t_{PD}$ delay, so, $t_{CP} \geq n\, t_{PD}$

Ex. Design the following counter: $000 \rightarrow 100 \rightarrow 111 \rightarrow 010 \rightarrow 011$ using T ffs.

{ Treat unused states as don't cares }

Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+	T_2	T_1	T_0
0	0	0	1	0	0	1	0	0
0	0	1	X	X	X	X	X	X
0	1	0	0	1	1	0	0	1
0	1	1	0	0	0	0	1	1
1	0	0	1	1	1	0	1	1
1	0	1	X	X	X	X	X	X
1	1	0	X	X	X	X	X	X
1	1	1	0	1	0	1	0	1

Q_2 \ Q_1Q_0	00	01	11	10
0	1	X		
1		X	1	X

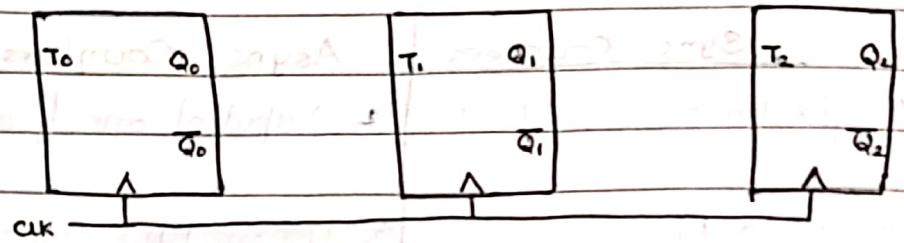
$T_2 = \bar{Q}_2\bar{Q}_1 + Q_2Q_0$

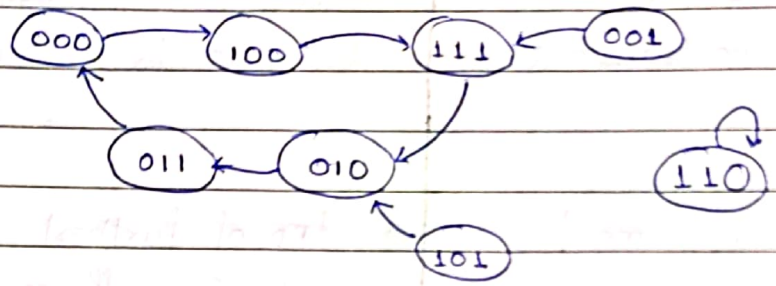
Q_2 \ Q_1Q_0	00	01	11	10
0		X	1	
1	1	X		X

$T_1 = \bar{Q}_2Q_0 + Q_2\bar{Q}_1$

Q_2 \ Q_1Q_0	00	01	11	10
0		X	1	1
1	1	X		X

$T_0 = Q_2\bar{Q}_1 + \bar{Q}_2Q_1$





* Ring & Johnson Counter

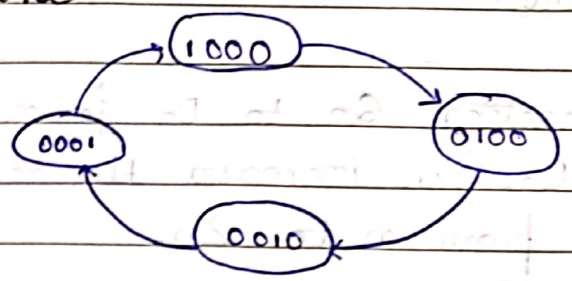
- During an inst^n execution cycle (FDE cycle), multiple stages w/ different H/W exist, say: Fetch, Decode, Execute, M/M Access, WB. So, for a particular inst^n at a time only a single set of h/w should be activated to indicate & run current stage.

We can depict it as follows:

CLK	F	D	Ex	MA	WB
0	1	0	0	0	0
1	0	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	0
4	0	0	0	0	1

So, at a time only one of the stages is active. This type of counting can be achieved using Ring & Johnson counter.

- Ring Counter :

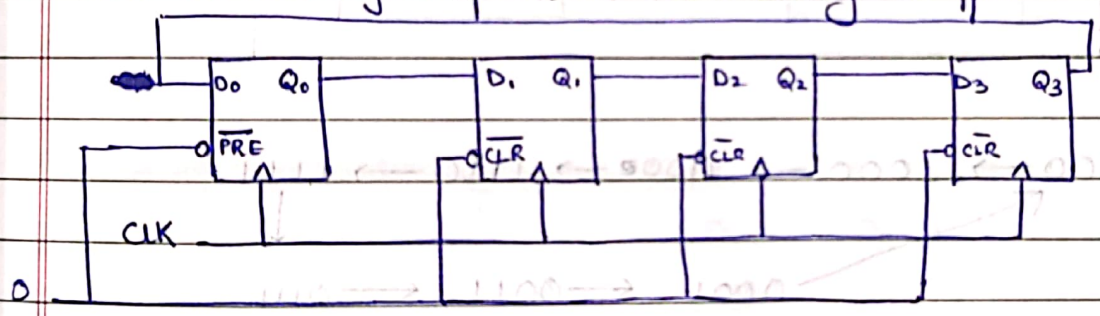


Initial Cond: 1000

At a time ~~only~~ exactly one bit is '1' & if in a state of nbit Ring Counter i^{th} bit is 1, then in the next state $(i+1)\%n$ bit would be '1'.

Implementation 1 :

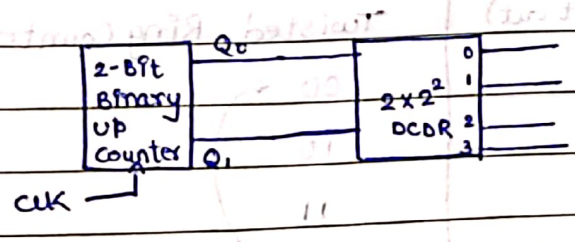
We can easily implement this using D ffs.



Give 0 & 1 resp to $\overline{PRE}$ & $\overline{CR}$ of FF_0 & to rest give 1 & 0 resp. as $\overline{PRE}$ & $\overline{CLR}$, to initialize.

For n-bit Ring counter, we need n FFs.

Implementation 2 :



At a time only a single DCDR o/p is 1.

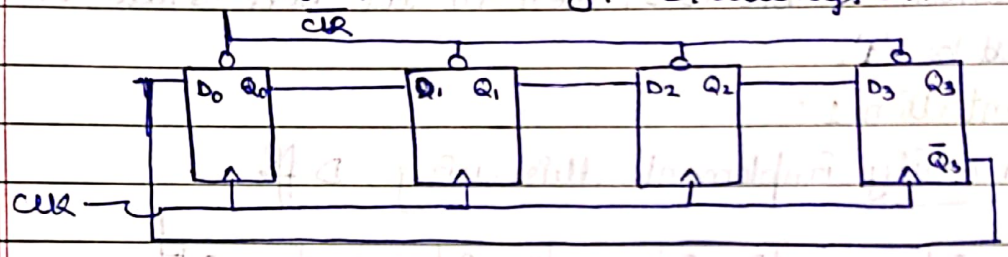
Twisted Ring Counter

Rather than connecting Q_n to D_0 for n ffs, we rather connect $\bar{Q}_n$ to D_0. This increases the no. of states in counting seq. from n to 2n.

Initial Cond^n: 0000

Say 4 bit:

Start signal (0 initially, then always 1).



0000 → 1000 → 1100 → 1110 → 1111 → 0111 → 0011 → 0001 → 0000

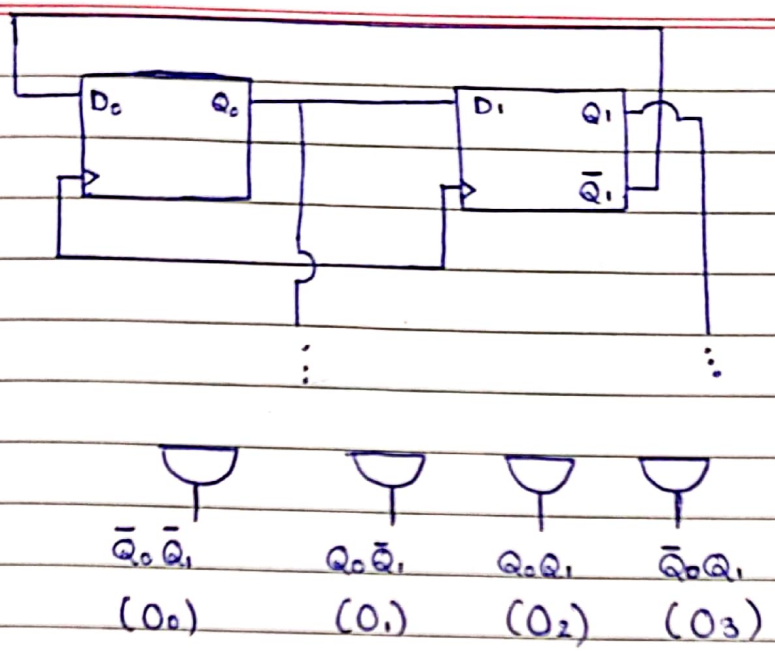
So, with 4ffs we get 8 states.

Johnson Counter

(n bit Twisted Ring Counter) + (2n 2input AND gate)

Say 2 bits:

Johnson Counter (4bit out)	Twisted Ring Counter (2bit out)
1000	00
0100	10
0010	11
0010	01



Q_0	Q_1	O_0	O_1	O_2	O_3	Q_0^+	Q_1^+
0	0	1	0	0	0	1	0
1	0	0	1	0	0	1	1
1	1	0	0	1	0	0	1
0	1	0	0	0	1	0	0