1 LOGIC GATE

1.1. Logic Operations

In Boolean algebra, all the algebraic functions performed is logical. The AND, OR and NOT are the basic operations that are performed in Boolean algebra. There are some derived operations such as NAND, NOR, EX-OR, EX-NOR that are also performed in Boolean algebra.

1.1. NOT Operation

Symbol:

Fig. 1.1.

$A \xrightarrow{\text{NOT}} \bar{A}$ or A' (Complementation law)

and $\bar{\bar{A}} = A \Rightarrow$ Double complementation law

Truth table for NOT operation

Input A	Output $Y = \bar{A}$
0	1
1	0

A NOT gate can be represented using switch whose circuit representation is shown in figure below.

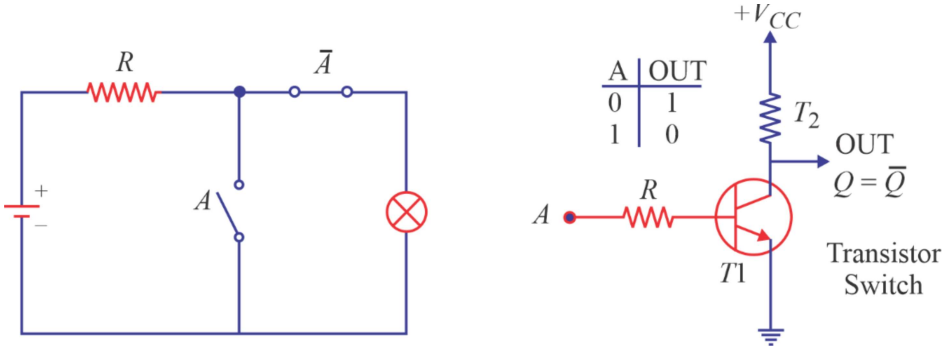
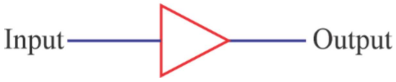


Fig. 1.2.

A buffer is a basic logic gate that passes its input, unchanged, to its output. Its behaviour is the opposite of a NOT gate.

The main purpose of a buffer is to regenerate the input, usually using a strong high and a strong low. Buffers are also used to increase the propagation delay of circuits by driving the large capacitive loads.

"Buffer" gate



Input	Output
0	0
1	1

1.1.2. AND Operation

Symbol:

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

Truth table for AND operation:

Input		Output
A	**B**	**Y = AB**
0	0	0
0	1	0
1	0	0
1	1	1

1.1.3. OR Operation

Symbol:

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

Truth table for OR operation:

Input		Output
A	**B**	**Y = A+B**
0	0	0
0	1	1
1	0	1
1	1	1

Example: Reduce the combinational logic circuit shown figure such that the desired output can be obtained using only one gate.

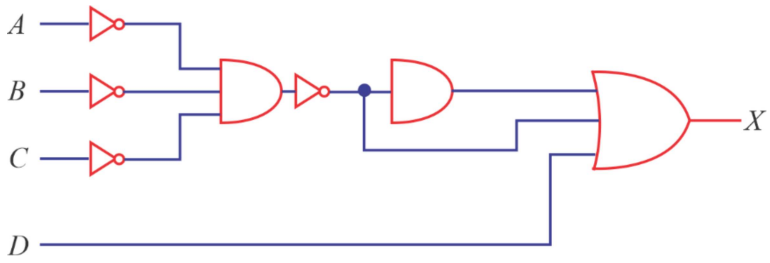


Fig. 1.3.

Solution:

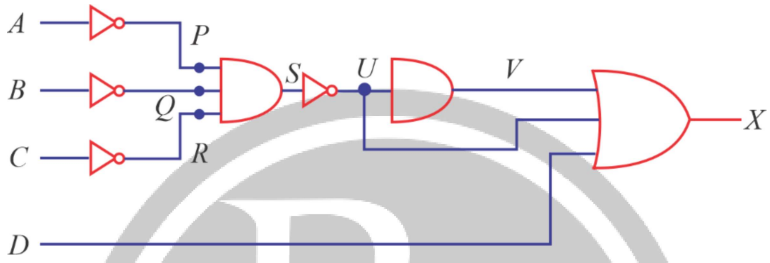


Fig. 1.4.

$$P = \overline{A},\ Q = \overline{B},\ R = \overline{C}$$

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

$$V = U = \overline{\overline{A}\overline{B}\overline{C}}$$

$$X = U + V + D$$

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

$$= A + B + C + D$$

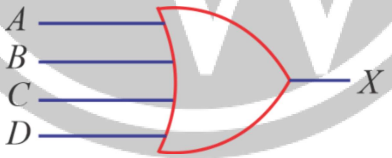


Fig. 1.5.

Enable or Disable Input:

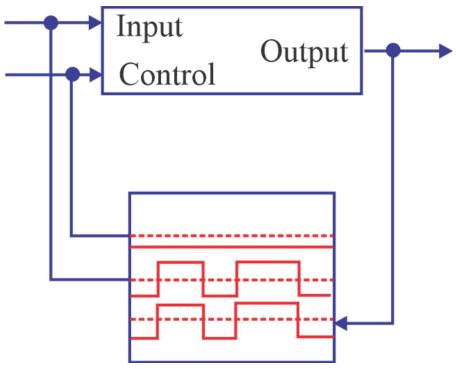


Fig. 1.6.

Enable:

- Allow a signal to pass when the control signal is HIGH.
- Prevent a signal from passing when the control signal is LOW.

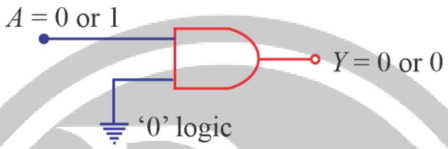
Disable:

- Prevent a signal from passing when the control signal is HIGH.
- Allow a signal to pass when the control signal is LOW.
- Enable and Disable Functions:
- AND and OR gates can both be used to enable or disable a transmitted waveform.

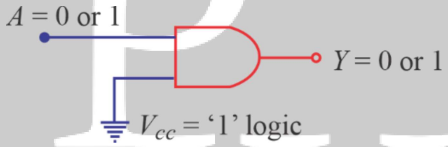
For a two input AND gate:

For a two input OR gate:

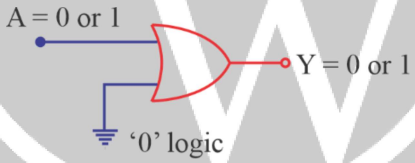
- **Control '0' disable**



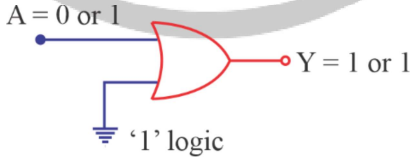
- **Control '1' enable (Buffer)**

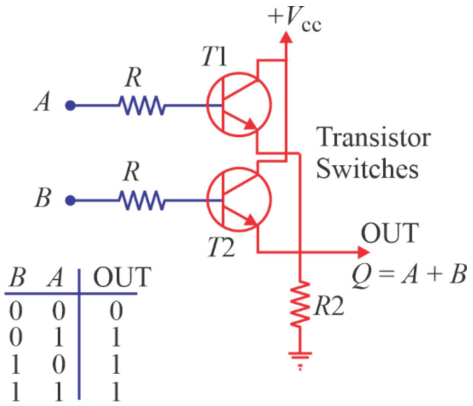


- **Control '0' enable (Buffer)**



- **Control '1' Always enable**





1.1.4. Switch Diagram for AND/OR gate

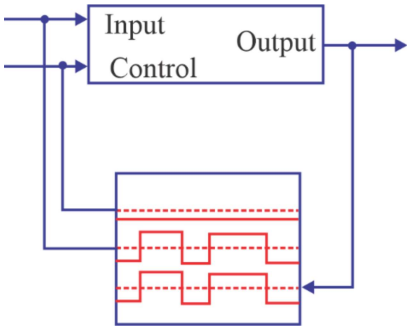


Fig. 1.7.

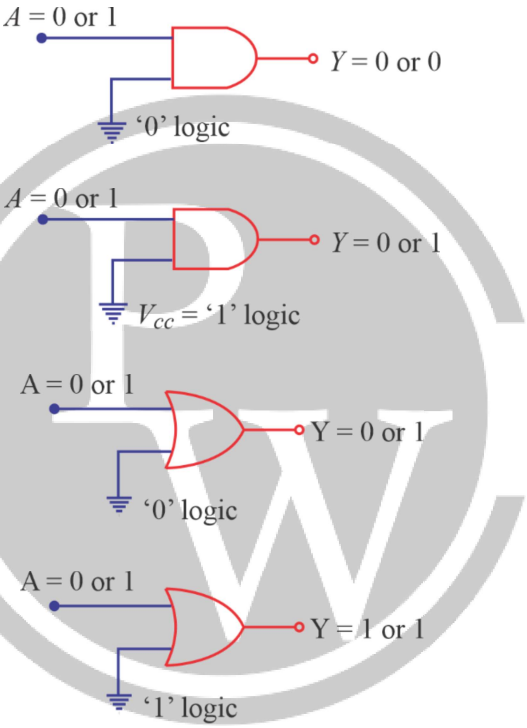


Fig. 1.8.

1.1.5. Basic law applications in AND/OR gate.

(a) Commutative Law:

The commutative law allows change in position of AND or OR variables. There are two commutative laws.

$$A + B = B + A$$

$$A \times B = B \times A$$

(b) Associative Law:

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

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

1.1.6. Circuit Diagram for AND/OR gate.

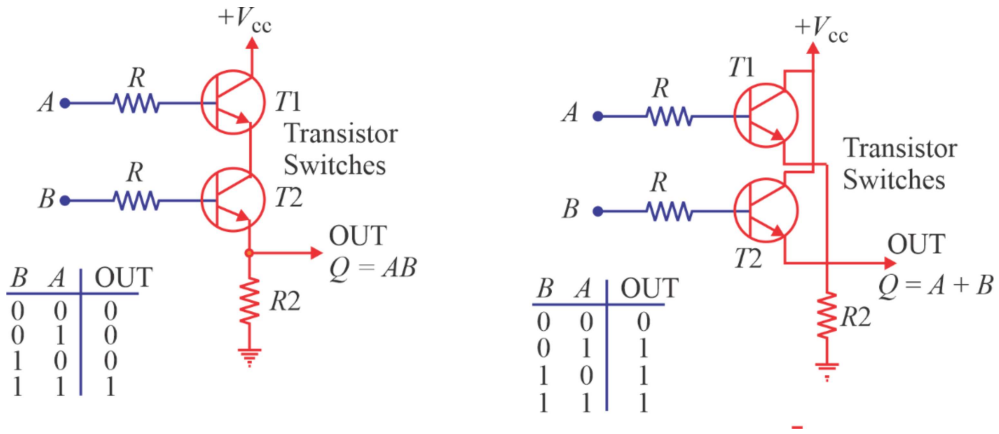


Fig. 1.9.

1.1.7. Switch Diagram for AND/OR Gate

The circuit shown below shows the switch representation of AND gate which is basically the series connection of switches A and B.

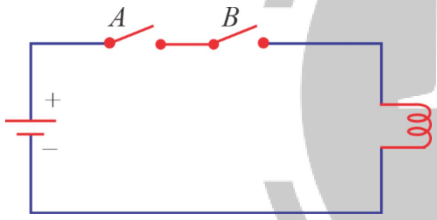


Fig. 1.10.

The circuit shown below shows the switch representation of OR gate which is basically the parallel connection of switches A and B.

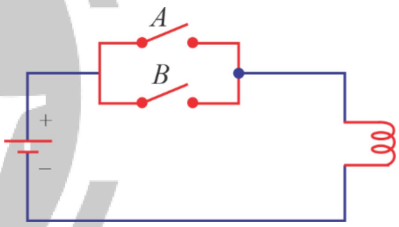
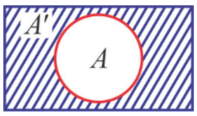
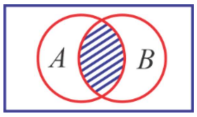
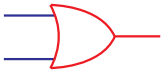


Fig. 1.11.

1.1.8. Venn Diagram:

NOT			$\bar{A}$	A		Output
				0 1		1 0
AND			A.B	A	B	Output
				0 0 1 1	0 1 0 1	0 0 0 1
OR			A + B	A	B	Output
				0 0 1 1	0 1 0 1	0 1 1 1

1.2. Logic Gates

Logic gates are the fundamental building blocks of digital systems.

Types of logic gates: There are three basic logic gates, namely

- OR gate
- AND gate
- NOT gate

And other logic gates that are derived from these basic gates are:

- NAND gate
- NOR gate
- Exclusive OR gate
- Exclusive NOR gate

1.2.1. NAND gate:

The term NAND gate equivalent to AND gate followed by a NOT gate, implies NOT-AND

Symbol:

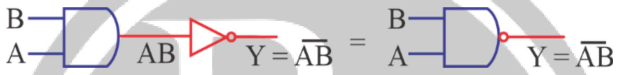


Fig. 1.12.

Truth table of 2-input NAND gate.

Input		Output
A	B	$Y = \overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

Switching and Circuit Diagram for NAND gate.

B	A	OUT
0	0	1
0	1	1
1	0	1
1	1	0

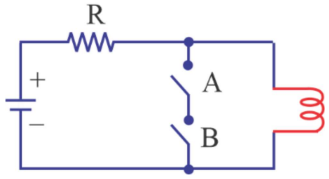


Fig. 1.13.

NAND gate acts as Universal Gate

Logic Gates using only NAND Gates

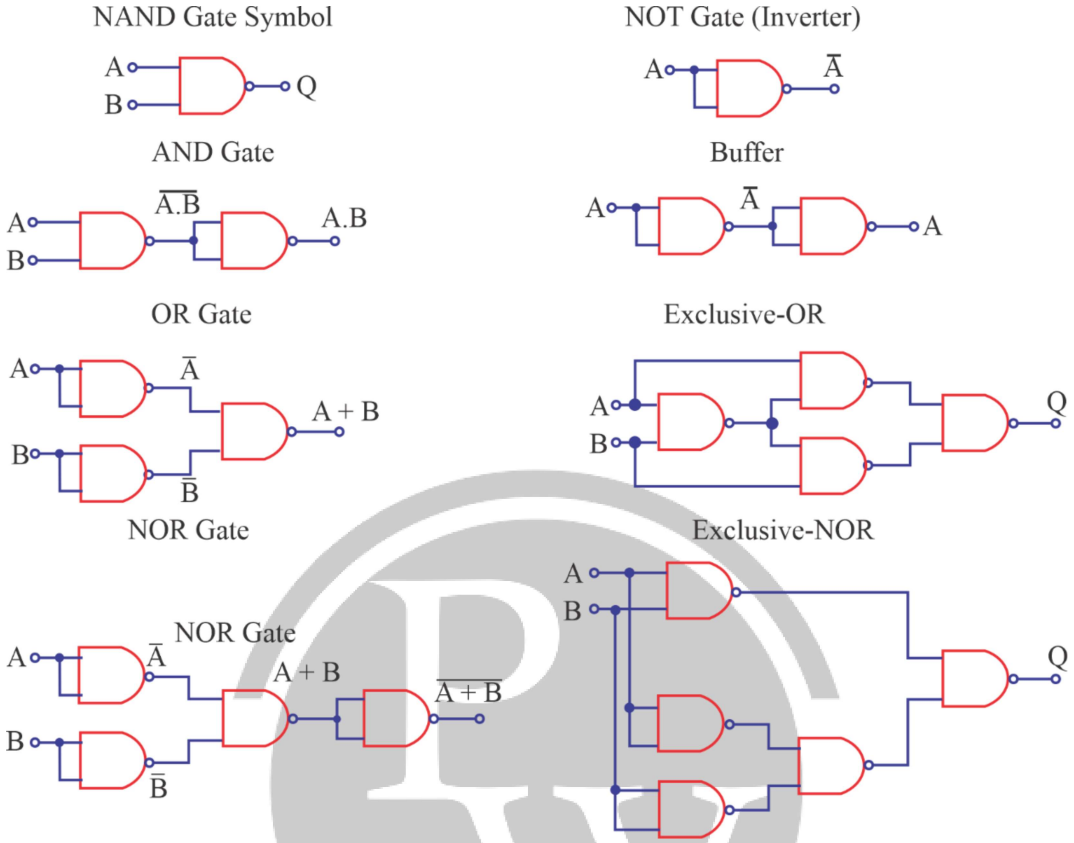


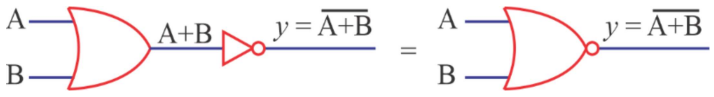
Fig. 1.14.

All the logic gate functions can be created using only NAND gates. Therefore, it is also known as a Universal logic gate.

1.2.2. NOR Gate

A NOR gate is equivalent to OR gate followed by a NOT gate.

Symbol:



Truth Table for 2-input NOR gate

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

Switching and Circuit Diagram for NOR gate

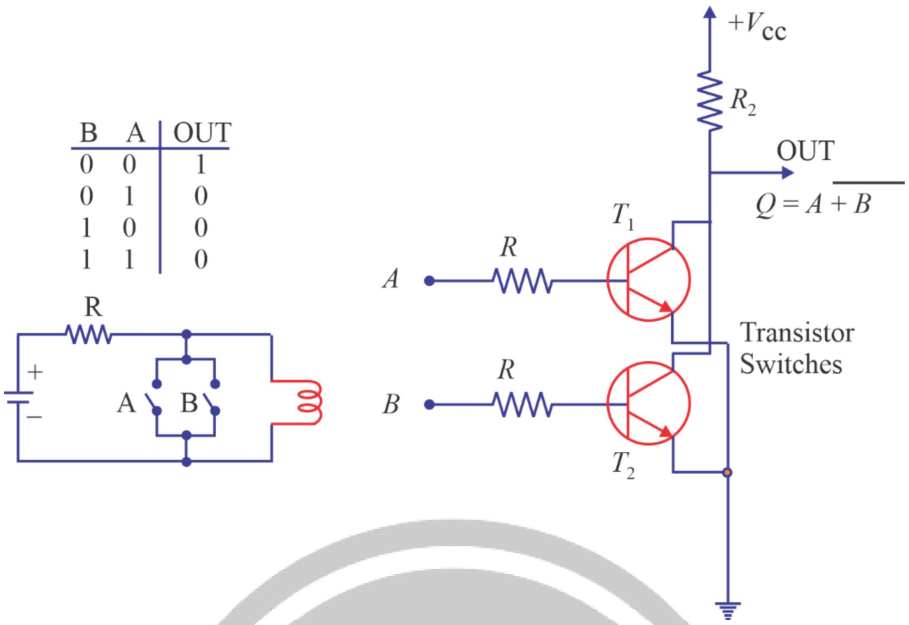


Fig. 1.15.

NOR gate acts as Universal Gate.

Logic Gates using only NOR Gates

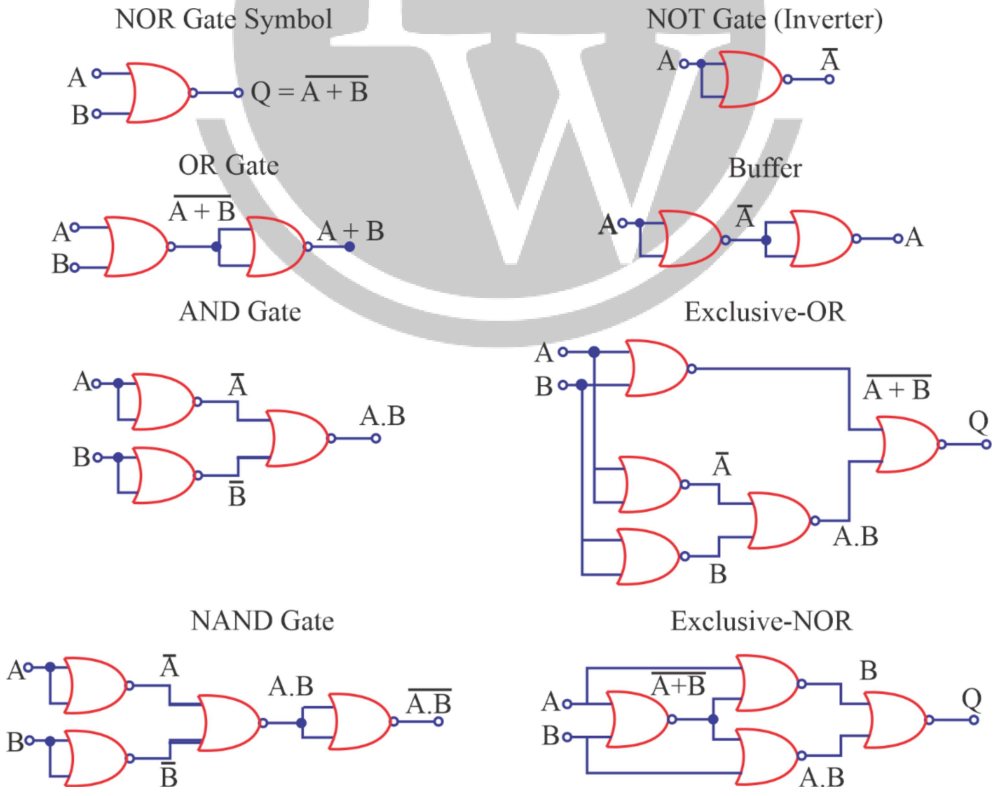
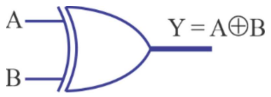


Fig. 1.16.

All the logic gate functions can be created using only NOR gates. Therefore, it is also known as a Universal logic gate.

1.2.3. XOR Gate

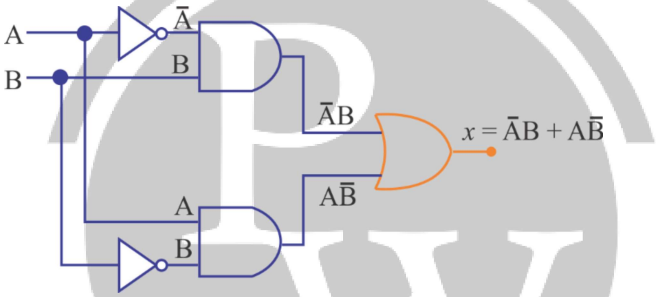
Symbol of two input XOR gate



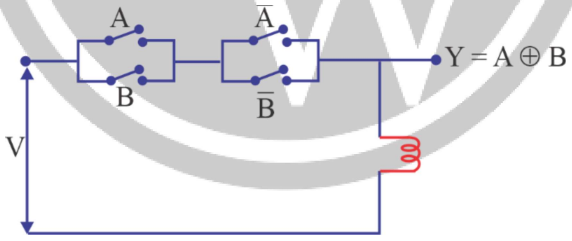
Truth table for 2-input XOR gate

Input		Output
A	**B**	**Y = A ⊕ B**
0	0	0
0	1	1
1	0	1
1	1	0

XOR gate using AND, OR and NOT gate



Switching diagram of XOR gate



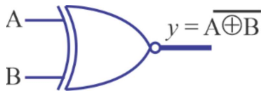
Truth Table:

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

$$Y = (A + B)(\bar{A} + \bar{B})$$
$$= \bar{A}B + A\bar{B}$$
$$= A \oplus B$$

1.2.4. X-NOR gate:

Symbol for two input X-NOR gate



Truth table for 2-input X-NOR gate

Input		Output
A	B	Y = A ⊙ B
0	0	1
0	1	0
1	0	0
1	1	1

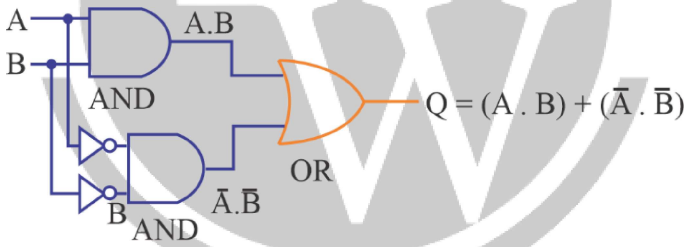
Boolean expression for EX-NOR gate is $Y = \overline{A \oplus B}$

Apply De-Morgan's theorem:

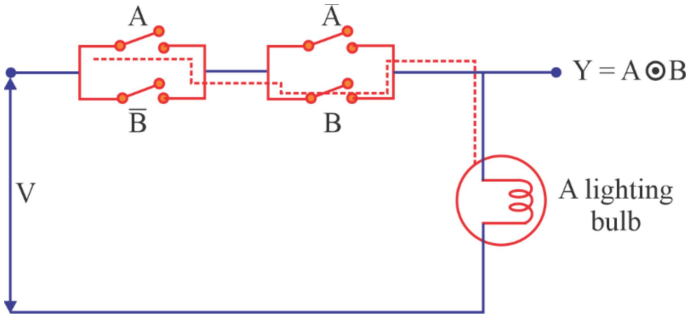
$$\overline{A \oplus B} = \overline{\bar{A}B + A\bar{B}} = \overline{\bar{A}B} \cdot \overline{A\bar{B}} = \left(A + \bar{B}\right)\left(\bar{A} + B\right) = AB + \bar{A}\bar{B}$$

The output of a two input EX-NOR gate is logic '1' when the inputs are same and a logic '0' when they are different.

X-NOR gate using AND OR and NOT gate



Switching Diagram of X-NOR gate

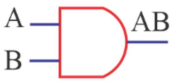
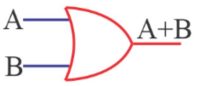
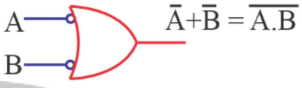
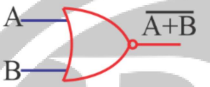
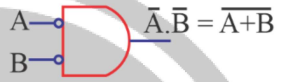


Truth Table:

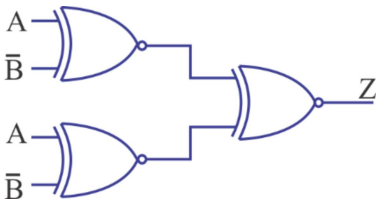
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

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

1.3. Alternate Logic Gate Representation

Logic	Normal symbol	Alternate symbol
NOT		
AND		
OR		
NAND		
NOR		

Example: In the following circuit, the find the output Z?

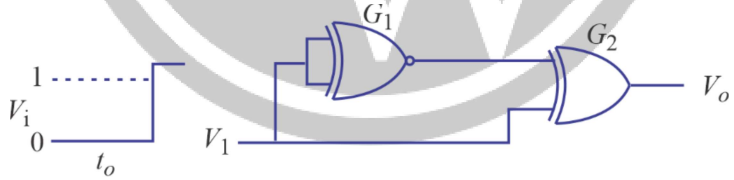


Solution: From the given circuit, we can observe that input to last XNOR is same, so, the XNOR output is given by (let input is X)

$$Z = X.X + \overline{X}.\overline{X} = X + \overline{X} = 1$$

i.e. the output will be high [logic 1] irrespective of the inputs A and B.

Example: The gate G_1 and G_2 in figure shown below have propagation delays of 10ns and 20ns respectively.

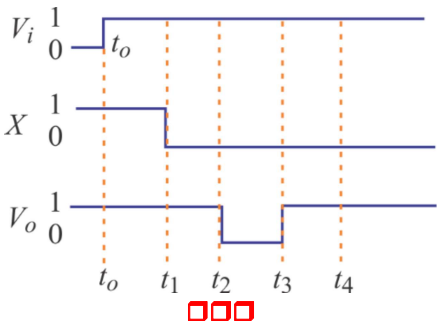


If input V_i makes an abrupt change from logic 0 to 1 at t = t_0, then find the output waveform V_0?

Here, $t_1 = t_0 + 10$ ns, $t_2 = t_1 + 10$ns, $t_3 = t_2 + 10$ ns.

Solution: Let the output of $G_1 = X$

The output waveform will be as shown in figure below.



2 MINIMIZATION OF BOOLEAN FUNCTION

2.1. Boolean Algebra

Boolean algebra is a system of mathematical logic. It is an algebraic system consisting on the set of elements (0, 1) two binary operators called OR, AND and one unary operator NOT. It is the basic mathematical tools in the analysis and the synthesis of switching circuits. It is a way to express logic functions algebraically.

Note: Any functional relation in Boolean algebra can be provided by the method of perfect induction perfect inductions the method of proof, where by a function relation is verified for every possible combination of values that the value may assume.

Axioms of Boolean Algebra:

Axioms of Boolean algebra are a set of logical expressions that we accept without proof and upon which we can build a set of useful theorem.

	AND operator	OR operator	NOT operators
Axioms 1:	0.0 = 0	0 + 0 = 0	$\bar{1} = 0$
Axioms 2:	0.1 = 0	0 + 1 = 1	$\bar{0} = 1$
Axioms 3:	1.0 = 0	1 + 0 = 1	
Axioms 4:	1.1 = 1	1 + 1 = 1	

Logic operations: In Boolean Algebra all the algebraic function performed is logical. These actually represents logic operations. The AND, OR and NOT are the basic operations that are performed in Boolean Algebra. In addition to these operations, there are some derived operations such as NAND, NOR, EX-OR and EX-NOR that are also performed in Boolean Algebra.

1.1.1. NOT Operation

The NOT operation in Boolean algebra is similar to inversion in ordinary algebra

$1 : \bar{0} = 1$

$2 : \bar{1} = 0$

3 : if A = 0 then $\bar{A} = 1$

4 : $\bar{\bar{A}}$ = A (Double inversion)

1.1.2. AND Operation

It is a logical operation that are performed by AND gate. The AND operation in Boolean Algebra is similar to multiplication in ordinary algebra.

1: $A \cdot 0 = 0$ (Null Law)

2: $A \cdot 1 = A$ (Identity law)

3: $A \cdot A = A$

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

1.1.3. OR Operation

It is the logical operation that are performed by OR gate. The OR operation in Boolean Algebra is similar to addition in ordinary algebra.

1: A + 0 = A (Null law)

2: A + 1 = 1 (Identity law)

3: A + A = A

4: $A + \bar{A} = 1$

1.1.4. NAND Operation:

The NAND operation in Boolean Algebra is performed by AND operation followed by NOT operation i.e., the negation of AND operation is performed by NAND gate.

1.1.5. NOR Operation:

The NOR operation in Boolean Algebra is performed by OR operation followed by NOT operation i.e., the negation of OR operation is performed by NOR gate.

2.2. Laws of Boolean Algebra

2.2.1. Commutative Law

1. A + B = B + A

   A + B + C = B + C + A = C + A + B = B + A + C

2. AB = BA

   $A \cdot BC = B \cdot CA = C \cdot AB = B \cdot AC$

Violation: Inhibition (1) for Example x/y (x but not y) is not commutative law it means $x/y \neq y/x$

2.2.2. Associative law:

This law arrows grouping of variables

1. (A + B) + C = A + (B + C)

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

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

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

   $A\,(BCD) = (ABC) \cdot D$

   $A\,(BCD) = AB \cdot CD$

   Note:- NAND and NOR gates are not Associative

2.2.3. Distributive Law

1: A(B + C) = AB + AC

A + BC = (A + B) (A + C)

2. 2.4. Redundant Literal Rule

1. $A + \bar{A}B = A + B$
2. $A(\bar{A} + B) = AB$

2.2.5. Idempotent Law

1. $A \cdot A = A$
2. $A + A = A$

2.2.6. Absorption Law

1. $A + AB = A$
2. $A(A + B) = A$

2.2.7. Involutionary Law

The law that for any variable A.

$\bar{\bar{A}} = (A')' = A$

2.2.8. Consensus theorem:

There are two consensus theorems

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

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

2.3. De-Morgan's theorem:

De-Morgan's theorem represents two of the most important rules of Boolean algebra.

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

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

The above two laws can be extended for 'n' variables as,

$\overline{A_1.A_2.A_3... + A_n} = \bar{A}_1 + \bar{A}_2 + ...\bar{A}_n$ and $\overline{A_1 + A_2 + ... + A_n} = \bar{A}_1.\bar{A}_2...\bar{A}_n$

2.3.1 Duality theorem:

Duality Theorem states that,

(a) Change each OR sign by an AND sign and vice versa.

(b) Compliment any '0' or '1' appearing in expression

(c) Keep literals as it is.

Note: With n variables, maximum possible distinct logic function = 2^{2^n}

Example : If a function is given as $f = AB + \bar{A}\bar{B}$ then find its complement.

Solution : Given $f = \left(AB + \bar{A}\bar{B}\right)$

$$\text{Complement of } \bar{f} = \overline{AB + \bar{A}\bar{B}} = \overline{AB} \cdot \overline{\bar{A}\bar{B}}$$

$$= \left(\bar{A} + \bar{B}\right)\left(A + B\right)$$

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

Example : Show that

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

Solution : LHS $= AB + B\bar{C} + AC$

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

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

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

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

$$= \text{RHS}$$

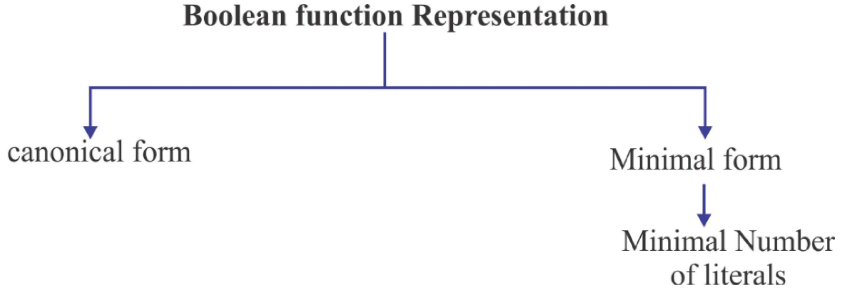
2.4. Minimization of Boolean function:

Every Boolean function expression must be reduced to as simple form as possible before realization because every logic operation in the expression represents a corresponding elements of hardware. Realization of digital circuit with minimal expression has several advantages as:

1. The number of logic gates will reduced.
2. The speed of operation will increase
3. power dissipation will decrease
4. The FAN IN may reduced
5. The complexing of the circuit reduces

The simple method of minimization of Boolean function using certain Algebraic rules which results in the reduction of number of term and/or number of arithmetic operations the various theorem and rules that are already discussed are very useful for the simplification of Boolean expression.

A function of n Boolean variables denoted by $f(A_1, A_2 .... A_n)$ is another variable of Algebra and takes one of the two possible values either 0 or 1. The various way of representing a given function are discussed below.



All the terms contain all the variable either in complementary or in uncomplimentary form

The literal means the Binary variable either in complementary or in uncomplimentary form.

2.4.1. Minimization of Boolean function using k-map

- **Using K-map:** The Boolean function can be simplified Algebraically but being not a symmetric method, we can never be sure that whether the minimal expression obtained is the real minimal or not.
- **Karnaugh Map (k-map):** A k-map is a graphical representation of Boolean expression, A two variable k-map will have four cell or squares 3-variable k-map will have 8-cells, n-variable k-map will have 2^n cells.

Note: Adjacent cells differ by 1 bit to maintain adjacently property gray code sequence is used in k-maps (Any two adjacent cells will differ by only one bit)

Min terms & Max terms:

1. n-binary variable have 2^n possible combinations.
2. Min term is a product term, it contains all the variables either complementary or un complementary form for that combination the function output must be '1'.
   - Max term is a sum term, it contains all the variables either complementary or uncomplimentary form for that combination the function output must be '0'.

   For two variable

x	y	Min terms	Max terms
0	0	$m_0 = \bar{x}\,\bar{y}$	$M_0 = x + y$
0	1	$m_1 = \bar{x}\,y$	$M_1 = x + \bar{y}$
1	0	$m_2 = x\,\bar{y}$	$M_2 = \bar{x} + y$
1	1	$m_3 = x\,y$	$M_3 = \bar{x} + \bar{y}$

- In min terms we assigns '1' to each uncomplemented variables and '0' to each complemented variable.
- In Max terms we assign '0' to each uncomplemented variable and '1' to each complemented variables.

2.5. Representation of Boolean Functions

Any Boolean expression can be expressed in two forms

- Sum of Product form (SOP)
- Product of Sum form (POS)

2.5.1. SOP Form

The SOP expression usually takes the forms of two or more variables OR together.

$$Y = \overline{A}BC + A\overline{B} + AC$$

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

SOP forms are used to write logical expression for the output becoming logic '1'.

Example:

Input (3-variables)			Output (Y)
A	B	C	Y
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

∴ Notation of SOP expression is:

$$f(A, B, C) = \Sigma m(3, 5, 6, 7)$$

$$Y = m_3 + m_5 + m_6 + m_7$$

Also, $Y = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$

2.5.2. POS Form

The POS expression usually takes the form of two or more OR variables within parentheses, ANDed with two or more such terms.

Example: $Y = \left(A+\overline{B}+C\right)\left(B\overline{C}+D\right)$

Each individual term in standard POS form is called maxterm.

POS forms are used to write logical expression for output be coming logic '0'.

we get $f(A, B, C) = \pi M(0, 1, 2, 4)$

$$Y = M_0.M_1.M_2.M_4$$

$$Y = \left(A + B + C\right)\left(A + B + \overline{C}\right)\left(A + \overline{B} + C\right)\left(\overline{A} + B + C\right)$$

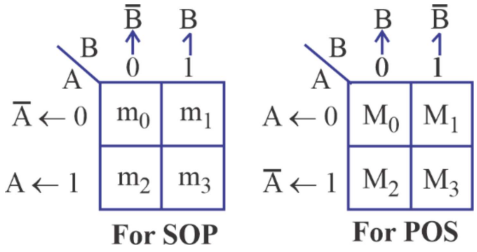
∴ We can also conclude from Table 2 and from above equations:

if $Y = \Sigma m(3, 5, 6, 7)$ or $Y = \pi M(0, 1, 2, 4)$

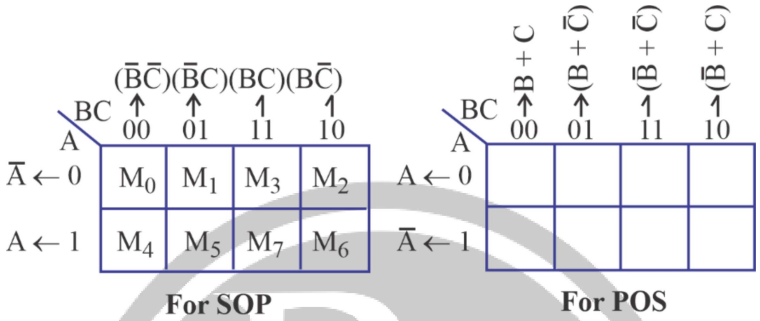
2.6. Karnaugh Map (K-MAP)

The K-map is a graphical method which provides a systematic method for simplifying the Boolean expressions. In n variable K-map, there are 2^n cells.

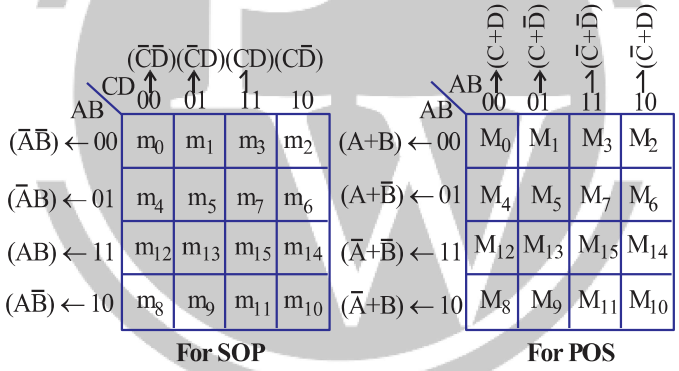
2.6.1. Two variable K-map



2.6.2. Three variable K-map



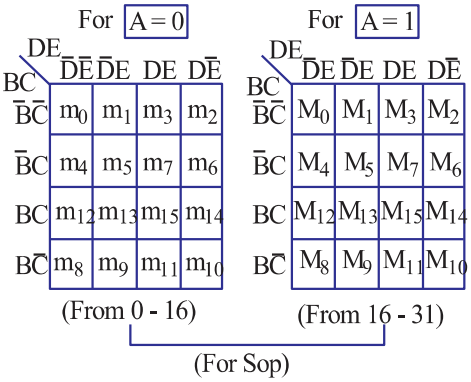
2.6.3. Four Variable K-map



2.6.4. Five variable K-map

- 32 cells
- 32 Minterms (Maxterms)

Here, we have $f(A, B, C, D, E)$



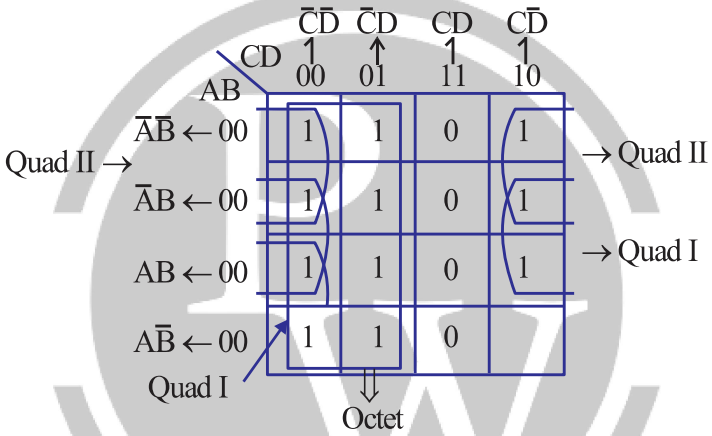
2.7. Simplification Rules

1. Construct the K-map and place 1's in those cells corresponding to the 1's in the truth table. Place the 0's in the other cells.
2. Examine the map for adjacent 1's and loop those 1's which are not adjacent to any other 1's. These are called isolated 1's.
3. Next, look for those 1's which are adjacent to only one other 1. Loop any pair containing such a 1.
4. Loop any octet even if it contains some 1's that have already been looped.
5. Loop any quad that contains one or more 1's which have not already been looped, making sure to use the minimum number of loops.
6. Form the OR sum of all the terms generated by each loop.

Example: Simply a four variable logic function using K-map

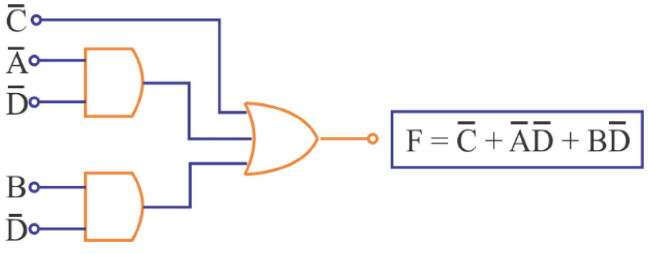
$f(A, B, C, D) = \Sigma m(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$ also implement the simplified expression with AND-OR logic.

Solution:



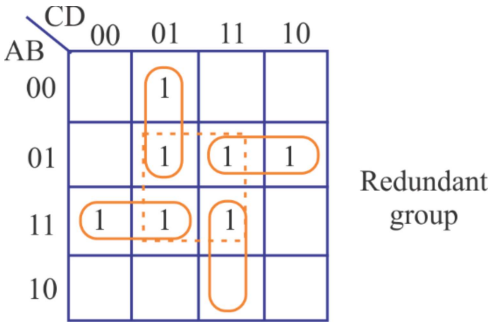
$$\therefore \quad f = \underset{\text{octet}}{\bar{C}} + \underset{\text{quad - II}}{\bar{A}\bar{D}} + \underset{\text{quad - I}}{B\bar{D}}$$

⇒ Gate implementation:



2.8. Redundant Group

If all the 1's in a group are already involved in some other groups, then that group is caused as a redundant group. A redundant group has to be eliminated, because it increases the no of gates required.



2.9. Don't Care Condition

The combinations for which the values of the expression are not specified are called don't care conditions.

Example: Simply the given equation in part (i) and (ii)

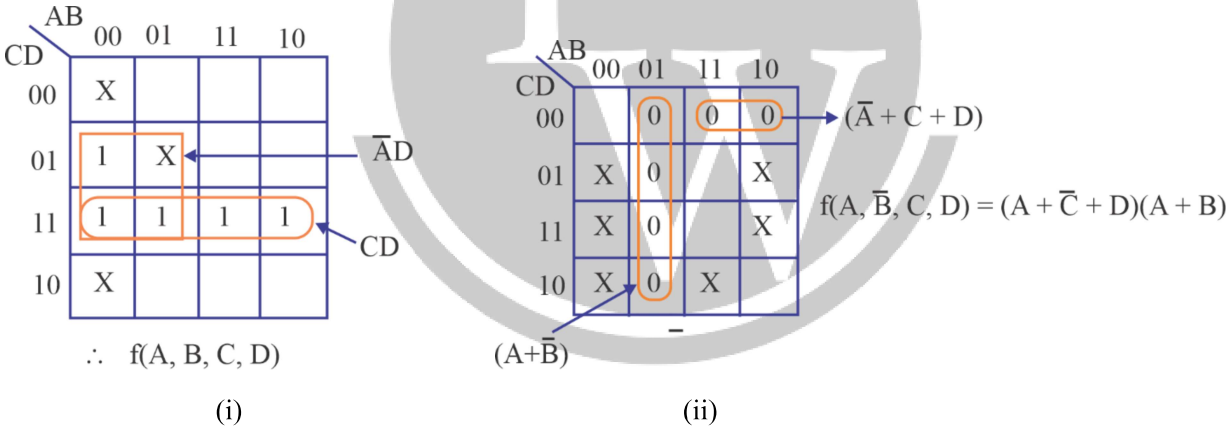
(i) In terms of SOP. and don't care conditions

$f(A, B, C, D) = \Sigma m(1, 3, 7, 11, 15) + d(0, 2, 5)$

(ii) In terms of POS and don't care conditions.

$f(A, B, C, D) = \pi M(4, 5, 6, 7, 8, 12) + d(1, 2, 3, 9, 11, 14)$

Solution:



2.10. Implicants, Prime Implicants and Essential Prime Implicants

2.10.1. Implicants

Implicants is a product term on the given function for that combination the function output must be 1.

2.10.2. Prime Implicant (PI)

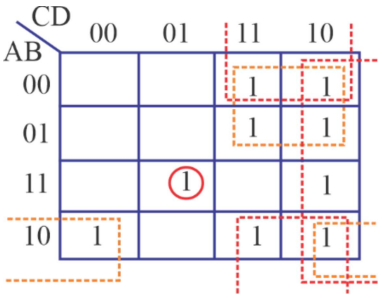
Prime implicant is a smallest possible product term of the given function,

2.10.3. Essential Prime Implicants (EPI)

EPI is a prime implicant it must cover at least one minterms, which is not covered by other PI.

Example: Reduce the expression using mapping $F = \Sigma m(2, 3, 6, 7, 8, 10, 11, 13, 14)$

Solution :

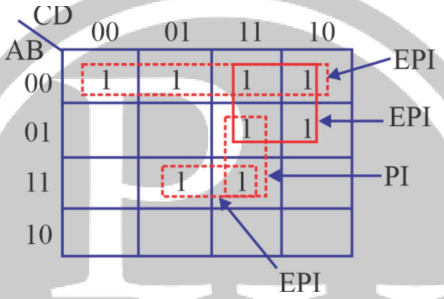


$F = AB\bar{C}D + A\bar{B}\bar{D} + \bar{A}C + \bar{B}C + C\bar{D}$

Example : Reduce the following expression using k-map and identify PI's and EPI

$F = \Sigma m(0, 1, 2, 3, 6, 7, 13, 15)$

Solution :



EPI = $\bar{A}\bar{B}$, $\bar{A}C$, ABD

PI = BCD

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

Example: Given the following Karnaugh map, which one of the following represents the minimal sum of products of the map.

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

A. $xy + \bar{y}z$

B. $\omega\bar{x}\bar{y} + xy + xz$

C. $\bar{\omega}x + \bar{y}z + xy$

D. $xy + y$

Solution :

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

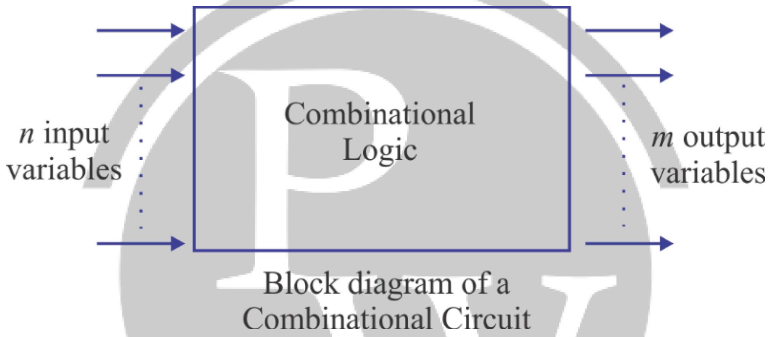
$= xy + \bar{y}z$

❐❐❐

3 COMBINATIONAL CIRCUITS

3.1. Combinational Circuits

The combinational circuit has 'n' input variables and 'm' output variables. Since, the number of input variables is n, there are 2^n possible combinations of bits at the input. Each output can be expressed in terms of input variables by a Boolean expression.



Block diagram of a Combinational Circuit

Fig. 3.1. Block diagram of a Combinational Circuit

3.2. Adders

The most basic arithmetic operation is the addition of two binary digits. A combinational circuit that performs the addition of two 1-bit numbers is called as half adder, and the logic circuit that adds three 1-bit numbers is called as full adder.

3.2.1. Half Adder

The logic circuit that performs the addition of two 1-bit numbers is called as half adder. It is the basic building block for addition of two single bit numbers. This circuit has two outputs namely carry (C) and sum (S).

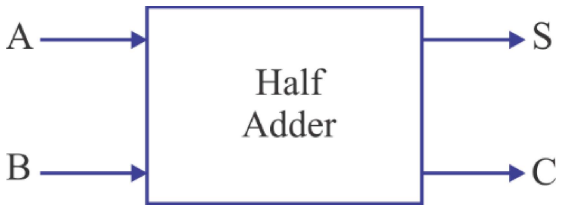


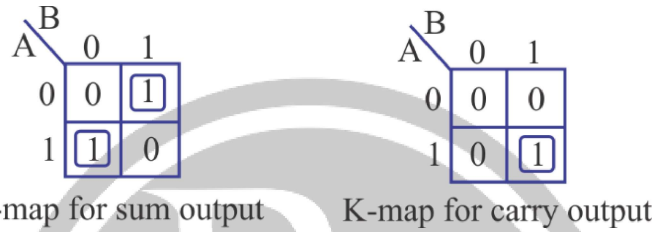
Fig. 3.2. Block Diagram of a 2-bit Half Adder

The truth table of half adder: where A and B are the inputs and sum and carry

Table 1: Truth Table of Half Adder

Inputs		Outputs	
A	**B**	**Sum (S)**	**Carry (C)**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

K–map simplification for Carry and Sum: Boolean expressions for the sum (S) and carry (C) outputs from K – maps:



Sum, $S = \bar{A}B + A\bar{B} = (A \oplus B)$

Carry, $C = AB$

Logic Diagram:

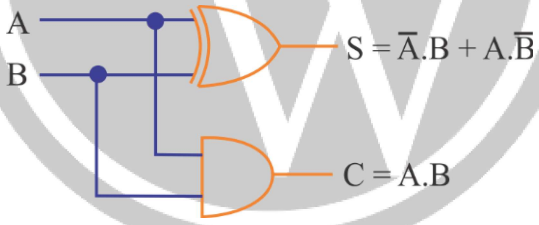


Fig. 3.3. Logic Diagram of Half Adder

3.2.2. Full Adder

A full adder circuit is an arithmetic circuit block that can be used to add three bits to produce a sum and a carry output. Let us consider A and B as two 1-bit inputs & C_{in} is a carry generated from the previous order bit additions. Let S (sum) and C_{out} (carry) are the outputs of the full adder.

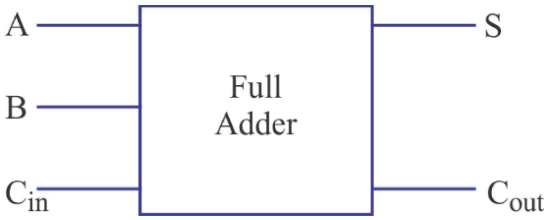


Fig. 3.4. Block Diagram of a Full Adder

The Truth Table for Full Adder is given as:

Table: Truth Table for Full Adder

Inputs			Outputs	
A	B	C_{in}	Sum (S)	Carry C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

K – map Simplification for Carry and Sum:

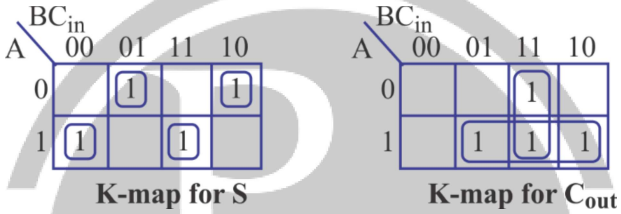


Fig. 3.5.

Simplified Boolean expressions for the sum (S) and carry (C_{out}) output from K-maps is

$$\text{Sum,} \quad S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + ABC_{in} + A\bar{B}\bar{C}_{in} = C_{in}\left(\bar{A}\bar{B} + AB\right) + \bar{C}_{in}\left(\bar{A}B + A\bar{B}\right)$$

$$= C_{in}\left(A \odot B\right) + C_{in}\left(A \oplus B\right)$$

$$= C_{in}\left(\overline{A \oplus B}\right) + C_{in}\left(A \oplus B\right)$$

$$\text{Sum,} \quad S = C_{in} \oplus A \oplus B$$

$$\text{Carry,} \quad C_{out} = AB + AC_{in} + BC_{in}$$

Logic Diagram:

We can realize logic diagram of a full adder using gates as shown in below figure:

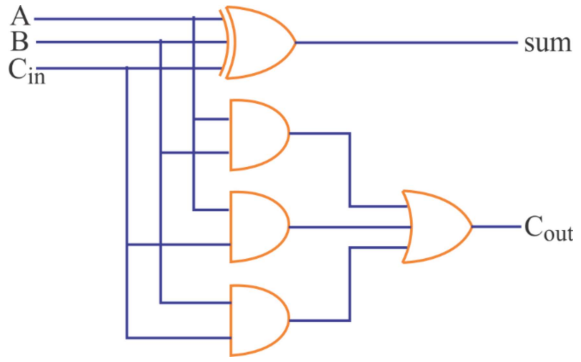


Fig. 3.6. Logic Diagram of Full Adder

Example: A full adder is implemented using two input OR gate and two half adders. Half adder is implemented using two input XOR and two input AND gate. The propagation delays of XOR gate, AND gate and OR gate respectively are 2ns, 1.5ns. and 1ns. The propagation delay of full adder is ….. ns.

Solution:

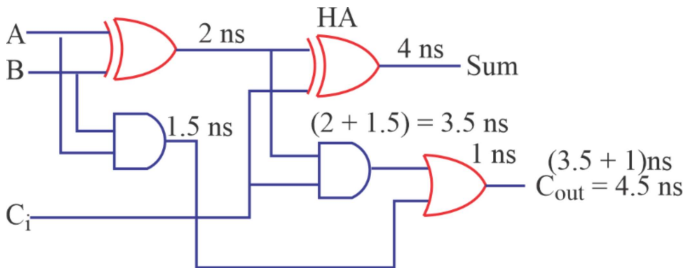


Fig. 3.7.

3.3. Subtractors

3.3.1. Half subtractor

A half subtractor is a combinational logic circuit, which performs the subtraction of two 1-bit numbers. It subtracts one binary digit from another to produce a DIFFERENCE output and a BORROW output.

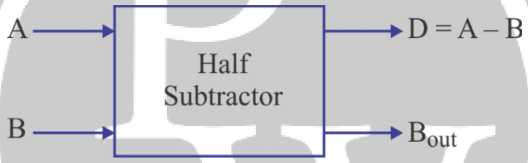


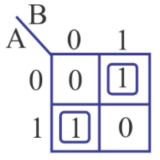
Fig. 3.8. Block diagram of a half subtractor

The truth table of half – subtractor, where A, B are the inputs, and difference (D) and borrow (B) are the outputs.

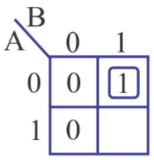
Table: Truth table of Half – Subtractor

Inputs		Outputs	
A	**B**	**D**	**B_{out}**
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

K – map Simplification for Difference and Borrow:



K-map for diffrence output



K-map for borrow output

Difference, $D = \bar{A}B + A\bar{B} = A \oplus B$

Borrow, $B_{out} = \bar{A}B$

Logic Diagram:

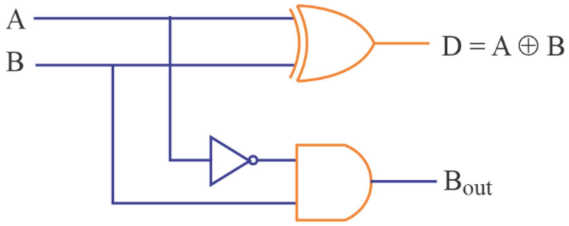


Fig. 3.9. Logic Diagram of a Half subtractor

3.3.2. Full Subtractor

A full subtractor performs subtraction operation on two bits, a minuend and a subtrahend.

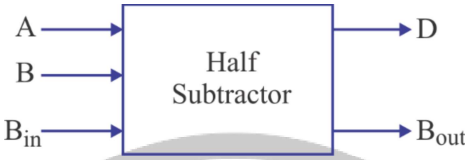


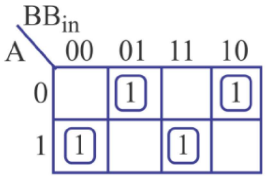
Fig. 3.10. Block Diagram of a Full subtractor

Truth table of Full subtractor:

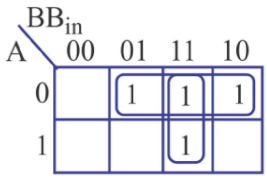
ble: Truth table of Full subtractor

Inputs			Outputs	
A	B	B_{in}	D	B_{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

K – map simplification for Difference and Borrow:



K-map for difference output



K-map for borrow output

Difference, $D = \bar{A}\bar{B}B_{in} + \bar{A}B\bar{B}_{in} + A\bar{B}\bar{B}_{in} + ABB_{in}$

$= B_{in}\left(AB + \bar{A}\bar{B}\right) + \bar{B}_{in}\left(A\bar{B} + \bar{A}B\right)$

$= B_{in}\left(\overline{A \mathring{A} B}\right) + B_{in}\left(A \mathring{A} B\right)$

$D = A \oplus B \oplus B_{in}$

Borrow, $B_{in} = \bar{A}B + \bar{A}B_{in} + BB_{in}$

Logic Diagram

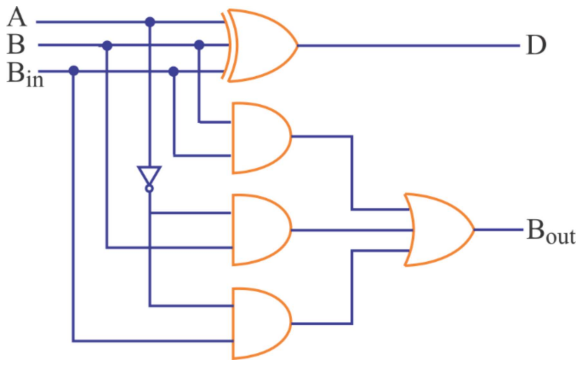


Fig. 3.11. Logic Diagram of a Full Subtractor

3.4. Binary Parallel Adder

An n-bit parallel adder can be constructed using n number of full adders are connected in parallel and hence; it is also known as parallel adder such that the previous carry or carry input for adder 0 is set to zero. The carry output of each adder is connected to the carry input of the next higher order adder. Hence, it is also known as carry propagate adder.

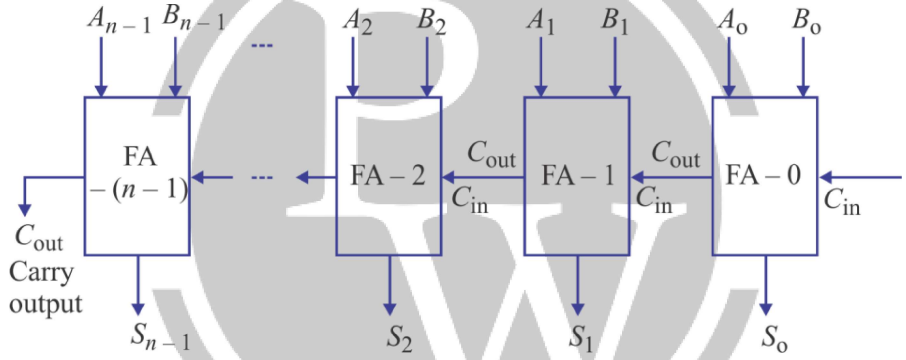


Fig. 3.12. n-bit Binary Adder

3.4.1. Propagation Delay in Parallel Adder:

Parallel adders suffer from propagation delay problem because higher bit additions depend on the carry generated from lower bit addition. In effect, carry bits must propagate or ripple through all stages before the most significant sum bit is valid. Thus, the total sum (the parallel output) is not valid until after the cumulative delay of all the adder.

3.5. Carry Look Ahead Adder

The look ahead carry adder speeds up the operation by eliminating this ripple carry delay. It examines all the input bits simultaneously and also generates the carry in bits for all the stages simultaneously. The method of speeding up the addition process is based on the two additional functions of the full adder called the carry generate and carry propagate functions.

3.5.1. Carry Generation

Carry is generated only if both the input bits are 1, that is, if both the bits A and B are 1's, a carry has to be generated in this stage regardless of whether the input carry C_{in} is a 0 or a 1. Let G as the carry generation function,

$$G = A \cdot B$$

Consider the present bit as the n^{th}, then

$$G_n = A_n \cdot B_n$$

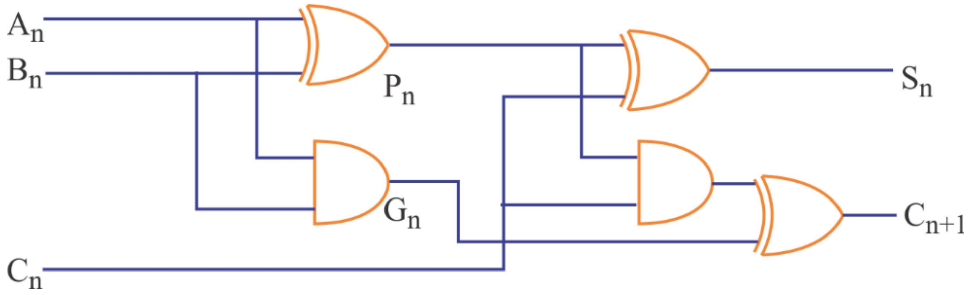


Fig. 3.13. Carry Look – ahead Generator Circuit

3.5.2. Carry Propagation

A carry is propagated if any one of the two input bits A or B is 1. If both A and B are 0, a carry will never be propagated. On the other hand, if both A and B are 1, then will not propagate the carry but will generate the carry. Let P as the carry – propagation function, then

$$P_n = A_n \oplus B_n$$

3.5.3. Look ahead Expressions

Let n^{th} bit adder, the sum (S) and the carry out (C) for the n^{th} bit may be expressed in terms of the carry generation function (G) and the carry propagation function (P) as

$$S_n = P_n \oplus C_n$$

$$C_{n+1} = G_n + P_n \cdot C_n$$

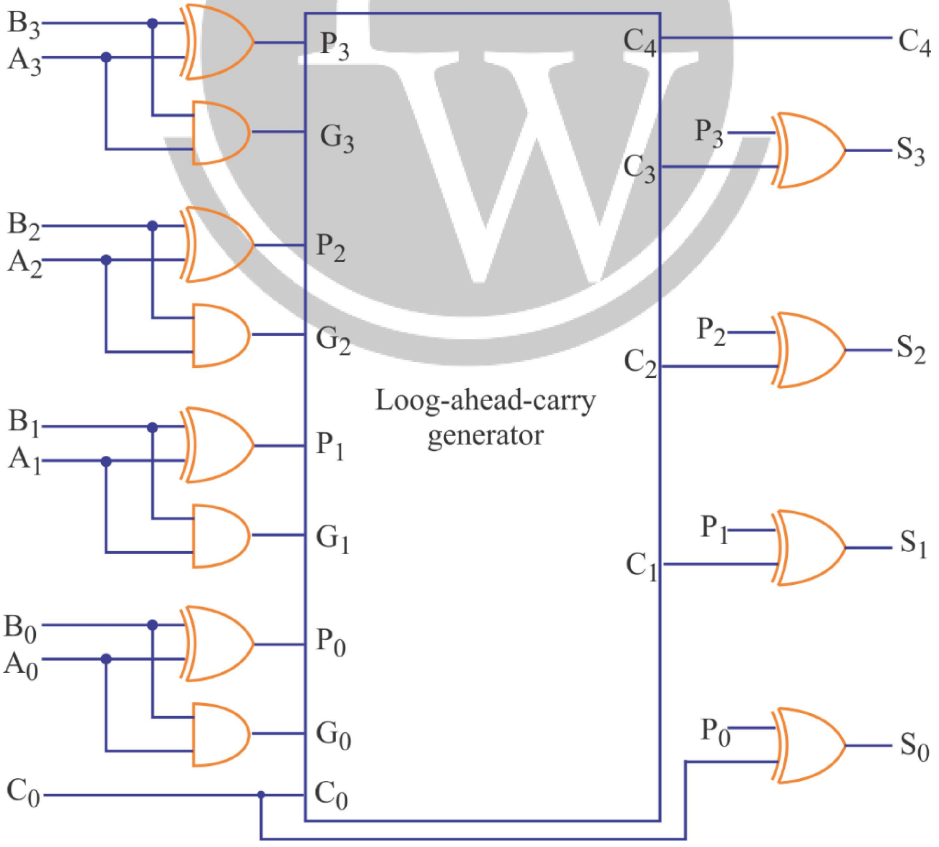


Fig. 3.14. 4-bit Full Adder with a look Ahead Carry Generator

Example: A full adder can be realized using half adder? Explain it in detail.

Solution: A full adder realization using half adder is given by:

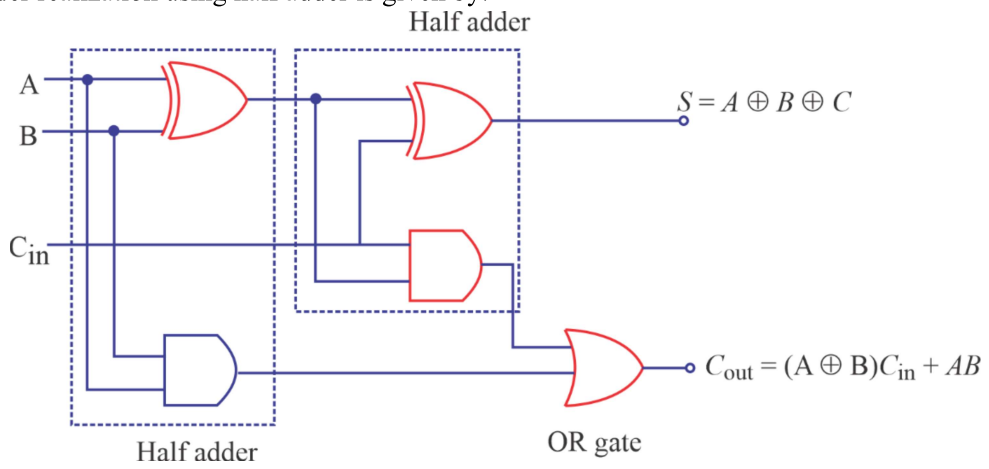


Fig. 3.15.

3.6. Comparator

The comparator is a combinational logic circuit. It compares the magnitude of two n-bit numbers and provides the relative result as the output. Let A and B are the two n-bit inputs. The comparator has three outputs namely A > B, A = B and A < B. Depending upon the result of comparison, one of these outputs will go high.

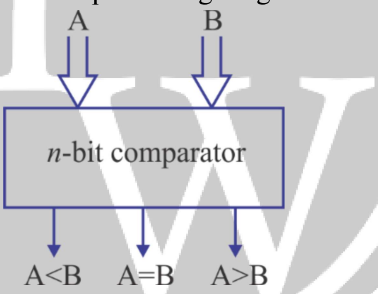


Fig. 3.16. Block diagram of digital comparator

3.6.1. 1-bit Magnitude Comparator

The 1-bit comparator is a combinational logic circuit with two inputs A and B and three outputs namely A < B, A= B and A > B.

Table : Truth Table of a 1-bit Comparator

Inputs		Outputs		
A	B	X (A < B)	Y (A = B)	Z (A > B)
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

Design of 1-bit Magnitude Comparator: We can write the expressions for the three outputs as under:

For $(A < B)$, $X = \bar{A}_0 B_0$

For $(A = B)$, $Y = \bar{A}_0\bar{B}_0 + A_0 B_0 = \overline{A_0 \oplus B_0}$

For $(A > B)$, $Z = A_0\bar{B}_0$

Logic Diagram of 1-bit Comparator:

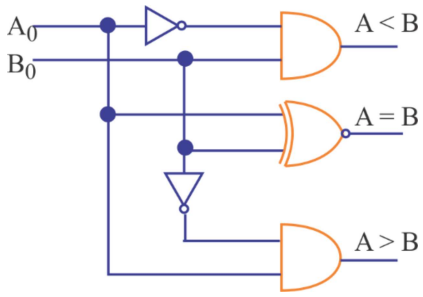


Fig. 3.17. Logic Diagram of 1-bit Comparator

Example: The circuit shown in given figure is:

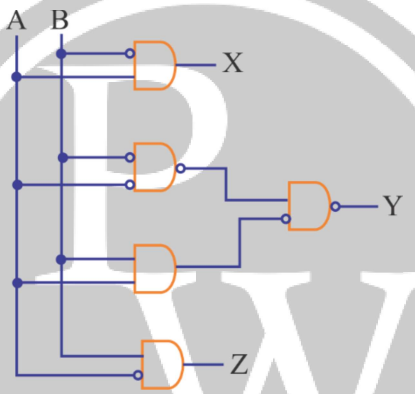


Fig. 3.18.

Solution: From the logic circuit shown in the figure, we obtain the following results,

$X = A\bar{B}$

$Y = \overline{(\bar{A}\bar{B})(\overline{AB})} = \bar{A}\bar{B} + AB = A \odot B$

$Z = \bar{A}B$

So, we obtain the truth table for the above function as shown below.

From truth table, we deduce the following results.

It $A > B$, then $x = 1$

It $A = B$, the $y = 1$

It $A < B$, then $z = 1$

A	B	X	Y	Z
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

Therefore, it is a comparator circuit.

3.7. Multiplexer

A multiplexer, abbreviated as MUX, is a digital switch which selects one of the many inputs to a single output. A number of control lines determine which input data is to be routed to the output. If there are n select lines, then the number of maximum possible input lines is 2^n and the multiplexer is referred to as a 2^n–to-1 multiplexer or $2^n \times 1$ multiplexer.

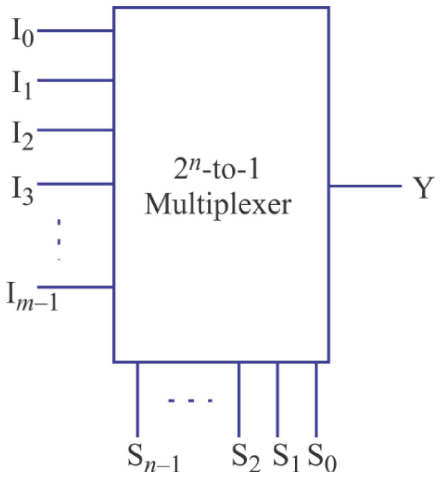


Fig. 3.19. Block diagram of a 2^n to 1 multiplexer

3.7.1. 2 × 1 MUX

A 2 to 1 multiplexer has 2 inputs. Since $2 = 2^1$, this multiplexer will have one control (select) line. It has two data inputs I_0 and I_1, one select input S, and one output.

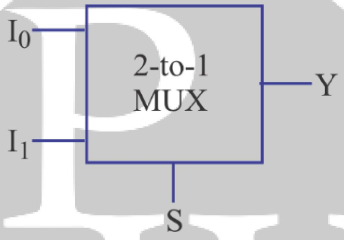


Fig. 3.20. Schematic block diagram of 2:1 Multiplexer

The truth table of this MUX is given below,

Select Line (S)	Output Y
0	I_0
1	I_1

Thus, the SOP expression for the output Y is,

$$Y = I_0\bar{S}_0 + I_1S_0$$

Realization of a 2:1 MUX using Logic Gates:

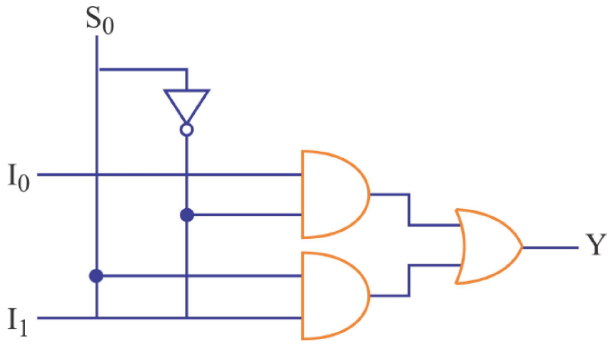


Fig. 3.21. Logic Diagram of a 2 x 1 Multiplexer

2.7.2. 4 × 1 MUX

A 4-to-1 multiplexer has 4 inputs and two select lines, where I_0 to I_3 are the four inputs to the multiplexer, and S_0 and S_1 are the select lines.

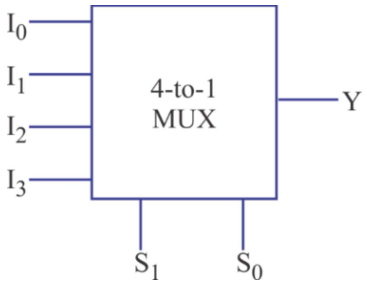


Fig. 3.22. Schematic block diagram of 4 x 1 MUX

Truth Table of a 4-to-1 Multiplexer

Select Inputs S_1	Select Inputs S_0	Output Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Output Y for a 4-input multiplexer is

$$Y = I_0\bar{S}_1\bar{S}_0 + I_1\bar{S}_1S_0 + I_2S_1\bar{S}_0 + I_3S_1S_0$$

3.8. Implementation of Higher Order Mux Using Lower Order MUX

The methods for implementing higher order MUX using lower order MUX are

Step 1: If 2^n is the number of input lines in the available lower order multiplexer and 2^N is the number of input lines in the desired multiplexer, then the number of lower order multiplexers required to construct the desired multiplexer circuit would be $2^N - n$.

Step 2: From the knowledge of the number of selection inputs of the available multiplexer and that of the desired multiplexer, connect the less significant bits of the selection inputs of the desired multiplexer to the selection inputs of the available multiplexer.

Step 3: The most significant bits of the selection inputs of the desired multiplexer circuit are used to enable or disable the individual multiplexers so that their outputs when OR produce the final output.

Example: In realization of 32 : 1 MUX using 2 : 1 MUX, the required number of 2 : 1 MUX is ?

Solution: In realization of 2^n : 1 MUX using 2 : 1 MUX, the required number of 2 : 1 MUX is 2^n -1, since, we have to realize 32 : 1 MUX, so we have

$$n = 5$$

Hence, the required number of 2 : 1 MUX is

$$2^n - 1 = 2^5 - 1 = 31$$

3.9. Applications of Multiplexers

1. It is used as a data selector to select one out of many data inputs.
2. They are used in designing the combinational circuits.
3. They are used in digital-to-analog and analog-to-digital converters.
4. They can be used for simplification of logic design.
5. Multiplexers are also used in data acquisition systems.

3.10. Demultiplexer

The demultiplexer is a combinational logic circuit that performs the reverse operation of a multiplexer. The demultiplexer has one input line and m output lines. Again $m = 2^n$, so it requires n select lines. A demultiplexer with one input and m output is called a 1-to-m demultiplexer.

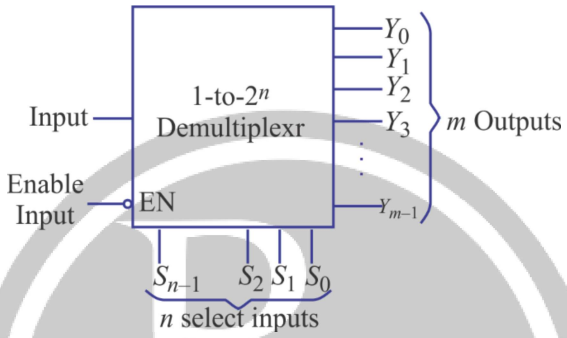


Fig. 3.23. Block Diagram of m-output Demultiplexer

The demultiplexer has one input line and m output lines. Again $m = 2^n$, so it requires n select lines. A demultiplexer with one input and m outputs is called a 1-to-m demultiplexer.

3.10.1. 1 × 2 Demultiplexer

A 1 to 2 demultiplexer has one input and two outputs. Since 2 = 2 × 1, it requires only one control (select) line.

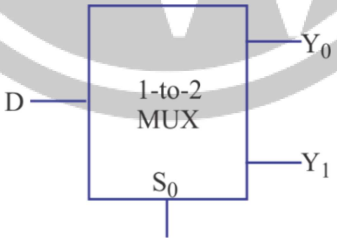


Fig. 3.24. Logic Diagram of 1 x 2 De-MUX

Truth table of a 1-to-2 demultiplexer

Table: Truth table of a 1-to-2 demultiplexer

Input S	Select input S_0	Output Y_1	Y_0
D	0	0	D
D	1	D	0

Thus, the Boolean expressions for the outputs can be written as

$$Y_0 = D\bar{S}_0 \ \& \ Y_1 = DS_0$$

Realization of a 1 x 2 Demultiplexer using Logic Gates:

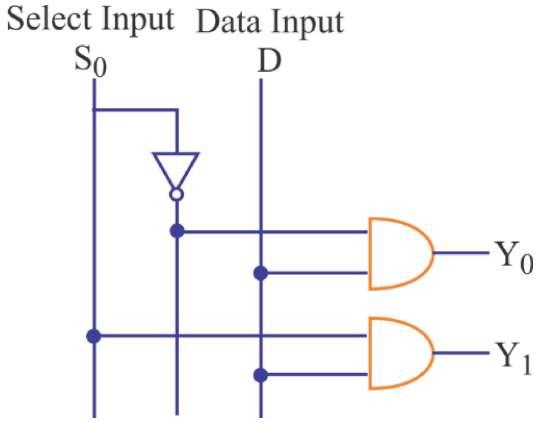


Fig. 3.25. Logic Diagram of 1 × 2 Demultiplexer

3.10.2. Applications of Demultiplexers

Demultiplexers are used in

1. Data transmission
2. Implementation of Boolean Functions
3. Combinational logic circuit design
4. Generate enable signals (enable one out of many). The application of enable signals in microprocessor systems are:
   (a) Selecting different banks of memory
   (b) Selecting different input/output devices for data transfer
   (c) Enabling different functional units
   (d) Enabling different rows of memory chips depending on address

3.11. Comparison Between Multiplexer and Demultiplexer

Table : Comparison between Multiplexer and Demultiplexer

S.No.	Parameter of comparison	Multiplexer	Demultiplexer
1.	Type of logic circuit	Combinational	Combinational
2.	Number of data inputs	m	1
3.	Number of select inputs	n	N
4.	Number of data output	1	M
5.	Relation between input/output lines and select lines	$m = 2^n$	$M = 2^N$
6.	Operation principle	Many to 1 or as data selector	1 to many or data distributor

3.12. Decoder

A decoder is a combinational circuit that converts an n-bit binary input data into 2^n output lines, such that each output line will be activated for only one of the possible combinations of inputs. Decoders are usually represented as n-to-2^n line decoders, where n is the number of input lines and 2^n is the number of maximum possible output lines.

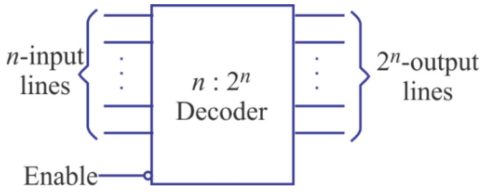


Fig. 3.26. Block Diagram of n-to-2ⁿ Decoder

If there are some unused or 'don't care' combinations in the n-bit code, then there will be less than 2n output lines. In general, if n and m are respectively the numbers of input and output lines, then m ≤ 2n.

3.12.1. 2 to 4 Line Decoder

Consider a 2 to 4-line decoder, where *A* and *B* are two inputs whereas Y_0 through Y_3 are the four outputs.

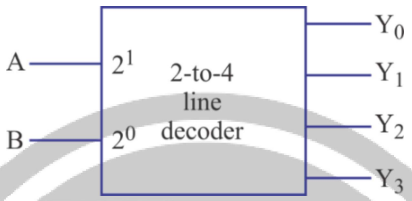


Fig. 3.27. Block Diagram of a 2 to 4 Line Decoder

Truth Table of a 2 to 4 Line Decoder

Table : Truth Table of a 2 to 4 Line Decoder

Inputs		Outputs			
A	B	Y_0	Y_1	Y_2	Y_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

The Boolean expressions for the four outputs is given as:

$$Y_0 = \overline{A}\,\overline{B} \text{ and } Y_1 = \overline{A}B$$

$$Y_2 = A\overline{B} \text{ and } Y_3 = AB$$

Realization of a 2 to 4 Line Decoder using Logic Gates

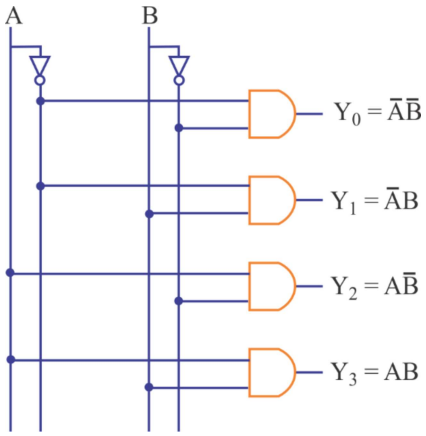


Fig. 3.28. Logic Diagram of a 2 to 4 Line Decoder

3.12.2. Applications of Decoder

Some of important applications of decoder are as follows:

1. When the decoder inputs come from a counter which is being continually pulsed, the decoder outputs will be activated sequentially. Hence, they can be used as timing or sequencing signals to turn devices on or off at specific times.
2. Decoder are use in memory system of a computer where they respond to the address code generated by the microprocessor to activate a particular memory location.
3. They are also used in computers for selection of external devices that include printers, modems, scanners, internal disk drives, keyboard, video monitor etc.

3.13. Encoders

An encoder is a combinational logic circuit that performs the inverse operation of a decoder. An encoder has 2^n (or fewer) input lines and n output lines.

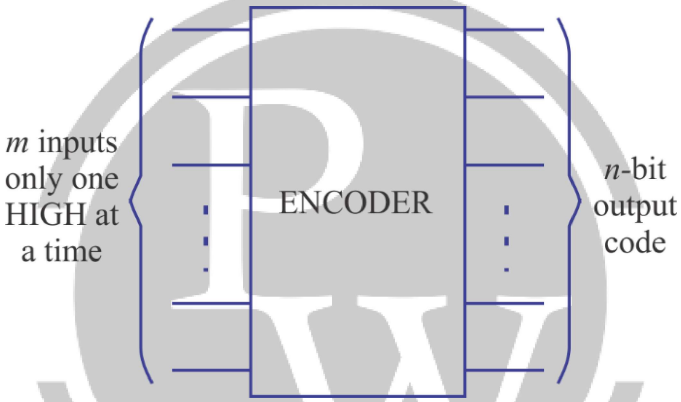


Fig. 3.29. Block Diagram of Encoder

3.13.1. Octal to Binary Encoder

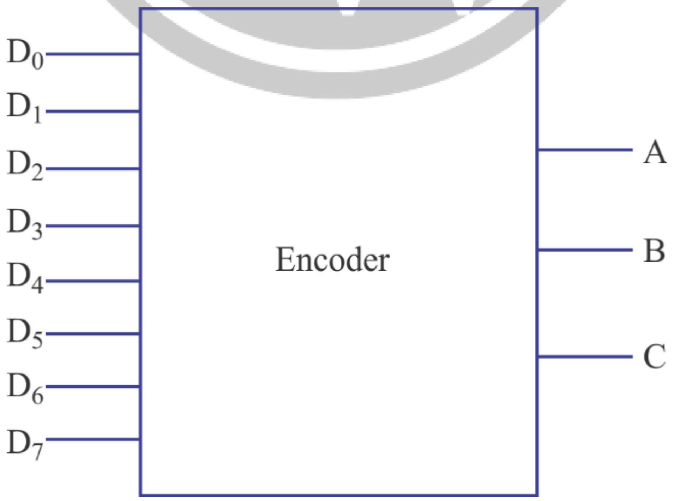


Fig. 3.30. Octal to Binary Encoder

Truth Table of an Octal to Binary Encoder:

Table: Truth Table of an Octal to Binary Encoder

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	A	B	C
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

The logical expressions for the outputs as follows:

$A = D_4 + D_5 + D_6 + D_7$

$B = D_2 + D_3 + D_6 + D_7$

$C = D_1 + D_3 + D_5 + D_7$

3.13.2. Octal to Binary Encoder

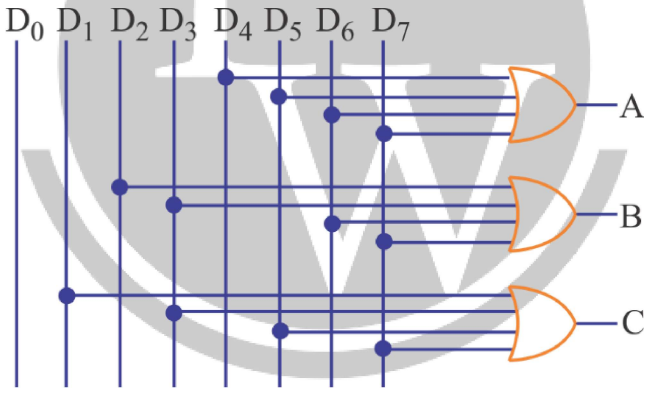


Fig. 3.31. Logic Diagram of Octal-to Binary Encoder

3.13.3. Decimal to BCD Encoder

This type of encoder has 10 inputs one for each decimal digit and 4 outputs corresponding to the BCD code.

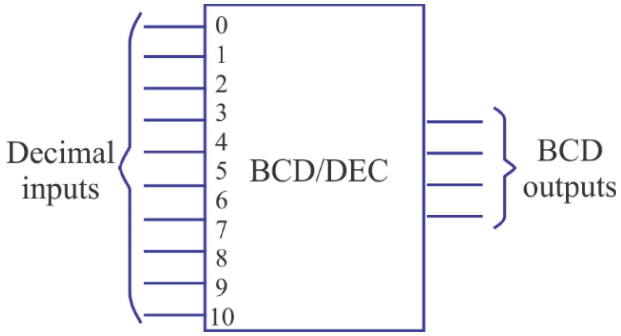


Fig. 3.32. Block Diagram of a Decimal-to-BCD Encoder

Truth Table of a Decimal to Binary Encoder:

Table : Truth Table of a Decimal to Binary Encoder

Input										Output			
0	1	2	3	4	5	6	7	8	9				
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	A	B	C	D
1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	1
0	0	0	0	0	0	1	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	1

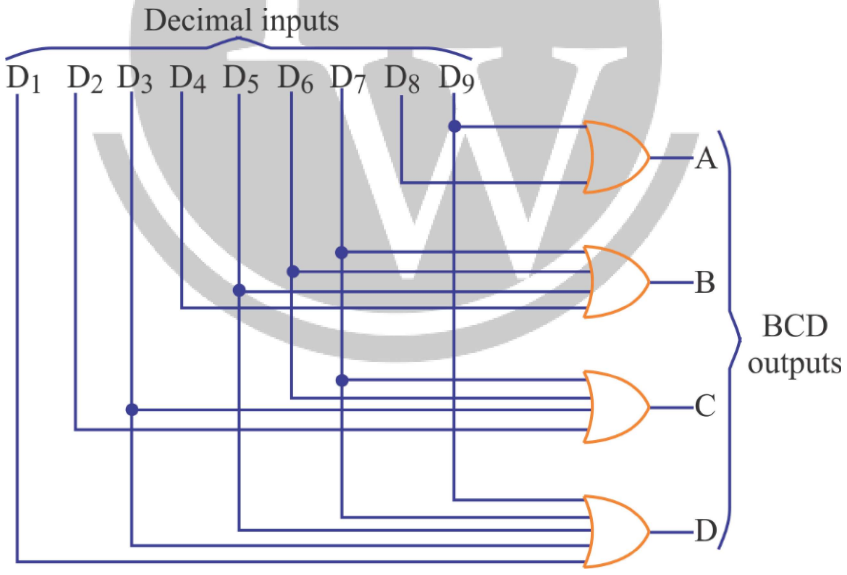


Fig. 3.33. Logic Diagram of Decimal-to-BCD encoder

The outputs of a decimal-to-BCD encoder:

$A = D_8 + D_9$

$B = D_4 + D_5 + D_6 + D_7$

$C = D_2 + D_3 + D_6 + D_7$

$D = D_1 + D_3 + D_5 + D_7 + D_9$

3.14. Priority Encoder

3.14.1. Truth Table of a Four Input Priority Encoder: (Taking LSB as priority)

Table: Truth Table of a Four Input Priority Encoder

Inputs				Outputs	
D_0	D_1	D_2	D_3	A	B
0	0	0	0	X	X
1	0	0	0	0	0
X	1	0	0	0	1
X	X	1	0	1	0
X	X	X	1	1	1

According to the truth table, the higher the subscript number, the higher the priority of the input.
The X's are don't care conditions indicating that the binary values they represent may be equal to 0 or 1.

3.15. Code Converters

A code converter is a combinational logic circuit which accepts the input information in one binary code, converts it and produces an output into another binary code.

3.15.1. The truth table for 4-bit Binary and its Equivalent BCD

Table: Truth table for 4-bit Binary and its Equivalent BCD

Decimal	Binary Input				BCD Output				
	A	B	C	D	B_4	B_3	B_2	B_1	B_0
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	1
2	0	0	1	0	0	0	0	1	0
3	0	0	1	1	0	0	0	1	1
4	0	1	0	0	0	0	1	0	0
5	0	1	0	1	0	0	1	0	1
6	0	1	1	0	0	0	1	1	0
7	0	1	1	1	0	0	1	1	1
8	1	0	0	0	0	1	0	0	0
9	1	0	0	1	0	1	0	0	1
10	1	0	1	0	1	0	0	0	0
11	1	0	1	1	1	0	0	0	1
12	1	1	0	0	1	0	0	1	0
13	1	1	0	1	1	0	0	1	1
14	1	1	1	0	1	0	1	0	0
15	1	1	1	1	1	0	1	0	1

The minimized expression of outputs are as follows:

$$B_4 = AB + AC$$

$$B_1 = \overline{A}C + AB\overline{C}$$

$$B_2 = \overline{A}B + BC$$

$$B_3 = A\overline{B}\,\overline{C}$$

$$B_0 = D$$

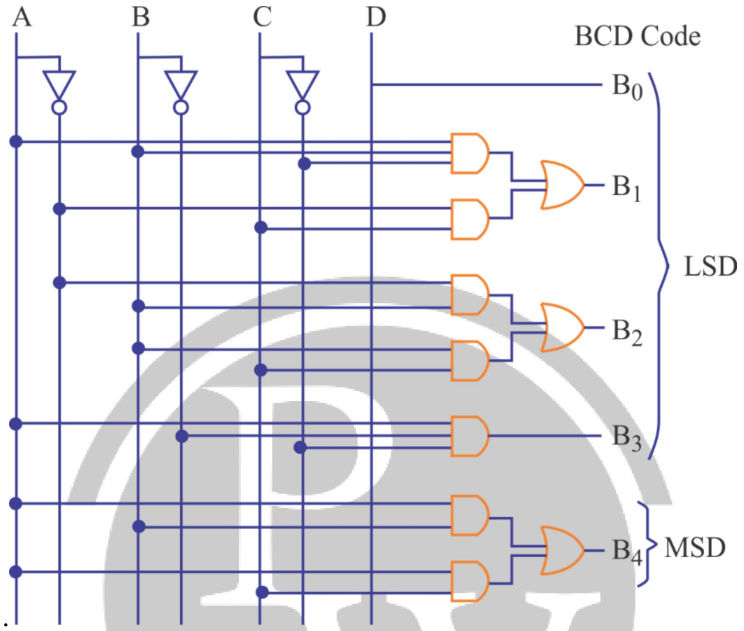


Fig. 3.34. Logic Diagram of a Binary-to-BCD Code Converter

3.16. Parity Generator

Parity generators are circuits that accept an (n-1) bit data stream and generate an extra bit that is transmitted with the bit stream. This extra bit is referred to as the parity bit. The parity added in binary message is such that the total number of 1's in the message can be either odd or even according to the type of parity used.

3.16.1. Even Parity Generator

The even parity generator is a combinational logic circuit that generates the parity bit such that the number of 1's in the message becomes even. The parity bit is '1' if there are odd number of 1's in the data stream and the parity bit is '0' if there are even number of 1's in the data stream.

Truth table for 4-bit data with Even Parity:

Table: Truth table for 4-bit data with Even Parity

4-bit data				Even Parity
A	B	C	D	P
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0

0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

The minimized expression for even parity generator is

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

The logic diagram for the even parity generator is given as

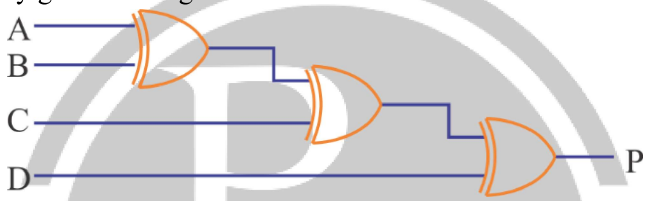


Fig. 3.35. Logic diagram of even parity generator

3.16.2. Odd Parity Generator

The odd parity generator is a combinational logic that generates the parity bit such that the number of 1's in the message becomes odd. The parity bit is '0' for odd number of 1's and '1' for even number of 1's in the bit stream.

Example: A parity generation circuit required to generator on odd parity bit may use ————?

Solution: Odd parity generation circuit consists of combination of EX-OR and EX-NOR gates, whereas even priority generator consists only EX-OR gates.

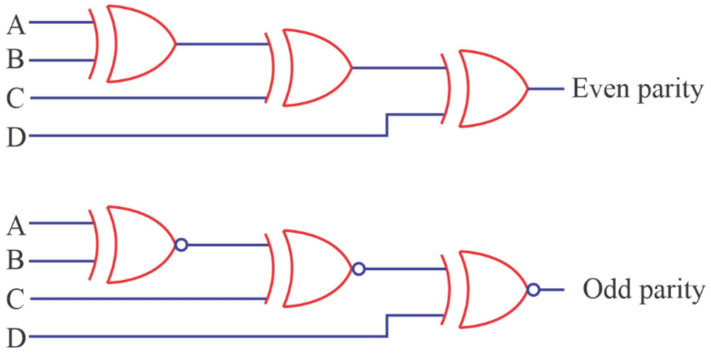


Fig. 3.36

∴ It is combination of EX-OR and EX-NOR gates.

❐❐❐

4 SEQUENTIAL LOGIC CIRCUITS

4.1. Sequential Logic Circuits

In sequential logic circuits, the output is a function of the present inputs as well as the inputs and outputs. Sequential circuit include memory elements to store past data. The flip-flop is a basic element of sequential logic circuits.

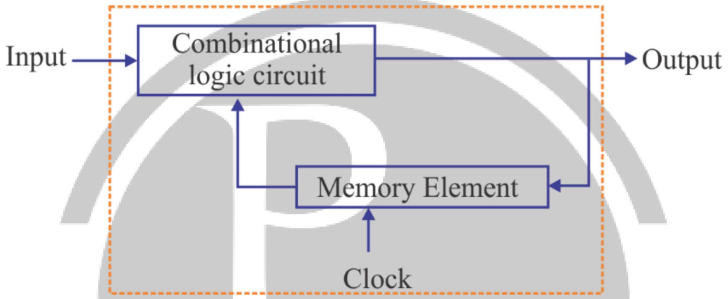


Fig. 4.1. General Block diagram of Sequential Logic Circuit

There are two types of sequential circuits:

4.1.1. Synchronous Circuits

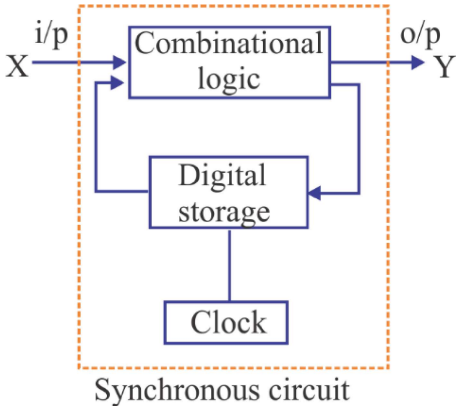
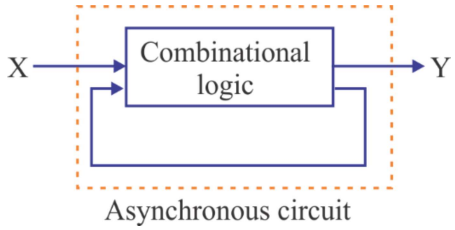
The sequential circuits which are controlled by a clock are called synchronous sequential circuits. These circuits get activated only when clock signal is present.

4.1.2. Asynchronous Circuits

The sequential circuits which are not controlled by a clock are called asynchronous sequential circuits, i.e. the sequential circuits in which events can take place any time the inputs are applied are called asynchronous sequential circuits.

4.2. Difference Between Synchronous and Asynchronous Sequential Circuits

S.No.	Synchronous Sequential Circuits	Asynchronous Sequential Circuits
1.	In synchronous circuits, the change in input signals can affect memory elements upon activation of clock signal.	In asynchronous circuits, change in input signals can affect memory elements at any instant of time.
2.	In synchronous circuits, memory elements are clocked FF's.	In asynchronous circuits, memory elements are either unlocked FF's or time delay elements.
3.	The maximum operating speed of the clock depends on time delays involved.	Since the clock is not present, asynchronous circuits can operate faster than synchronous circuits.
4.	They are easier to design.	More difficult to design.

5.		

4.3. Latches

Flip-flop is an electronic circuit or device which is used to store a data in binary form. Actually, flip-slop is a one-bit memory device and it can store either 1 or 0. Flip-flops is a sequential device that changes its output only when a clocking signal is changing. On the other hand, latch is a sequential device that checks all its inputs continuously and changes its outputs accordingly at any time independent of a clock signal. It refers to non-clocked flip-flops, because these flip-flops, because these flip-flops 'latch on' to a 1 or a 0 immediately upon receiving the input pulse.

4.3.1. General Block Diagram of a Latch or Flip-flop

Figure shown below is the general type of symbol used for a latch. In case of a flip-flop, a clock signal must be shown at input side. It has many inputs and two outputs, labelled Q and $\overline{Q}$. The Q output is the normal output of the latch and $\overline{Q}$ is the inverted output.

Note: A flip-flop is said to be in HIGH state or logic 1 state or SET state when Q = 1, and in LOW state or logic 0 state or RESET state or CLEAR state when Q = 0.

4.3.2. Difference between Latches and Flip-flops

S.No.	Latch	Flip-flop
1.	A latch is an electronic sequential logic circuit used to store information in an asynchronous arrangement.	A flip-flop is an electronic sequential logic circuit used to store information in a synchronous arrangement. It has two stable states and maintains its states for an indefinite period until a clock pulse is applied.
2.	One latch can store one-bit information, but output state changes only in response to data input.	One flip-flop can store one-bit data, but output state changes with clock pulse only.
3.	Latch is an asynchronous device and it has no clock input.	Flip-flop has clock input and its output is synchronised with clock pulse.
4.	Latch holds a bit value and it remains constant until new inputs force it to change.	Flip-flop holds a bit value and it remains constant until a clock pulse is received.
5.	Latches are level-sensitive, and the output tracks the input when the level is high. Therefore, as long as the level is logic level 1, the output can change if the input changes.	Flip-flops are edge sensitive. They can store the input only when there is either a rising or falling edge of the clock.

4.4. Latch

A latch is a type of bistable logic device or multivibrator that is most often used in applications that require data storage. The main characteristics of latch is that the output is not dependent solely on the on the present state of the input but also on the proceeding output state.

Latches are sometimes used for multiplexing data onto a bus. For example, data being input to a computer from a external source have to share the data bus with data from other sources. When the data bus becomes unavailable to external source, the existing data must be temporarily stored, and hence the latches are placed between the external source and data bus.

4.4.1. SR Latch

For the SR latch (S stands for set and R for reset). The logic circuit for SR latch is shown in figure below:

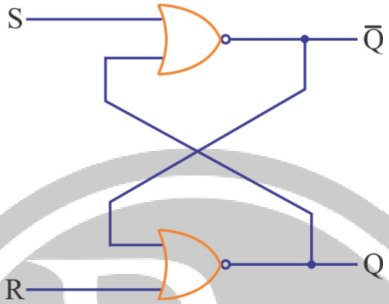


Fig. 4.1. Logic circuit of SR latch.

The state table for the SR latch is:

S	R	Q	Q^+	$\overline{Q^+}$
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	0	1
1	1	1	0	0

The symbol for SR Latch is:

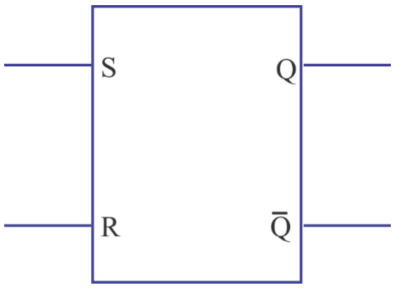


Fig. 4.2.

Obtaining the characteristic equations of the NOR gate based latch are; we get

$$Q^+ = \overline{R} \times S + \overline{R} \times Q = \overline{R} \times (S + Q) \text{ and } \overline{Q}^+ = \overline{S} \times R + \overline{S} \times \overline{Q} = \overline{S} \times (S + \overline{Q})$$

Note: It must be noted that the complementing Q^+ does not yield $\overline{Q}^+$.

Hence, the truth table for SR latch is

S	Q	Q^+	$\overline{Q^+}$	
0	0	Q	$\bar{Q}$	⇒ No change
1	1	0	1	⇒ Reset Q^+ to 0
1	0	1	0	⇒ Set Q^+ to 1
1	1	0	0	⇒ Forbidden state

However, the forbidden state (S = R = 1) is considered a don't care state.

Consider a Timing diagram for SR latch

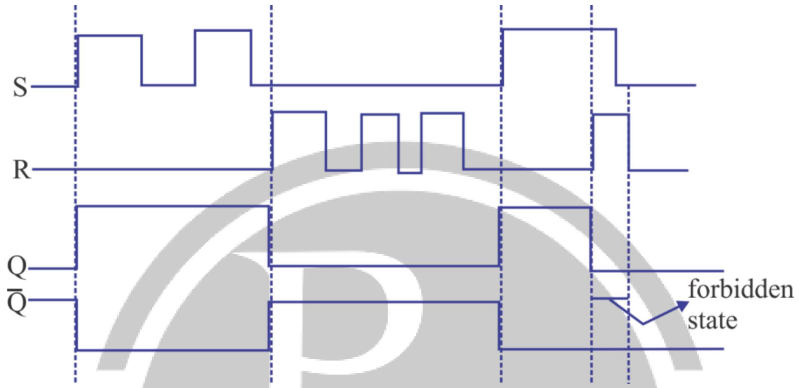


Fig. 4.3.

4. 4.2. $\bar{S}\bar{R}$ Latch:

An $\bar{S}\bar{R}$ latch can be implemented using NAND gates, as shown in figure below

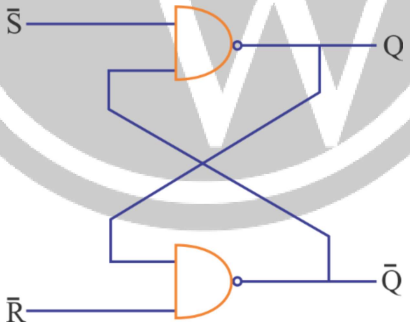


Fig. 4.4. Logic circuit for $\bar{S}\bar{R}$ Latch

The $\bar{S}\bar{R}$ latch is said to be set-dominant 1,

The symbol for $\bar{S}\bar{R}$ latch is shown below:

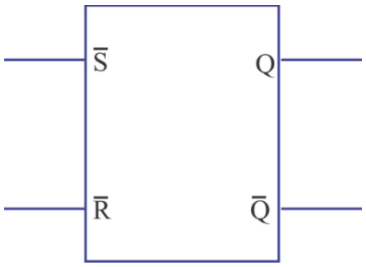


Fig. 4.5.

The truth table for $\overline{S}\overline{R}$ latch is given as:

$\overline{S}$	$\overline{R}$	Q^+	$\overline{Q^+}$	
1	1	Q	$\overline{Q}$	⇒ No change
1	0	0	1	⇒ Reset Q^+ to 0
0	1	1	0	⇒ Set Q^+ to 1
1	1	1	1	⇒ Forbidden state

Application of $\overline{S}\overline{R}$ latch: The application of $\overline{S}\overline{R}$ latch is in switch bouncing i.e. contact bounces of a push-button switch during its opening or closing can be eliminated by using $\overline{S}\overline{R}$ latch.

4.4.3. Gated SR Latch on Enable SR Latch or clocked SR Latch

A gated or level-sensitive SR latch uses a control signal C that can be used as a clock signal or can be used as enable input. The logic circuit diagram, symbol and truth is given as

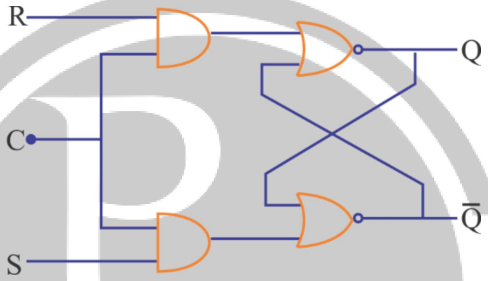


Fig. 4.6. Logic circuit of clocked SR latch.

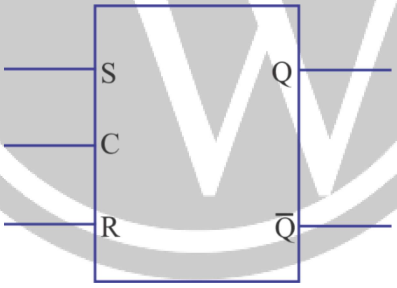


Fig. 4.7. Symbol for clocked SR latch

C	S	R	Q^+	$\overline{Q^+}$	
0	×	×	Q	$\overline{Q}$	} No change state
1	0	0	Q	$\overline{Q}$	
1	0	1	0	1	⇒ Reset
1	1	0	1	0	⇒ Set
1	1	1	0	0	⇒ Forbidden state

4.4.4. Gated $\overline{S}\overline{R}$ Latch or enable $\overline{S}\overline{R}$ Latch or clocked $\overline{S}\overline{R}$ Latch

Gated $\overline{S}\overline{R}$ latch is implemented using two NAND gates and an $\overline{S}\overline{R}$ latch.
The logic circuit diagram, symbol and truth is given as

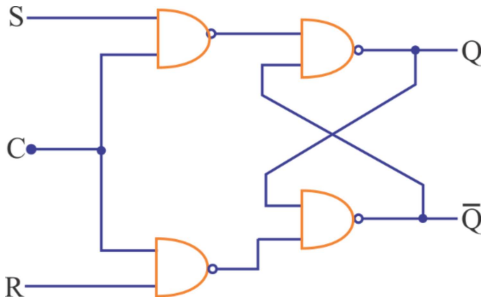


Fig. 4.8. Logic diagram of clocked $\overline{S}\overline{R}$ latch.

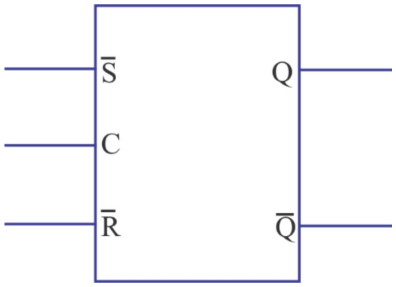


Fig. 4.9.

The truth table of the gated SR latch based on a $\overline{S}\overline{R}$ latch:

C	J	K	Q	$\overline{Q^+}$
O	X	X	Q	$\overline{Q}$
1	0	0	Q	$\overline{Q}$
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1

The characteristic equation for SR flip-flop is given as

$$Q^+ = Q_{n+1} = S + \overline{R}Q_n = S + \overline{R}Q$$

4.5. Flip-Flops

Flip-flops are synchronous bistable devices also known as bistable multivibrator. Its output change its state only at a verified paint (i.e. leading or trailing edge) on the triggering input called the clock (CLK), i.e. changes in the output occur in synchronization with the clock.

Flip-flops are edge-triggered or edge-sensitive whereas gated latches are level-sensitive.

4.5.1. Edge-triggered flip-flop

An edge-triggered flip-flop changes its state either at positive edge (rising edge) or at negative edge (falling edge) of the check pulse.

There are two type of edge-triggered flip-flops. The key to identify an edge-triggered flip-flop is by its logic symbol by small triangle inside the block at the clock input C. This triangle is called the dynamic input indicator.

Positive edge triggered has no bubble at input C whereas negative edge triggered has bubble at input C.

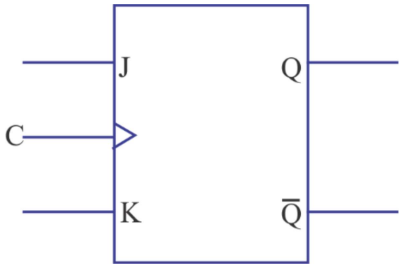


Fig. 4.10. Positive edge triggered flip-flop.

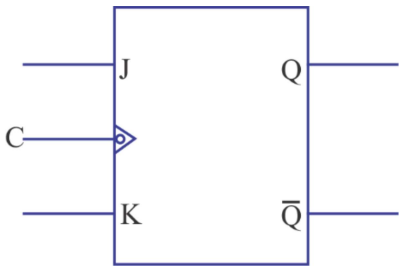


Fig. 4.11. Negative edge-triggered flip-flop.

4.5.2. Basic JK flip-flop

JK Flip-flop (J as a set input and K as a reset input) is the most versatile of the basic flip-flops.

The logic circuit of the gated JK flip-flop is shown in figure below:

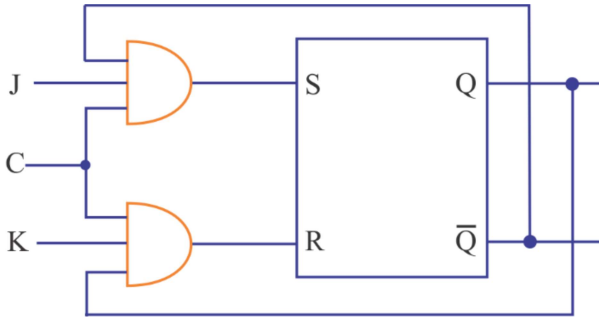


Fig. 4.12. Logic circuit diagram of clocked JK flip-flop.

The state table for the JK flip-flop is given as

C	J	K	Q	Q^+
O	X	X	X	Q
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Hence, the truth table becomes,

C	S	R	Q^+	$\overline{Q}^+$	
0	X	X	Q	$\overline{Q}$	} No change state
1	0	0	Q	$\overline{Q}$	
1	0	1	0	1	⇒ Reset
1	1	0	1	0	⇒ Set
1	1	1	0	0	⇒ Forbidden state

Note: The forbidden state, inherent to SR flip-flop is eliminated by adding two feedback loops such that the output becomes 1 only if Q = 0 and reset to only if Q = 1.

It should also be noted that when the inputs (J & K) are set to 1 and clock signal change to 1, then the feedback value of Q & $\overline{Q}$ forced the flip-flop is toggle its value.

(i.e. to switch its state to its logical complement) hence, to ensure this operation in smooth fashion, the pulse width of the clock must be smaller than the propagation delay of the flip-flop.

The characteristic equation of the JK flip-flop

$$Q_{n+1} = J\overline{Q}_n + \overline{K}Q_n \text{ or } Q^+ = J\overline{Q} + \overline{K}Q$$

4.5.3. T-flip-flop

A JK flip-flop can be transformed into a T- flip-flop (T stands for Toggle). When T flip-flop is activated, its output changes its state at every time a pulse is applied to the input T.
The logic circuit of the gated T flip-flop is.

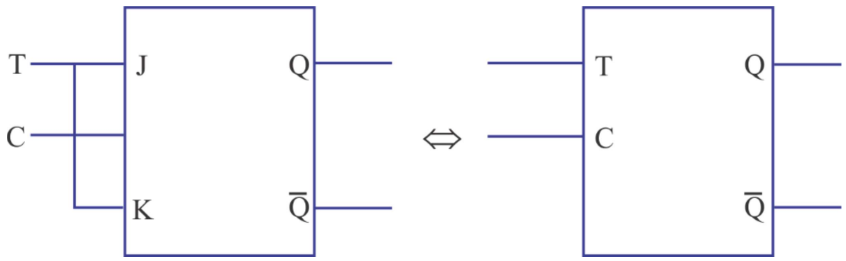


Fig. 4.13.

The state or characteristic table for T flip-flop is

C	T	Q	Q^+
O	X	X	Q
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

As J = K = T, we obtain the characteristic equation as

$$Q^+ = T \times \overline{Q} \times C + (\overline{T} + \overline{C}) \times Q$$

If C = 1, the characteristic the equation is reduced to

$$Q^+ = T \oplus Q$$

$$\text{If } C = 0,\ Q^+ = Q$$

Hence, the truth table of the T-flip flop is given as

C	T	Q^+	$\overline{Q^+}$	
0	×	Q	$\overline{Q}$	⇒ No change state
1	0	Q	$\overline{Q}$	
1	1	$\overline{Q}$		⇒ Toggle

Fig. 4.14.

4.5.4. D Flip-Flop

D-flip-flop can be obtained by use of only two combinations of S-R or J-K flip-flop. It has only one input i.e. D-input or data input.

The logic symbol for D- flip-flop is given as

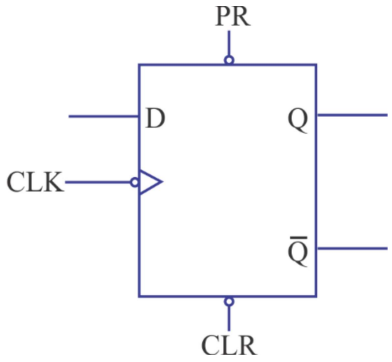


Fig. 4.15.

The truth table for D-flip-flop is

Input	Output
D	Q_{n+1}
0	0
1	1

The characteristic equation of D-flip-flop is:

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

4.5.5. Excitation table of Flip-flops

The truth table of a flip-flop is sometimes referred as characteristic table as it specifies the operational characteristics of the flip-flop there may occurs some situations in which the present state and the next state of the circuit is desired and known. Then the designing of input conditions to as to fulfil the requirements of the circuit, there is a table called excitation table.
It is very important and useful design aid for sequential circuit.

The excitation table for flip-flops:

Present state	Next state	SR Flip-flop		JK Flip-flop		T Flip-flop	D Flip-flop
		S	R	J	K	T	D
0	0	0	×	0	×	0	0
0	1	1	0	1	×	1	1
1	0	0	1	×	1	1	0
1	1	×	0	×	0	0	1

4.6. Operating Characteristics of Flip-Flops

4.6.1. Propagation Delay Time:

Propagation delay time is the time interval required after an input signal has been applied for the resulting output change to occur.

There are four categories of propagation delay times which are as follows:

A. Propagation delay t_{PLH}, it is measured from the triggering edge of the clock pulse to Low-to-High transition of the output.

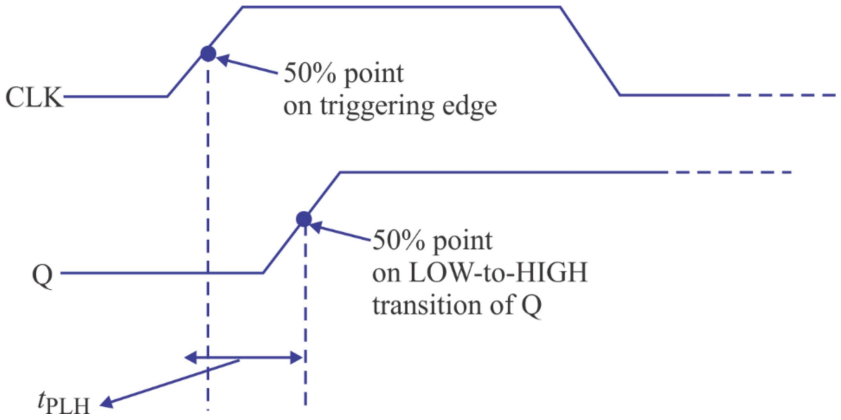


Fig. 4.16.

B. Propagation delay t_{PHL}, it is measured from the triggering edge of the clock pulse to HIGH-to-LOW transition of the output.

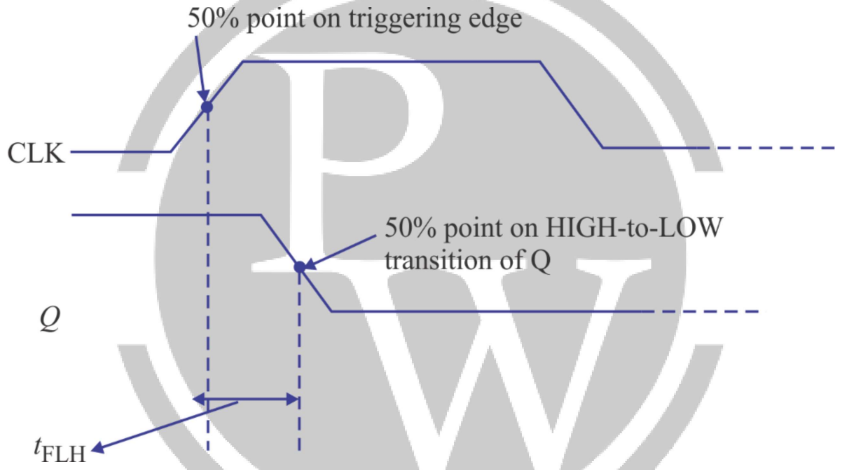


Fig. 4.17.

C. Propagation delay t_{PHL}, it is measured from the leading edge of the PRESET input to LOW-to-HIGH transition of the output.

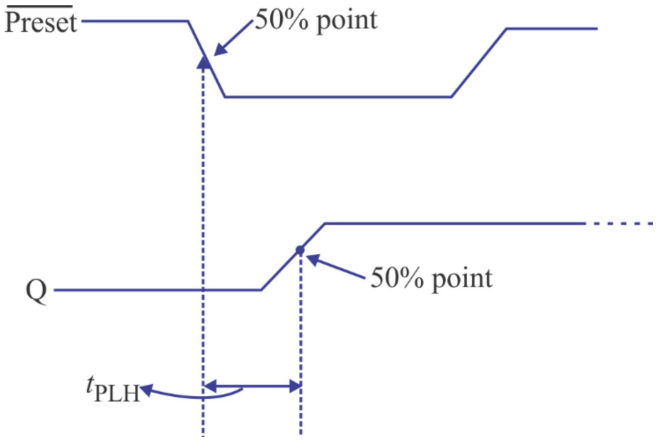


Fig. 4.18.

D. Propagation delay t_{PHL}, it is measured from the leading edge of the clear input to the HIGH-to-LOW transition of the output.

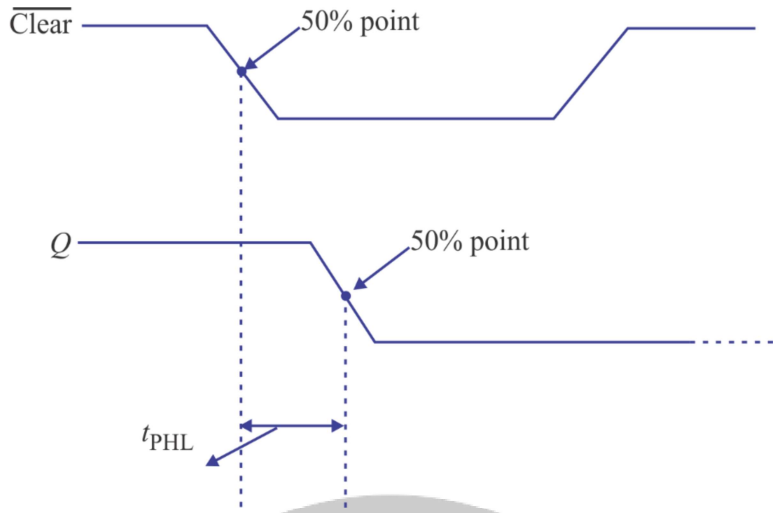


Fig. 4.19.

4.6.2. Set-up time (t_s)

It is the minimum time interval required for the logic levels (0 or 1) to be maintained constantly on the inputs (J, K or D) prior to the triggering edge of the clock pulse in order for the levels to be reliably clocked into the flip-flop.

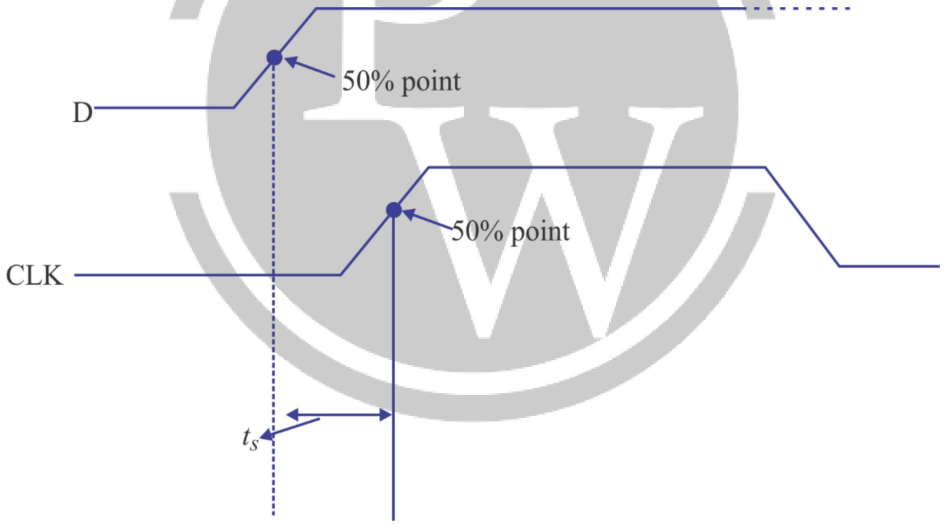


Fig. 4.20.

4.6.3. Hold time (t_h)

It is the time for which the data must remain stable after the triggering edge of the clock.

4.6.4. Clock-pulse width

The minimum time duration for which the clock pulse must remain HIGH and LOW which are designed by manufacturers. Failure to clock pulse width results in unreliable triggering.

4.6.5. Maximum clock frequency

The maximum clock frequency (f_{max}) is the highest rate at which flip-flop can be reliably operated.

4.7. Applications of Flip-Flops

Some of the common applications of flip-flops are as follows:

1. Switch bouncing.
2. Registers.
3. Counters.
4. Memory elements.

4.8. Race Around Condition

JK flip flop suffers from the problem of race around condition. When J = 1 & K = 1, is applied to the JK flip flop and JK flip flop is level triggered then output of the JK flip flop toggles so many times during the pulse width of the clock and output of the flip flop settled either at 1 or 0 depending upon the pulse width of the clock and propagation delay of the flip flop is called race around condition.

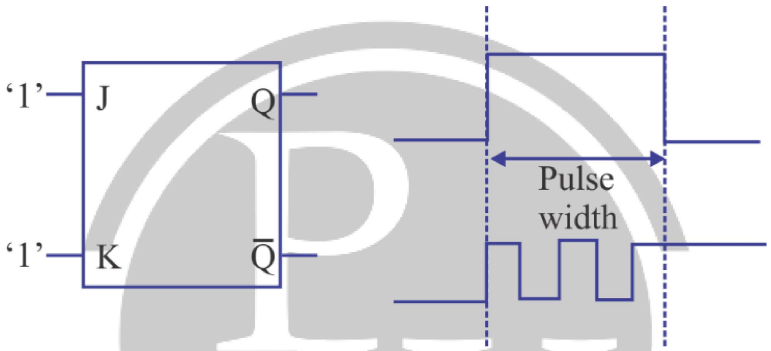


Fig. 4.21. Race Around Condition in JK Flip Flop

To avoid Race Around condition:

- $T_{\text{pulse-width}} < T_{\text{pd}} < T_{\text{clock}}$
- Master Slave flip flop

4.8.1. Master Slave Flip flop:

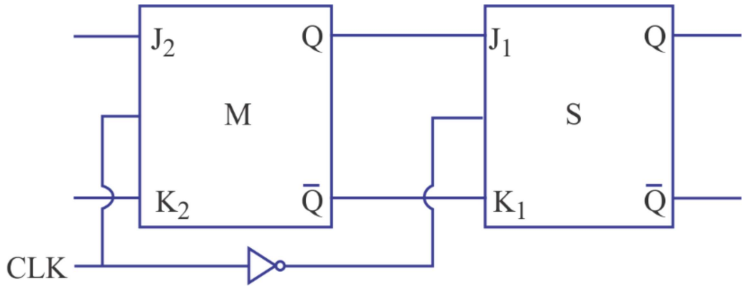


Fig. 4.22. Logic Diagram for Master Slave JK Flip Flop

(a) In master slave flip flop, inverted clock is given to the slave.

(b) Master slave flip flop is used to store single bit because output is taken only from slave flip flop.

(c) Here, master flip flop is level triggered while slave is negative edged triggered.

Note: JK flip flop is also known as Universal flip flop.

4.9. Designing of One Flip Flop by Other Flip Flop

The steps for designing of one flip flop or new flip flop using existing or same existing flip flop.

Step 1: Write the characteristic table for the designed flip flop.

Step 2: Write the excitation table for the available flip flop.

Step 3: Write the logical expression.

Step 4: Minimize the logical expression.

Step 5: Circuit Implementation.

4.10. Shift Registers

An array of flip-flops in required to store binary information, and the number of flip-flops required is equal to the number of bits used to store is referred as registers.
Examples of registers are general purpose registers flags, etc.

Now, the information or data can be stored or entered in serial form (one-bit at a time) or in parallel form (all the bits simultaneously) and can be retrieved like this manner too. The data will be entered or retrieved in serial form is known as temporal code and which is in parallel form is called special code.
Hence, registers can be classified into four categories depending upon the data being entered or retrieved.

4.10.1. SISO (serial-in, serial-out) Shift Register:

In serial-in, serial-out shift register, data input is in serial form and common clock pulse is applied to each of the flip-flop. After each clock pulse, data moves by one position. The output can be obtained in serial form, as shown in figure below:

Fig. 4.23. SISO Shift Register

- It is the slowest shift register among all the shift registers.
- To store n-bits in a n-bit SISO register, then the minimum "n" clock pulses are required.
- To retrieve n-bits from a n-bit SISO register, then the minimum "(n-1)" clock pulses are required.

4.10.2. SIPO (serial-in, parallel-out) Shift Register

In serial-in, parallel-out shift register, data is applied at the input of register in serial form and the output can be obtained in parallel form after the completely shifting of data in register. Figure below shows the serial input data, and then parallel output.

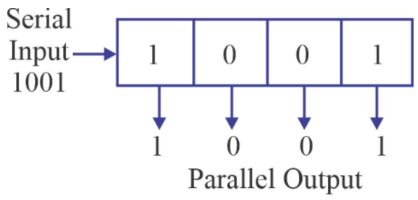


Fig. 4.24. SIPO Shift Register

- To store n-bits in a n-bit SIPO register, the minimum "n" clock pulses are required.
- To retrieve n-bits from a n-bit SIPO register, there is no pulse required.

4.10.3. PISO (parallel-in, serial out) Shift Register

In parallel-in, serial-out shift register, data is loaded into shift register in parallel form and the data output obtained will be serial form as shown in figure below:

- To store n-bit in a n-bit PISO register, a single clock pulse is required.
- To retrieve n-bit from n-bit PISO register, the minimum "(n-1)" clock pulses are required.

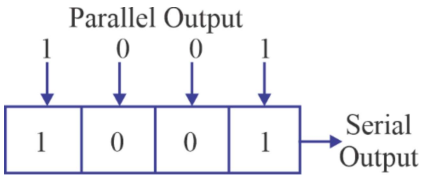


Fig. 4.25. PISO shift register

4.10.4. PIPO (parallel in, parallel out) Shift Register

In parallel-in, parallel-out shift register, data is loaded in parallel form and the data output obtained will be in parallel, as shown in figure below:

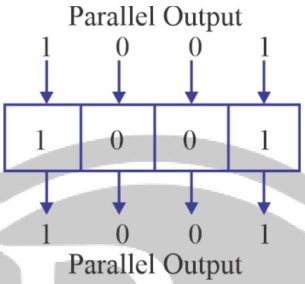


Fig. 4.26. PIPO Shift Register

- To store n-bit in n-bit PIPO register, only a single clock pulse is required.
- To retrieve n-bits from n-bit PIPO register, no clock pulse is required.

Serial Input: The data in the serial form is applied at the serial input after clearing the flip-flops using CLR.

The waveform of serial input shift register is shown below:

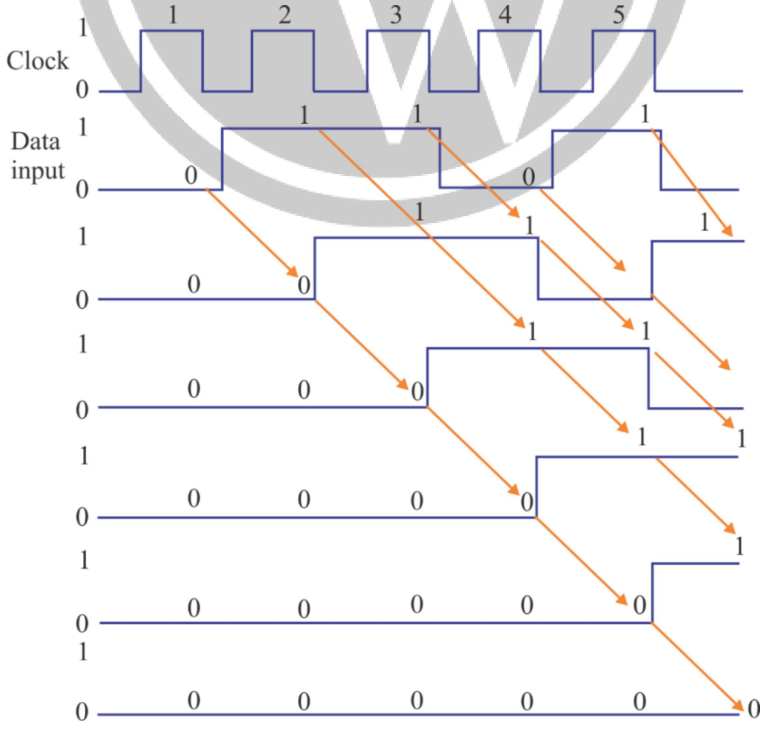


Fig. 4.27.

Parallel Input: Data can be entered in the parallel form making use of the pre-set inputs. Then after clearing the flip-flops, if the data lines are connected to the parallel lines and '1' is applied to the PRESET input.

4.10.5. Universal Shift Register

If the flip-flop outputs of a shift register are accessible, then information entered serially by shifting can be taking out in parallel from the outputs of the flip-flops. If a parallel

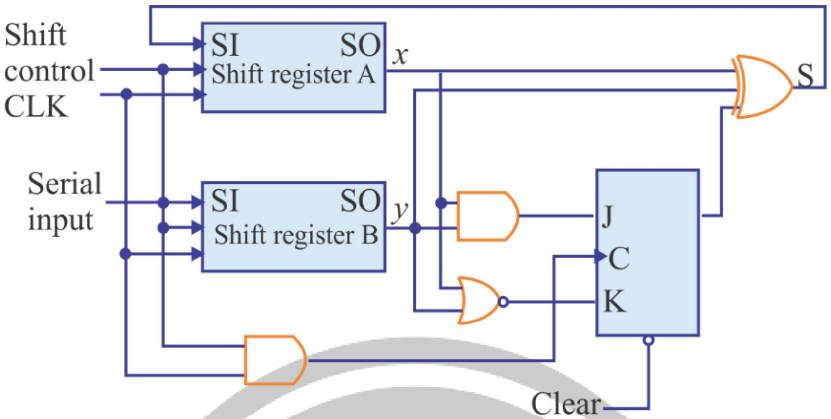


Fig. 4.28. Second form of serial adder

load capability is added to a shift register, then data entered in parallel can be taken out in serial fashion by shifting the data stored in the register. Some shift registers provide the necessary input and output terminals for parallel transfer. They may also have both shift-right and shift-left capabilities.

The most general shift register has the following capabilities:

1. A clear control to clear the register to 0.
2. A clock input to synchronize the operations.
3. A shift-right control to enable the shift-right operation and the serial input and output lines associated with the shift right.
4. A shift-left control to enable the shift-left operation and the serial input and output lines associated with the shift left.
5. A parallel-load control to enable a parallel transfer and the n input lines associated with the parallel transfer.
6. "n" parallel output lines.
7. A control state that leaves the information in the register unchanged in response to the clock.

Other shift registers may have only some of the preceding functions, with at least one. shift operation. A register capable of shifting in one direction only is a unidirectional shift register. One that can shift in both directions is a bidirectional shift register. If the register can shift in both directions and has parallel-load capabilities, it is referred to as a universal shift register. The block diagram symbol and the circuit diagram of a four-bit universal shift register is shown in figure below:

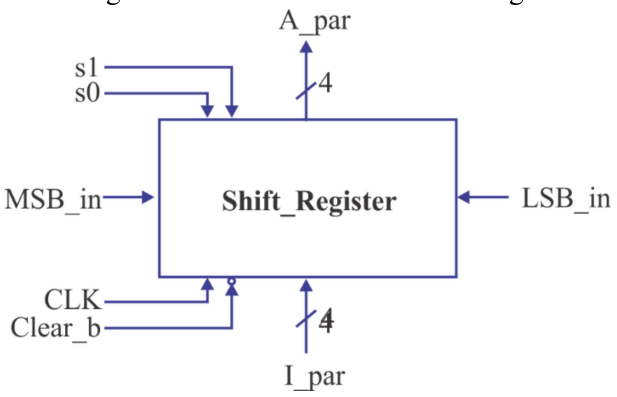


Fig. 4.29. 4-bit Universal Shift Register

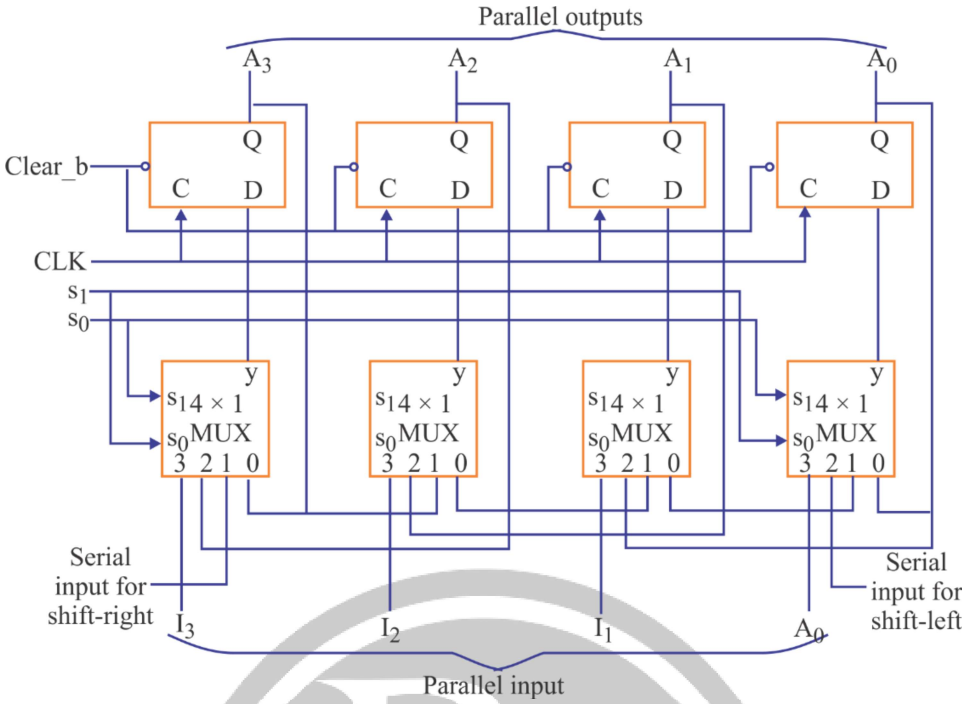


Fig. 4.30. Logic Diagram of 4-bit Universal shift register

The function for the Universal Shift Register is as follows:

Mode Control		Register Operation
S_1	S_0	
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

Shift registers are often used to interface digital systems situated remotely from each other. For example, suppose it is necessary to transmit an n-bit quantity between two points. If the distance is far, it will be expensive to use n lines to transmit the its bits in parallel. It is more economical to use a single line and transmit the information serially, one bit at a time. The transmitter accepts the n-bit data in parallel into a shift register and then transmits the data serially along the common line. The receiver accepts the data serially into a shift register. When all n bits are received, they can be taken from the outputs of the register in parallel. Thus, the transmitter performs a parallel-to-serial conversion of data and the receiver does a serial-to-parallel conversion.

4.10.6. Applications of Shift Registers

(a) **Delay line:** A shift register can be used to introduce a delay (Δt) in signals

$$\Delta t = N \times \frac{1}{f_c}$$

Where N is number of stages & f_c is the clock frequency.

(b) Serial-to-parallel converter

(c) Parallel-to-serial converter

(d) Ring counter
(e) Twisted ring counter
(f) Sequence counter

4.11. Asynchronous Counter Or Ripple Counter

A circuit which is used for counting the numbers or pulses is known as counter. Counter is referred to as modulo-N (or divide by N), where the word modulo indicates the number of states in the counter.

4.11.1. 3- Bit Binary Counter

Consider a 3-bit binary counter which has total '8' number of states which require three flip-flops and Q_2, Q_1 and Q_0 are the outputs of those flip-flops.

The circuit diagram or logic circuit diagram for 3-bit binary counter,

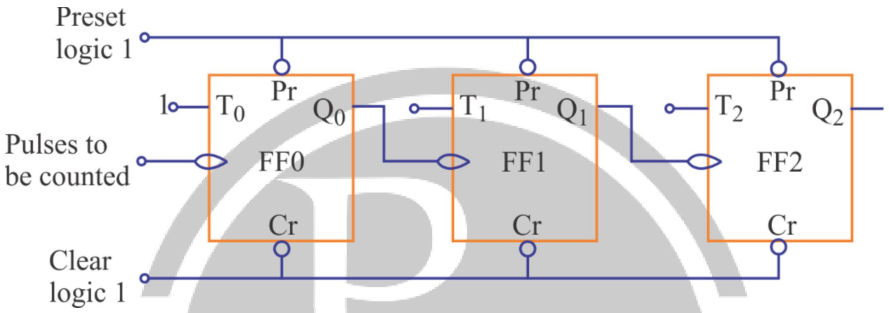


Fig. 4.31. A 3-bit Binary Counter

The truth table for 3-bit binary counter is given as:

Counter state	Count Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Output waveforms of the above counter is:

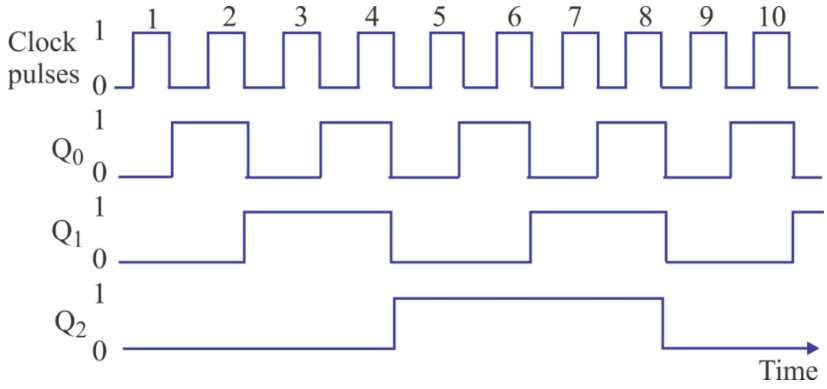


Fig. 4.32.

The frequency 'f' of clock pulses for reliable operation of the counter is given as

Where, N = number of flip-flops

t_{pd} = propagation delay of one flip-flop.

T_s = strobe pulse width.

If during the operation of counter, if some pulses are falsely operated for short duration, known as spikes, which change the state of the flip-flop. It may happen when the propagation delay of each flip-flop may vary and may happen that, all the flip-flops may not change their states or may be only one flip-flop changes its state during the pulse time.

This problem of spikes can be eliminated by using a strobe pulse with the help of strobe pulse, the state will charge only when flip-flops of the counter are in steady state.

Example: In a 4-stage ripple counter, the propagation delay of a flip-flop is 50n sec. If the pulse width of the strobe is 30n sec. Find the maximum frequency at which the counter operator reliably.

Solution: The maximum frequency is

$$f_{max} = \frac{1}{nt_{pd} + t_s}$$

n = number of flip-flops or stage = 4

t_{pd} = propagation delay of each flip-flop = 50 nsec.

t_s = Strobe pulse width = 30 nsec

$$f_{max} = \frac{1}{(4 \times 50 + 30) \times 10^{-9}} = \frac{1000}{(200 + 30)} \text{ MHz} = \frac{1000}{230} \text{ MHz}$$

4.11.2. Modulo-6 Asynchronous Down Counter

Down counter is the counter which counters the values of pulses in descending order. Consider or Stable-8 counter ($2^3 = 8$), which uses three flip-flops.

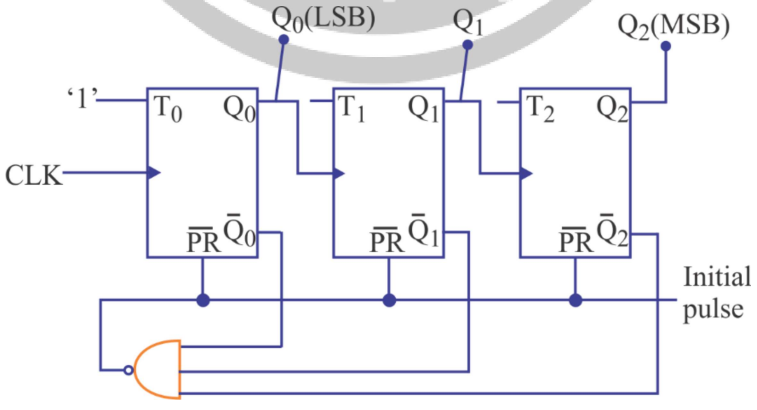


Fig. 4.33.

1. Only sequential counter can be designed. Random counter cannot be designed.
2. Glitch (undesirable state) would appear in case of asynchronous counter.
3. Speed of asynchronous counter is not fast.

4.11.5. BCD Ripple Counter

A decimal counter follows a sequence of 10 states and returns to 0 after the count of 9. Such a counter must have at least four flip-flops to represent each decimal digit, since a decimal digit is represented by a binary code with at least four bits. The sequence of states in a decimal counter is dictated by the binary code used to represent a decimal digit. If the BCD code is used, the sequence of states is as shown in the state diagram. A decimal counter is similar to a binary counter, except that the state after 1001 (the code for decimal digit 9) is 0000 (the code for decimal digit 0).

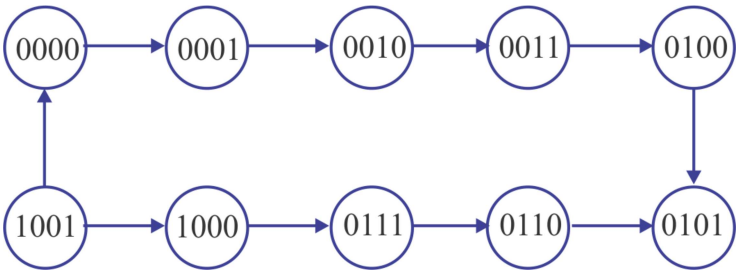


Fig. 4.34. State diagram of a decimal BCD counter.

The logic diagram of a BCD ripple counter using JK flip-flops is shown in figure below. The four outputs arc designated by the letter symbol Q, with a numeric subscript equal to the binary weight of the corresponding bit in the BCD code. Note that the output of Q_1 is applied to the C inputs of both Q_2 and Q_4 and the output of Q_2 is applied to the C and output of Q_2 and Q_3 applied to J through a two input AND gate.

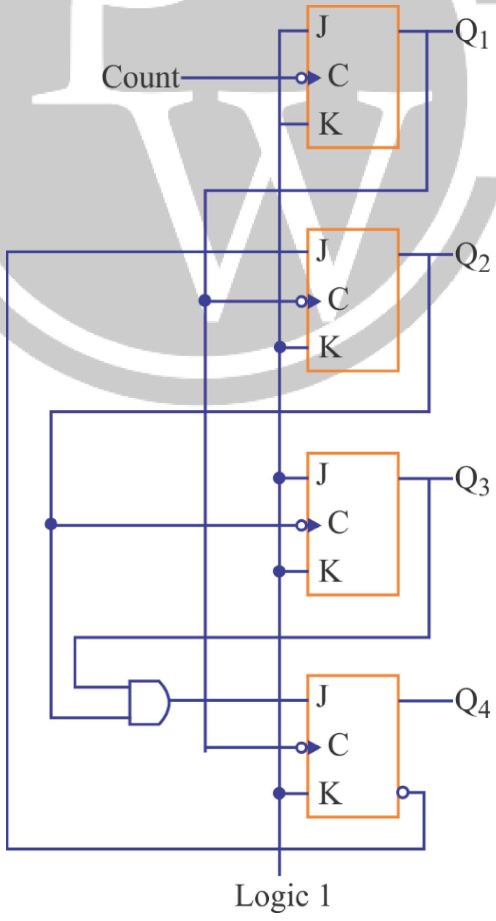


Fig. 4.35. BCD ripple counter

4.12. Synchronous Counter

The ripple counters have the advantage of simplicity (only FLIP-FLOP's are required) but their speed is low because of ripple action. The maximum time is required when the output changes from 111....1 to 00....0 and this limits the frequency of operation of ripple counters.

The speed of operation improves significantly if all the FLIP-FLOPS are clocked simultaneously. The resulting circuit is known as a synchronous counter. Synchronous counters can be designed for any count sequence (need not be straight binary).

The output Q_0 of the least-significant FLIP-FLOP changes for every clock pulse. This can be achieved by using a T-type FLIP-FLOP with $T_0 = 1$. The output Q_0 changes whenever Q_0 changes from 1 to 0. Therefore, if Q_0 is connected to T input (T_1) of the next FLIP-FLOP, Q_1 will change from 1 to 0 (or 0 to 1) when $Q_0 = 1$ ($T_1 = 1$) and will remain unaffected when $Q_0 = T_1 = 0$. Similarly, Q_2 changes whenever Q_1 and Q_0 are both "1". This can be achieved by making the T-input (T_2) of the most-significant FLIP-FLOP equal to $Q_1.Q_0$.

In addition to FF's, synchronous counters require some gates also. JK FLIP-FLOPS are the most commonly used FLIP-FLOP's for the design of synchronous counters. In this, each FLIP-FLOP has two control inputs (J and K) and circuit is required to be designed for each control input. Many programmable logic devices (PLDs) used for the design of digital systems utilise D FLIP-FLOPS for their memory elements, therefore, counter design using D FLIP-FLOPS will be useful for programming inside a PLD. It has only one control input which makes its design simpler than the design using J-K FLIP-FLOPS.

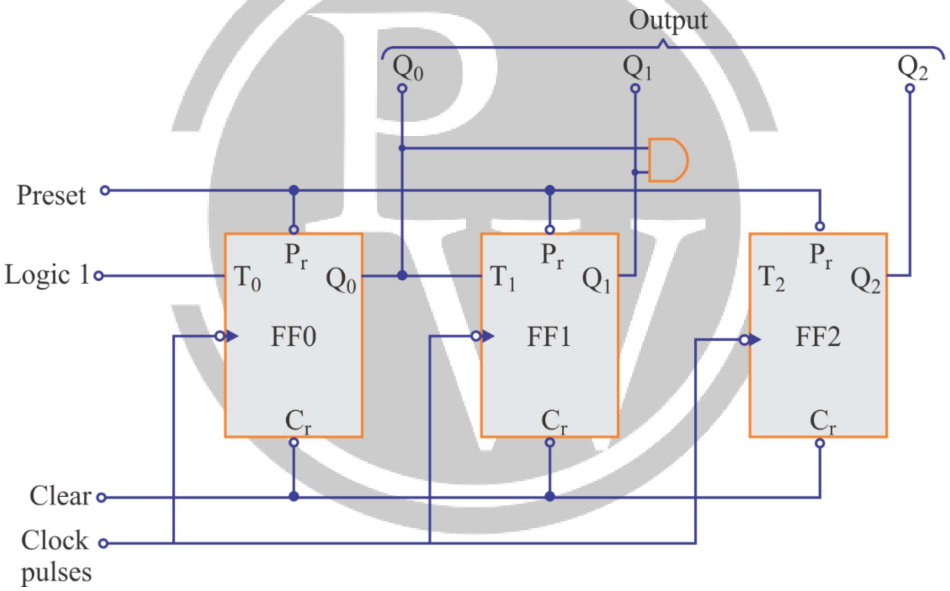


Fig. 4.36. A 3-bit Synchronous Counter

4.12.1. Synchronous Counter Design

Synchronous counters for any given count sequence and modulus can be designed in the following way:

1. Find the number of FLIP-FLOPs required.
2. Write the count sequence in the tabular form.
3. Determine the FLIP-FLOP inputs which must be present for the desired next state from the present state using the excitation table of the FLIP-FLOPS.
4. Prepare K-map for each FLIP-FLOP input in terms of FLIP-FLOP outputs as the input variables.
5. Simplify the K-maps and obtain the minimized expressions.
6. Connect the circuit using FLIP-FLOPS and other gates corresponding to the minimized expressions.

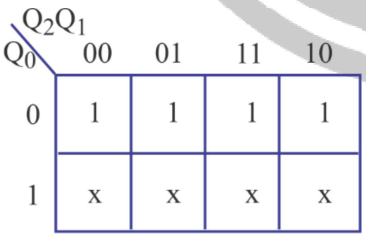
Example: Design a 3-bit synchronous counter using JK Flip-Flops.

Solution: The number of FLIP-FLOPs required is 3. Let the FLIP-FLOPs be FF0, FF1, FF2 and their inputs and outputs are given below:

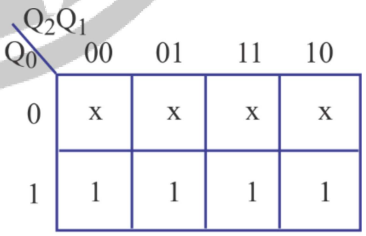
FLIP-FLOP	Inputs	Outputs
FF0	J_0, K_0	Q_0
FF1	J_1, K_1	Q_1
FF2	J_2, K_2	Q_2

The count sequence and the required inputs of FLIP-FLOPs is shown below.

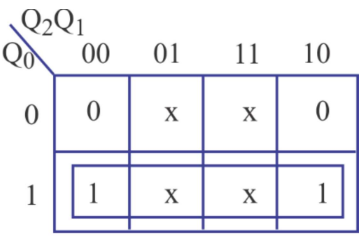
			FLIP-FLOP INPUTS					
Counter state			FF0		FF1		FF2	
Q_2	Q_1	Q_0	J_0	K_0	J_1	K_1	J_2	K_2
0	0	0	1	X	0	X	0	X
0	0	1	X	1	1	X	0	X
0	1	0	1	X	X	0	0	X
0	1	1	X	1	x	1	1	X
1	0	0	1	X	0	X	X	0
1	0	1	X	1	1	X	X	0
1	1	0	1	X	X	0	X	0
1	1	1	x	1	X	1	x	1
0	0	0						



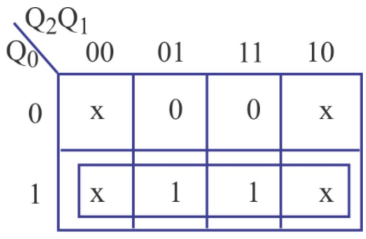
$J_0 = 1$
(a)



$K_0 = 1$
(b)



$J_1 = Q_0$
(c)



$K_1 = Q_0$
(d)

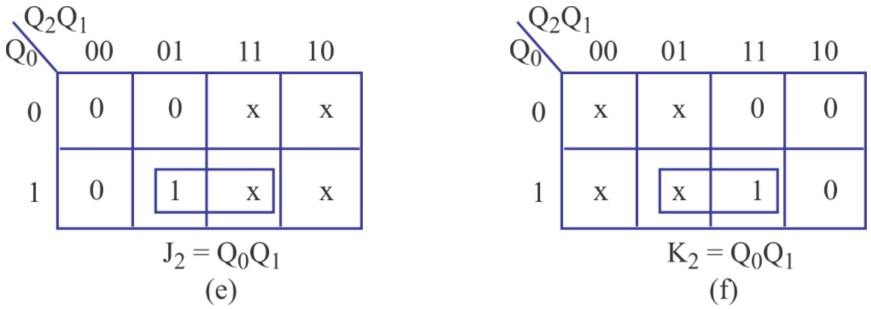


Fig. 4.37. K-Maps of 3-bit Synchronous Counter

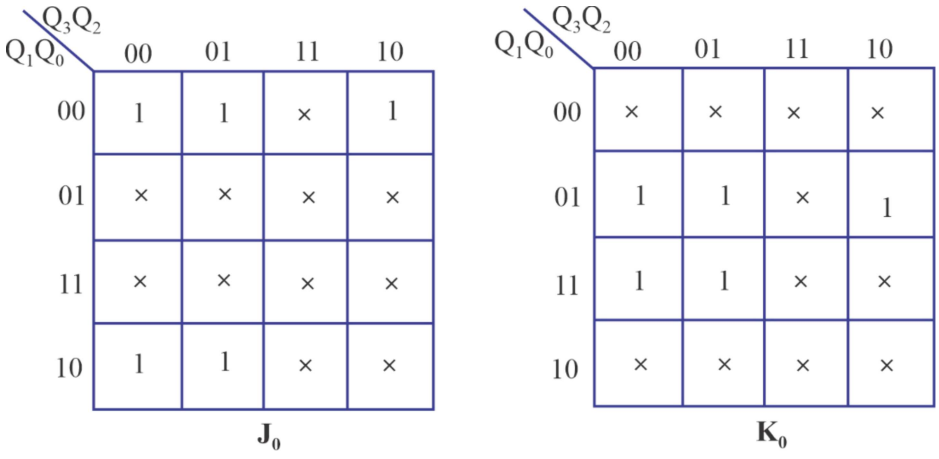
Example:

Design a natural binary sequence mod-8 synchronous counter using D FLIP—FLOPS.

Solution:

The number of FLIP-FLOPS required is 3. Let the FLIP—FLOPS be FF0, FF1 and FF2 with inputs D_0, D_1 and D_2, respectively. Their outputs are Q_0, Q_1, and Q_2 respectively

Counter State			FLIP-FLOP inputs		
Q_2	Q_1	Q_0	D_0	D_1	D_2
0	0	0	1	0	0
0	0	1	0	1	0
0	1	0	1	1	0
0	1	1	0	0	1
1	0	0	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1
1	1	1	0	0	0



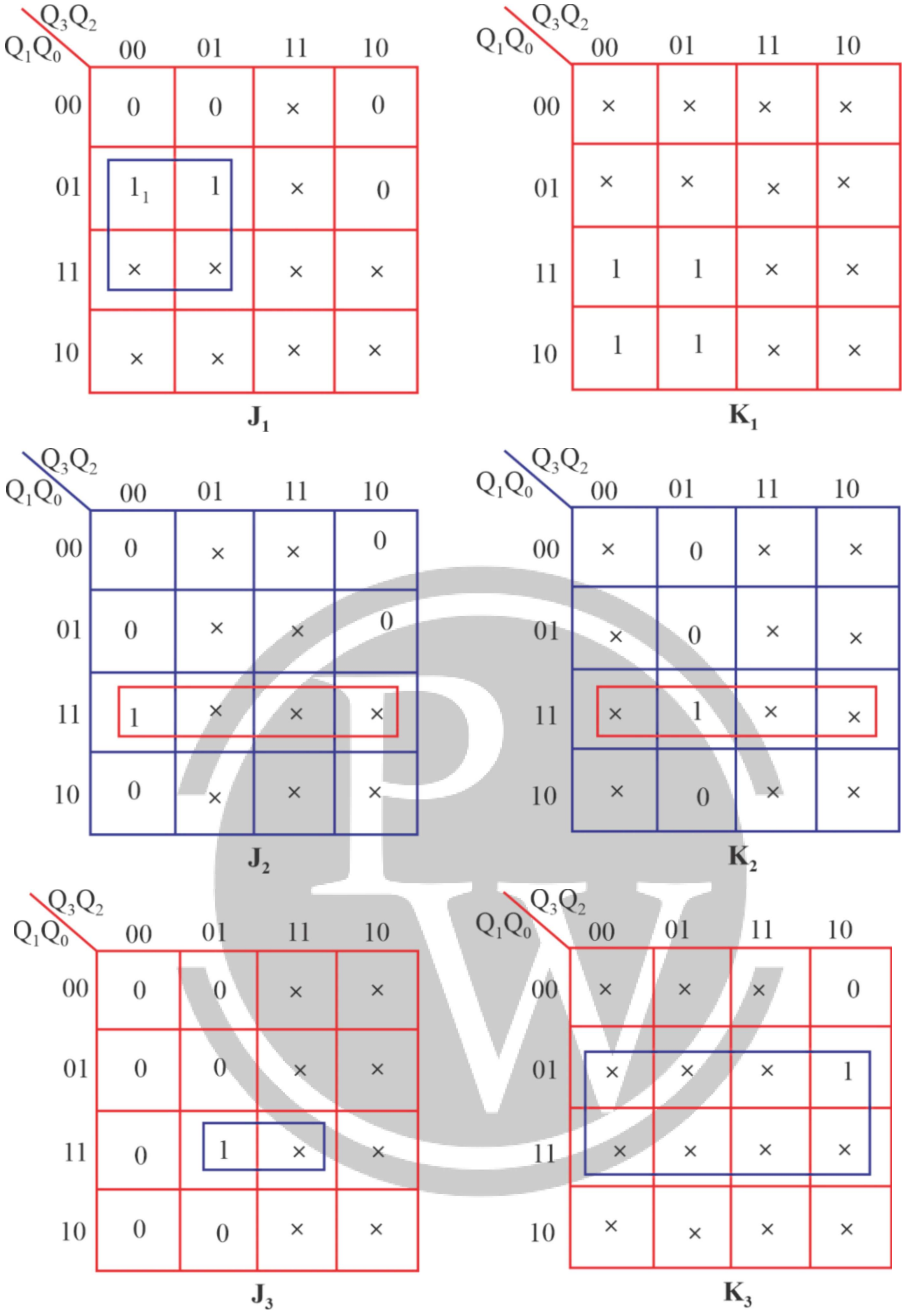


Fig. 4.38. K-Maps for 8-bit Synchronous counter

The K – maps for D_0, D_1 and D_2 are given as,

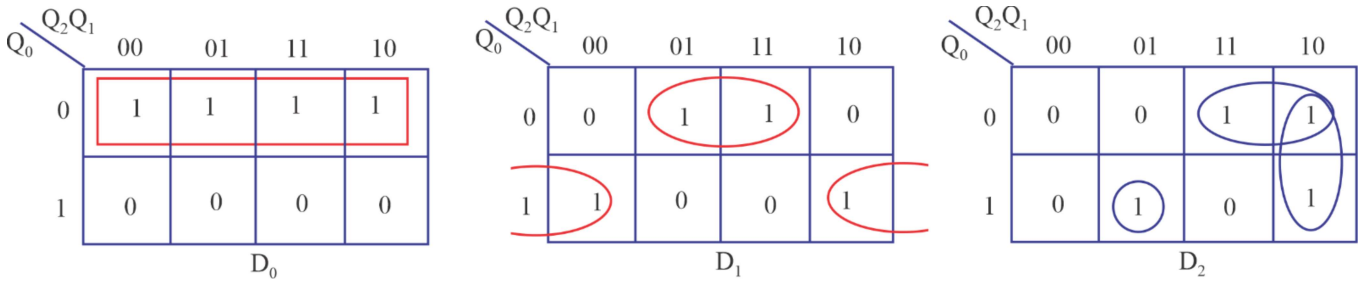


Fig. 4.39.

The minimised expressions for D0, D1 and D2 are:

$$D_0 = \bar{Q}_0$$

$$D_1 = Q_1\bar{Q}_0 + \bar{Q}_1 Q_0$$

$$D_2 = Q_2\bar{Q}_0 + Q_2\bar{Q}_1 + Q_2Q_1Q_0 = Q_2(\bar{Q}_0 + \bar{Q}_1) + \bar{Q}_2Q_1Q_0 = Q_2(\overline{Q_0 \cdot Q_1}) + \bar{Q}_2(Q_1Q_0)$$

$$= Q_2 \oplus Q_1 \cdot Q_0$$

The complete circuit of the synchronous counter using positive edge triggered D FLIP-FLOPs is shown in figure below as

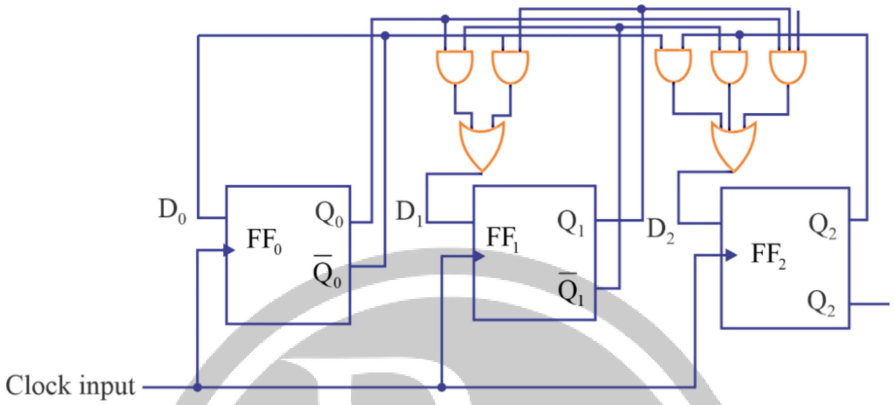


Fig. 4.40. 8-bit Synchronous Counter Circuit

4.12.2 Synchronous Sequential Circuit Models

A general block diagram of clocked sequential circuit is also known as finite stable machine (FSM). Depending upon the external outputs, there are two types of models of sequential circuits.

Mealy Model

In mealy model, the next of the function depends on present state as well as present inputs.

Moore Model

In more model, the next state depends on the present state. The block diagram of a Moore model is given as

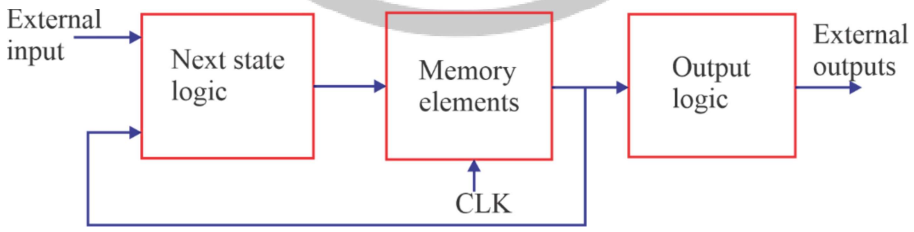


Fig. 4.41.

The systematic procedure for designing of clocked sequential circuit is based on the concept of 'state'. Hence the sequence of inputs, present & next states and output is represented by a state table or state diagram & if the procedure follows in the form of flow chart, it is known as algorithms state machine (ASM).

4.12.3. State Diagram

It is a directed graph, consisting of vertices (or nodes) and directed are between the nodes. Every state of the circuit is represented by a node in the graph. A node is represented by a circle with the name of the state written inside the circle. The directed are represents the state transitions.

With the circuit in may one state, at the occurrence of a clock pulse, there will be a state transition to the next state and there will be an output, corresponding to the requirement of the circuit. This state transition is represented by a directed line and we use each (/) for representing present state and the next state.

Example:

Draw the state diagram of D-flip-flop.

Solution:

The D flip-flop has only input (D) & two output states (Q = 0 & Q = 1).

Using the state table or characteristic table of the D-flip flop. The state diagram is given as

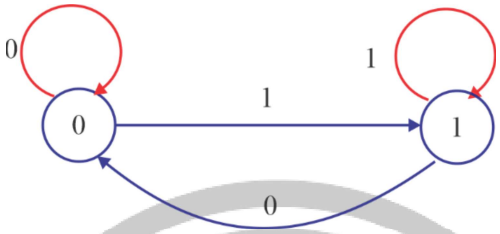


Fig. 4.42.

Example:

Draw the state diagram of a JK flip-flop.

Solution:

A JK flip-flop has inputs (J & K) and one clock input (CLK) and the two output states (Q = 0 & Q = 1).

Using the state table or characteristic table of the JK flip-flop, the state diagram is given as

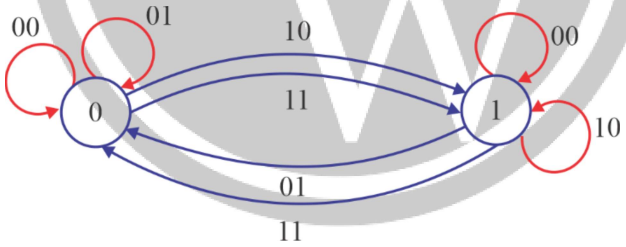


Fig. 4.43.

5 NUMBER SYSTEM

5.1. NUMBER SYSTEM

5.1.1. Base (Radix)

Total number of digit used in the system

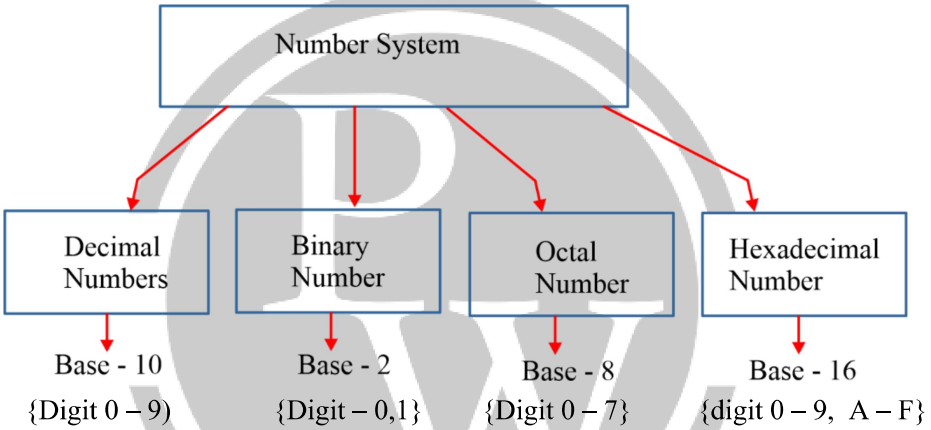


Fig. 5.1.

5.1.2. Decimal Number System

$\ldots \quad 10^4 \quad 10^3 \quad 10^2 \quad 10^1 \quad 10^0 \quad 10^{-1} \quad 10^{-2} \quad 10^{-3} \ldots$

$\ldots \quad a_4 \quad a_3 \quad a_2 \quad a_1 \quad a_0 \quad a_{-1} \quad a_{-2} \quad a_{-3} \ldots$

$a_i \rightarrow$ Coefficient of decimal number system

$10^i \rightarrow$ Weight of decimal number system

Example: - $(501.23)_{10}$

10^2	10^1	10^0	10^{-1}	10^{-2}
5	0	1	2	3

Base	Digit
2	0, 1
3	0, 1, 2
4	0, 1, 2, 3
5	0, 1, 2, 3, 4
6	0, 1, 2, 3,4,5

7	0, 1, 2, 3, 4, 5, 6
8	0, 1, 2, 3, 4, 5, 6, 7
9	0, 1, 2, 3, 4, 5, 6, 7, 8
10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
11	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A
12	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B
13	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C
14	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D
15	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E
16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

5.1.3 Binary Number System (Base (Radix) = 2)

…	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3} …
…	a_4	a_3	a_2	a_1	a_0	a_{-1}	a_{-2}	a_{-3} …

2^i → Weight of Binary number system

a_i → Coefficient of Binary number system {0, 1}

Example:- $(101.11)_2$

2^2	2^1	2^0	2^{-1}	2^{-2}
1	0	1	1	1

5.1.4. Octal Number System (Base (Radix) = 8)

…	8^3	8^2	8^1	8^0	8^{-1}	8^{-2}	8^{-3}…
…	a_3	a_2	a_1	a_0	a_{-1}	a_{-2}	a_{-3}…

8^i → Weight of Octal number system

a_i → Coefficient of Octal number system {0 -7}

Example:- $(728.64)_8$

8^2	8^1	8^0	8^{-1}	8^{-2}
7	2	8	6	4

5.1.5 Hexadecimal Number System (Base (Radix) = 16):

…	16^3	16^2	16^1	16^0	16^{-1}	16^{-2}	16^{-3}…
…	a_3	a_2	a_1	a_0	a_{-1}	a_{-2}	a_{-3}…

16^i → Weight of Hexadecimal number system

a_i → Coefficient of Hexadecimal number system {0 – 9, A–F}

Example: $(A2C.F)_{16}$

16^2	16^1	16^0	16^{-1}
A	2	C	F

5.1.6. In base conversion 2 key points are there:

(A) Any base to Decimal conversion

(B) Decimal to any other base conversion

(A) Any base to Decimal conversion:

$$(a_3\ a_2\ a_1\ a_0\ .\ a_{-1}\ a_{-2})=(\quad)_{10}$$

$$\left(a_3\times r^3+a_2\times r^2+a_1\times r^1+a_0\times r^0+a_{-1}\times r^{-1}+a_{-2}\times r^{-2}\right)_{10}$$

Case (1) : Binary to Decimal conversion

Ex. $(1011.11)_2=(\quad)_{10}$

$\Rightarrow \left[(1\times2^3)+(0\times2^2)+(1\times2^1)+(1\times2^0)+(1\times2^{-1})+(1\times2^{-2})\right]$

$\Rightarrow [8+0+2+1+0.5+0.25]_{10}$

$\Rightarrow (11.75)_{10}$

Case (2) : Octal to Decimal conversion

Ex. $(721.4)_8=(\quad)_{10}$

$\Rightarrow \left[(7\times8^2)+(2\times8^1)+(1\times8^0)+(4\times8^{-1})\right]_{10}$

$\Rightarrow [448+16+1+0.5]_{10}$

$\Rightarrow (465.5)_{10}$

Case (3) : Hexadecimal to Decimal conversion

Ex. $(A2B.C)_{16}=(\quad)_{16}$

$\Rightarrow \left[(A\times16^2)+(2\times16^1)+(B\times16^0)+(C\times16^{-1})\right]_{10}$

$\Rightarrow \left[(10\times256)+(2\times16)+(11\times1)+(12\times16^{-1})\right]_{10}$

$\Rightarrow [2560+32+11+0.75]_{10}$

$\Rightarrow (2603.75)_{10}$

Case (4) : Base 5 to Decimal conversion

Ex. $(432.22)_5=(\quad)_{10}$

$\Rightarrow \left[(4\times5^2)+(3\times5^1)+(2\times5^0)+(2\times5^{-1})+(2\times5^{-2})\right]_{10}$

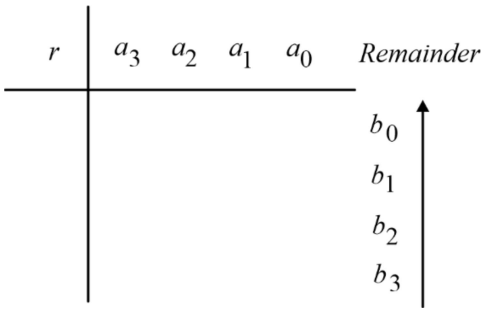
$\Rightarrow [100+15+2+0.4+0.08]_{10}$

$\Rightarrow (117.48)_{10}$

(B) Decimal to any other Base conversion

$$(a_3\ a_2\ a_1\ a_0\ .\ a_{-1}\ a_{-2}\ a_{-3})_{10}=(\quad)_r$$

Before Decimal — After Decimal



$$0 \cdot a_{-1}\, a_{-2}\, a_{-3} \times r = x_0 \cdot x_{-1}\, x_{-2}$$
$$0 \cdot x_{-1}\, x_{-2} \times r = x_1 \cdot x_{-3}\, x_{-4}$$
$$0 \cdot x_{-3}\, x_{-4} \times r = x_2 \cdot x_{-5}\, x_{-6}$$

$$(a_3 a_2 a_1 a_0 \cdot a_{-1} a_{-2} a_{-3})_{10} = (b_3 b_2 b_1 b_0 \cdot x_0 x_1 x_2)_r$$

Case (1) : Decimal to Binary Base conversion.

Ex.

$(19.75)_{10} = (\quad)_2$

Before Decimal R After Decimal

2	19	1
2	9	1
2	4	0
2	2	0
	1	1

$0.75 \times 2 = 1.5$ 1
$0.5 \times 2 = 1.0$ 1

$(19.75)_{10} = (10011.11)_2$

Case (2) : Decimal to Octal Base conversion.

Ex.

$(210.23)_{10} = (\quad)_8$

Before Decimal Remainder After Decimal

8	210	2
8	26	2
	3	3

$0.23 \times 8 = 1.84$ 1
$0.84 \times 8 = 6.72$ 6
$0.72 \times 8 = 5.76$ 5

$(210.23)_{10} = (322.165)_8$

Case (3): Decimal to Hexadecimal Base conversion.

Ex.

$(1228.55)_{10} = (\quad)_{16}$

Before Decimal Remainder After Decimal

16	1228	12 (C)
16	76	12 (C)
	4	4

$0.55 \times 16 = 8.8$ 8
$0.8 \times 16 = 12.8$ 12(C)

$(1228.56)_{10} = (4CC.8C)_{16}$

5.2. Some Special Case

Case (1): Binary to Octal base conversion

Example: $(10110111)_2 = (\quad)_8$

Octal → means base 8

$8 = 2^3$

Every three digits of binary represent one digit of octal

010 110 111

2 6 7

Hence $(10110111)_2 = (267)_8$

Case (2): Binary to Hexadecimal base conversion

Example: $(1011011)_2 = (\quad)_{16}$

Hexadecimal → means base 16

$16 = 2^4$

Every four digits of binary represent one digit of Hexadecimal.

0101 1011

5 11(B)

Hence $(1011011)_2 = (5B)_{16}$

5.1.2. BCD (Binary Coded Decimal)

- In this each digit of the decimal number is represented by its four-bit binary equivalent. It is also called natural BCD or 8421 code. It is weighted code.
- **Excess – 3 Code:** This is an non weighted binary code used for decimal digits. Its code assignment is obtained from the corresponding value of BCD after the addition of 3.
- **BCO (Binary Coded Octal):** In this each digit of the Octal number is represented by its three-bit binary equivalent.
- **BCH (Binary Coded Hexadecimal):** In this each digit of the hexadecimal number is represented by its four bit binary equivalent.

Decimal Digits	BCD 8421	Excess – 3	Octal digits	BCO	Hexadecimal Digits	BCH
0	0000	0011	0	000	0	0000
1	0001	0100	1	001	1	0001
2	0010	0101	2	010	2	0010
3	0011	0110	3	011	3	0011
4	0100	0111	4	100	4	0100
5	0101	1000	5	101	5	0101
6	0110	1001	6	110	6	0110
7	0111	1010	7	111	7	0111
8	1000	1011			8	1000

9	1001	1100			9	1001
					A	1010
					B	1011
					C	1100
					D	1101
					E	1110
					F	1111

Don't care values or unused states in BCD code are 1010, 1011, 1100, 1101, 1110, 1111.

Don't care values or unused states in excess – 3 code are 0000, 0001, 0010, 1101, 1110, 1111.

The binary equivalent of a given decimal number is not equivalent to its BCD value.

Example: $25_{10} = 11001_2$.

The BCD equivalent of decimal number 25 = 00100101 from the above example the BCD value of a given decimal number is not equivalent to its straight binary value.

The BCO (Binary Coded Octal) value of a given Octal number is exactly equal to its straight binary value.

Example: $25_8 = 21_{10} = 010101_2$

The BCO Value of 25_8 is 010101.

From the above example, the BCO value of a given Octal number is same as binary equivalent of the same number.

The BCH (Binary Coded Hexadecimal) value of a given hexadecimal number is exactly equal to its straight binary.

Example: $25_{16} = 37_{10} = 100101_2$

The BCH value of hexadecimal number 25_{16} = 00100101.

From this example the above statement is true.

	Binary	Octal	Decimal	Hexadecimal
Complement	r =2	r = 8	r = 10	r = 16
(r – 1)'s	1's	7's	9's	15's
r's Complement	2's	8's	10's	16's

Example: Add the two Binary numbers 101101_2.

Augned 101101
addend 100111
1111
Sum 1010100

Example: Subtract the Binary number 100111_2 from 101101_2.

Minuend : 101101
Subtracted: 100111
Difference: 000110

Example: Multiple the Binary number 1011_2 from 101_2.

Multiplicand: 1011
Multiplier: X101

```
        1011
       0000
      1011
+
Product: 110111
```

While storing the signed binary numbers in the internal registers of a digital computer} most significant bit position is always reserved for sign bit and the remaining bits are used for magnitude.

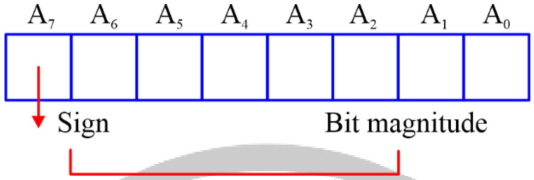


Fig. 5.2.

When the binary number is positive, the sign is represented by '0'. When the number is negative, the sign is represented by ' 1'.

5.2.2. Fixed-Point Representation and Floating-Point Representation;

The representation of the decimal point (ordinary point) in a register is complicated by the fact that it is characterized by a position between two flip- flops in the register.

There are two ways of specifying the position of the decimal point in a register.

(1) Fixed Point and

(2) Floating Point.

The fixed point method assumes that the decimal point (or binary point) is always fixed in one position. The two positions most widely used are (1) a decimal point in the extreme left of the register to make the stored number a fraction, and (2) a decimal point in the extreme right of the register to make the stored number an integer.

The floating-point representation uses a second register to store a number that designates the position of the decimal point in the first register.

Positive numbers are stored in the registers of digital computer in sign magnitude form only.

Negative number can be represented in one of three possible ways.

1. Signed – magnitude representation.
2. Signed – 1's complement representation.
3. Signed – 2's complement representation.

Example: +9 –9

Signed – magnitude 0 0001001 (a) 1 000 1001 signed – magnitude
(b) 1 111 0110 signed – 1's complement
(c) 1 111 0111 signed – 2's complement

The 2's complement of a given binary number can be formed by leaving all least significant zeros and the first non-zero digit unchanged, and then replacing 1's by 0's and 0's by 1's in all other higher significant digits.

Example: The 2's complement of 10011000_2 is 01101000.

Subtraction using 2's complement: Represent the negative number in signed 2's complement from, add the two numbers, including their sign bit, and discard any carry out of the most significant bit.

Since negative numbers are represented in 2's compliment form, negative results also obtained in signed 2's compliment form.

Example: 1's complement:

+ 6 0000110	– 6 1111001	+ 6 0000110	– 6 1111001
+ 9 0001001	+ 9 0001001	– 9 1110110	– 9 1110110
+ 15 0001111	+3 (i) 0000010	– 3 1111100	– 15 (1) 1101111
	Carry + 1		Carry + 1
	+ 3 0000011		1110000
	carry		carry

The advantage of signed 2's complement representation over the signed 1's compliment from (and the signed – magnitude form) is that it contains only one type of zero.

The general form of floating – point number is mr^e. Where M = Mantissa, r = base, e = exponent.

Example: $+ 0.3574 \times 10^5$.

The mantissa can be a fixed point fraction or fixed point integer.

Normalization: Getting non-zero digit in the most significant digit position of the mantissa is called Normalization.

- If the floating point number is normalized, more number of significant digits can be stored, as a result accuracy can be improved.
- A zero cannot be normalized because it does not contain a non-zero digit. The hexadecimal code is widely used in digital systems because it is very convenient to enter binary data in a digital system using hexcode.
- The parity of a digital word is used for detecting error in digital transmission. Hollerith code is used for punched card data.
- In weighted codes, each position of the number has specific weight. The decimal value of a weighted code number is the algebraic sum of the weights of those positions in which 1's appears.
- Most frequently used weighted codes are 8421, 2421 code, 5211 code and 8421 code.
- **Reflective Code:** A code is called reflective or self-complimenting, if the code for 9 is the compliment for the code for 0, code for 8 is the compliment from 1 and so on. $2421, 842'1', 5211$ are examples for reflected codes.
- **Sequential Code:** A code is called sequential, if each successive code-is one binary number greater than its preceding code.

Example: 8421

❐❐❐