DIGITAL LOGIC

Ankit Sir
Uncode GATE Computer Science
GATE AIR 1 2014, 2018
58 lectures

STARTING DATE 16/07/2023

ANKIT Sir

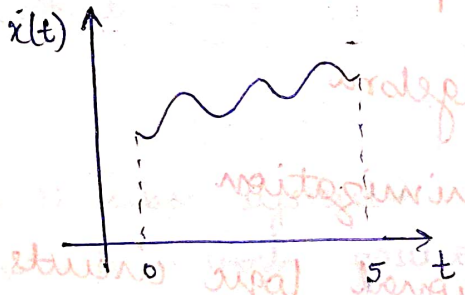
Digital logic –

- Digital has been derived from the word digit
- It represents a system with finite number of possibilities

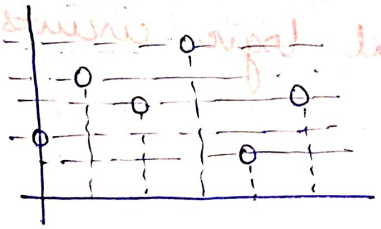
Analog v/s Digital –

* all real world signals are analog.

→ Analog signals are continuous while digital signals are discrete.

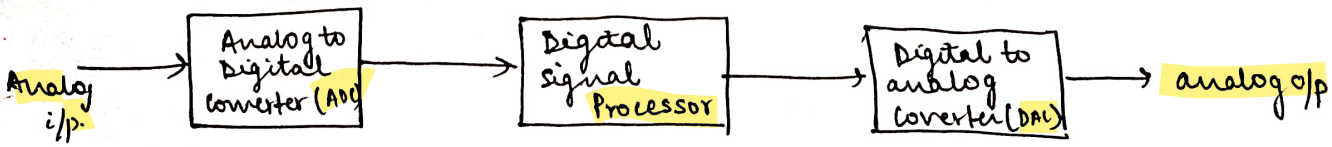


every value is possible.
∴ analog (continuous)



5 possible values of function
∴ digital.

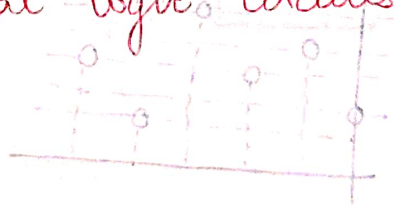
→ Analog = infinite possibilities
Digital = finite possibilities

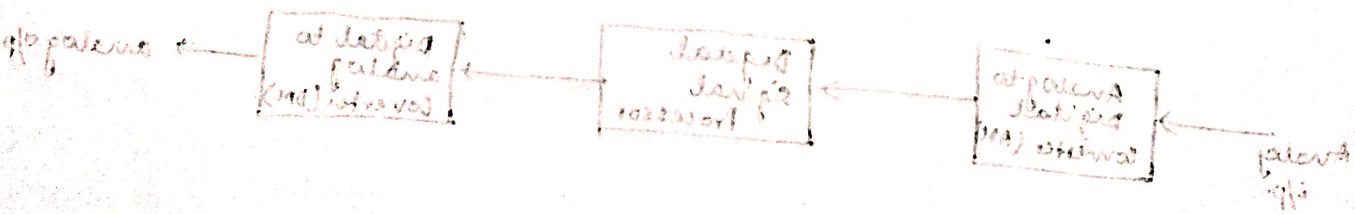


और भी गम हैं जमाने में।

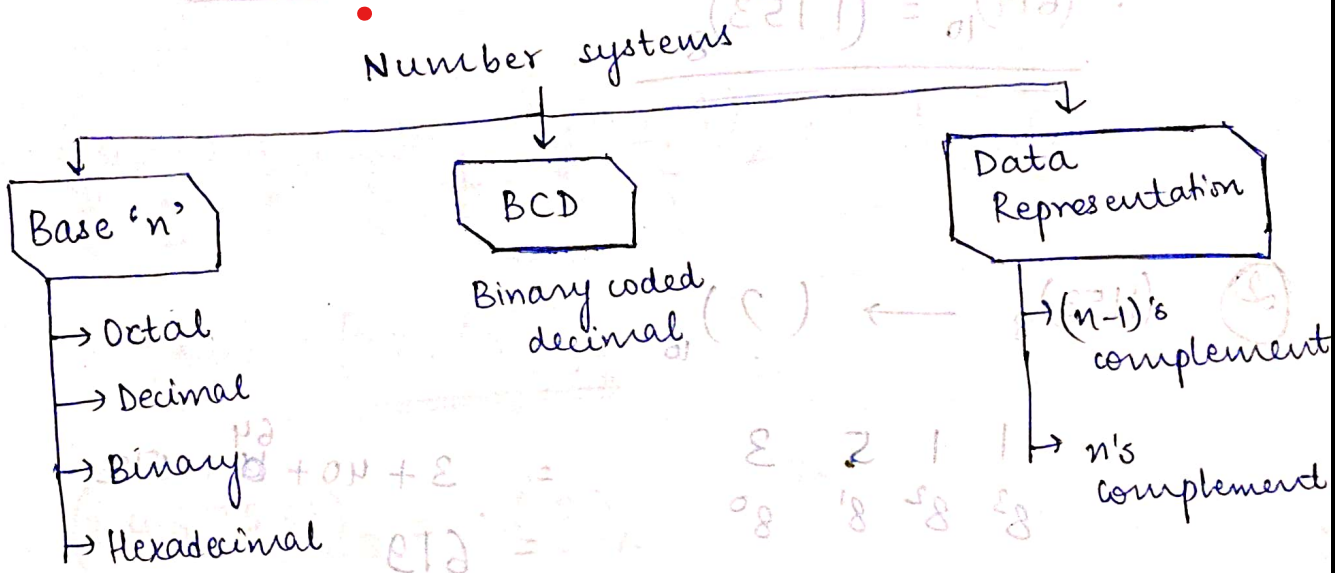
Syllabus

- Number system
- Computer Arithmetic (fixed and floating point)
- Boolean Algebra
- Logic Minimization
- Combinational logic circuits
- Sequential logic circuits





NUMBER SYSTEMS



Positional Number system –
system in which position of a digit determines the weightage of the digit in the number.

Base :- number of unique digits in a no. system.

Base	Terminology	Range
2	Binary	0, 1
3	Ternary	0, 1, 2
8	Octal	0 – 7
10	Decimal	0 – 9
16	Hexadecimal	0 – 9, A – F

① $(619)_{10} \longrightarrow (?)_8$

8	619	
8	77	— 3
8	9	— 5
	1	— 1 ↑

$\therefore (619)_{10} = (1153)_8$

② $(1153)_8 \longrightarrow (?)_{10}$

1	1	5	3
8^3	8^2	8^1	8^0

$= 3 + 40 + 64 + 512$

$= 619$

$\therefore (1153)_8 = (619)_{10}$

③ $(1001)_2 = (?)_3$

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

$= 1 + 8 = 9$

$\therefore (1001)_2 = (9)_{10}$

$(9)_{10} = (?)_3$

3	9	
3	3	— 0
	1	— 0 ↑

$\therefore (9)_{10} = (100)_3$

$\therefore (1001)_2 = (100)_3$

④ $(1100101)_2 = (?)_8$

$\underline{001}\ \underline{100}\ \underline{101}$

$\downarrow \quad \downarrow \quad \downarrow$

$1 \quad 4 \quad 5$

$\therefore (1100101)_2 = (145)_8$

⑤ $(1001000111001101000 1)_2 = (?)_8$

$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow$

$2 \quad 2 \quad 1 \quad 6 \quad 3 \quad 2 \quad 1$

$\therefore$ Ans = $(2216321)_8$

⑥ $(2467531)_8 = (?)_2$

$(10\ 100\ 110\ 111\ 101\ 011\ 001)_2$

⑦ $(7542106)_8 = (?)_2$

$(111\ 101\ 100\ 010\ 001\ 000\ 110)_2$

⑧ $(0100\ 1011\ 1011\ 0001\ 0010\ 1100)_2 = (?)_{16}$

$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow$

$4 \quad B \quad B \quad 1 \quad 2 \quad C$

$\therefore$ Ans → $(4BB12C)_{16}$

⑨ $(4BC\ DA975)_{16} = (?)_2$

$(0100\ 1011\ 1100\ 1101\ 1010\ 1001\ 0111\ 0101)_2$

Ques

If $(2.3)_{base4} + (1.2)_{base4} = (y)_{base4}$.
What is value of y?

$(2.3)_4 = 2 + 0.25 \times 3 = 2.75_{10}$

$(1.2)_4 = 1 + 0.25 * 2 = 1.5_{10}$

$2.75 + 1.5 = (4.25)_{10}$

$0.25 \times 4 = 1$ ↑

4 | 4
1 – 0 ↑

∴ Ans – $(10.1)_4$

$$\begin{array}{r} \overset{1}{2.3} \\ 1.2 \\ \hline 5 \end{array}$$

10.1

4 | 5
1 – 1

4 | 4
1 – 0

Ques

Given: $(135)_x + (144)_x = (323)_x$
What is value of x?

$(135)_x = (x^2 + 3x + 5)_{10}$

$(323)_x = (3x^2 + 2x + 3)_{10}$

$(144)_x = (x^2 + 4x + 4)_{10}$

$x = 2$ or $x = 3$

$x = 6$

$x^2 + 3x + 5 + x^2 + 4x + 4 = 3x^2 + 2x + 3$

$2x^2 + 7x + 9 = 3x^2 + 2x + 3$ $\quad (x+1)(x-6) = 0$

$x^2 + x - 6x - 6 = 0$ $\quad x^2 - 5x - 6 = 0$ $\quad$ ~~$x^2 - 2x - 3x - 6 = 0$~~

$x(x+1) - 6(x+1) = 0$ $\quad$ ~~$x(x-2) - 3(x-2) = 0$~~ $\quad (x-2)(x-3) = 0$

(10) $(43)_x = (y3)_8$

$\Rightarrow 4x+3 = 8y+3 \quad \Rightarrow 4x = 8y \quad \Rightarrow x = 2y$

$0 \le y \le 7 \quad \therefore \quad 0 \le x \le 14$

No. of possible solutions = 8

$x \ge 5 \qquad y \ge 3$

possible soln = $y = 3, 4, 5, 6, 7$

5 soln

(11) $(37.625)_{10} = (?)_2$

2	37
2	18 – 1
2	9 – 0
2	4 – 1
2	2 – 0
2	1 – 0

$0.625 \times 2 = 1.25 \quad 1$

$0.25 \times 2 = 0.5 \quad 0$

$0.5 \times 2 = 1.0 \quad 1$

$\therefore$ Ans :- $(100101.101)_2$

(12) $(37.625)_{10} = (?)_8$

8	37
	4 – 5

$0.625 \times 8 = 5.000 \quad 5$

Ans – 45.5

(13) $(78.025)_{10} = (\ ?\ )_{16}$

~~78 × 16 = 124~~

16 | 78
4 — DE ↑

$0.025 \times 16 = 0.4$ 0
$0.4 \times 16 = 6.4$ 6 ↓
$0.4 \times 16 = 6.4$ 6

Ans → $(4E.066\ldots)_{16}$

(14) $(0110110.1010)_2 \longrightarrow (x)_{10}$

0 1 1 0 1 1 0
$2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

$32+16+4+2 = 54$

1 0 1 0
$2^{-1}\ 2^{-2}\ 2^{-3}\ 2^{-4}$

$= \frac{1}{2} + \frac{1}{8}$

$0.5 + 0.125$
$= 0.625$

Ans → 54.625

(15) $(7632.15)_8 = (\ ?\ )_{10}$
$8^3\ 8^2\ 8^1\ 8^0\ \ 8^{-1}\ 8^{-2}$ 8

$= 2 + 24 + 64 \times 6 + 512 \times 7 + \frac{1}{8} + \frac{5}{64}$

$= 2 + 24 + 384 + 3584 + \frac{13}{64}$

$= 3994 + 0.203125$

Ans → $(3994.203125)_{10}$

64 | 130 → 0.203125
128
200
192
80
64
160
128
32

(16) $(DC4F.16)_{16} = (?)_{10}$

256 × 16 = 4096, 4096 × 12 = 49152, 49152 + 4096 = 53248

256 × 12 = 3072, 3072 + 256 = 3328

2^{12}

D	C	4	F
16^3	16^2	16^1	16^0

$= 15 + 16 \times 4 + 256 * 12 + 16^3 * 13$

$= 15 + 64 + 3072 + 53248$

$= 56399$

53248 + 3072 + 64 + 15 = 56399

$0.1\ 6$ (16^{-1}, 16^{-2})

$= \frac{1}{16} + \frac{6}{256} = \frac{16+6}{256} = \frac{22}{256} = 0.086$

$\therefore$ Ans $\rightarrow (56399.086)_{10}$

(17) $(01110011.101100)_2 = (?)_8$

01 | 110 | 011 . 101 | 100

1 | 6 | 3 | 5 | 4

Ans – $(163.54)_8$

Binary Coded Decimal

Express each ~~bit~~ digit as a combination of 4 bits

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

$$(74)_{10} = (0111\ 0100)_{BCD}$$

Weighted Codes / Non Weighted Codes

⊙ Weighted Codes :-

binary codes which obey positional weight princi
Each position has specific weight.

Ex → (8421) code.

b_3	b_2	b_1	b_0
2^3	2^2	2^1	2^0
8	4	2	1

⊙ Non weighted codes :-

In these types of binary codes, positional weights are not assigned.

Ex → Excess 3 code, gray code.

Excess 3 Code –

also called XS-3

Decimal number → 8421 BCD → (Add 0011) → XS-3.

0 → 3 → 0011
5 → 8 → 1000
9 → 12 → 1100

6 sequences are wasted!

0000 1101
0001 1110
0010 1111

Ques Decimal equivalent of XS-3 no. –

1100 1010 0011 . 0111 0101

12 10 3 7 5 ← BCD

9 7 0 4 2 ← Decimal

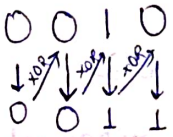
Ans → 970.42

Gray code

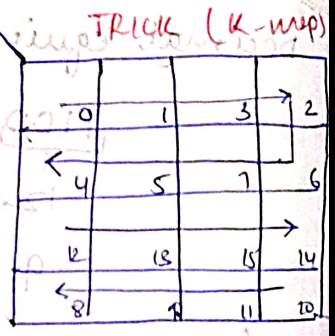
minimum error code

- non weighted code (not arithematic codes)
- no specific weight assigned to a bit position
- only one bit will change each time the decimal number is incremented.
- Also known as
  - unit distance code
  - cyclic code
- arithematic operations cannot be performed!

Decimal	BCD	Gray	Gray code.
0	0000	0	0000
1	0001	1	0001
2	0010	3	0011
3	0011	2	0010
4	0100	6	0110
5	0101	7	0111
6	0110	5	0101
7	0111	4	0100
8	1000	12	1100
9	1001	13	1101



∴ 2 in binary = 0011 in gray code.



Reflective codes

- Code is reflective when the code is self complementing
- In other words,
  code for 9 = complement of 0.
  8 for 1 7 for 2 6 for 3 5 for 4

Decimal	BCD	Excess 3 (BCD + 0011)
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

Example- 2421 BCD, 5421 BCD, Excess 3 code

Sequential codes –

each succeeding code is one binary number greater than the preceding code.

Ex → Excess 3, 8421 BCD.

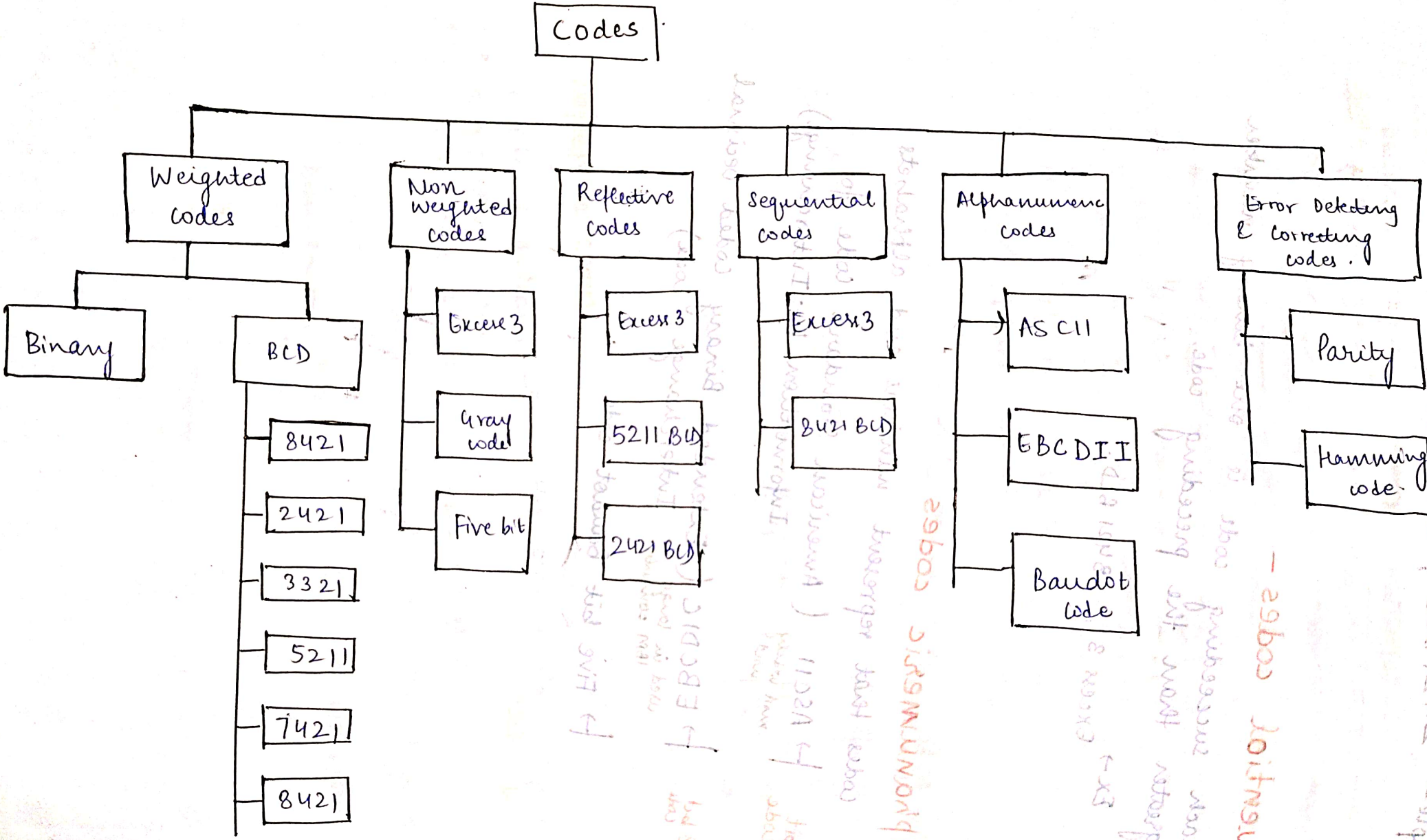
Alphanumeric codes

codes that represent numbers and alphabets

7 bit code → ASCII (most widely used) (American Standard Code for Information Interchange)

8 bit code → EBCDIC (used in large IBM computers) (Extended Binary Coded Decimal Interchange Code)

→ Five bit Baudot code



Data Representation

- Unsigned Magnitude Representation
- Signed Magnitude Representation.
- 1's complement Representation
- 2's complement Representation.

1. Unsigned Magnitude Representation:

Only +ve numbers can be represented.

N bits $\Rightarrow$ 0 to $(2^N - 1)$ numbers.

2. Signed Magnitude Representation:

MSB is reserved for sign.

0 $\rightarrow$ +ve 1 $\rightarrow$ -ve.

Both +ve and -ve nos can be represented.

N bits we can represent $-(2^{N-1} - 1)$ to $(2^{N-1} - 1)$

Representation	Range
Unsigned Magnitude Representation.	0 to $2^N - 1$
Signed Magnitude Representation	$-(2^{N-1} - 1)$ to $2^{N-1} - 1$
1's complement	$-(2^{N-1} - 1)$ to $2^{N-1} - 1$
2's complement.	-2^{N-1} to $2^{N-1} - 1$

Overflow –

- ⊙ occurs when more bits are required to store the result.
- ① Overflow cannot occur in addition (or subtraction) if operands have opposite signs.

Storing Real Numbers

2 major approaches of storing real numbers –

- ↦ Fixed point representation
- ↦ Floating point representation

1. Fixed point Representation –

fixed no. of bits for integer part and fractional part

Ex → IIII · FFFF

Min value = .0000 · 0001

Max value = 1111 · 1111

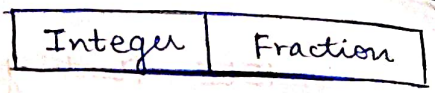
3 parts

- ↦ Sign field
- ↦ Magnitude field
- ↦ fractional field

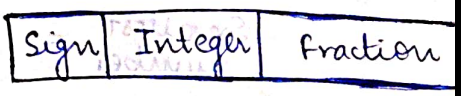
$-(45.5)$

1 101101 100000

Unsigned Fixed Point Representation



Signed Fixed Point Representation



Ques Assume number is using 32 bit format which reserves
1 bit for sign
16 bits for integer part.
15 bits for fractional part.
Find the value in decimal of the min & max that can be stored.

Min value = $1\ 000\ldots0\ \underset{2^{-1}\,2^{-2}\,2^{-3}}{000\ldots01}_{2^{-15}} = -2^{-15}$

Max value = $0\ 111\ldots1\ \ 111\ldots11$

$= (2^{16}-1) + \frac{1}{2}\,\frac{1-\left(\frac{1}{2}\right)^{15}}{1-\frac{1}{2}}$

$= 2^{16} - 1 + 1 - \left(\frac{1}{2}\right)^{15} = 2^{16} - \frac{1}{2^{15}}$

2. Floating point Representation

does not have specific number of bits for the integer part and fractional part.

Floating point representation has 3 parts
- Sign bit
- Mantissa
- Exponent.

IEEE Floating Point Representation

Sign	Exp.	Mantissa
←1→	←8→	←23→

Smallest number = 1 000.....0 00.....1

Exponent is in excess 127 notation

$\therefore$ Smallest no = $-1 * 2^{(1-127)} = -1 * 2^{-126}$

$= 2^{-126}$

Largest Number = 0 1 1 1 ..... 1 1 1 ..... 1 0

exponent = 254 − 127 = 127

Largest number = $(2^{24} - 1) * 2^{127}$

~~BOOLEAN ALGEBRA~~

LOGIC GATES

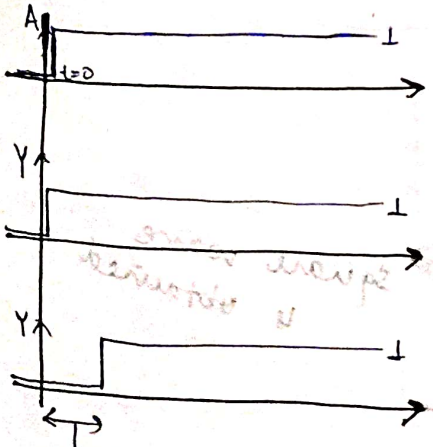
Logic gates -
Basic building block of a digital circuit which is used to make all logic decisions.

ⓐ NOT gate
ⓑ BUFFER
ⓒ AND
ⓓ NAND
ⓔ OR
ⓕ NOR
ⓖ XOR
ⓗ XNOR

① Buffer

A buffer has the same output as the input but it increases signal strength so that it can travel for longer distance.

Relation: $Y = A$



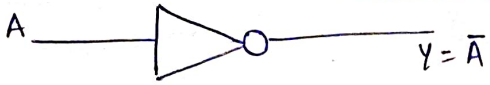
t_{pd} = propagation delay
time taken by signal to go from

A	Y
0	0
1	1

← Truth table
↳ tabular representation which tells us when o/p is 1
it gives value of o/p for all possible combination of inputs.

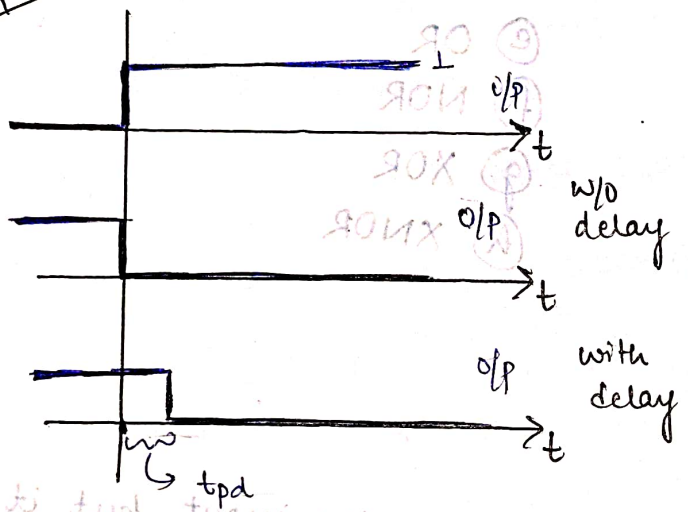
(2) Inverter (NOT)

- It negates all logic
- Complements the logic



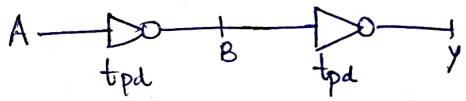
In any logic circuit, bubble implies inversion.

Waveform



A	Y
0	1
1	0

Cascading of inverters

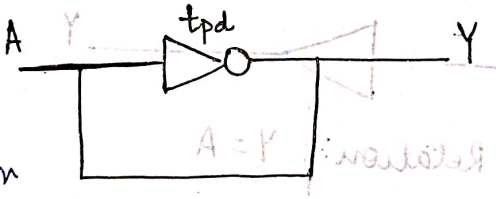


$B = \bar{A}$

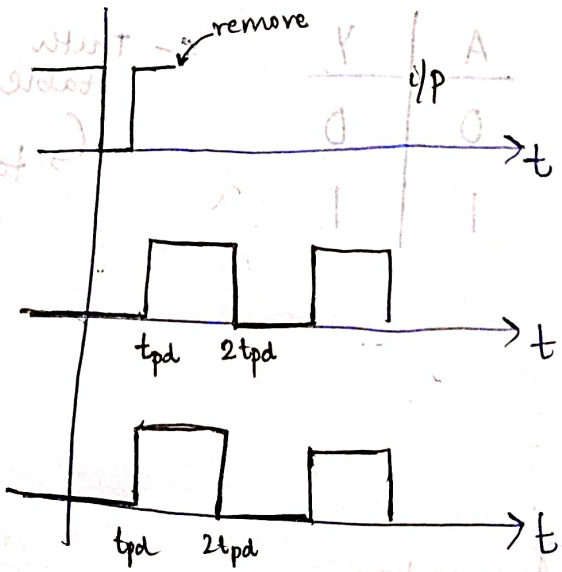
$Y = \bar{B} = \bar{\bar{A}} = A$ (1)

2 inverters in cascade behave like a buffer with delay = 2 tpd

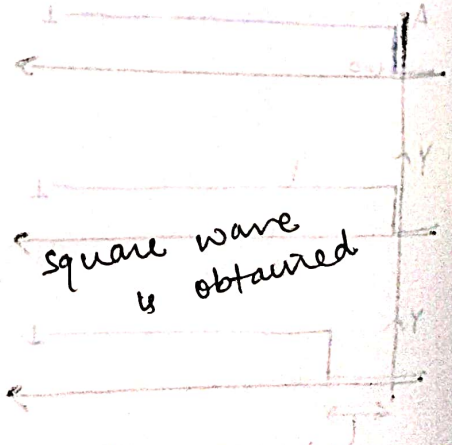
NOT gate with feedback.



* apply trigger input A i.e. an input for short duration



square wave is obtained



* if even number of NOT gates are connected in a loop, the output remains stable at 0 or 1.

* if odd number of NOT gates are connected in a loop, o/p becomes square wave.

↳ period of square wave = $2n t_{pd}$

n = no. of NOT gates

(3) AND gate

Both i/p or conditions must be true in order for o/p to be true or high.

$Y = A \cdot B$

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

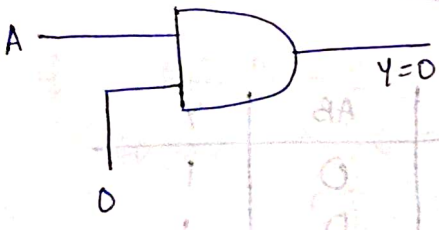
* follows commutative law

$A \cdot B = B \cdot A$

* follows assosiative law

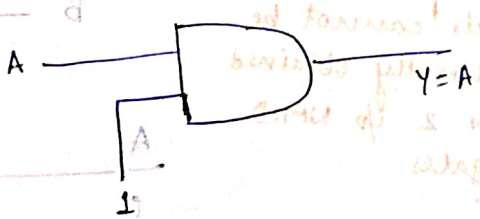
$(A \cdot B) \cdot C = A \cdot (B \cdot C)$

Enable and Disable AND gate.



The i/p A is not passed to o/p

so, AND gate is disabled.



The i/p directly passes to o/p

↳ enabled AND gate

↳ acts as a buffer.

(4) OR gate

- atleast one of the inputs must be high in order for o/p to be high.

A	B	Y = A+B
0	0	0
0	1	1
1	0	1
1	1	1

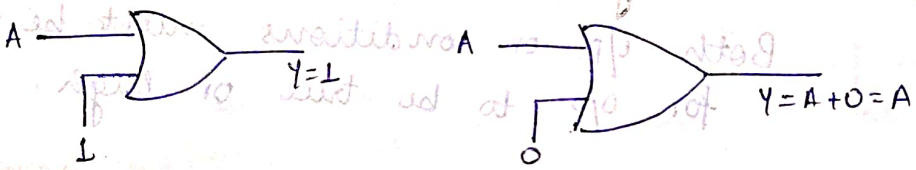
* follows commutative law

$A+B = B+A$

* follows associative law

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

Enable and Disable OR gate.



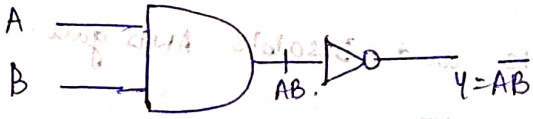
disabled OR gate.

behaves as buffer enabled OR gate.

(5) NAND gate

combination of inverter and AND gate.

NAND = AND → NOT



* 3 input NAND gate cannot be directly obtained for 2 i/p NAND gates.

A	B	AB	Y
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

* follows commutative law

$\overline{AB} = \overline{BA}$

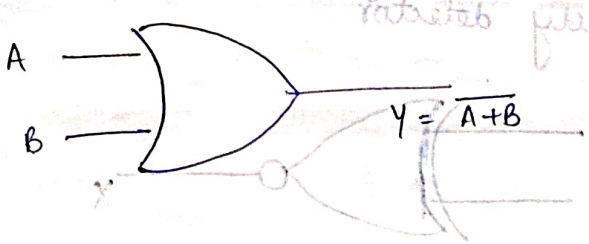
* does not follows associative law

$\overline{\overline{AB}C} \neq \overline{A\overline{BC}}$

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

⑥ NOR gate

combination of OR gate and NOT gate.



*3 i/p NOR gate cannot be constructed directly from 2 i/p NOR gates.

A	B	A+B	Y = $\overline{A+B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

* follows commutative law

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

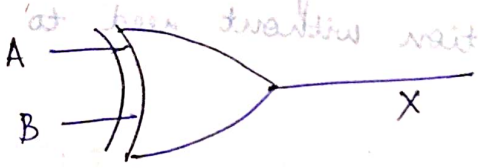
* does not follow associative law

$\overline{\overline{A+B}+C} \neq \overline{A+\overline{B+C}}$

⑦ XOR gate

Exclusive - OR

called as inequality detector.



odd no. of 1s in i/p → o/p is 1

even no. of 1s in i/p → o/p is 0

A	B	X = A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

* follows commutative law

$A \oplus B = B \oplus A$

* follows associative law

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

if $A \oplus B = C$

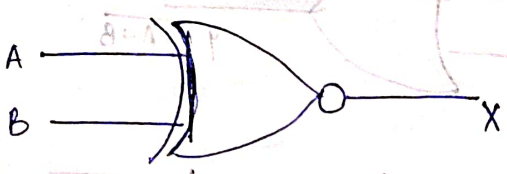
then, $A \oplus C = B$

$B \oplus C = A$

⑧ XNOR gate

Exclusive NOR gate represented by ⊙

equality detector.



A	B	X = A⊙B
0	0	1
0	1	0
1	0	0
1	1	1

* follows associative law

$A \odot (B \odot C) = (A \odot B) \odot C$

* follows commutative law.

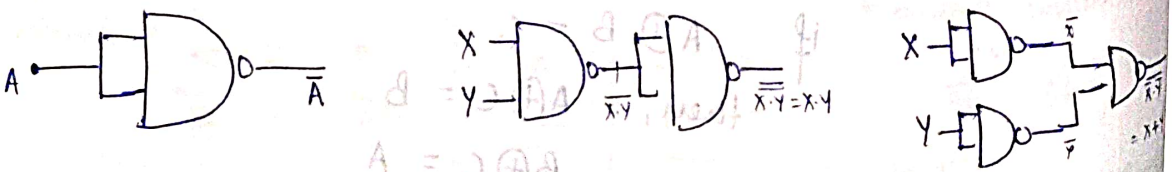
$A \odot B = B \odot A$

Universal gates

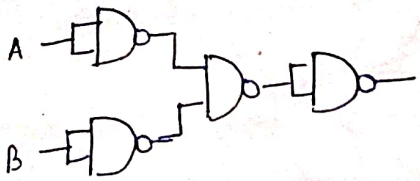
* universal gate is a gate which can implement any boolean function without need to use any other gate type.

* AND & OR gates are designed from NAND and NOR gates (not other way round!!)

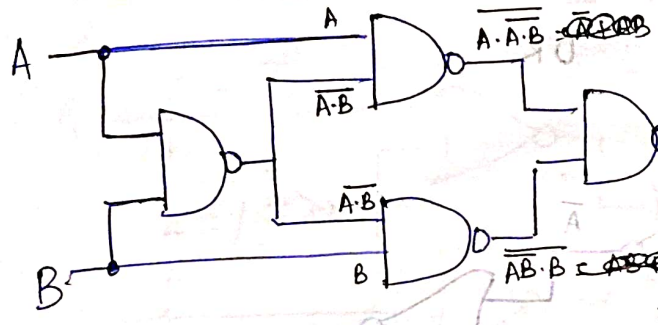
NAND gate as ~~inverter~~ universal gate –



NOT AND OR



NOR gate.



XOR gate.
(4 NAND gates)

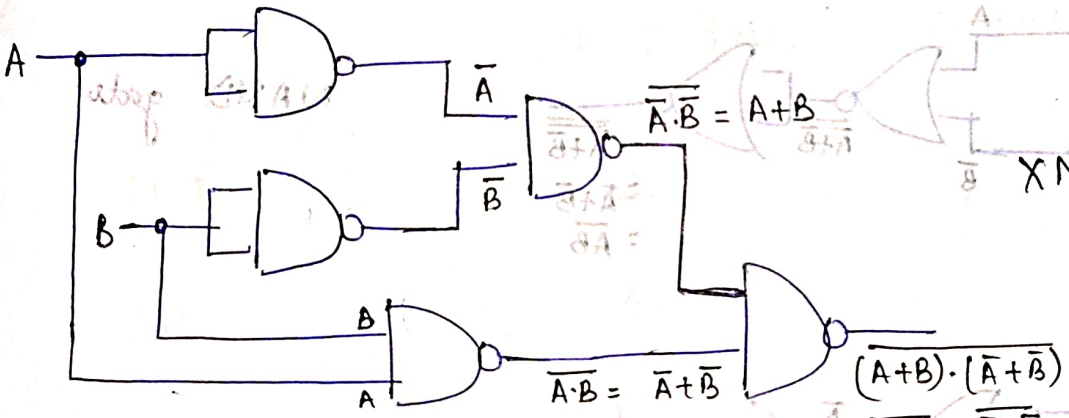
$A \cdot \overline{A \cdot B} \cdot \overline{AB \cdot B}$

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

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

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

$\because (\overline{A}+\overline{B})(A+B) = A \oplus B$



XNOR gate.

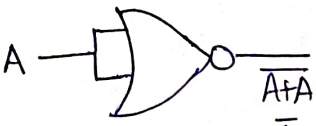
$\overline{(A+B) \cdot (\overline{A}+\overline{B})}$

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

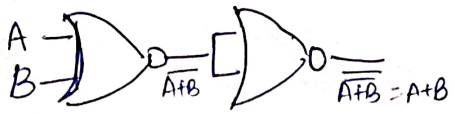
$= \overline{A}\overline{B} + AB$

$= A \odot B$

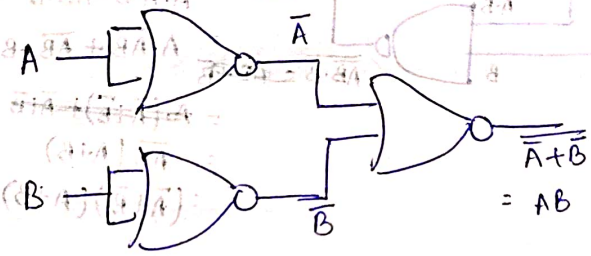
NOR gate as universal gate.



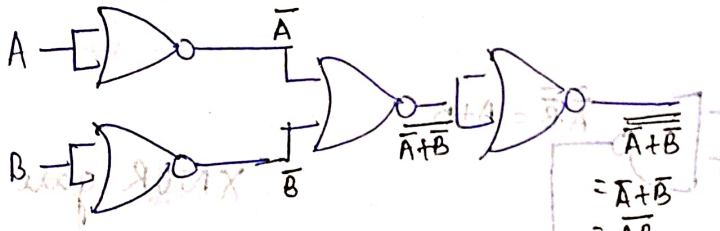
NOT



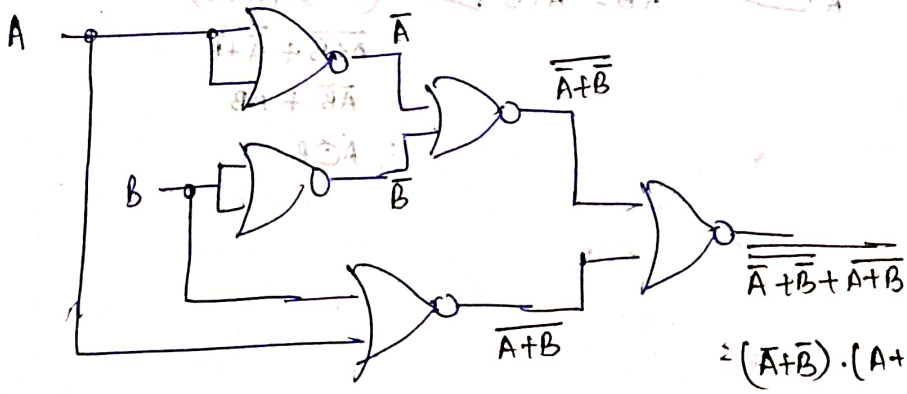
OR



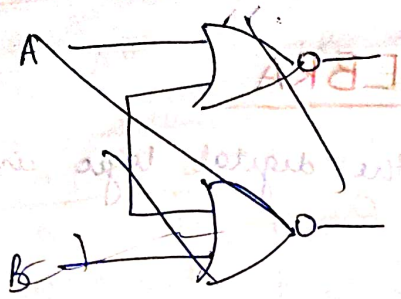
AND

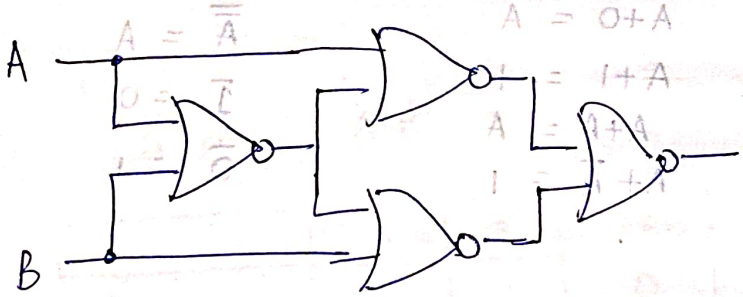


NAND gate.



XOR gate.





XNOR gate

For odd no. of inputs, XOR and XNOR are same

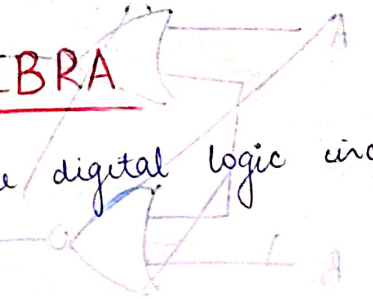
$A \oplus B \oplus C = A \odot B \odot C$

★

	No. of NAND gates	No. of NOR gates
AND	2	3
OR	3	2
NOT	1	1
XOR	4	5
XNOR	5	4
NAND	1	4
NOR	4	1

BOOLEAN ALGEBRA

⊙ used to analyze and simplify the digital logic circuits.
uses 2 binary nos — 0 & 1.



AND Laws —

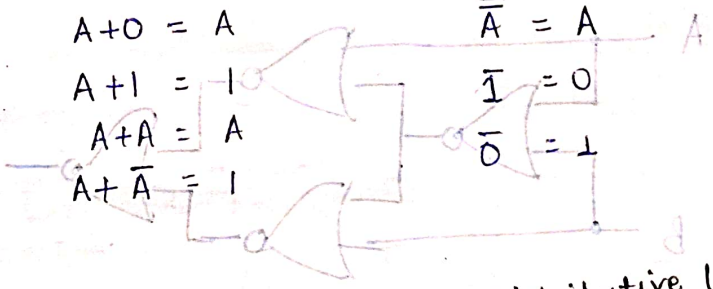
$A \cdot 0 = 0$
$A \cdot 1 = A$
$A \cdot A = A$
$A \cdot \bar{A} = 0$

OR Laws —

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

Inversion law

$\bar{\bar{A}} = A$
$\bar{1} = 0$
$\bar{0} = 1$



Commutative law

$A + B = B + A$
$A \cdot B = B \cdot A$

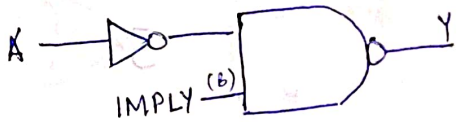
Associative law

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

Distributive law

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

Implication gate and inhibition gate.

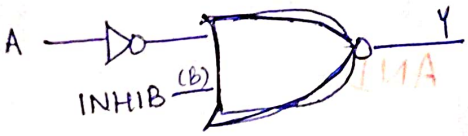


$Y = \overline{\bar{A} \cdot B} = \bar{\bar{A}} + \bar{B} = A + \bar{B}$

if $B = 0$, $Y = 1$
$B = 1$, $Y = A$

↳ if B is true, then, A comes as o/p.

Implication gate



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

if $B = 0$, $Y = A$
$B = 1$, $Y = 0$

↳ if B = 1, A does not reach o/p.

Inhibition gate.

De Morgan's first Theorem –

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

A	B	AB	$\bar{A}$	$\bar{B}$	$\overline{AB}$	$\bar{A}+\bar{B}$
0	0	0	1	1	1	1
0	1	0	1	0	1	1
1	0	0	0	1	1	1
1	1	1	0	0	0	0

De Morgan's second Theorem –

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

A	B	$\bar{A}$	$\bar{B}$	A+B	$\overline{A+B}$	$\bar{A}\cdot\bar{B}$
0	0	1	1	0	1	1
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	1	0	0	1	0	0

Principle of duality –

dual of an expression

- → changing AND to OR & vice versa.
- → changing zeros to ones and vice versa
- → variables & complements remain unchanged.

Concensus Theorem / Redundancy Theorem

↦ 3 variables must be present in the expression.
↦ Each variable is repeated twice.
↦ One variable is present in complemented and uncomplemented form.

After applying this theorem, we take only those terms which contain complemented variable.

Proof:

$$Y = AB + \bar{A}C + BC$$
$$= AB + \bar{A}C + BC(A+\bar{A})$$
$$= AB + \bar{A}C + ABC + \bar{A}BC$$
$$= AB(1+C) + \bar{A}C(1+B)$$
$$= AB + \bar{A}C$$

Redundant Literal Rule

Redundant Literal Rule says that

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

Shannon's Expansion Theorem

MINTERMS —

product of the variables in which each operand appears exactly once in true or complemented form.

A	B	C	Minterm	
0	0	0	$\bar{A}\bar{B}\bar{C}$	m_0
0	0	1	$\bar{A}\bar{B}C$	m_1
0	1	0	$\bar{A}B\bar{C}$	m_2
0	1	1	$\bar{A}BC$	m_3
1	0	0	$A\bar{B}\bar{C}$	m_4
1	0	1	$A\bar{B}C$	m_5
1	1	0	$AB\bar{C}$	m_6
1	1	1	ABC	m_7

MAXTERMS —

sum of variables in which each variable appears exactly once in true or complemented form.

A	B	C	Maxterm	
0	0	0	$A+B+C$	M_0
0	0	1	$A+B+\bar{C}$	M_1
0	1	0	$A+\bar{B}+C$	M_2
0	1	1	$A+\bar{B}+\bar{C}$	M_3
1	0	0	$\bar{A}+B+C$	M_4
1	0	1	$\bar{A}+B+\bar{C}$	M_5
1	1	0	$\bar{A}+\bar{B}+C$	M_6
1	1	1	$\bar{A}+\bar{B}+\bar{C}$	M_7

★ $M_j = \overline{m_j}$

Canonical SoP → sum of products → sum of minterms.

Canonical PoS → product of sums → product of maxterms.

Standard SoP → simplified → each term need not contain all the literals.

Standard PoS → simplified → each term need not contain all the literals.

K-map -

- Karnaugh map.
- graphical method which consists of 2^n cells for n variables.

The adjacent cells differ only in single bit position ⇒ gray code.

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

Important terminologies –

1. Implicant :- minterm or maxterm.
2. Prime implicant :- it is formed by combining maximum possible adjacent cells.
3. Essential Prime implicant :- prime implicant which contains atleast one minterm or maxterm which cannot be covered by ~~any~~ any other prime implicant.

Design binary to gray code converter using logic gates

A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1

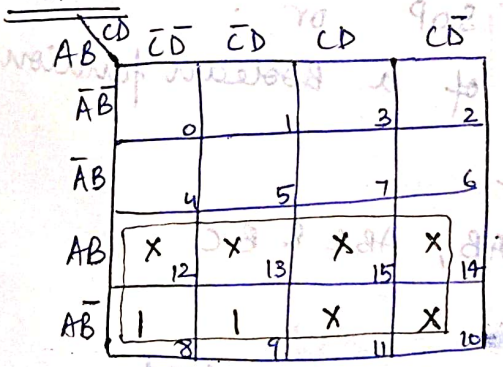
$W = \Sigma m(8,9) + d(10,11,12,13,14,15)$

$X = \Sigma m(4,5,6,7,8,9) + d(10,11,12,13,14,15)$

$Y = \Sigma m(2,3,4,5) + d(10,11,12,13,14,15)$

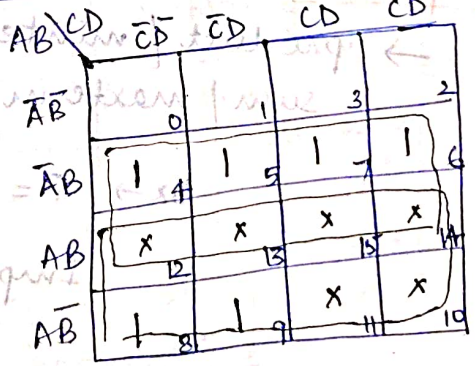
$Z = \Sigma m(1,2,5,6,9) + d(10,11,12,13,14,15)$

For W –



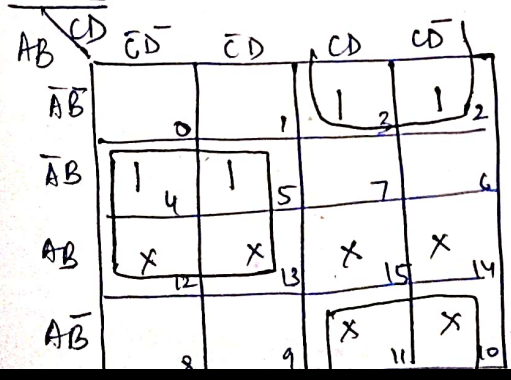
$\therefore W = A$

For X –



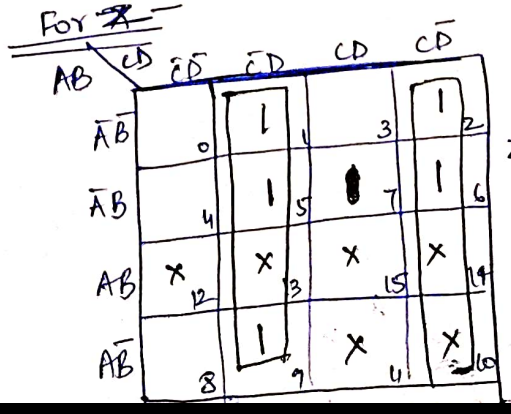
$X = A + B$

For Y



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

For Z



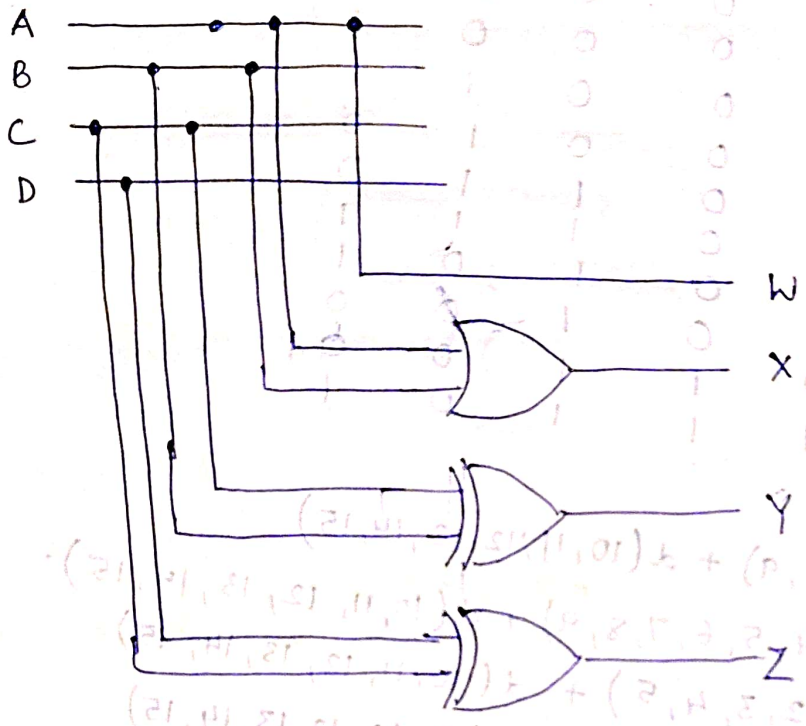
$Z = \bar{C}D + C\bar{D}$

$W = A$

$X = A + B$

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

$Z = \bar{C}D + C\bar{D}$



Implicant —

→ product / minterm term in SOP or sum / maxterm term in POS of a Boolean functio

Ex → $F = AB + ABC + BC$

Implicants are AB, ABC & BC

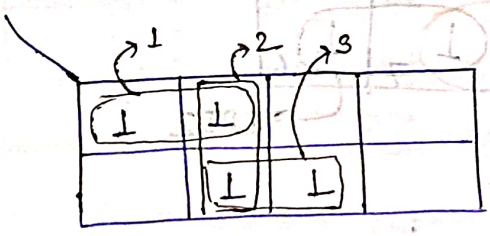
every term in logic expression is called implicant.

PRIME IMPLICANTS -

group of square or rectangle made up of bunch of adjacent minterms / maxterms.

→ Form as large group as possible
→ Form all possible combinations
→ do not form any smaller group within larger group.

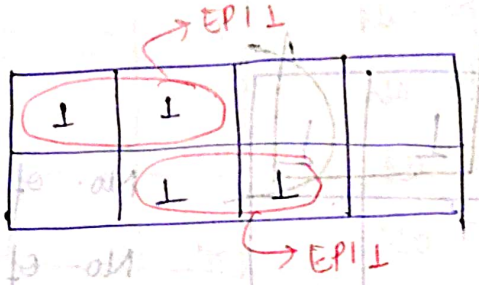
Ex



3 prime implicants.

ESSENTIAL PRIME IMPLICANTS

Those prime implicants which cover atleast one minterm that can't be covered by any other prime implicant.

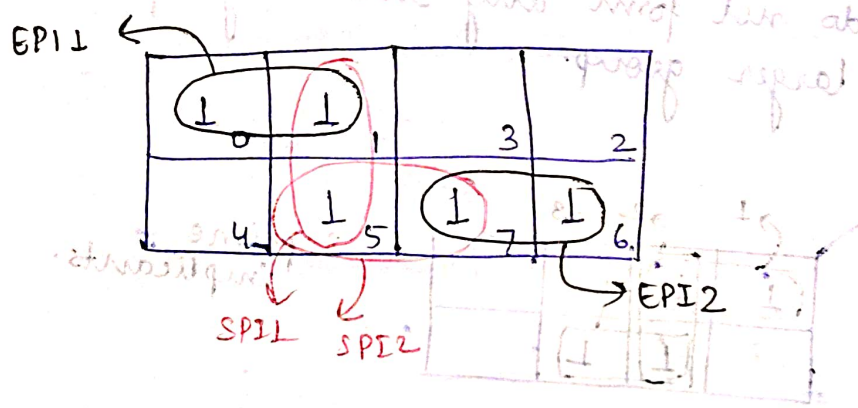


No. of essential prime implicants = 2

Redundant prime implicant do not appear in minimized logic expression.

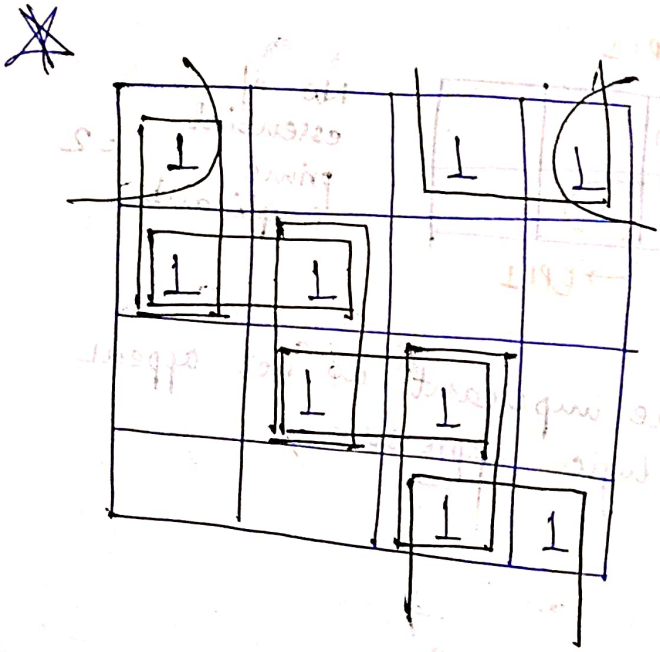
SELECTIVE PRIME IMPLICANTS

prime implicants which are neither essential nor redundant prime implicants are called Selective Prime Implicant (SPI).



For maxterm, we replace prime implicant by false prime implicant.

★ Smaller group within larger group is implicand.



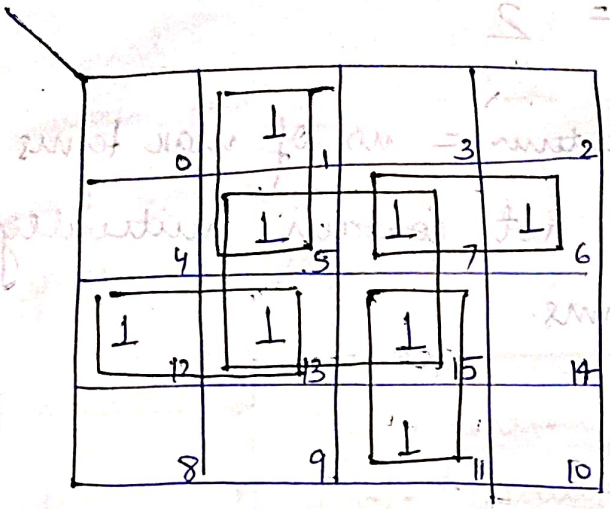
No. of PI = 7

No. of EPI = 1

No. of RPI = 0

No. of SPI = 6

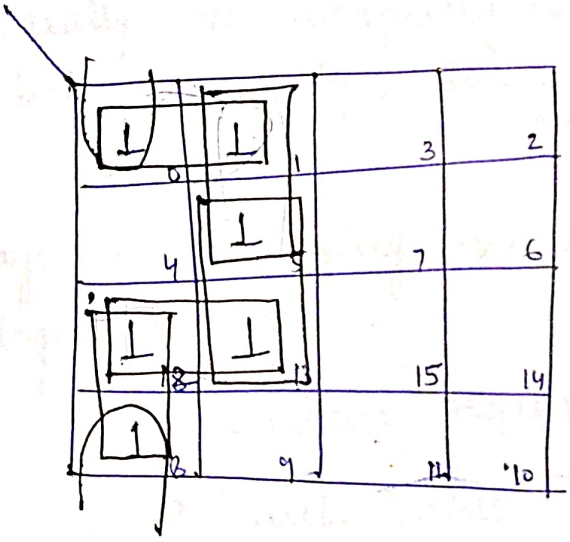
Ques Given $F = \Sigma m(1, 5, 6, 7, 11, 12, 13, 15)$, find number of PI, EPI, RPI and SPI.



No. of implicants = 8+5+4 = 17
No. of PI = 5
No. of EPI = 4
No. of RPI = 1
No. of SPI = 0

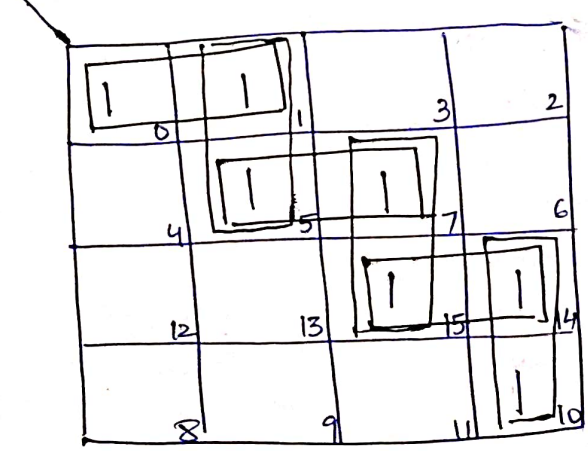
Ques $\Sigma m(0, 1, 5, 8, 12, 13)$

* If EPI is not there, RPI cannot exist.



No. of implicants = 6+6 = 12
No. of PI = 6
No. of EPI = 0
No. of RPI = 0
No. of SPI = 6.

Ques $\Sigma(0, 1, 5, 7, 15, 14, 10)$

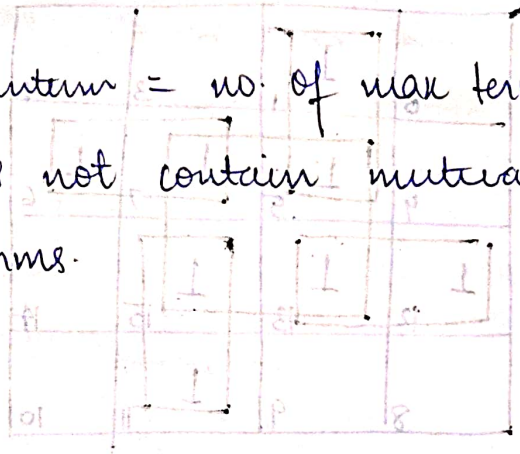


No. of implicants = 6+7 = 13
No. of PI = 6
No. of EPI = 2
No. of RPI = 0
No. of SPI = 4

★ Number of self dual functions $= 2^{2^{n-1}}$

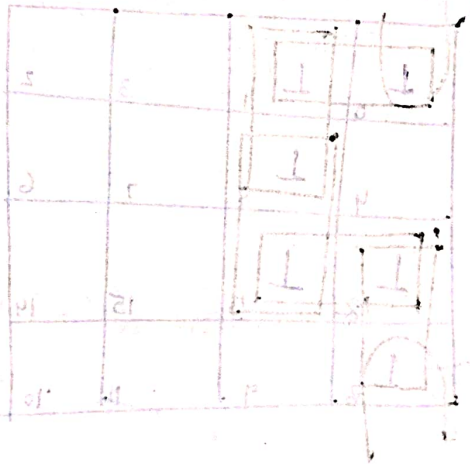
↳ Number of minterm = no. of max terms

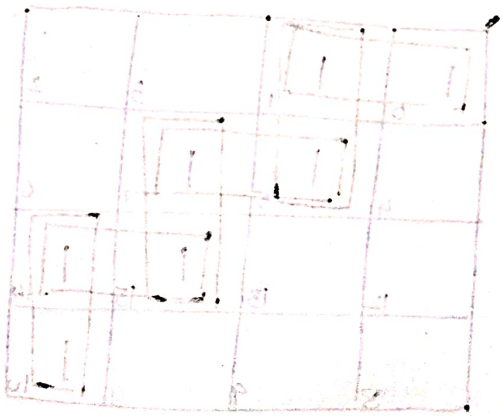
↳ function does not contain mutually exclusive terms.



Functionally complete boolean function

↳ practice

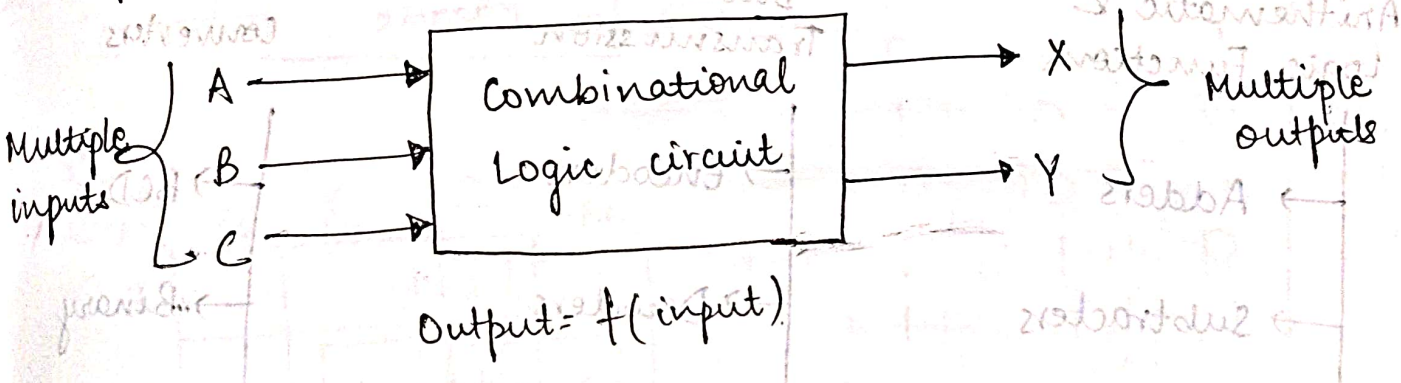




COMBINATIONAL CIRCUITS

Combinational circuits consist of logic gates. These circuits operate with binary ~~variables~~ values.

The o/p of combinational circuit depends upon the combination of present inputs.



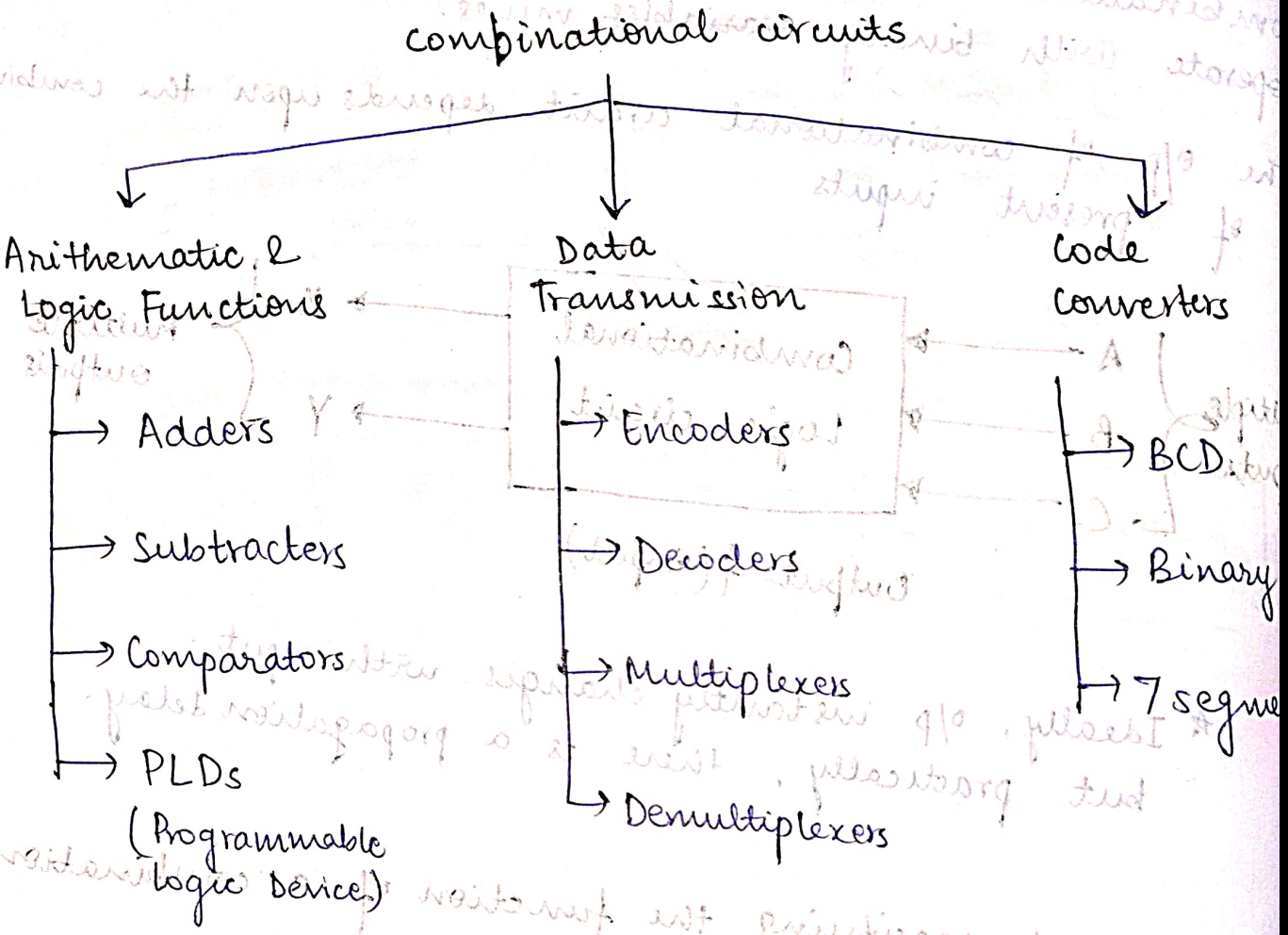
Output = f(input)

* Ideally, o/p instantly changes with input but practically, there is a propagation delay.

Ways of specifying the function of a combinational logic circuit

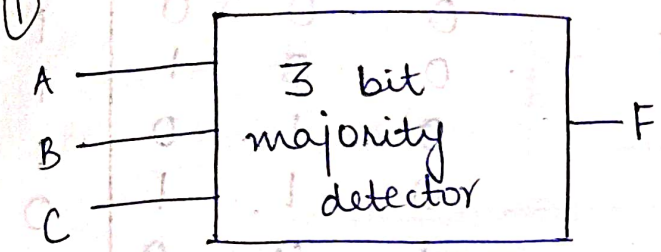
- Boolean Algebra
- Truth Table
- Logic diagram.

Classification of combinational circuits



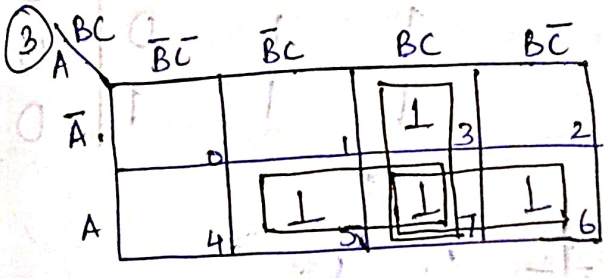
Majority Detectors —

This circuit gives the output 1 when inputs have more number of 1's than 0's.

① 

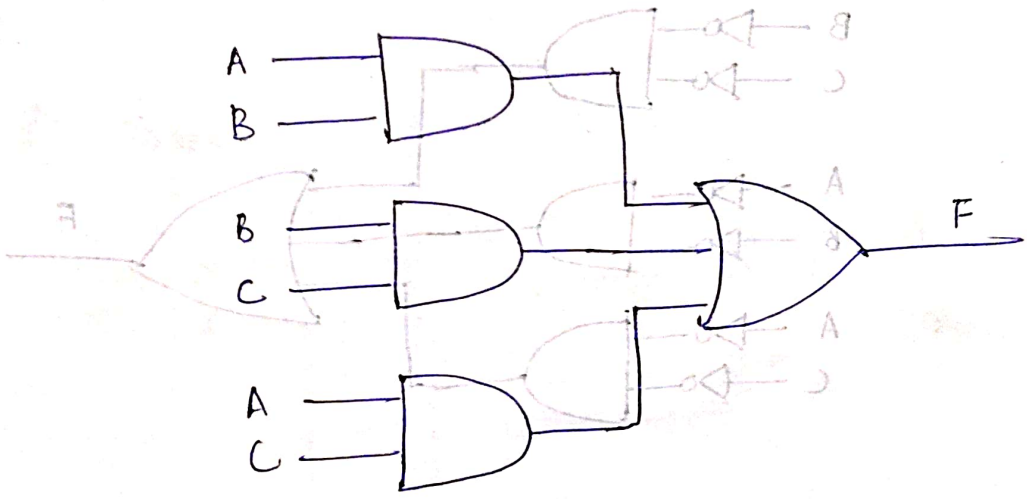
②

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

③ 

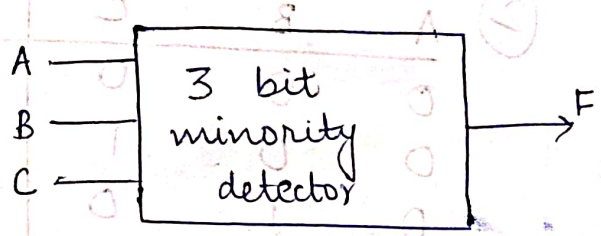
$\therefore F = AC + AB + BC$

④ Circuit diagram —



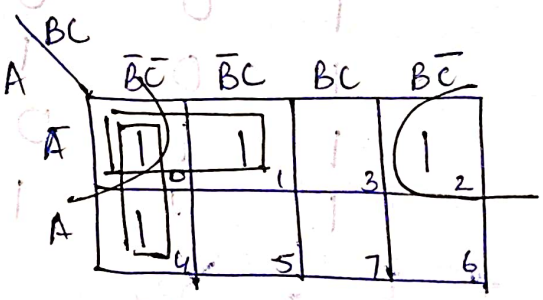
Minority detector –

gives o/p 1 when inputs have lesser no. of 1s than 0s.

① 

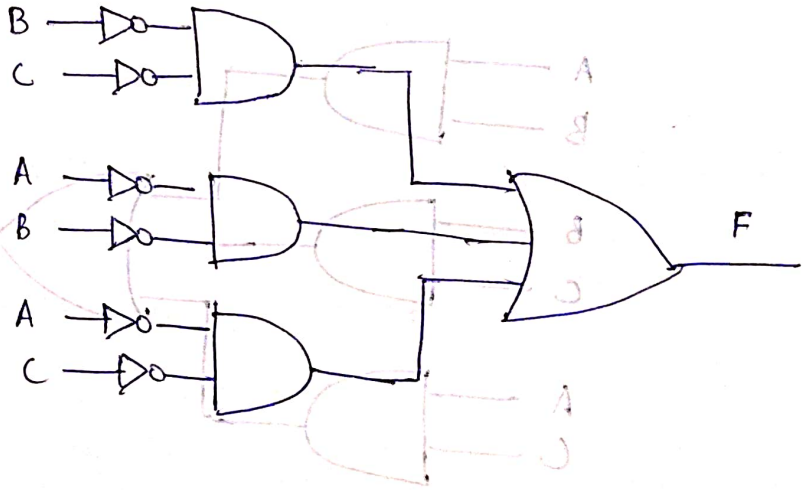
②

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

③ 

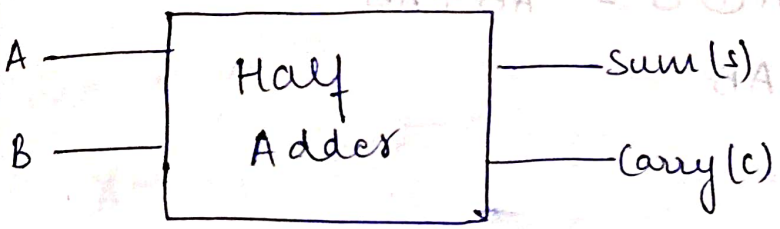
$\therefore F = \bar{B}\bar{C} + \bar{A}\bar{B} + \bar{A}\bar{C}$

④ Logic circuit –



Half Adder —

device which adds 2 bits and produces 2 outputs — sum and carry.

① 

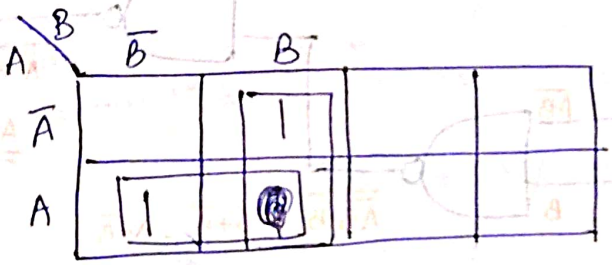
②

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

③

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

$C = AB$.

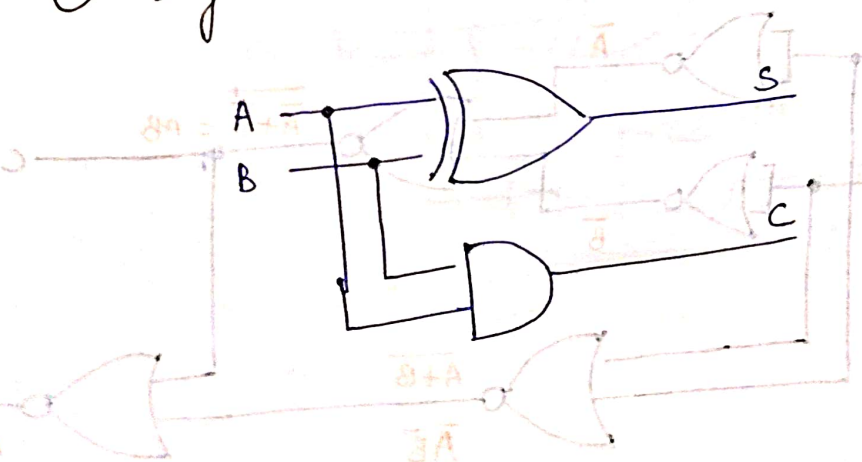


~~C = AB~~ Simplified —

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

$C = AB$.

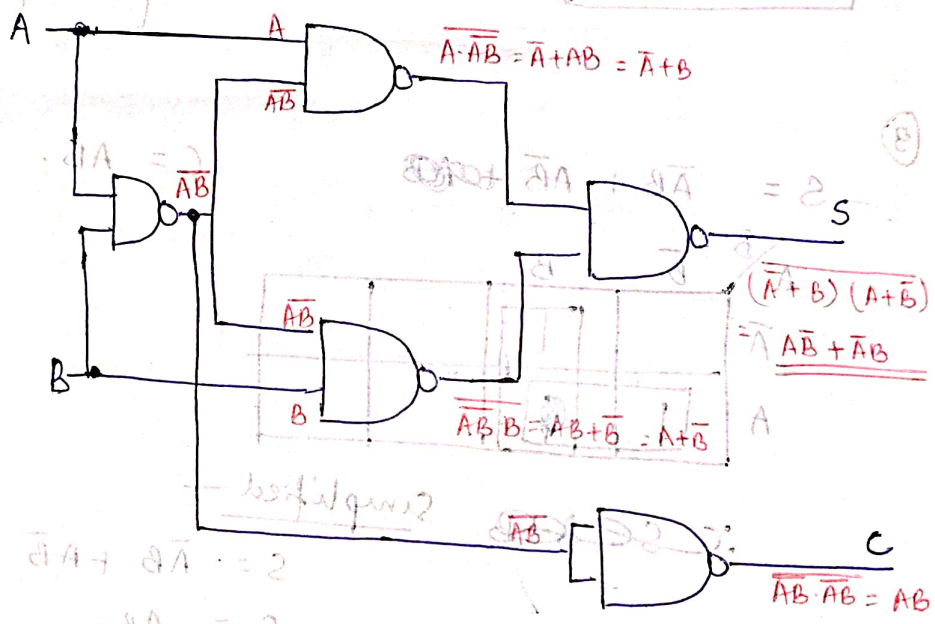
④ Logic circuit —



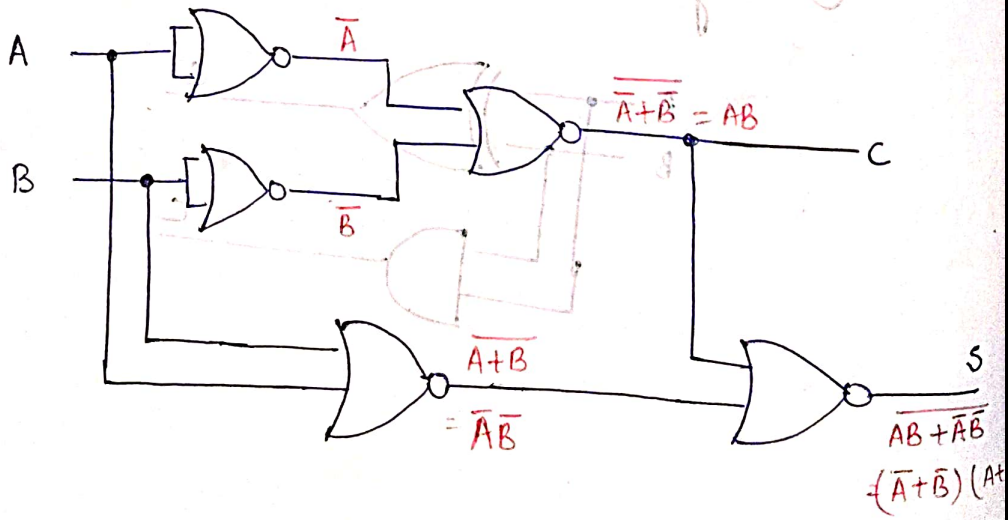
Half adder using only NAND and NOR gates

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

$$C = AB$$



∴ 5 NAND gates are required.



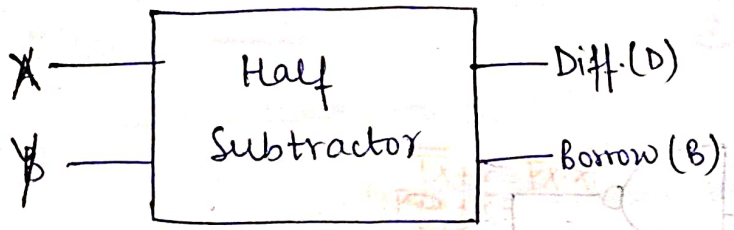
∴ 5 NOR gates are required.

Half Subtracter

circuit to perform subtraction of 2 binary bits.

O/p → Difference and borrow.

①



②

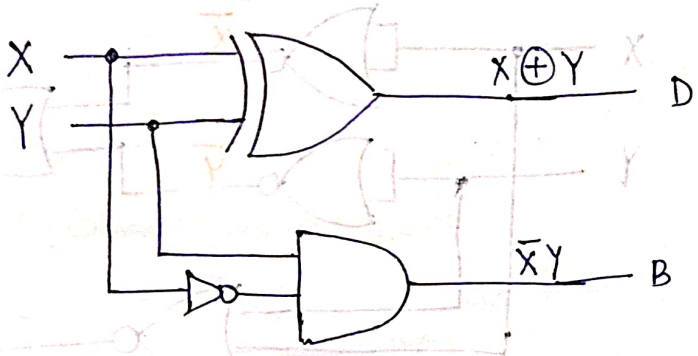
X	Y	D	Bo
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

③ Simplified expression –

$$D = X\bar{Y} + \bar{X}Y = X \oplus Y$$

$$B = \bar{X}Y$$

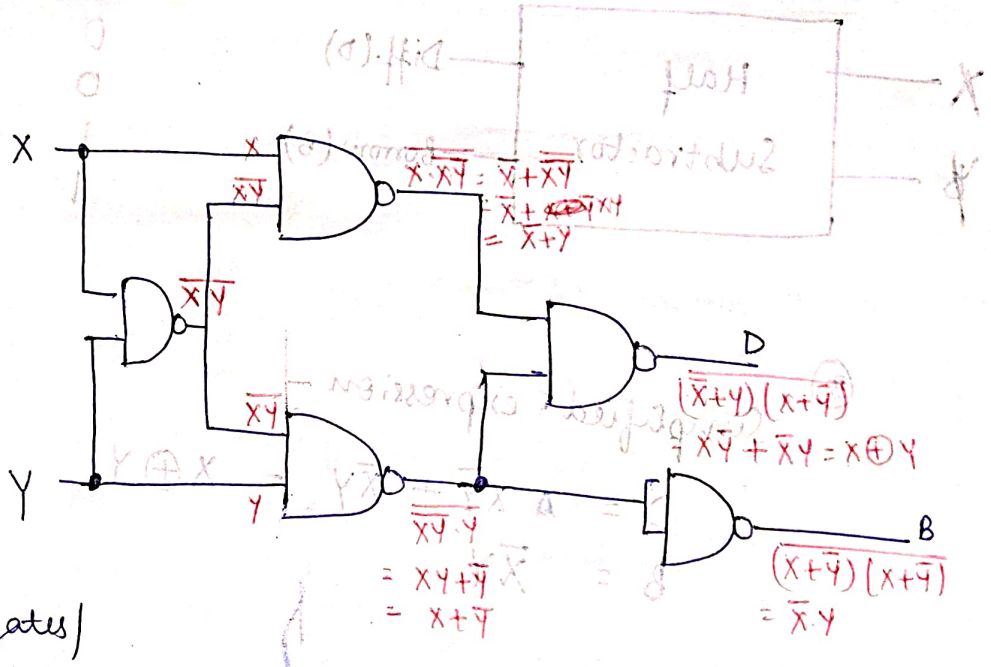
④ Logic circuit diagram



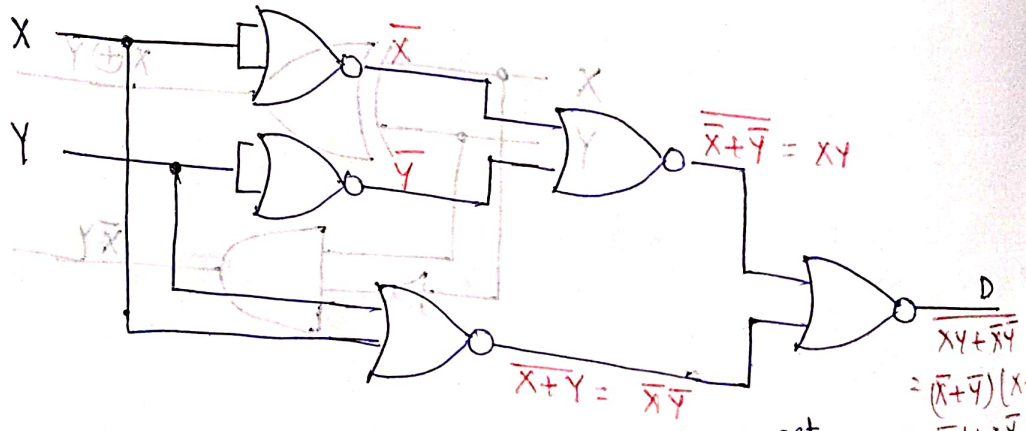
Half Subtractor using NAND and NOR gates.

$$D = X \oplus Y$$

$$B = \overline{X}Y$$

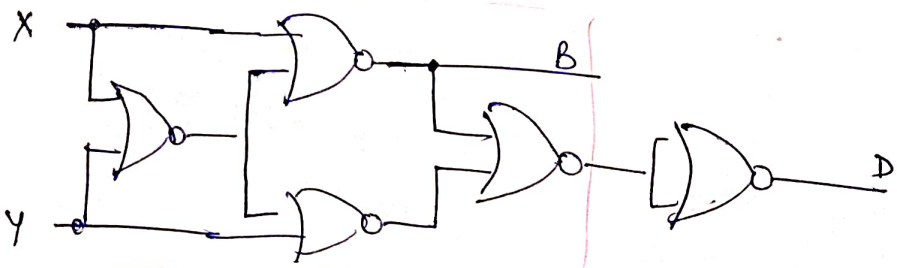


5 NAND gates / 5 NOR gates required.



No way to get B w/o using 2-3 more gates.

XNOR

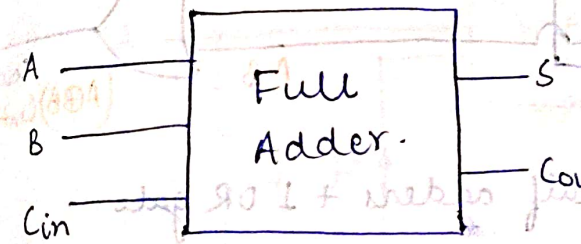


Full Adder

takes 3 inputs and produces 2 outputs — sum and carry.

Input — A, B, Cin

Output — S, Cout

① 

②

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

③

A \ BCin	$\bar{B}\bar{C}in$	$\bar{B}Cin$	$BCin$	$B\bar{C}in$
$\bar{A}$	0	⊥ 1	3	⊥ 2
A	⊥ 4	5	⊥ 7	6

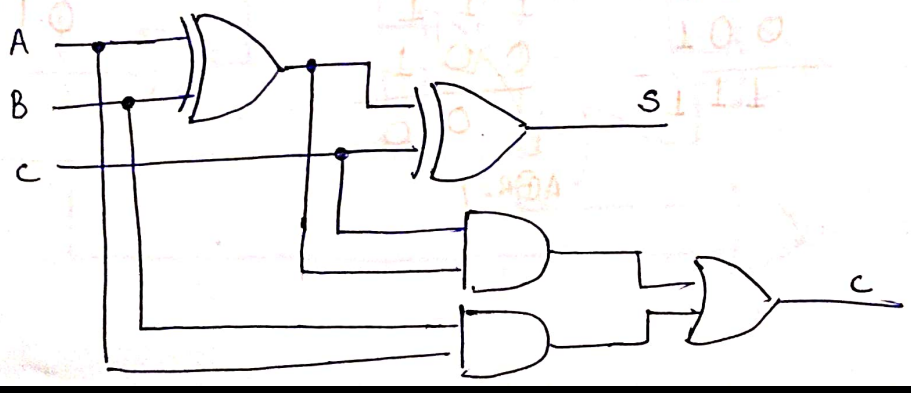
Sum = ⊥ when 1's at input are odd

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

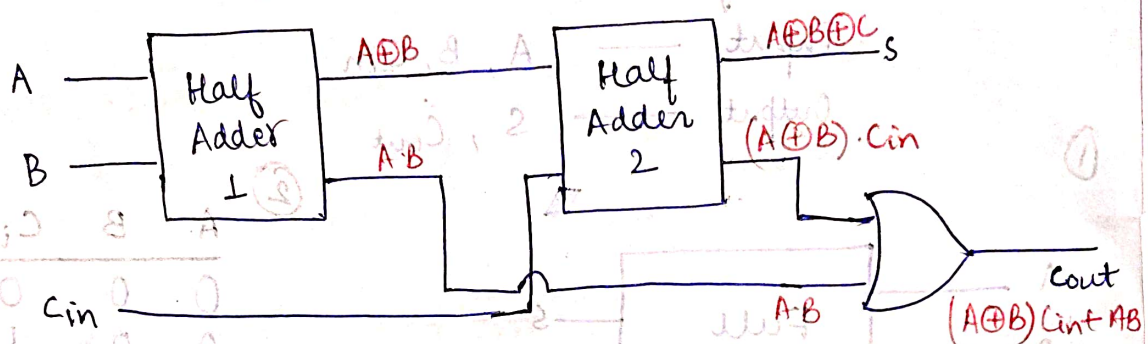
A \ BCin	$\bar{B}\bar{C}in$	$\bar{B}Cin$	$BCin$	$B\bar{C}in$
$\bar{A}$	0	1	⊥ 3	2
A	⊥ 4	⊥ 5	⊥ 7	⊥ 6

$\therefore Cout = BCin + ACin + AB = (A \oplus B)Cin + AB$

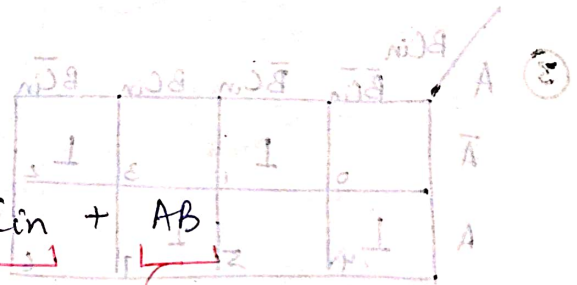
④ Logic circuit —



Implementation of Full Adder using Half Adders.



∴ 1 full adder = 2 Half adders + 1 OR gate.



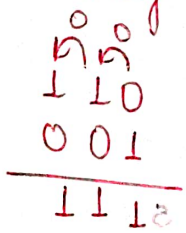
$C_{out} = (A \oplus B) C_{in} + AB$

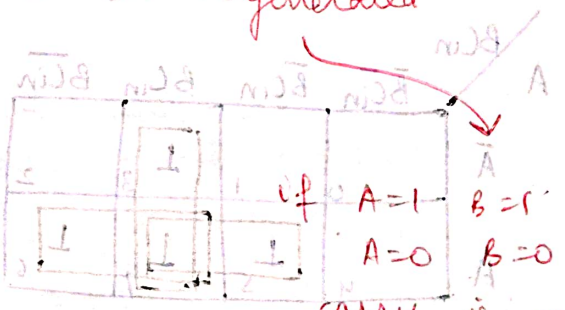
Carry propagated

Carry generated

If $A \oplus B = 1$ i.e.
when A=1, B=0
A=0 B=1

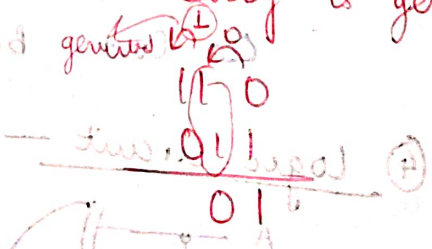
$C_{out} = C_{in}$ i.e.
carry propagates





if A=1 B=1
A=0 B=0

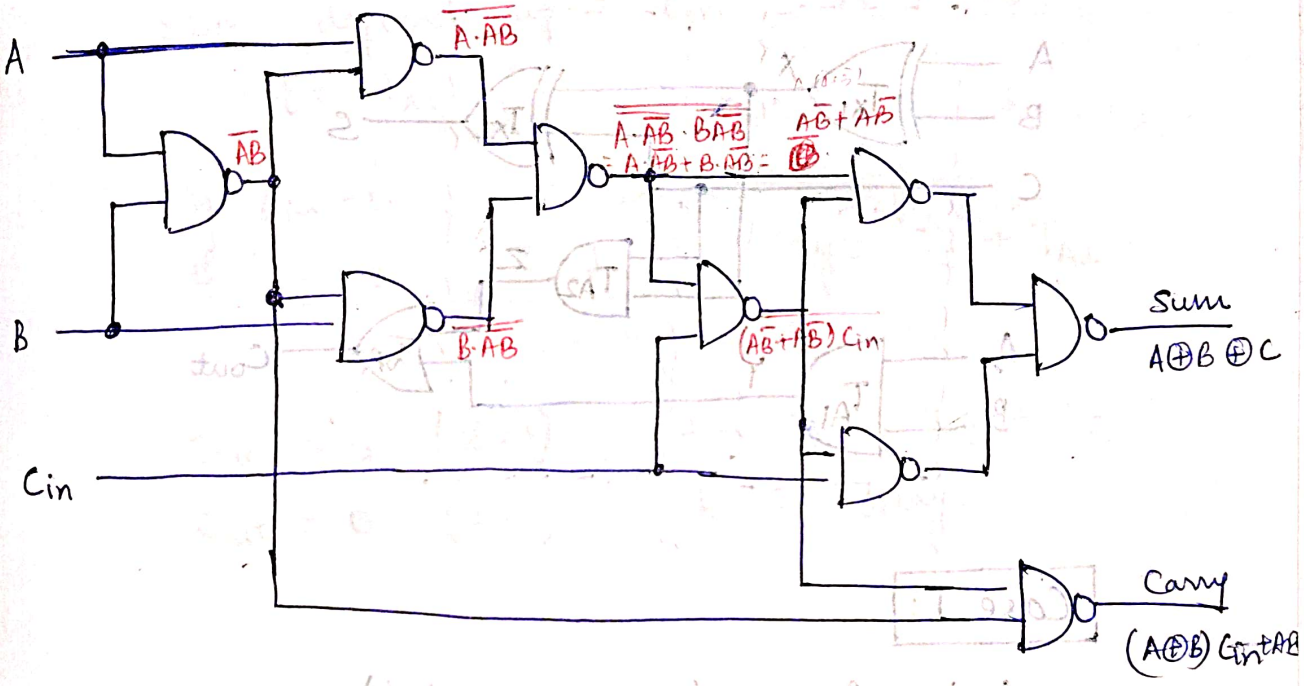
carry is generated.



Full adder using NAND gates

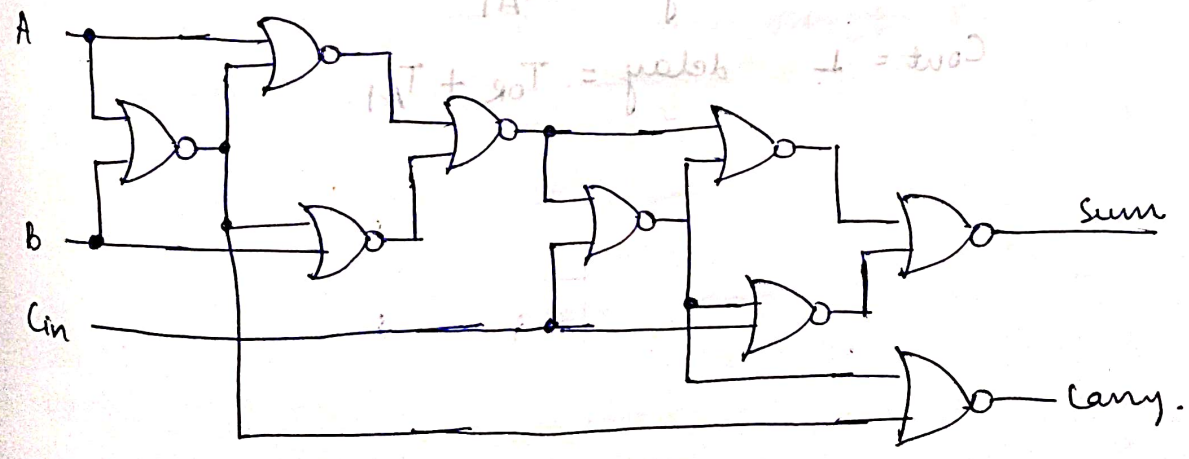
9 NAND gates are required.

4 for XOR1, 4 for XOR2, 1 for carry.



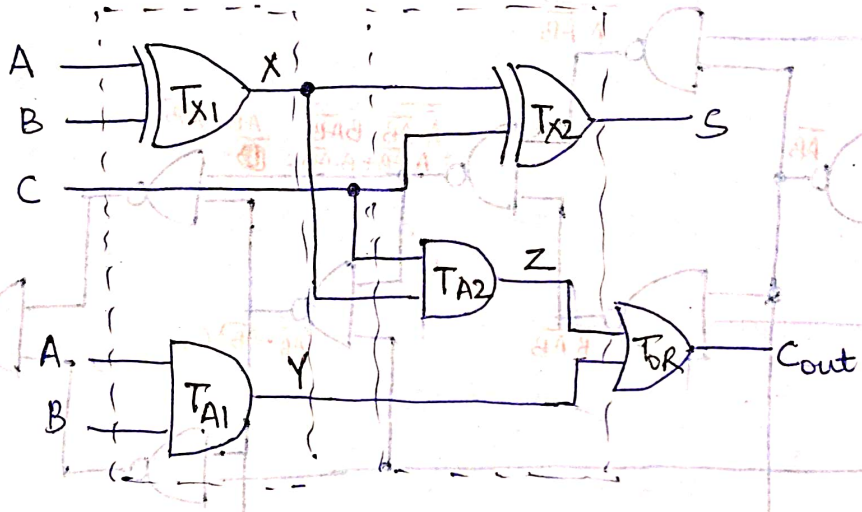
Full adder using NOR gates

9 NOR gates are required.



Propagation delay –

It is the difference b/w the instant at which input changes and the time at which o/p changes.



Case I:

A = 1 B = ⊥ (carry generated)

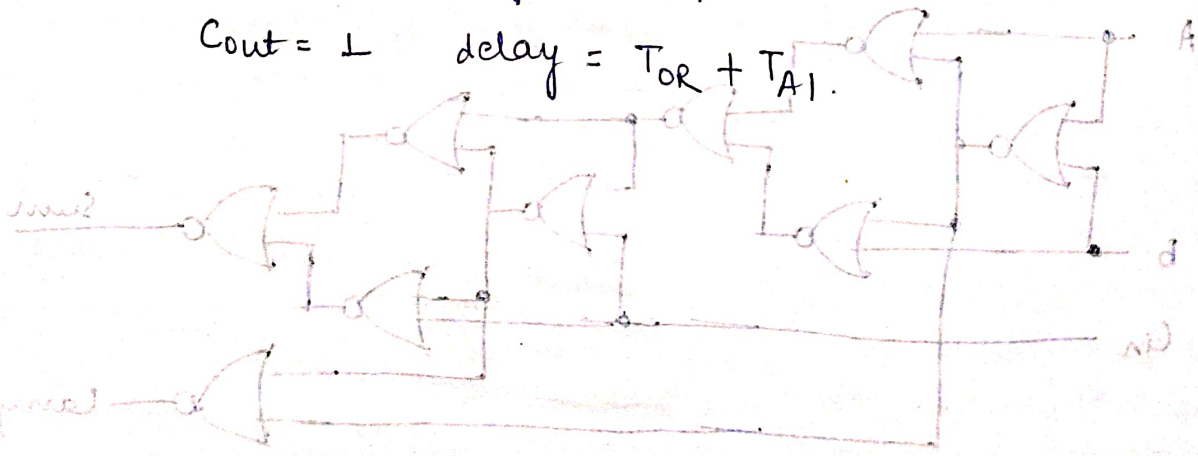
X = 0 (no edge ∴ no delay)

if $C_{in} = 0$, $\overset{sum}{Q} = 0$ no delay

if $C_{in} = ⊥$ $\overset{sum}{Q} = ⊥$ delay = T_{A2}

Y = ⊥ delay = T_{A1}

C_{out} = ⊥ delay = $T_{OR} + T_{A1}$.



Case II

A = 1 B = 0
A = 0 B = 1 [carry propagated]

X = 1 delay = T_{X1}

Y = 0 no delay

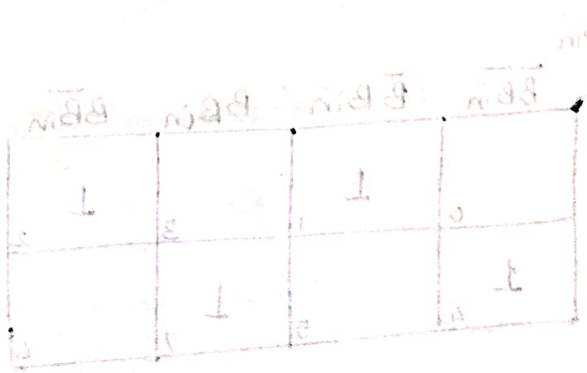
SUM: depends upon Cin

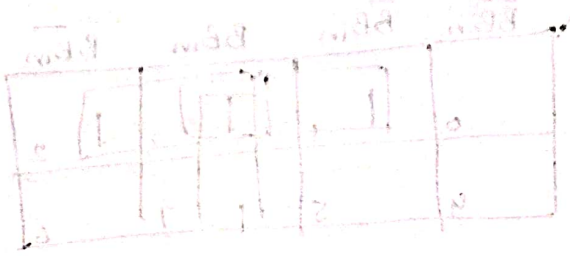
$t = T_{X1} + T_{X2}$

if Cin = 0 , Z = 0 no delay

if Cin = 1 , Z = 1 delay = $T_{X1} + T_{A2}$

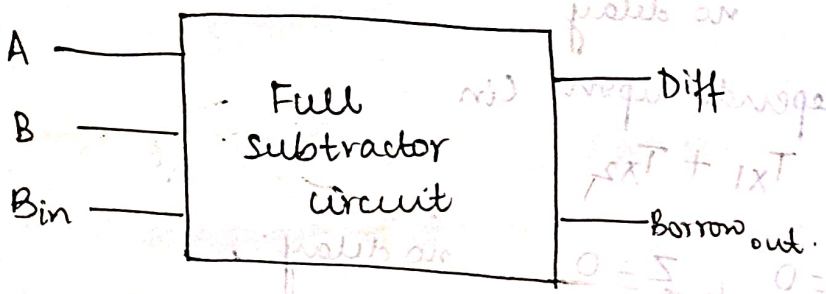
Cout = 1 (Z = 1)	$t = T_{X1} + T_{A2} + T_{OR}$
Cout = 0 (Z = 0)	t = no delay





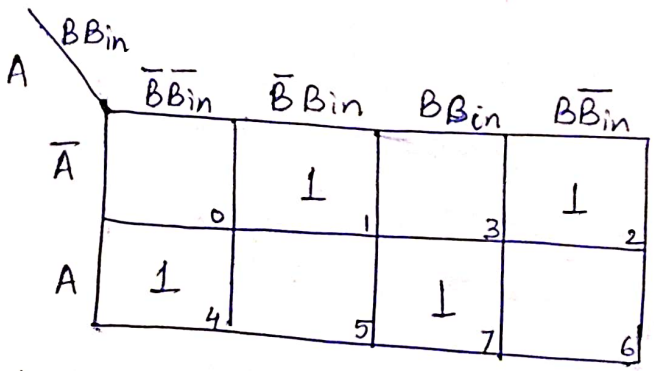
Full Subtractor

combinational circuit that performs subtraction of 3 bits — minuend, subtrahend & borrow of previous adjacent lower minuend bit.

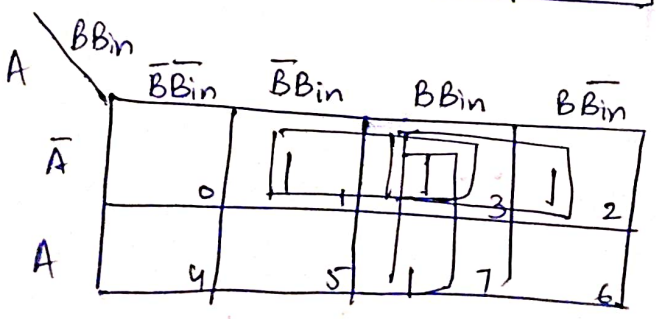


A − B − Bin

A	B	Borrow in	Diff	Borrow out
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1



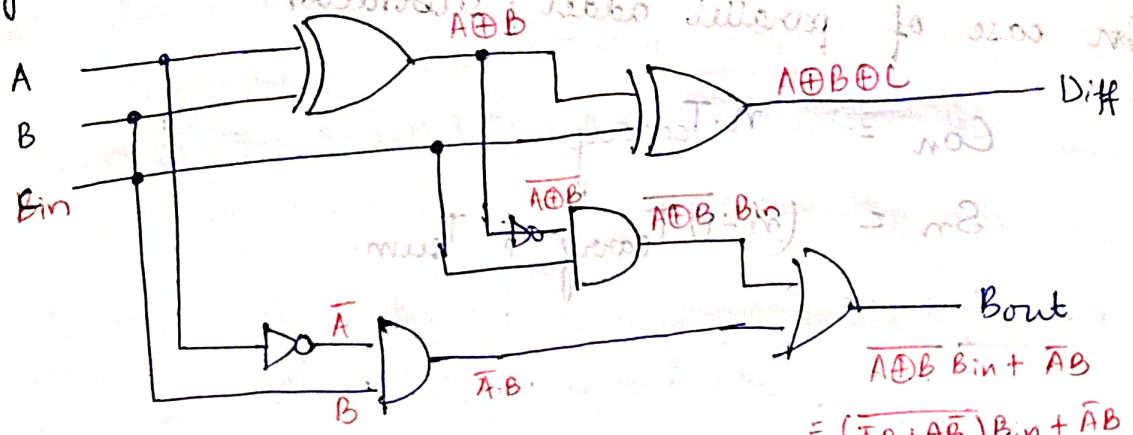
$\therefore D = A \oplus B \oplus C$



$\therefore B_{out} = \bar{A}B_{in} + \bar{A}B + BB_{in}$

$= \bar{A}B + (\bar{A}+B)B_{in}$

Logic circuit –



$\overline{A \oplus B}\, B_{in} + \bar{A}B$

$= (\overline{\bar{A}B + A\bar{B}})\, B_{in} + \bar{A}B$

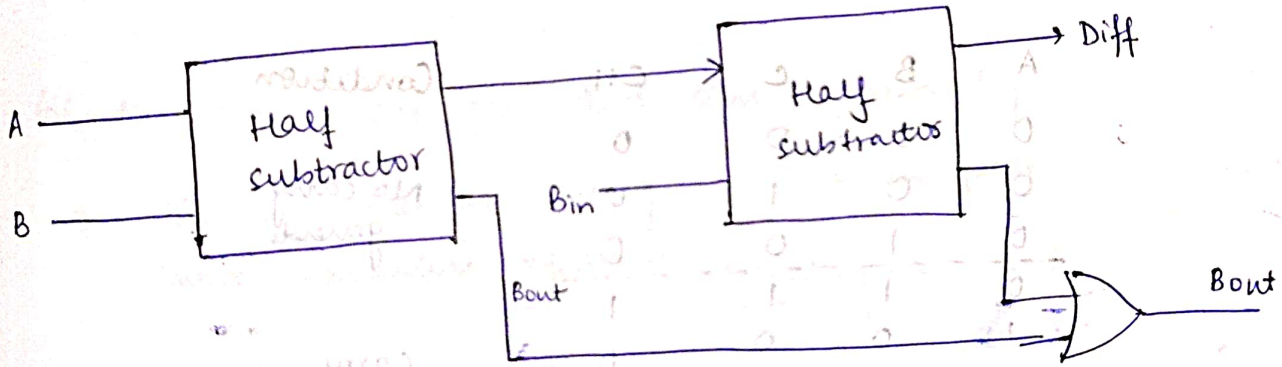
$= (A + \bar{B})(\bar{A} + B)\, B_{in} + \bar{A}B$

$= (AB + \bar{A}\bar{B})\, B_{in} + \bar{A}B$

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

$= B + \bar{A}\bar{B}B_{in} = B + \bar{A}B_{in}$

Full SubTractor using Half subtractors –

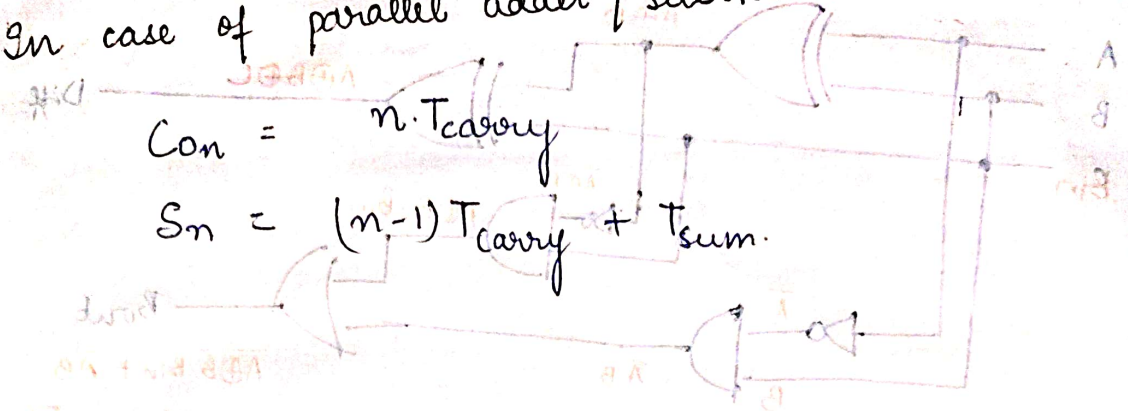


Full Subtractor using universal gates

→ 9 NAND gates are required.

→ 9 NOR gates are required.

In case of parallel adder / subtractor.



$C_{on} = n \cdot T_{carry}$

$S_n = (n-1) T_{carry} + T_{sum}$.

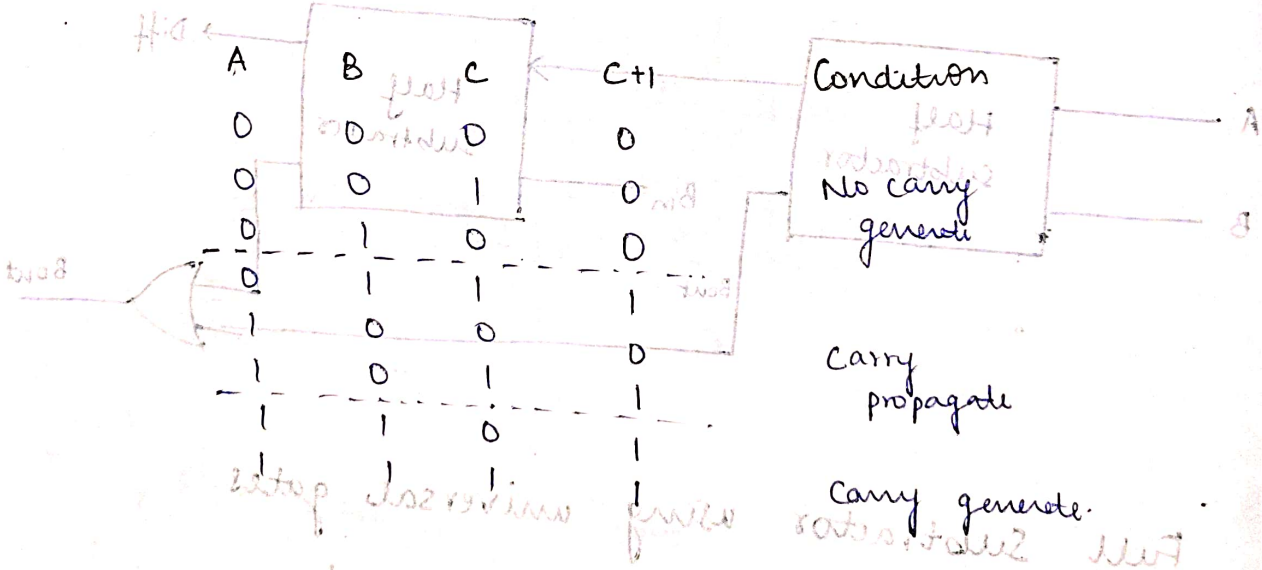
Carry Look Ahead Adder

⊙ reduces propagation delay by introducing more complex circuit/hardware.

$C_{out} = (A \oplus B) C_{in} + AB$

$(A \oplus B)$ → Carry propagate

AB → Carry generate.



$C_{i+1} = (A_i \oplus B_i) C_i + A_i B_i$

C_{i+1} → carry output of ith stage

C_i ← carry input to ith state

Let $P_i = A_i \oplus B_i$ $\quad G_i = A_i B_i$

$$S_i = A_i \oplus B_i \oplus C_i = P_i \oplus C_i$$

$$C_{i+1} = (A_i \oplus B_i) C_i + A_i B_i = P_i C_i + G_i$$

$$C_1 = G_0 + P_0 C_{in}$$

$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 (G_0 + P_0 C_{in})$$
$$= G_1 + P_1 G_0 + P_0 P_1 C_{in}$$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 (G_1 + P_1 G_0 + P_0 P_1 C_{in})$$
$$= G_2 + P_2 G_1 + P_1 P_2 G_0 + P_0 P_1 P_2 C_{in}$$

$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 (G_2 + P_2 G_1 + P_1 P_2 G_0 + P_0 P_1 P_2 C_{in})$$
$$= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in}$$

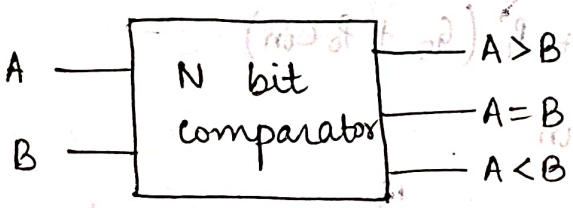
All the carries are generated simultaneously.

can be implemented using AND & OR gates.

more hardware required.

Comparators

- compare 2 numbers
- finds out whether one binary number is equal, less than or greater than the other binary number.



1 bit

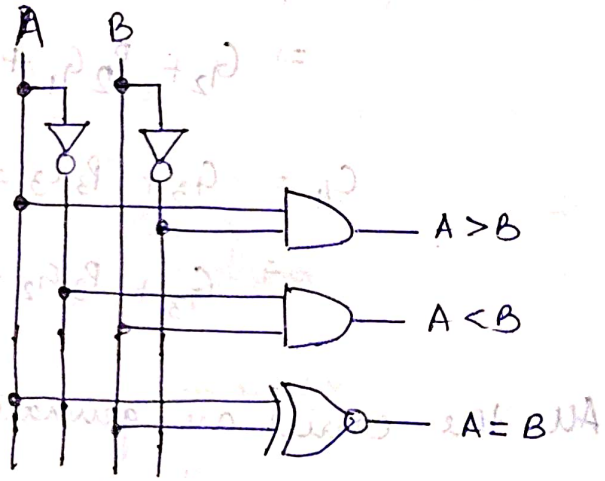
A	B	A>B	A=B	A<B
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

O/p —

$A>B : A\bar{B}$

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

$A<B : \bar{A}B$



2 bit magnitude comparator —

1st binary no. = $A_1 A_0$

2nd binary no. = $B_1 B_0$.

A_1	A_0	B_1	B_0	$A>B$	$A=B$	$A<B$
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	1	0	0
1	0	1	1	1	0	0
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

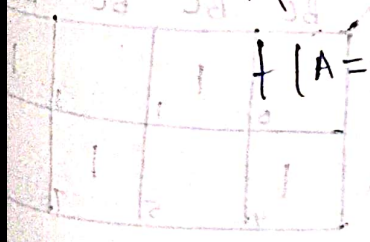
Priority Circuit

O/P

$$f(A>B) = \Sigma m(4, 8, 9, 12, 13, 14)$$

$$f(A<B) = \Sigma m(1, 2, 3, 6, 7, 11)$$

$$f(A=B) = \Sigma m(0, 5, 10, 15)$$



Simplified logic expression —

$A>B$: $A_1\overline{B_1} + (A_1 \odot B_1)\, A_0\overline{B_0}$

$A<B$: $\overline{A_1} B_1 + (A_1 \odot B_1)\, \overline{A_0} B_0$

$A=B$: $(A_1 \odot B_1)\,(A_0 \odot B_0)$

Similarly, for 3 bit comparator —

$A>B$: $A_2\overline{B_2} + (A_2 \odot B_2)\, A_1\overline{B_1} + (A_0 \odot B_0)\,(A_1 \odot B_1) A_0\overline{B_0}$

$A<B$: $\overline{A_2} B_2 + (A_2 \odot B_2)\, \overline{A_1} B_1 + (A_0 \odot B_0)(A_1 \odot B_1)\, \overline{A_0} B_0$

$A=B$: $(A_2 \odot B_2)\,(A_1 \odot B_1)\,(A_0 \odot B_0)$

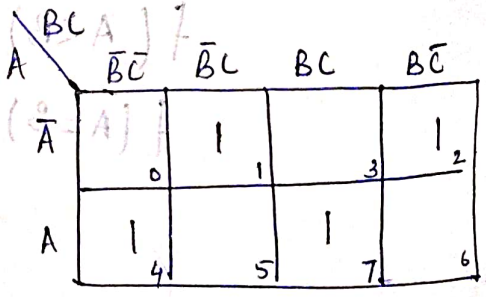
Parity Circuit

used for error detection

counting no. of 1's

3 bit message — Parity bit

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



$\therefore Y = A \oplus B \oplus C$

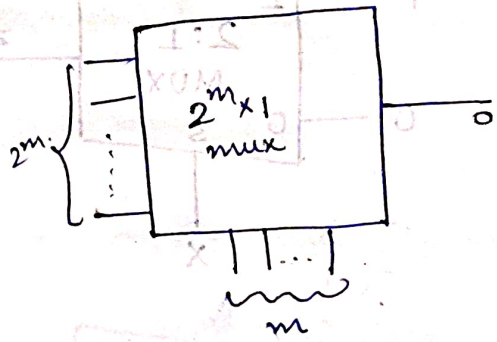
Parity bit = 1 if odd no. of 1s are present in data stream.

Multiplexer

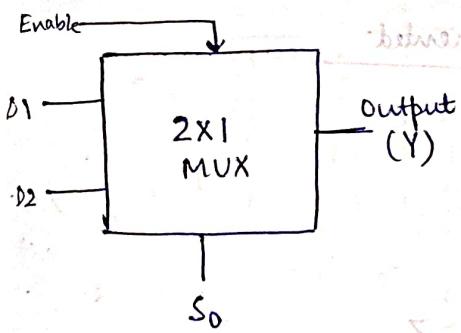
- also called data selector.
- combinational circuit with more than one input lines, one output line and more than one select lines.

Multiplexer → $2^m \times 1$ or $2^m : 1$ mux.

m = no. of select lines.



2X1 MUX



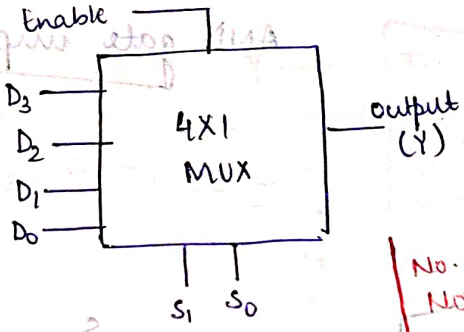
Select	Inputs		Output
	D_1	D_0	
0	0	0	0
0	0	1	0
1	1	0	1
1	1	1	1

$Y = D_0\bar{S} + D_1S$

Select	Y
0	D_0
1	D_1

4 NAND gates

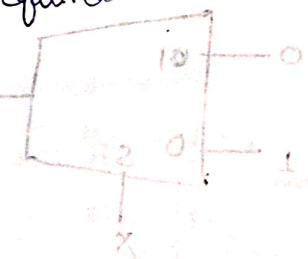
4X1 MUX



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

$\therefore Y = \bar{S}_1\bar{S}_0D_0 + \bar{S}_1S_0D_1 + S_1\bar{S}_0D_2 + S_1S_0D_3$

7 NAND gates required



No. of NOT gates req. = No. of select lines (m)

No. of AND gates req. = No. of inputs (2^m)

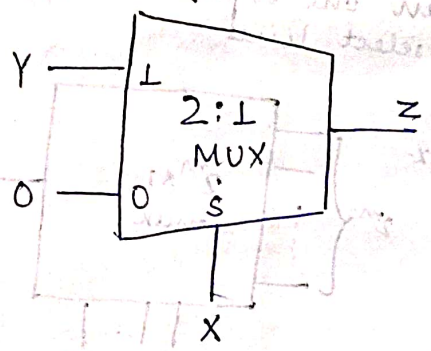
No. of OR gates = 1.

No. of NAND gates = $m + 2^m + 1$.

No. of NAND gates req. = $2^m + m + 1$ (↓ OR, ↙ OR, NOT)

Implementation of logic gates using MUX

1) AND gate.

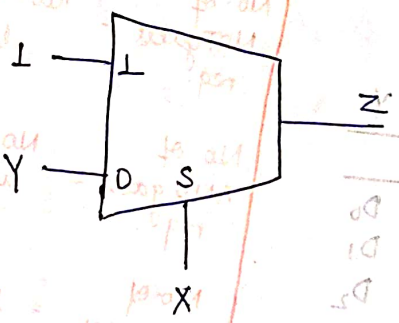


S	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

Also, $Z = \bar{X} \cdot 0 + X \cdot Y = XY$.

AND gate implemented.

2) OR gate.

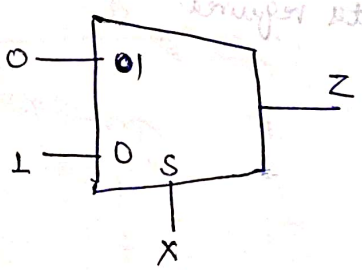


S	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

$Z = \bar{X} \cdot Y + X \cdot 1$

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

3) NOT gate



$Z = \bar{X} \cdot 1 + X \cdot 0$

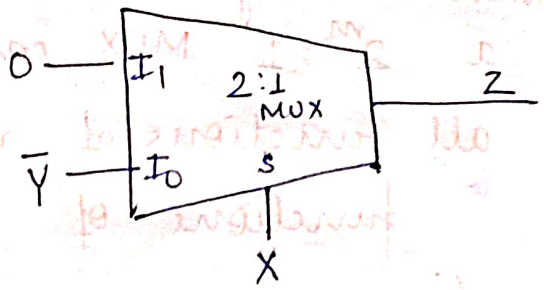
$= \bar{X}$

4) NOR gate

	X	Y	Z
S=0	0	0	1
	0	1	0
	1	0	0
S=1	1	1	0

$Z = \bar{Y} = I_0$

$Z = 0 = I_1$

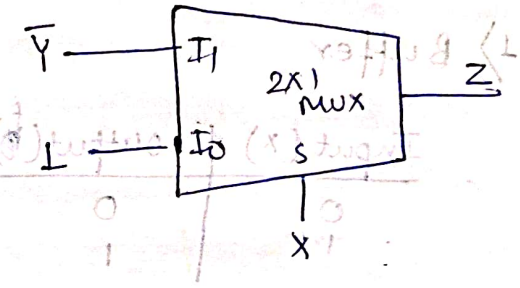


5) NAND gate

	X	Y	Z
S=0	0	0	1
	0	1	1
S=1	1	0	1
	1	1	0

$Z = 1 = I_0$

$Z = \bar{Y} = I_1$

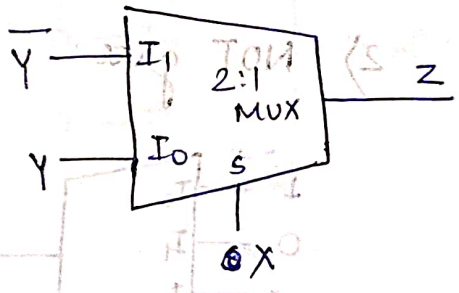


6) XOR gate

	X	Y	Z
S=0	0	0	0
	0	1	1
S=1	1	0	1
	1	1	0

$Z = Y = I_0$

$Z = \bar{Y} = I_1$

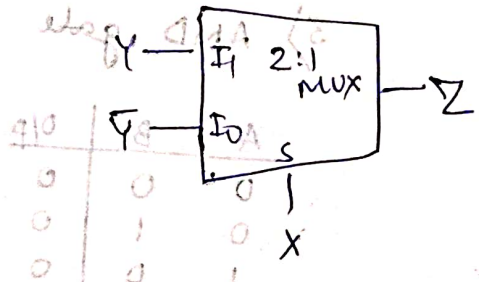


7) XNOR gate

	X	Y	Z
S=0	0	0	1
	0	1	0
S=1	1	0	0
	1	1	1

$Z = \bar{Y} = I_0$

$Z = Y = I_1$

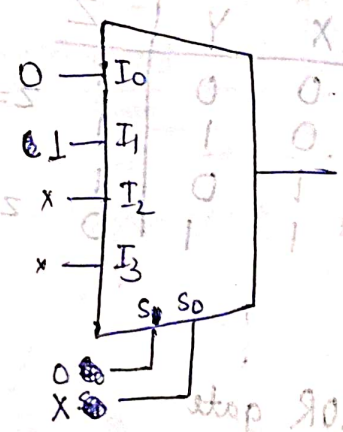


* Without any additional circuitry, a $2^m : 1$ MUX can be used to obtain all functions of m variables, only some functions of (m+1) variables.

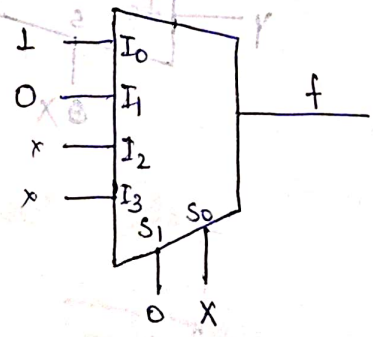
4:1 multiplexer as universal gate.

1) Buffer

Input (X)	Output (f)
0	0
1	1



2) NOT gate



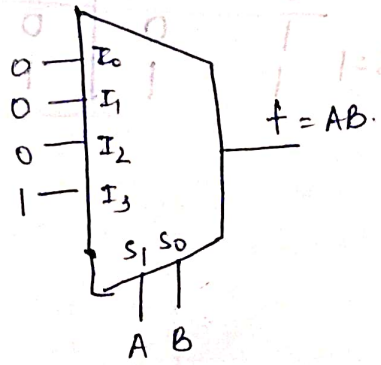
S_1	$S_0 = X$	f
0	0	$I_0 = 1$
0	1	$I_1 = 0$

* choose select lines as inputs of logic gate & then connect o/p of truth table as inputs of MUX.

3) AND gate

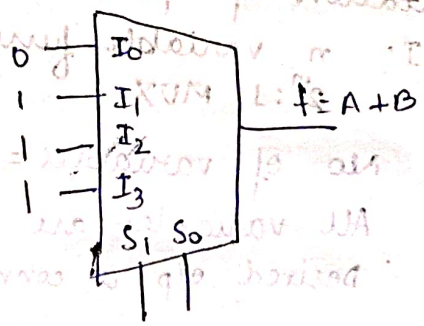
A	B	O/P
0	0	0
0	1	0
1	0	0
1	1	1

select lines

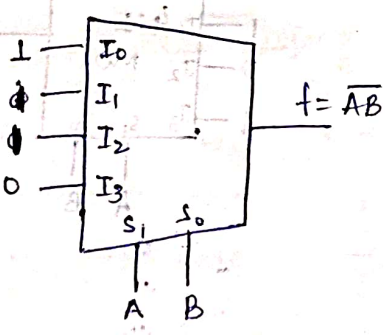


4) OR gate

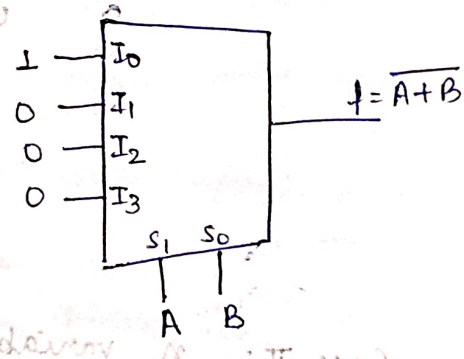
A	B	O/p
0	0	0
0	1	1
1	0	1
1	1	1



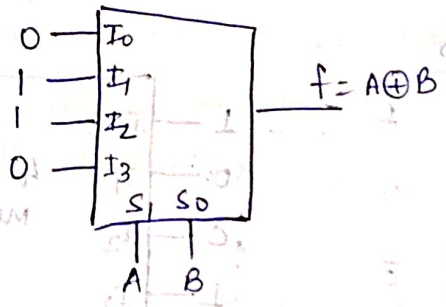
5) NAND gate.



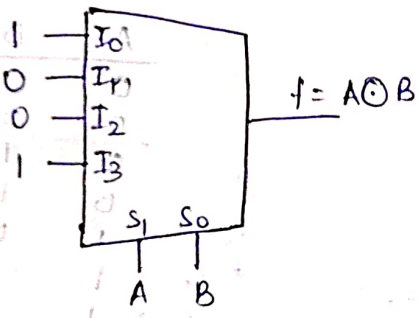
6) NOR gate.

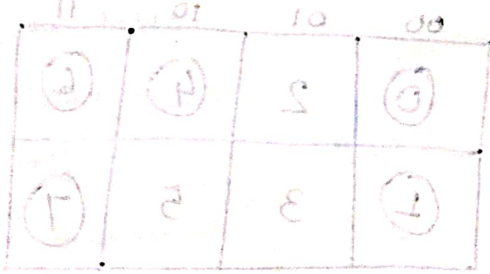


7) XOR gate



8) XNOR gate





Implementation of functions using multiplexer.

Case I: n variable function

$2^n : 1$ MUX

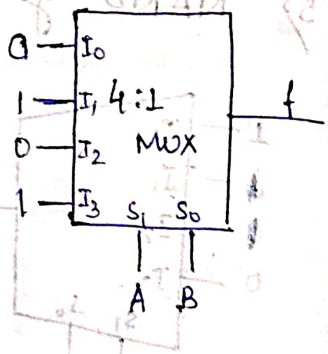
No. of variables = No. of select lines

All variables are connected to the select lines

Desired o/p is connected to the input lines.

Ex- $f(A,B) = \Sigma m(1,3)$ using 4:1 MUX

A	B	o/p
0	0	0
0	1	1
1	0	0
1	1	1

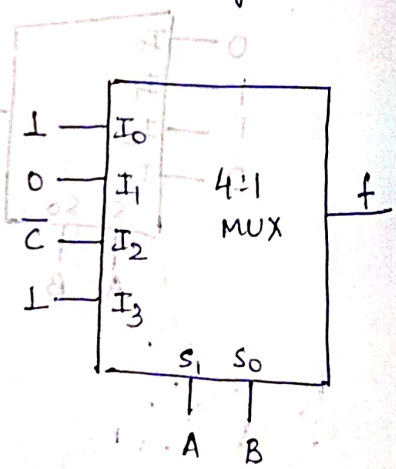


Case II: n variable function

$2^{n-1} : 1$ MUX

Ex- $f(A,B,C) = \Sigma m(0,1,4,6,7)$ using 4:1 MUX

A	B	C	o/p	
0	0	0	1	1
0	0	1	1	
0	1	0	0	
0	1	1	0	0
1	0	0	1	
1	0	1	0	$\bar{C}$
1	1	0	1	
1	1	1	1	1



variables in select lines.

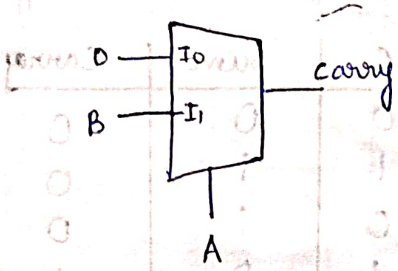
variable to be used as i/p

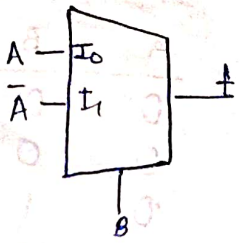
Implementation Table.

C \ AB	00	01	10	11
$\bar{C}$ 0	(0)	2	(4)	(6)
C 1	(1)	3	5	(7)
	1	0	$\bar{C}$	1

Half adder using 2:1 MUX

Input A	Input B	Output Sum	Output Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



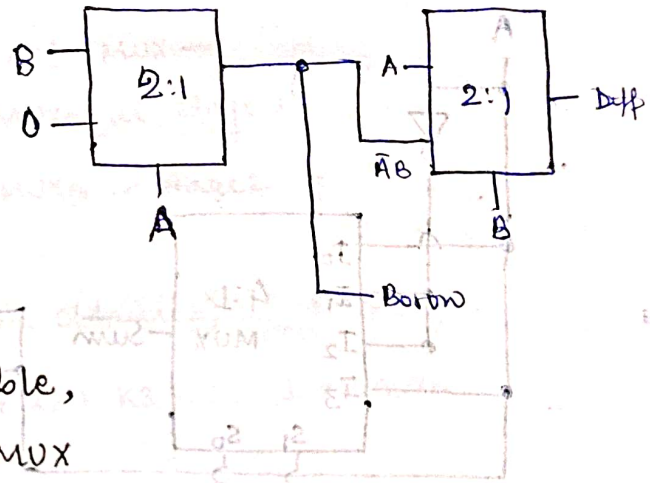


Sum = $A \oplus B$

Carry = AB

Half subtractor using 2:1 MUX

A	B	Diff	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0



Diff = $A \oplus B$, Borrow = $\bar{A}B$

** If complement is not available,
half adder requires - 3 2:1 MUX
half subtractor requires 2 2:1 MUX.

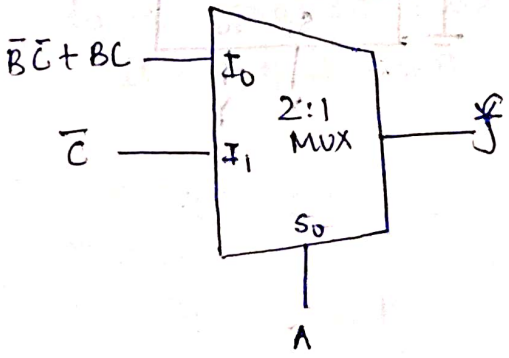
Case III : n variable function

2^{n-2} : 1 MUX

Ex- $F(A, B, C) = \Sigma m(0, 3, 4, 6)$ using 2:1 MUX

Implementation table

	$\bar{A}$	A
$\bar{B}\bar{C}$	(0)	(4)
$\bar{B}C$	1	5
$B\bar{C}$	2	(6)
BC	(3)	7
	$\bar{B}\bar{C}+BC$	$\bar{B}\bar{C}+B\bar{C}$
		$\bar{C}$



Implementation of full adder using multiplexer –

4:1 MUX
2 select line

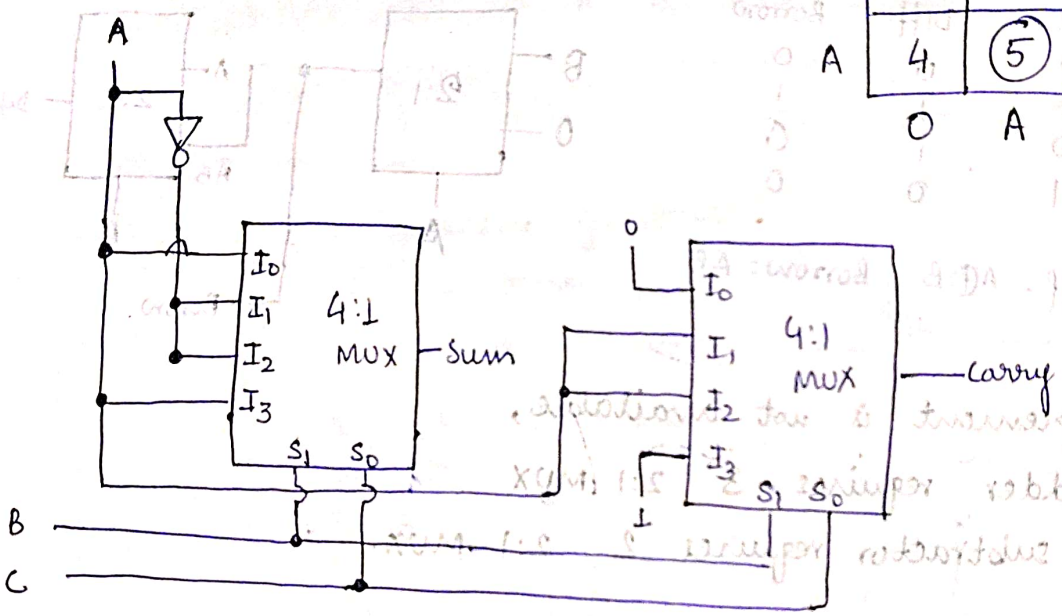
A	B	C	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Implementation table for sum

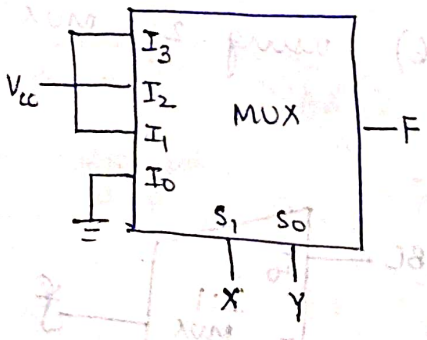
	$\bar{B}\bar{C}$	$\bar{B}C$	$B\bar{C}$	BC
$\bar{A}$	0	(1)	(2)	3
A	(4)	5	6	(7)
	A	$\bar{A}$	$\bar{A}$	A

Implementation table for carry

	$\bar{B}\bar{C}$	$\bar{B}C$	$B\bar{C}$	BC
$\bar{A}$	0	1	2	(3)
A	4	(5)	(6)	(7)
	0	A	A	1



Output of multiplexer –



$$\bar{X}\bar{Y}\cdot 0 + \bar{X}\cdot Y\cdot 1 + X\bar{Y}\cdot 1 + XY\cdot 1$$

$$= \bar{X}Y + X\bar{Y} + XY$$

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

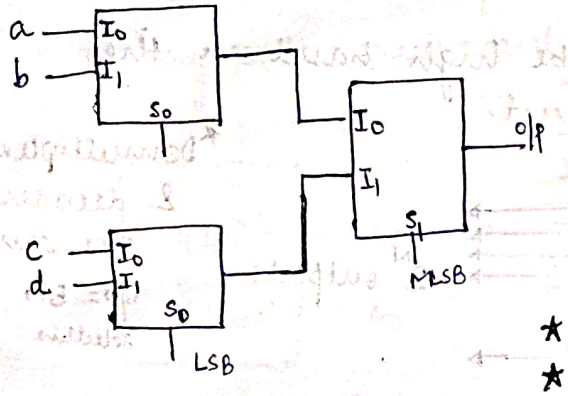
$$= \cancel{\bar{X}+Y+X}\ \bar{X}Y + X$$

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

$$= X+Y$$

Implementation of higher order multiplexer using lower order MUXes

1) 4:1 MUX using 2:1 MUX



S_1	S_0	f
0	0	a
0	1	b
1	0	c
1	1	d

★ First stage should use LSB as select lines
★ As we proceed further, more significant bits will be used as select lines.

General formula–

To implement B:1 MUX using A:1 MUX —

$B/A = K1$ → no. of MUXes in stage 1

$K1/A = K2$ → no of MUXes in stage 2

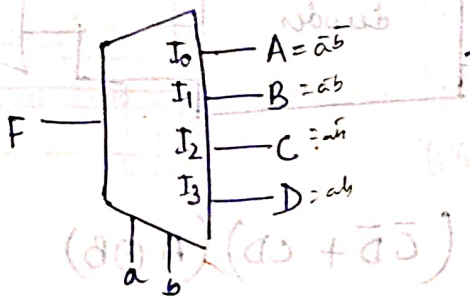
⋮

$K_{N-1}/A = K_N = 1$ (till we obtain 1)

No. of MUXes required = K1 + K2 + K3 + ... 1 MUXes

DEMULTIPLEXER

↦ takes one single input data line & then switches it to any of ~~the~~ a no. of individual o/p lines based on select lines.

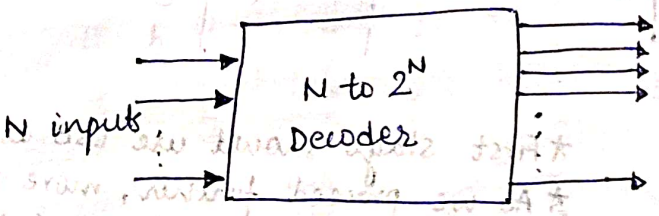


★ Converts serial ~~data~~ data signal at the input to a parallel data at its output lines.

DECODER

Combinational circuit that has n input lines and maximum 2^n output lines.

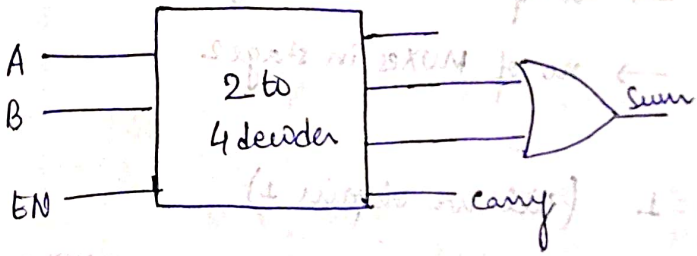
One of the o/p lines will be high based on the combination of inputs present.



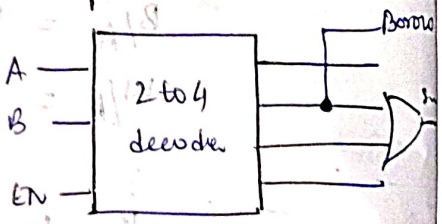
*Demultiplexer & decoder are same
i/p = EN
selection = i/p.

Implementation of functions using decoder.

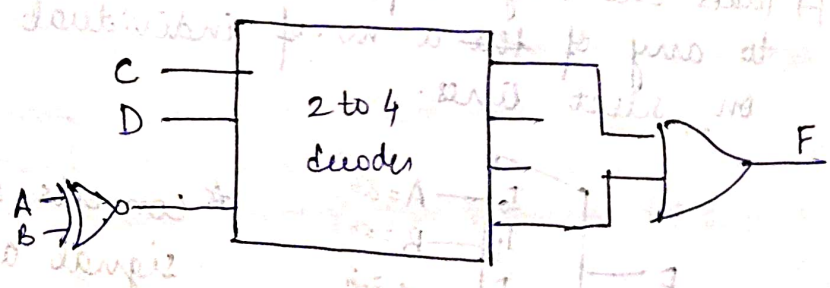
Half adder.



Half subtracter



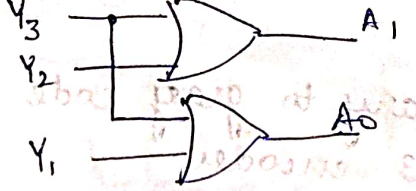
* If No. of variables > No. of inputs,
Then, few variables are used as enable input.



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

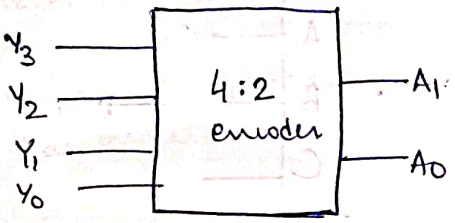
$$= (\bar{C}\bar{D} + CD)(\bar{A}\bar{B} + AB)$$

ENCODER



2^N inputs N outputs

binary code equivalent to the input.



Y_3	Y_2	Y_1	Y_0	A_1	A_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

$A_1 = Y_3 + Y_2$

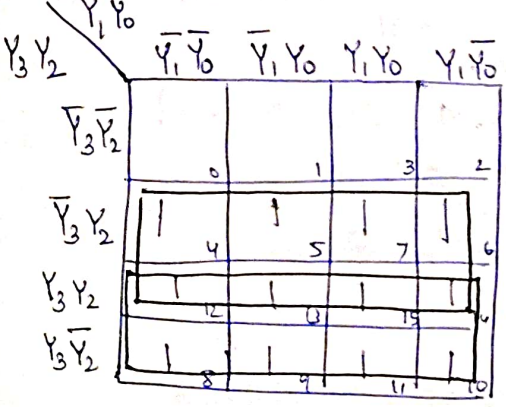
$A_0 = Y_1 + Y_3$

Priority encoder

If more than one i/p are 1, then, o/p depends upon the highest priority.

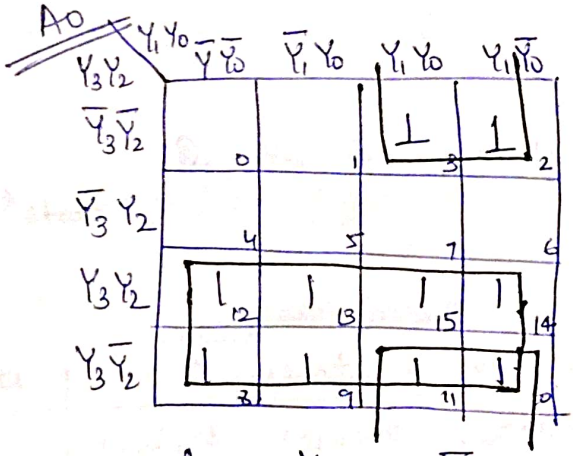
Y_3	Y_2	Y_1	Y_0	A_1	A_0	V
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

A_1



$A_1 = Y_2 + Y_3$

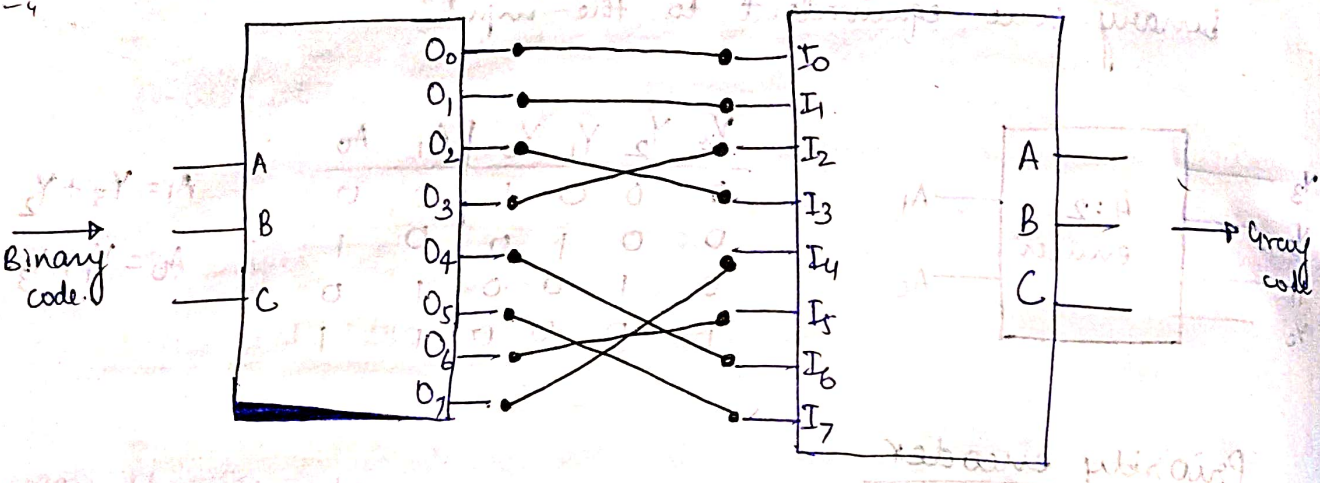
A_0

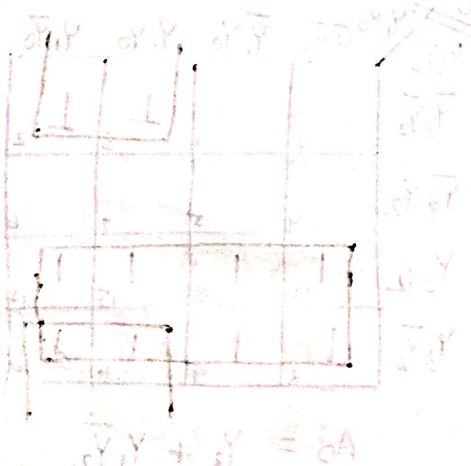


$A_0 = Y_3 + Y_1\bar{Y}_2$

0-0
1-1
2-3
3-2
4-6
5-7
6-5
7-4

Binary to gray code coverter using 3:8 decoder and 8:3 encoder

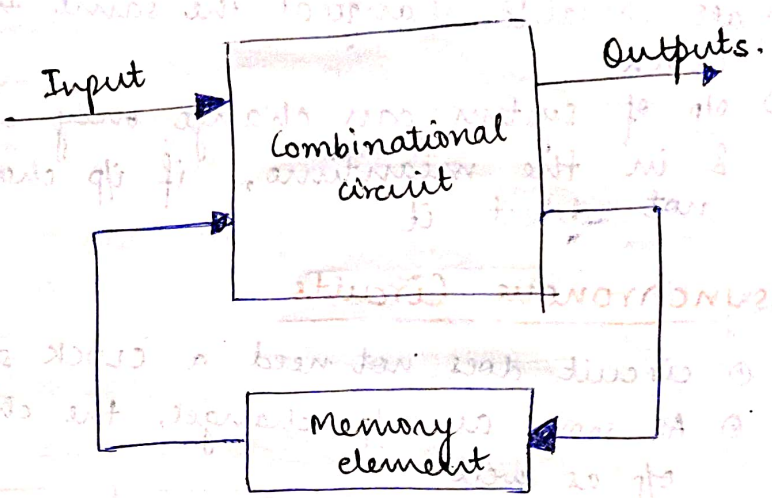




Sequential Circuit

Combinational logic –
output depends upon given combination of inputs.

Sequential logic –
output depends upon combination of inputs as well as present state of system



To store current state of system, in order to use it in next cycle, memory elements are needed.

memory elements → latches, flip flops.

State and State variable –

State :- present stored value in the system

State variable :- variable used to represent stored value in the system.

Ex → 3 bit counter

Q_2	Q_1	Q_0	
0	0	0	→ state
0	0	1	

Q_2, Q_1, Q_0 :- state variables

Combinational circuits	Sequential circuits
⊙ O/p only depends upon present state of inputs.	Output depends upon past output and present input.
⊙ No memory element required	⊙ Memory elements are required
⊙ Faster	⊙ slower
⊙ Easier to design	⊙ Difficult to design.

Sequential circuits

Synchronous ← → Asynchronous.

Synchronous Circuits

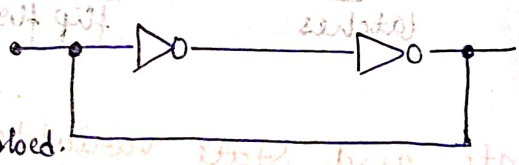
- all variables change at the same ~~by~~ time in synchronism with CLOCK.
- o/p of system can change only when clock signal arrives & in the meantime, if i/p changes, the system does not detect it.

Asynchronous Circuits

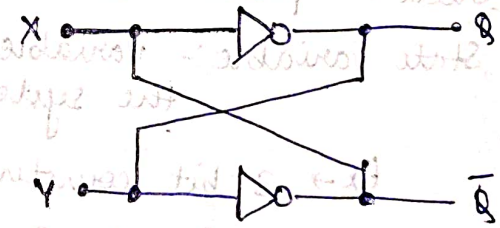
- circuit does not need a CLOCK signal to operate.
- As soon as i/p changes, the change is reflected to the o/p as well.

Bistable Multivibrator

↳ Both states 0 & 1 are stable & ckt can stay in any of these states for infinite time until disturbed.



In such a circuit, a particular value 0/1 gets locked/ stored/ latched.



* metastable state – undesirable state where ckt may get b/w 0 & 1.

[Ball Hill Analogy]

Stable state – The system remains in a state even after disturbance is applied.

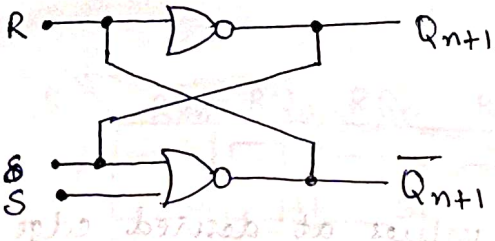
Unstable state – The system ~~and~~ automatically moves to other states even without disturbance.

Metastable state – The system remains in that state if ~~do~~ not disturbed. If disturbed, it changes state.

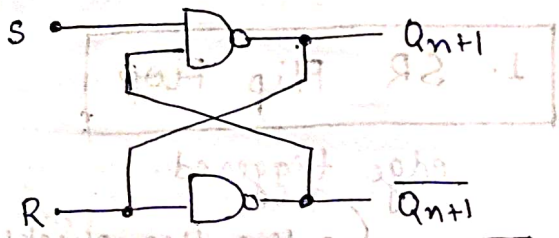
LATCHES

basic memory element that is used to store 1 bit of information!

Latches
- Active High – i/p becomes active when HIGH (using NOR gate)
- Active Low – i/p becomes active when LOW. (using NAND gate)



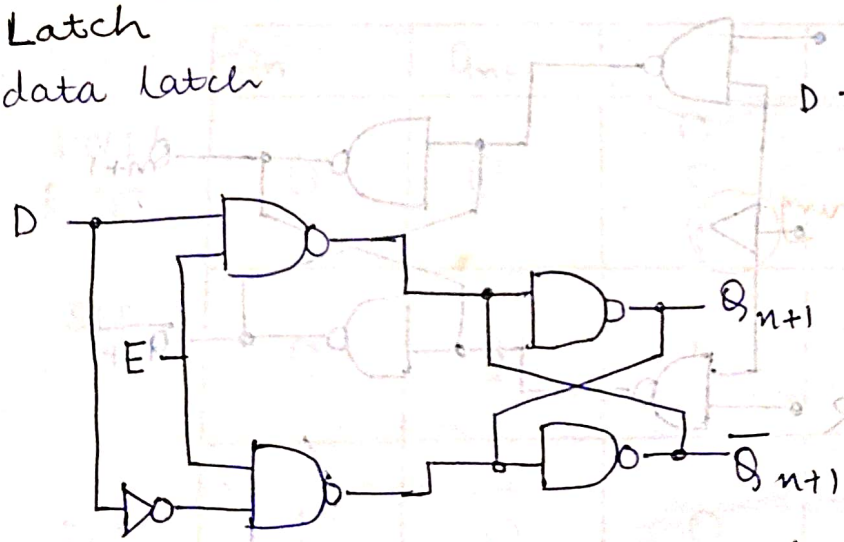
S	R	Q_{n+1}	$\overline{Q}_{n+1}$	State
0	0	Q_n	$\overline{Q}_n$	HOLD
0	1	0	1	RESET
1	0	1	0	SET
1	1	0	0	INVALID

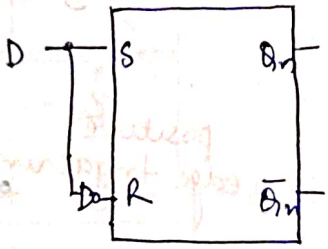


S	R	Q_{n+1}	$\overline{Q}_{n+1}$	State
0	0	1	1	INVALID
0	1	1	0	SET
1	0	0	1	RESET
1	1	Q_n	$\overline{Q}_{n+1}$	HOLD

D - Latch

data latch





↳ SR latch with enable i/p.

D	E	Q_{n+1}	$\overline{Q}_{n+1}$
X	0	Hold	Hold
0	1	0	1
1	1	1	0

↳ S = D R = $\overline{D}$

↳ also called transparant latch

$Q_{n+1} = D$

FLIP FLOPS

- SR Flip Flop
- JK Flip Flop
- D Flip Flop
- T Flip flop.

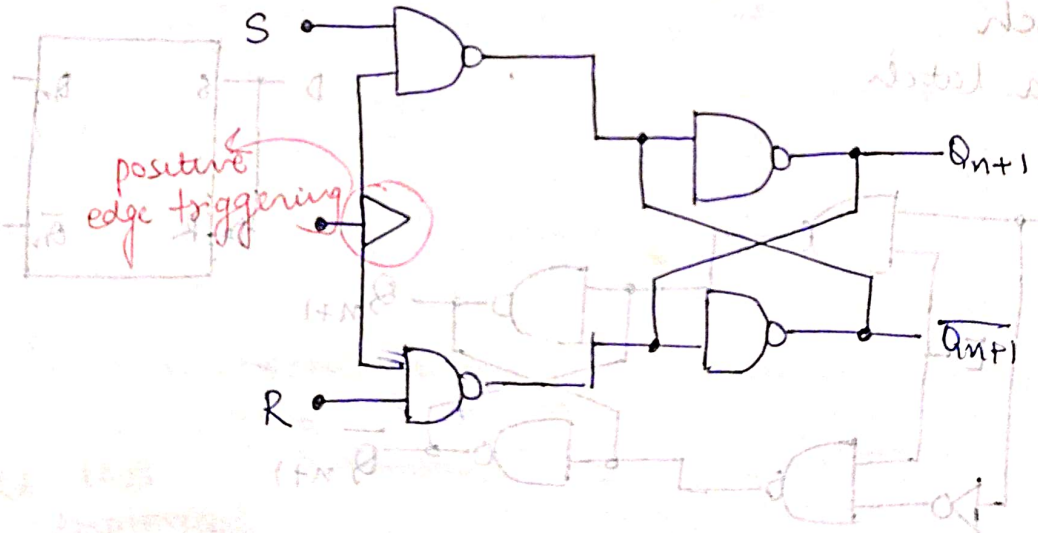
1. SR Flip Flop

edge triggered.

↳ flip flop checks the i/p values at desired edge of clock and produces corresponding o/p.

→ positive edge triggering

→ negative edge triggering.

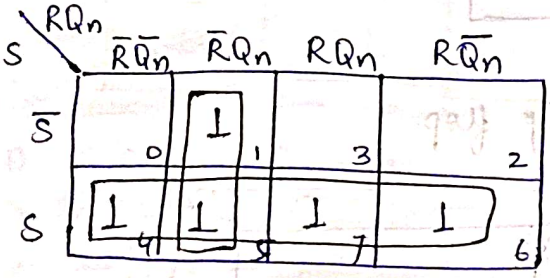


→ State Table

Clock	S	R	Q_{n+1}	$\overline{Q_{n+1}}$	
–	X	X	Q_n	$\overline{Q_n}$	HOLD
↑	0	0	Q_n	$\overline{Q_n}$	HOLD
↑	0	1	0	1	Reset
↑	1	0	1	0	Set
↑	1	1	1	1	Invalid.

↳ characteristic Table.

S	R	Q_n	Q_{n+1}	
0	0	0	0	} HOLD
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	1	} INVALID
1	1	1	1	



$Q_{n+1} = S + \bar{R}Q_n$

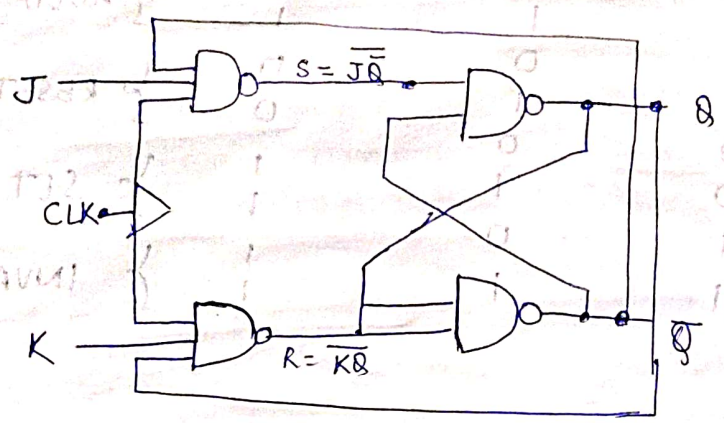
↑ characteristic equation

↳ Excitation Table

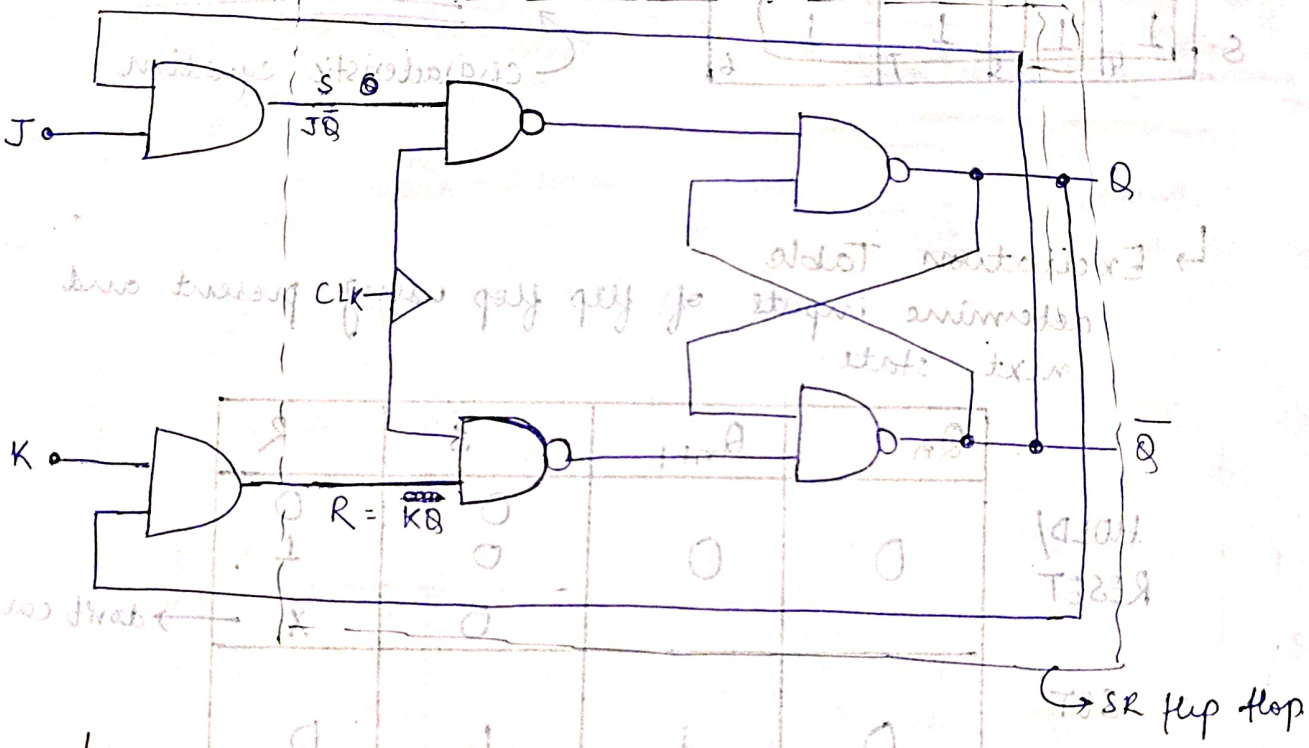
determine inputs of flip flop using present and next state.

	Q_n	Q_{n+1}	S	R
HOLD/ RESET	0	0	0	0
			0	1
			0	X → don't care
SET	0	1	1	0
RESET	1	0	0	1
HOLD/ SET	1	1	0	0
			1	0
			X	0

2. JK Flip Flop



JK flip flop using SR flip flop.



↳ SR flip flop

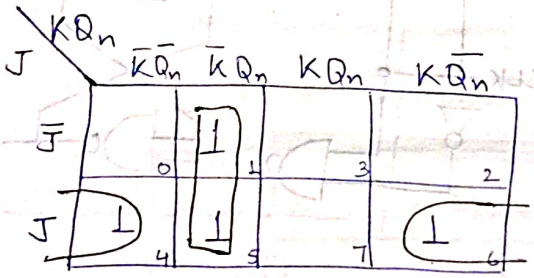
↳ State table

$S = J\overline{Q}$ $R = KQ$

J	K	S	R	Q_{n+1}	$\overline{Q}_{n+1}$	
0	0	0	0	Q_n	$\overline{Q}_n$	HOLD
0	1	0	1	0	1	RESET
1	0	1	0	1	0	SET
1	1	Toggle		$\overline{Q}_n$	Q_n	TOGGLE

↳ Characteristic Table –

J	K	Q_n	Q_{n+1}	
0	0	0	0	HOLD
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	1	TOGGLE
1	1	1	0	



$\therefore Q_{n+1} = \bar{K}Q_n + J\bar{Q}_n$

↳ Excitation Table –

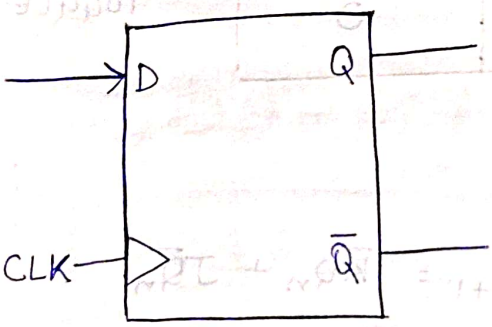
	Q_n	Q_{n+1}	J	K
HOLD/ RESET	0	0	0	0
			0	1
			0	X
TOGGLE/ SET	0	1	1	0
			1	1
			1	X
TOGGLE/ RESET	1	0	1	1
			0	1
			X	1
HOLD/ SET	1	1	0	0
			1	0
			X	0

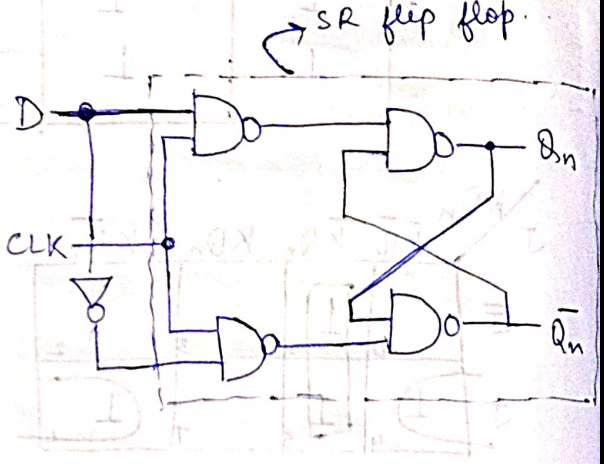
★ No race around condition in edge triggering

Condition for race around → $T_p > \frac{T_{CLK}}{2}$

3. D - Flip Flop

↳ Transparant flip flop.





↳ State Table.

D	CLK	S	R	Q_{n+1}	
X	—	X	X	Q_n	← HOLD
0	↑	0	1	0	← RESET
1	↑	1	0	1	← SET

★ The i/p D propagates as it is to the o/p so, it is called transparant flip flop.

↳ Characteristic Table.

CLK	D	Q_n	Q_{n+1}
–	X	0	0
–	X	1	1
↑	0	0	0
↑	0	1	0
↑	1	0	1
↑	1	1	1

$$Q_{n+1} = D$$

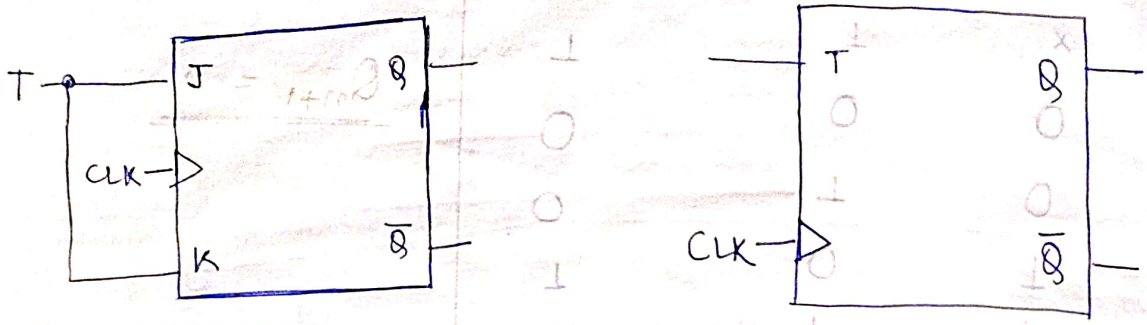
↳ Excitation Table.

Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

* if value of D changes at the same instant the clock edge arrives then, output is based on value of input <u>just before</u> the edge.

4. T - Flip Flop

↳ Toggle Flip Flop.



↳ State Table.

CLK	T	J	K	Q_{n+1}	
—	X	X	X	Q_n	
↑	0	0	0	Q_n	← HOLD
↑	1	1	1	$\overline{Q_n}$	← TOGGLE

↳ Characteristic Table.

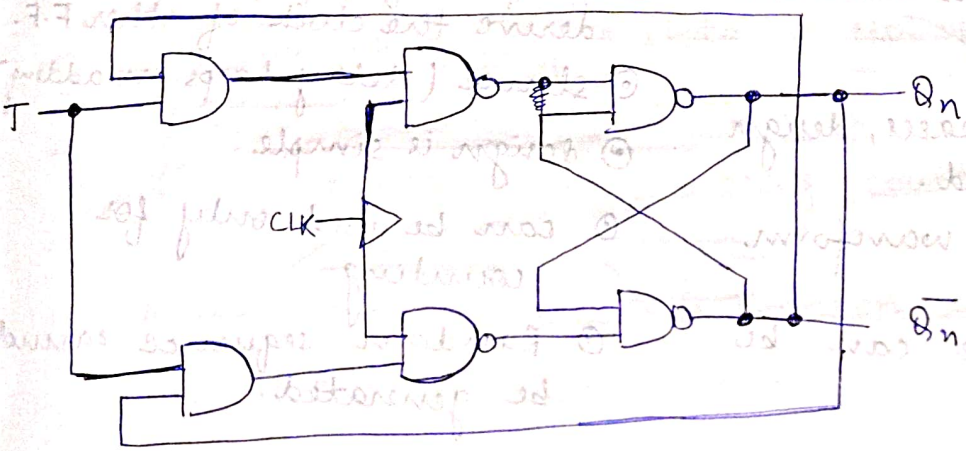
CLK	T	Q_n	Q_{n+1}
—	X	0	0
—	X	1	1
↑	0	0	0
↑	0	1	1
↑	1	0	1
↑	1	1	0

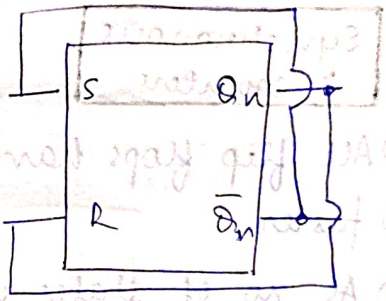
$Q_{n+1} = T \oplus Q_n$

↳ Excitation Table:

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Toggle circuit using SR flip flop –





Q_n	Q_{n+1}	S	R
0	1	1	0
1	0	0	1

* All flip flops are free from race around (because they are edge triggered)

* D-flip flops are used in buffer registers because they can store data temporarily (no hold state)

Counters (Application of flip flop)

Type of sequential circuit whose state represents the number of clock pulses given and can be used to count the number of clock pulses.

Types of counters

Synchronous counters

- All flip flops have the same clock.
- faster
- As no. of states increases, design becomes complicated.
- For counting and waveform generation.
- Random sequence can be generated.

Asynchronous counters

- o/p of one flip flop is used to derive the clock of other F.F
- slower (delay keeps on addi
- design is simple.
- can be used only for counting
- Random sequence cannot be generated.

Important Terminology

1. **Up Counter :-**
   at every clock pulse, the count increases until it reaches maximum.

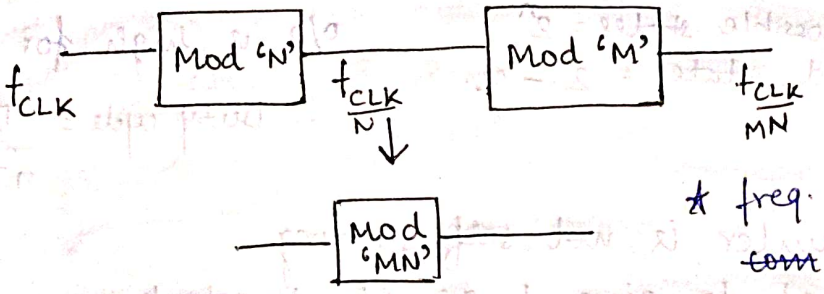
2. **Down Counter :-**
   at every clock pulse, count reduces by 1 until it reaches minimum.

3. **Modulus of a counter :-**
   number of states in the counting sequence of counter.

4. **Lock out :-**
   condition when the counter is struck in a sequence of states which are not part of the counting sequence.

- it cannot come back to desired states on its own.

- When 2 counters are cascaded, the overall modulus of combination is product of modulus of 2 counters



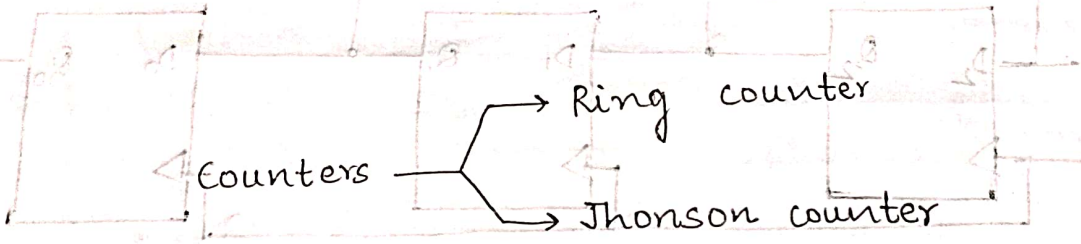
* freq. of o/p of mod 'N' ~~com~~ counter is $\frac{f_{CLK}}{N}$.

Asynchronous counter

With 'n' flip flops,

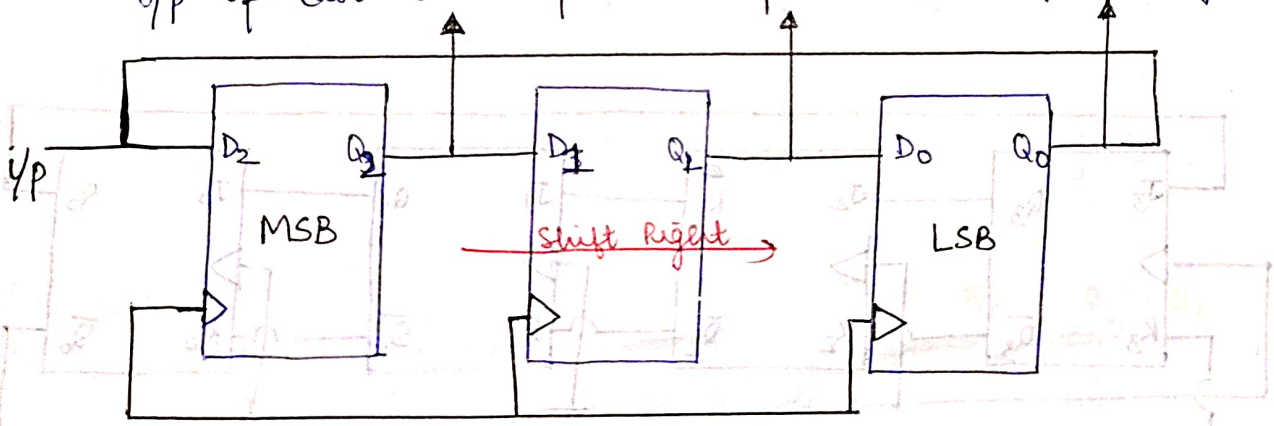
Max. ~~non~~ number of possible states = 2^n

The max. no. of possible states $\geq$ No. of desired states.



1. Ring counter

o/p of last bit is fed as input to the first stage.



also called shift register counter.

- only one o/p can be high at a time.
- Number of states = No. of flip flops.

- n bit ring counter.
  - ↳ n flip flops
  - ↳ no. of states
- total no. of possible states = 2^n
- No. of unused states = $2^n - n$

Period of each o/p = nT_{CLK}

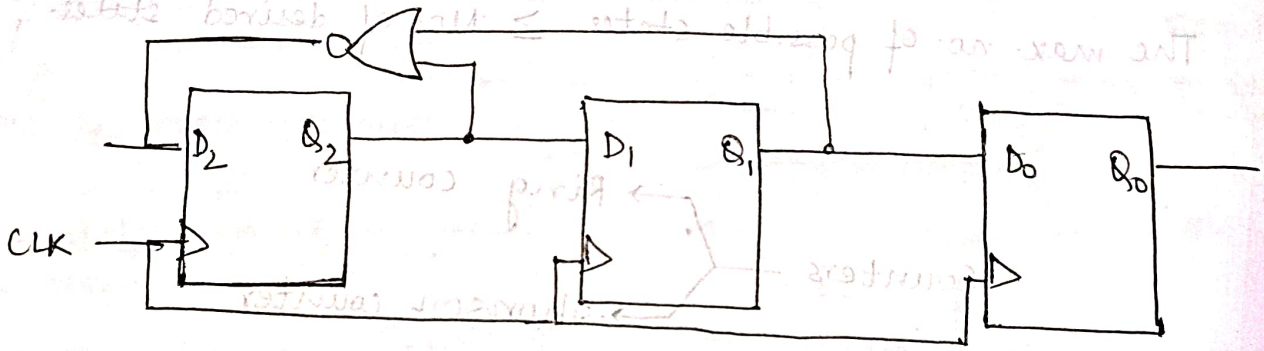
o/p is high for T_{CLK}

∴ Duty cycle = $\frac{T_{CLK}}{nT_{CLK}} = \frac{1}{n}$

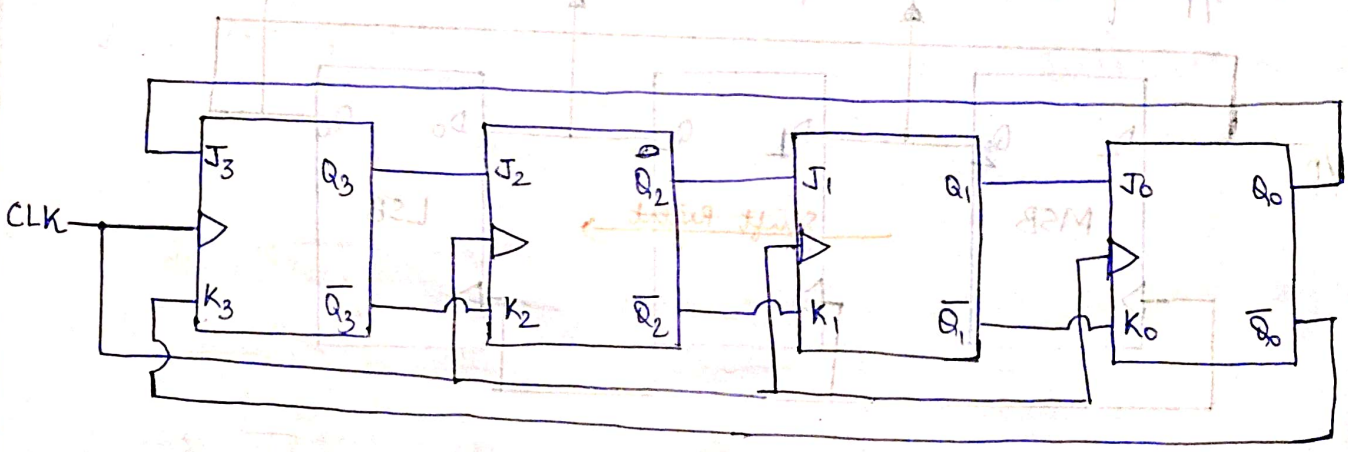
Ex:

* Ring counter is not self starting. We need to give 1 as i/p to start the counting sequence.

Self starting ring counter.



Ring counter with JK flip flop.



3 bit RING counter states

1 0 0

0 1 0

0 0 1

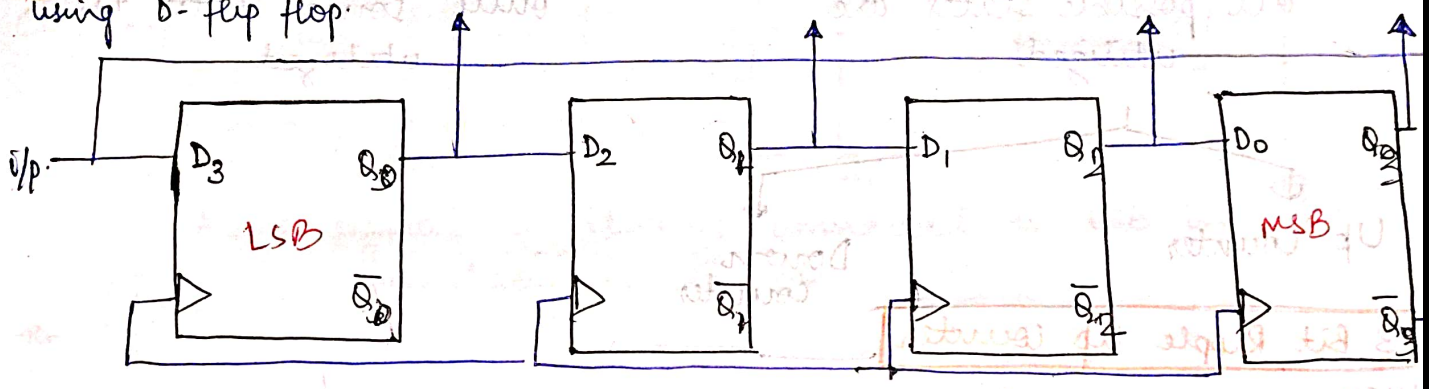
2. Jhonson counter

also known as Twisted Ring counter
Switched Tail counter
Mobius counter.

Logic diagram —

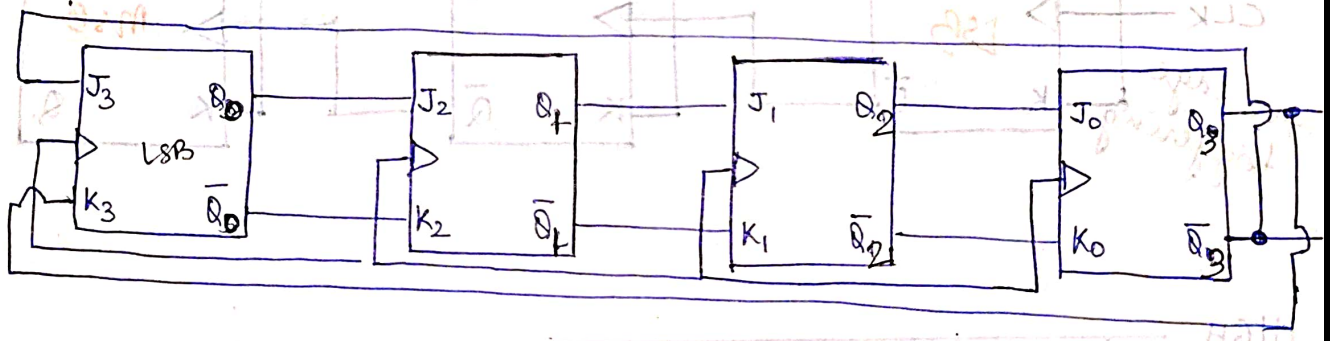
↳ only one change w.r.t. ring counter is that complement of o/p of last stage is fed as i/p to the first stage.

using D-flip flop.



using JK flip flop.

$J_3 = \overline{Q_0}$ & $K_3 = Q_0$.



For n flip flops,

no. of used states = $2n$

unused states = $2^n - 2n$

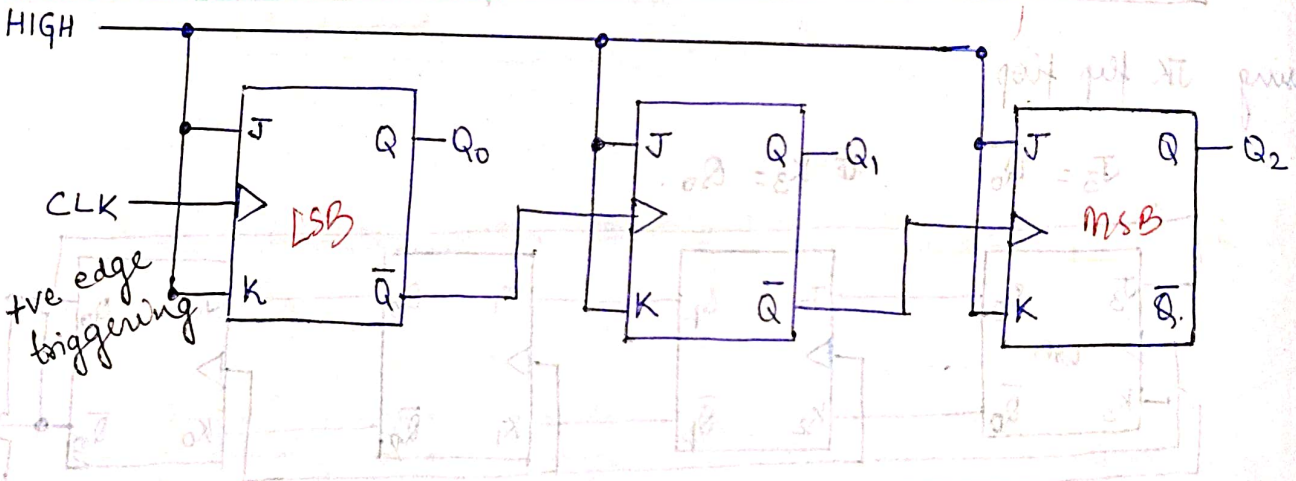
	Q_0	Q_1	Q_2	Q_3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

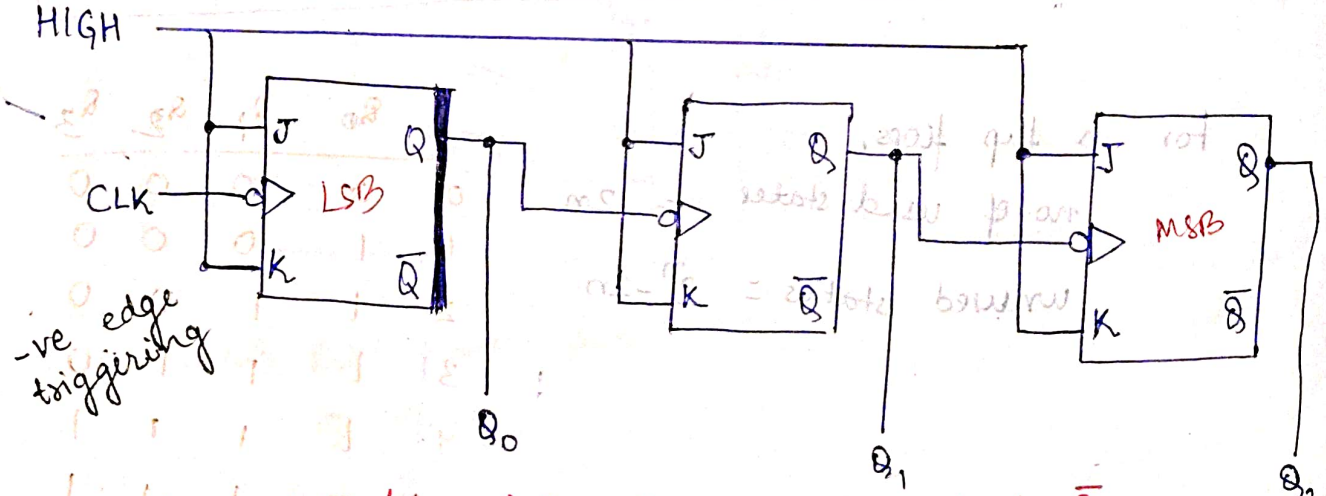
Asynchronous counters

In asynchronous counters, the output of previous state acts as the input to the next state.

- **Binary**
  - MOD = 2^n
  - all possible states are utilized
  - **Up Counter**
  - **Down Counter.**
- **Non binary**
  - MOD < 2^n
  - only some states are utilized.

3 Bit Ripple Up counter





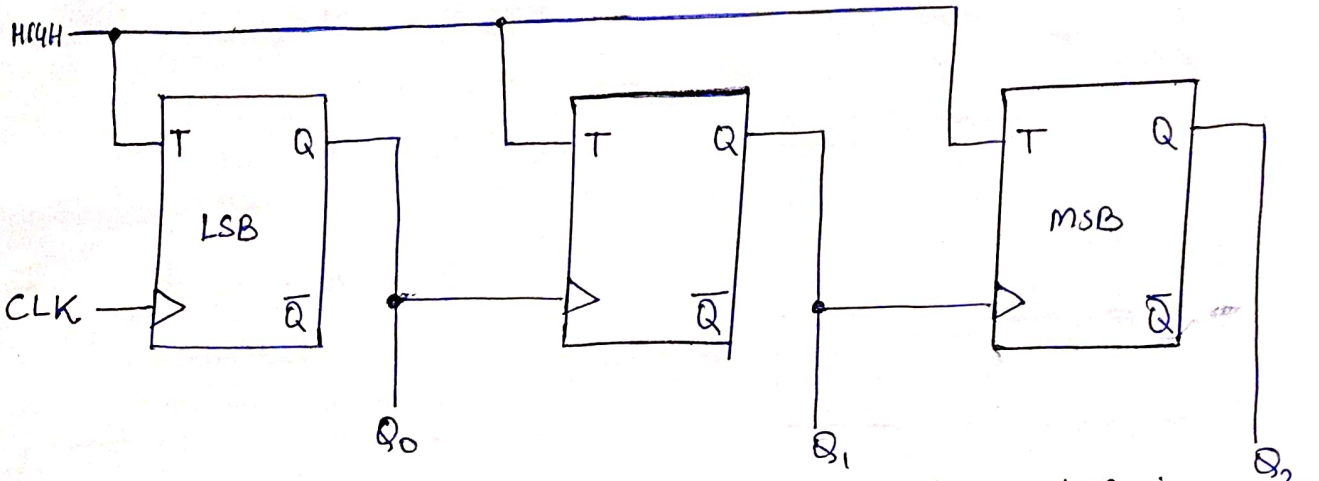
+ve edge triggering → CLK is connected to $\bar{Q}$

-ve edge triggering → CLK is connected to Q

all flip flops operate in toggle mode.

3 Bit Ripple Down Counter

Triggering logic is opposite to an up counter.
positive edge triggered → CLK is connected to Q
negative edge triggered → CLK is connected to $\overline{Q}$



*CLK signal is always connected to LSB in asynchronous circuit.

delay = no. of bits toggling * t_{pd}

DIGITAL LOGIC

Principle of duality –

If expression contains only AND OR and NOT, dual of expression can be obtained by

- ⊙ changing AND to OR and OR to AND
- ⊙ changing ⊥ to 0 and 0 to ⊥
- ⊙ variables and their complement remain unchanged

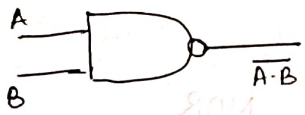
Consensus Theorem

- ↦ 3 variables in expression
- ↦ each variable is repeated twice
- ↦ one variable is present in complemented & uncomplemented form.

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

→ Terms containing complemented variable.

Logic gates



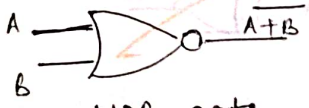
NAND gate.

Commutative

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

Not associative.

$\overline{\overline{A \cdot B} \cdot C} \neq \overline{A \cdot \overline{B \cdot C}}$



NOR gate

Commutative

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

Not associative.

$\overline{\overline{A+B}+C} \neq \overline{A+\overline{B+C}}$

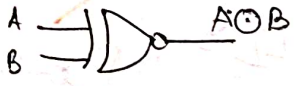
XOR gate

Commutative ✓

$A \oplus B = B \oplus A$

Associative ✓

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



XNOR gate

Commutative ✓

$A \odot B = B \odot A$

Associative ✗

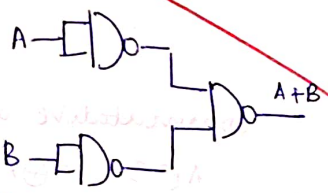
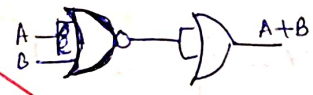
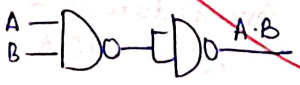
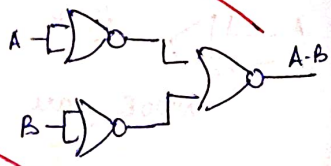
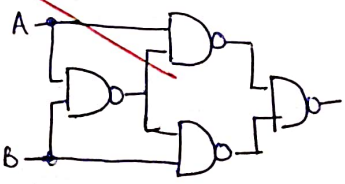
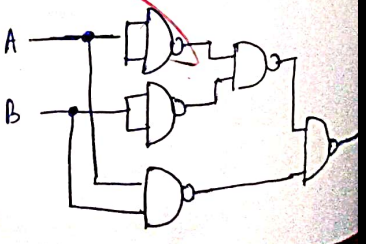
$A \odot (B \odot C) \neq (A \odot B) \odot C$.

* NAND and NOR gates are called universal gates.

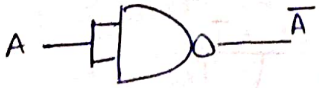
* NOR gates have advantages over NAND gates
  - ↳ becoz of simpler transistor structure, NOR gates are faster than NAND gates.
  - ↳ NOR gates consume less energy than NAND gates. ∴ beneficial for low power applications

* Advantages of NAND over NOR –
  - ↳ NAND has higher memory capacity than NOR.
  - ↳ NOR is more expensive than NAND
  - ↳ size of NOR is greater than size of NAND
  - ↳ all transistors of NAND are of equal sizes where as NOR gates don't.

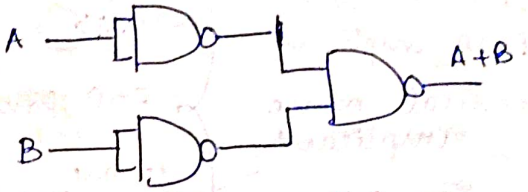
because of these reasons, NAND is preferred in industry.

Logic gate.	NAND	NOR
NOT A		
A+B		
A·B		
A⊕B		

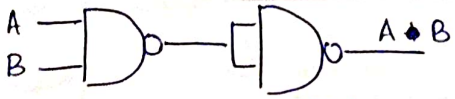
NAND gate circuits —



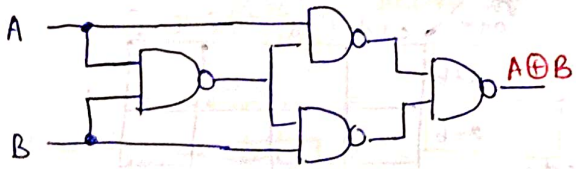
NOT gate (1)



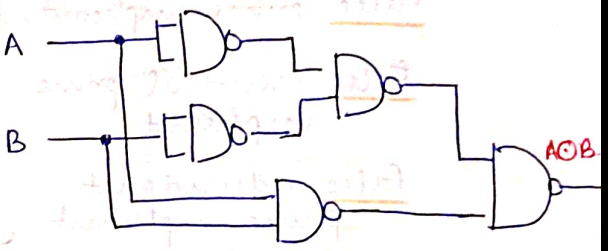
OR gate (3)



AND gate (2)

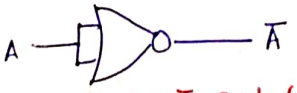


XOR gate (4)

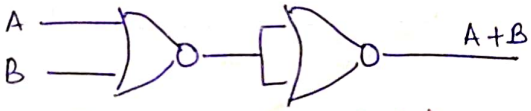


XNOR gate (5)

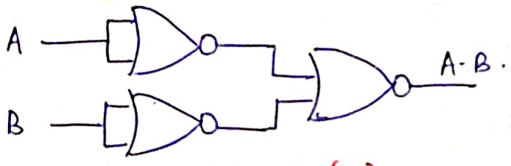
NOR gate circuits



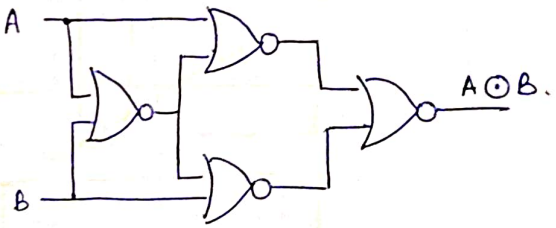
NOT gate (1)



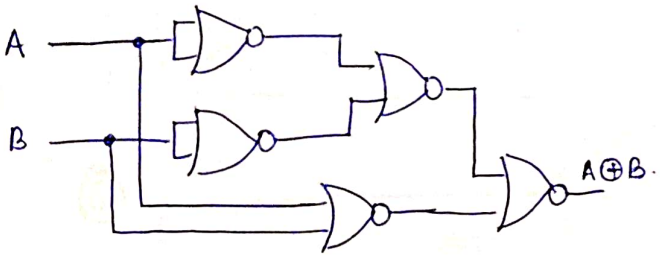
OR gate (2)



AND gate (3)



XNOR ~~gate~~ gate (4)

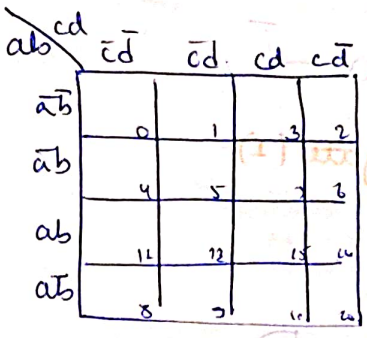


XOR gate (5).

K-Map important

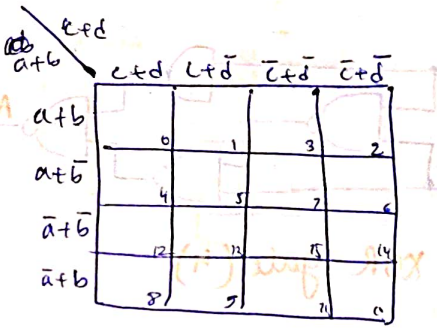
- Implicant
- Prime implicant
- Essential prime implicant
- Redundant prime implicant

} SOP form 1s are considered



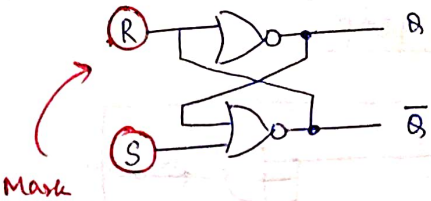
- False implicant
- False prime implicant
- False essential prime implicant
- False redundant prime implicant

} POS form 0s are considered.



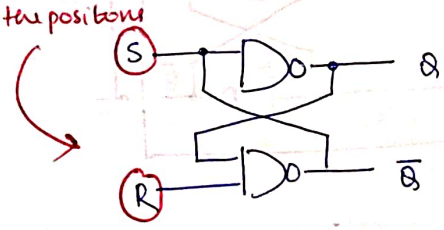
Sequential circuits

NOR gate – Active High
NAND gate – Active Low



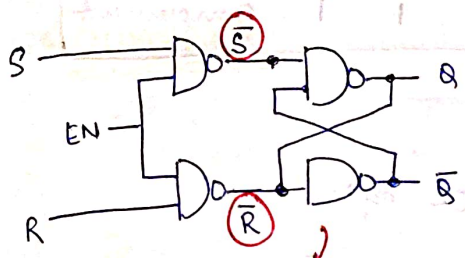
S	R	Q	$\bar{Q}$
0	0	No change	
0	1	0	1
1	0	1	0
1	1	—	—

Mark the positions



S	R	Q	$\bar{Q}$
0	0	—	—
0	1	1	0
1	0	0	1
1	1	No change	

Using ENABLE input & NAND gates

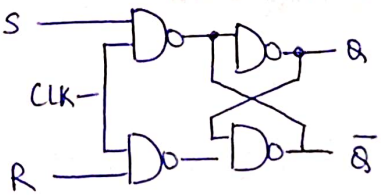


behaves as ACTIVE HIGH.

① Truth Table / Functions Table
② characteristic table
③ characteristic equation
④ Excitation Table.

1. SR Flip flop

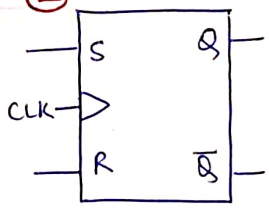
①



③

CLK	S	R	Q	$\bar{Q}$
0	X	X	No change	
↑	0	0	No change	
↑	0	1	0	1
↑	1	0	1	0
↑	1	1	—	—

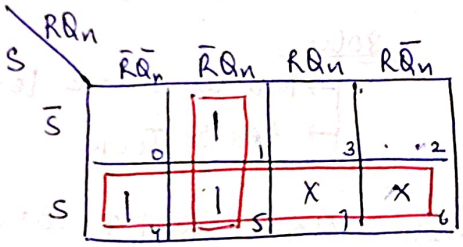
②



④

S	R	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	—
1	1	1	—

⑤ Characteristic equation —

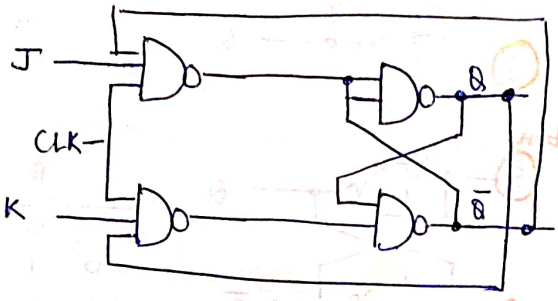


$\therefore$ $Q_{n+1} = S + Q_n\bar{R}$

⑥

Q_n	Q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

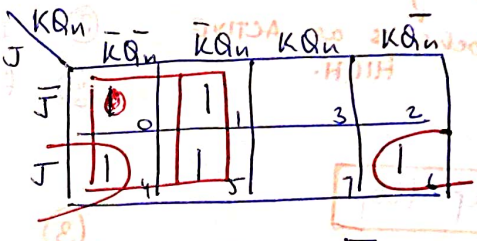
2. JK Flip flop



CLK	J	K	Q	$\bar{Q}$
0	X	X	No change	
↑	0	0	No change	
↑	0	1	0	1
↑	1	0	1	0
↑	1	1	Complement	

J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Characteristic eqn.



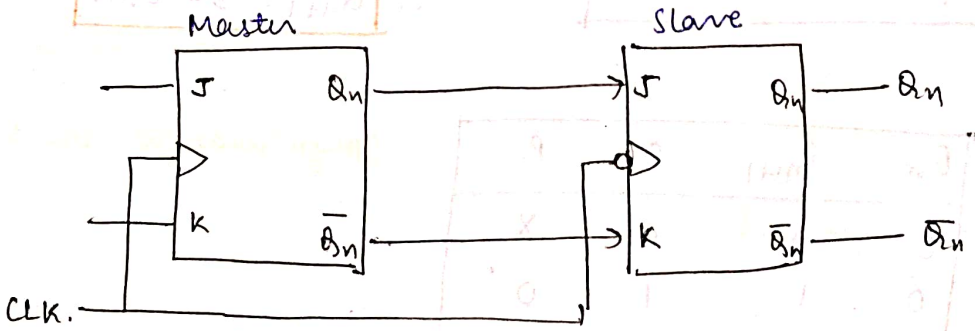
$$Q_{n+1} = \bar{K}Q_n + J\bar{Q}_n$$

★ Race around condition occurs in level triggered flip flop when $J = K = 1$ and $t_{pd} << T_{CLK}$

the state of flip flop oscillates b/w 0 and 1.

Soln
- Master slave level triggered flip flop
- $t_{pd} > T_{CLK}$.

Master slave configuration —



When CLK = 1, master is in operating mode

When CLK = 0, slave is in operating mode

(-ve level triggered)

In master slave FFs —
↳ o/p may change only once
↳ change in o/p is triggered by -ve edge of clock.
↳ change may occur only during clock's -ve level.

Counters

⊙ sequential logic circuit capable of counting no. of clock pulses arriving at its CLK input.

⊙ can be used as frequency divider.

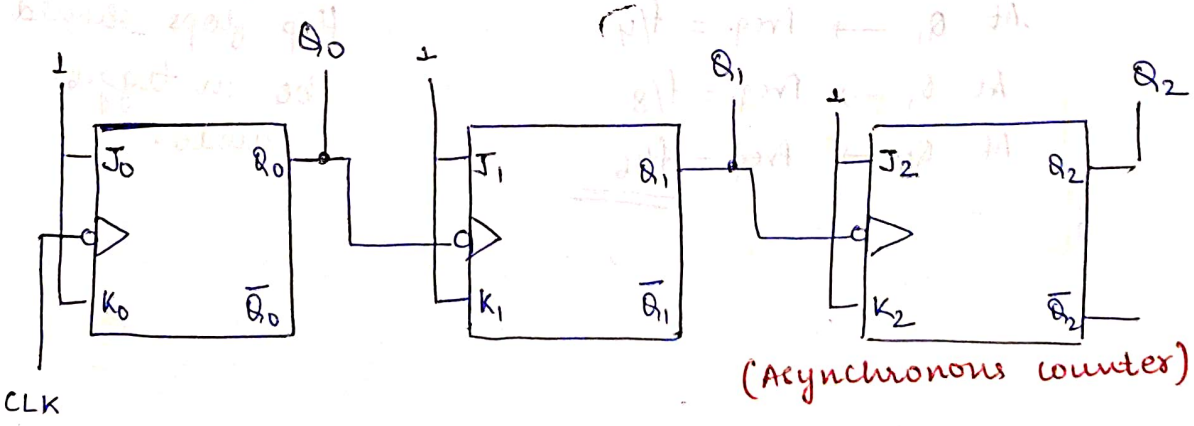
with n FFs, no. of possible states is

$N \leq 2^n$

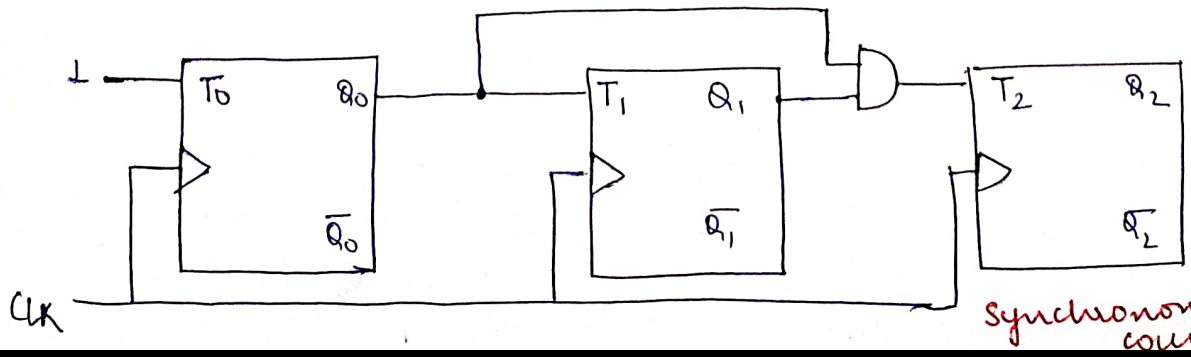
Mod of counter — No of states — No. of flip flops.

Depending upon clock pulse applied,

Counters →
- Synchronous counter (~~Ripple Counter~~) ⊙ fast ⊙ complex implem— if no of states ↑.
- Asynchronous counter (Ripple counter) ⊙ slow ⊙ implementation is simple.



(Asynchronous counter)



Synchronous counter.

Imp.

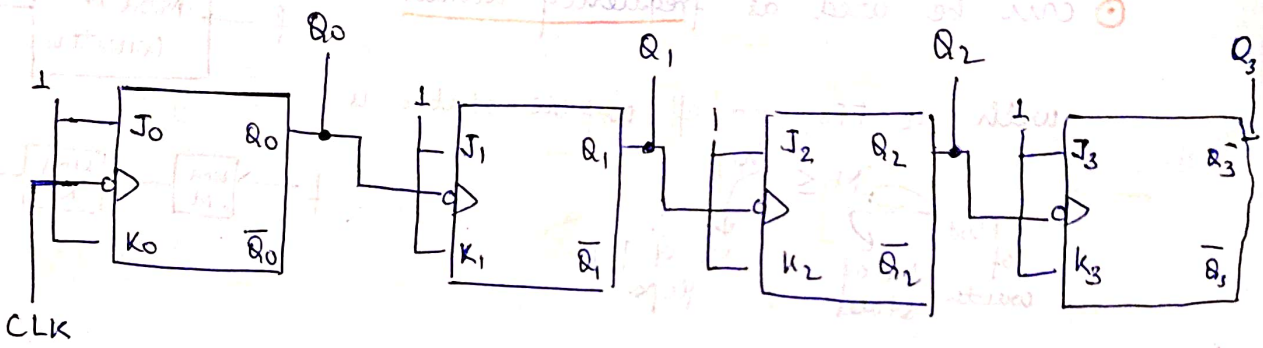
① -ve edge triggered FF + Q as clock — up counter

② -ve edge triggered FF + $\bar{Q}$ as clock — down counter

③ +ve edge triggered FF + Q as clock — down counter

④ +ve edge triggered FF + $\bar{Q}$ as clock — up counter

4 bit ripple counter



-ve edge triggered flip flops.

Q_0 is connected to clock

∴ up counter

Counting seq. → 0 → 1 → 2 → 3 ---- → 14 → 15

At Q_0 → Freq. = $f/2$

At Q_1 → Freq. = $f/4$

At Q_2 → Freq. = $f/8$

At Q_3 → Freq. = $f/16$

For ripple counter, flip flops should be in toggle mode.