

# Unit 3

## ALGEBRA AND LOGIC CIRCUITS:

Binary numbers, Number base conversion and Hexadecimal Numbers, Complements, Basic definitions, Basic theorems and properties of Boolean Algebra, Boolean functions, Canonical and Standard forms, Digital Logic gates, Demorgan's Laws, Ex-OR realization using NAND and NOR, Kmaps (Upto 4 variable)

## COMBINATIONAL LOGIC:

Introduction, Design procedure, Adders-Half adder, Full adder

# Number System

Number System is a way to represent numbers in computer architecture.

Four different types of the number system:

- 1.Binary number system (base 2)
- 2.Octal number system (base 8)
- 3.Decimal number system(base 10)
- 4.Hexadecimal number system (base 16)

# Binary Number System:

- According to digital electronics and mathematics, a binary number is defined as a number that is expressed in the binary system or base 2 numeral system.
- It describes numeric values by two separate symbols; 1 (one) and 0 (zero). The base-2 system is the positional notation with 2 as a radix.
- The binary system is applied internally by almost all latest computers and computer-based devices because of its direct implementation in electronic circuits using logic gates.
- Every digit is referred to as a **bit**.
- A binary number consists of several bits. Examples are:

EX: 10101 is a five-bit binary number

101 is a three-bit binary number

100001 is a six-bit binary number

# Decimal to Binary

Convert 4 in binary number system.

Step 1: Divide the number 4 by 2. Use the integer quotient obtained in this step as the dividend for the next step. Continue this step, until the quotient becomes 0.

Dividend	Remainder
$4/2 = 2$	0
$2/2 = 1$	0
$1/2 = 0$	1

Step 2: write the remainder in reverse chronological order. (i.e from bottom to top)

# Binary to Decimal

## 1. What is binary number 1.1 in decimal

**Step 1:** 1 on the left-hand side is on the one's position, so it's 1.

**Step 2:** The one on the right-hand side is in halves, so it's  $1 \times \frac{1}{2}$

**Step 3:** so,  $1.1 = 1.5$  in decimal.

## 2. Write $10.11_2$ in Decimal?

$$S1: 10.11 = 1 \times (2)^1 + 0 (2)^0 + 1 (\frac{1}{2})^1 + 1(\frac{1}{2})^2$$

$$= 2 + 0 + \frac{1}{2} + \frac{1}{2}$$

$$= 2.75$$

• So,  $10.11$  is  $2.75$  in Decimal.

# Octal Number System

- Has a base of eight and uses the numbers from 0 to 7.
- Any number with base 8 is an octal number like  $24_8$ ,  $109_8$ ,  $55_8$ , etc
- The octal numbers, in the [number system](#), are usually represented by binary numbers when they are grouped in pairs of three.
- For example, an octal number  $12_8$  is expressed as  $001010_2$  in the binary system, where 1 is equivalent to 001 and 2 is equivalent to 010.
- Solve an octal number, each place is a power of eight.

$$124_8 = 1 \times 8^2 + 2 \times 8^1 + 4 \times 8^0$$

- Only 3 bits are used to represent Octal Numbers. Each group will have a distinct value between 000 and 111

Note: **Beyond 7, such as 8 and 9 are not octal digits. For example, 19 is not an octal number.**

# Decimal to Octal Number

the octal dabble method is used- decimal number is divided by 8 each time, it yields or gives a remainder.

The first remainder we get is the least significant digit(LSD) and the last remainder is the most significant digit(MSD)

- Ex: 560 is a decimal number, convert it into an octal number.

**Solution:** If 560 is a decimal number, then,

$560/8 = 70$  and the remainder is 0

$70/8 = 8$  and the remainder is 6

$8/8 = 1$  and the remainder is 0

And  $1/8 = 0$  and the remainder is 1

So the octal number starts from MSD to LSD, i.e. 1060

Therefore,  $560_{10} = 1060_8$

Octal Digital Value	Binary Equivalent
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

# Decimal to Octal Number

Convert 0.52 into an octal number.

- **Solution:** The fraction part of the decimal number has to be multiplied by 8.
- $0.52 \times 8 = 0.16$  with carry 4
- $0.16 \times 8 = 0.28$  with carry 1
- $0.28 \times 8 = 0.24$  with carry 2
- $0.24 \times 8 = 0.92$  with carry 1
- So, for the fractional octal number, we read the generated carry from up to down.
- Therefore, 4121 is the octal number.



# Octal to Decimal

- multiply each digit of the given octal with the reducing power of 8.

**Example 1:** Suppose  $215_8$  is an octal number, then it's decimal form will be,

$$\begin{aligned} 215_8 &= 2 \times 8^2 + 1 \times 8^1 + 5 \times 8^0 \\ &= 2 \times 64 + 1 \times 8 + 5 \times 1 = 128 + 8 + 5 \\ &= 141_{10} \end{aligned}$$

**Example 2:** Let 125 is an octal number denoted by  $125_8$ . Find the decimal number.

$$\begin{aligned} 125_8 &= 1 \times 8^2 + 2 \times 8^1 + 5 \times 8^0 \\ &= 1 \times 64 + 2 \times 8 + 5 \times 1 = 64 + 16 + 5 \\ &= 85_{10} \end{aligned}$$

# Binary To Octal Number

Convert  $(100010)_2$  to an octal number.

100---4

010---2

Therefore,  $(100010)_2 = 42$

<i>Octal Digital Value</i>	<i>Binary Equivalent</i>
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

# Octal to Hexadecimal Number

- Conversion of the octal number to hexadecimal requires two steps.
- First, convert octal numbers to decimal numbers.
- Then, convert decimal numbers to hexadecimal numbers.

## Example

Convert the octal numbers to decimal

$$(55)_8 = (45)_{10}$$

Convert  $(45)_{10}$  into a hexadecimal number by dividing 45 by 16 until you get a remainder less than 16.

$$(45)_{10} = (2D)_{16}$$

$$\text{Or } (55)_8 = (2D)_{16}$$

$$\begin{array}{r|l} 16 & 45 \\ \hline 16 & 2 \text{ --- } 13 \\ \hline & 0 \text{ --- } 2 \end{array}$$

# Hexadecimal Number System

- Has a base value equal to 16. It is also pronounced sometimes as '**hex**'. Hexadecimal numbers are represented by only 16 symbols. These symbols or values are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F.
- Each digit represents a decimal value. For example, D is equal to base-10 13.

Decimal Numbers	4-bit Binary Number	Hexadecimal Number
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

# Decimal to Hexadecimal Conversion

- Divide the number by 16
- Take the quotient and divide again by 16
- The remainder left will produce the hex value
- Repeats the steps until the quotient has become 0

**Example:** Convert  $(242)_{10}$  into hexadecimal.

**Solution:** Divide 242 by 16 and repeat the steps, till the quotient is left as 0.

$$(242)_{10} = (F2)_{16}$$

$$\begin{array}{r|l} 16 & 242 \\ \hline 16 & 15 \quad 2 \rightarrow 2 \\ \hline & 0 \quad 15 \rightarrow F \end{array}$$

# Octal to Hexadecimal Conversion

To convert octal to hex, we have to first convert octal number to decimal and then decimal to hexadecimal.

- **Example:** Convert  $(121)_8$  into hexadecimal.
- **Solution:** First convert 121 into decimal number.
  - $\Rightarrow 1 \times 8^2 + 2 \times 8^1 + 1 \times 8^0$
  - $\Rightarrow 1 \times 64 + 2 \times 8 + 1 \times 1$
  - $\Rightarrow 64 + 16 + 1$
  - $\Rightarrow 81$

$$(121)_8 = 81_{10}$$

Now converting  $81_{10}$  into a hexadecimal number.

$$\text{Therefore, } 81_{10} = 51_{16}$$

16	81	
16	5	1 $\rightarrow$ 1
	0	5 $\rightarrow$ 5

# Hexadecimal to Binary Conversion

Use only 4 digits to represent each hexadecimal number, where each group has a distinct value from 0000 (for 0) and 1111 (for F= 15 =8 + 4 + 2 + 1)

Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Binary	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111



# Binary to Hexadecimal Conversion

- Binary to hexadecimal conversion is a simple method to do. You just have to put the values of the binary number to the relevant hexadecimal number.
- **Example:** Convert  $(11100011)_2$  to hexadecimal.  
**Solution:** From the table, we can write, 11100011 as E3.

Therefore,  $(11100011)_2 = (E3)_{16}$

# Logic Gates

- A logic gate is a basic building block of a digital circuit that has two inputs and one output.
- The relationship between the i/p and the o/p is based on a certain logic.
- These gates are implemented using electronic switches like transistors, diodes.
- But, in practice basic logic gates are built using CMOS technology, FETs and MOSFET(Metal Oxide Semiconductor FET)s.
- Logic gates are used in microprocessors, microcontrollers, embedded system applications and in electronic and electrical project circuits.

# *Behavior of gates and circuits*

## Boolean expressions

Uses Boolean algebra, a mathematical notation for expressing two-valued logic

## Logic diagrams

A graphical representation of a circuit; each gate has its own symbol

## Truth tables

A table showing all possible input value and the associated output values

# Types of Gates

## Basic Gates

1. NOT
2. AND
3. OR

## Universal Gates:

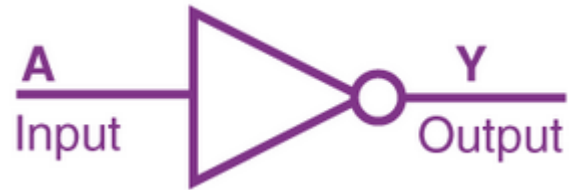
1. NAND
2. NOR

## Special Purpose gates

1. XOR
2. XNOR

# Not Gate

- Single i/p Single o/p



- The Boolean expression is  $Y = \bar{A}$

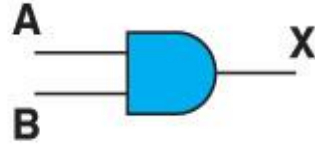
**Truth Table**

A	X
0	1
1	0

# AND Gate

An AND gate accepts two input signals

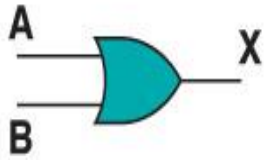
If both are 1, the output is 1; otherwise, the output is 0

Boolean Expression	Logic Diagram Symbol	Truth Table															
$X = A \cdot B$		<table><tr><th>A</th><th>B</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	X	0	0	0	0	1	0	1	0	0	1	1	1
A	B	X															
0	0	0															
0	1	0															
1	0	0															
1	1	1															

# OR Gate

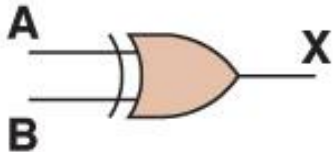
An OR gate accepts two input signals

If both are 0, the output is 0; otherwise, the output is 1

Boolean Expression	Logic Diagram Symbol	Truth Table															
$X = A + B$		<table><tr><th>A</th><th>B</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	X	0	0	0	0	1	1	1	0	1	1	1	1
A	B	X															
0	0	0															
0	1	1															
1	0	1															
1	1	1															

# XOR Gate

- An XOR gate accepts two input signals
- If both are the same, the output is 0; otherwise, the output is 1

Boolean Expression	Logic Diagram Symbol	Truth Table															
$X = A \oplus B$		<table><tr><th>A</th><th>B</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	X	0	0	0	0	1	1	1	0	1	1	1	0
A	B	X															
0	0	0															
0	1	1															
1	0	1															
1	1	0															

Note: The difference between the XOR gate and the OR gate; they differ only in one input situation

When both input signals are 1, the OR gate produces a 1 and the XOR produces a 0 XOR is called the *exclusive OR*



# NAND Gate

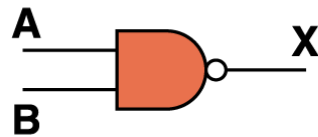
The NAND gate accepts two input signals

If both are 1, the output is 0; otherwise, the output is 1

**Boolean Expression**

$$X = (A \cdot B)'$$

**Logic Diagram Symbol**

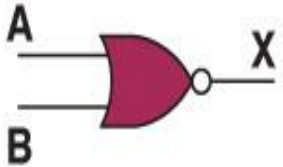


**Truth Table**

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

# NOR Gate

- The NOR gate accepts two input signals
- If both are 0, the output is 1; otherwise, the output is 0

Boolean Expression	Logic Diagram Symbol	Truth Table															
$X = (A + B)'$		<table><tr><th>A</th><th>B</th><th>X</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	X	0	0	1	0	1	0	1	0	0	1	1	0
A	B	X															
0	0	1															
0	1	0															
1	0	0															
1	1	0															

# Review of Gate Processing

A **NOT** gate **inverts** its single input

An **AND** gate produces **1** if **both** input values are **1**

An **OR** gate produces **0** if **both** input values are **0**

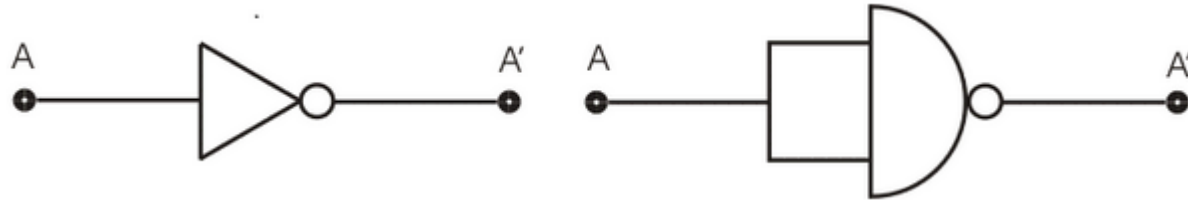
An **XOR** gate produces **0** if input values are the **same**

A **NAND** gate produces **0** if **both** inputs are **1**

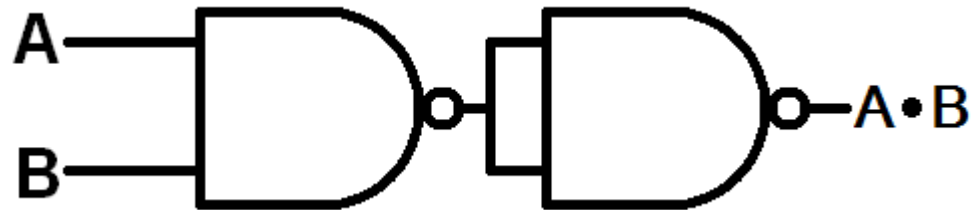
A **NOR** gate produces a **1** if both inputs are **0**

# Realization of Basic Gates using NAND Gate

NOT Gate using NAND Gate

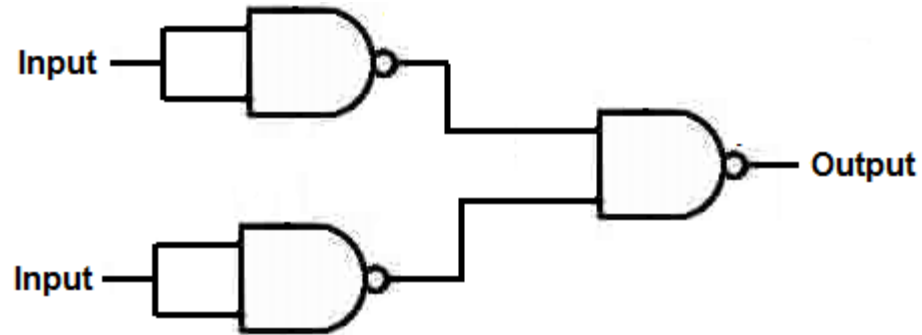


AND Gate using NAND Gate

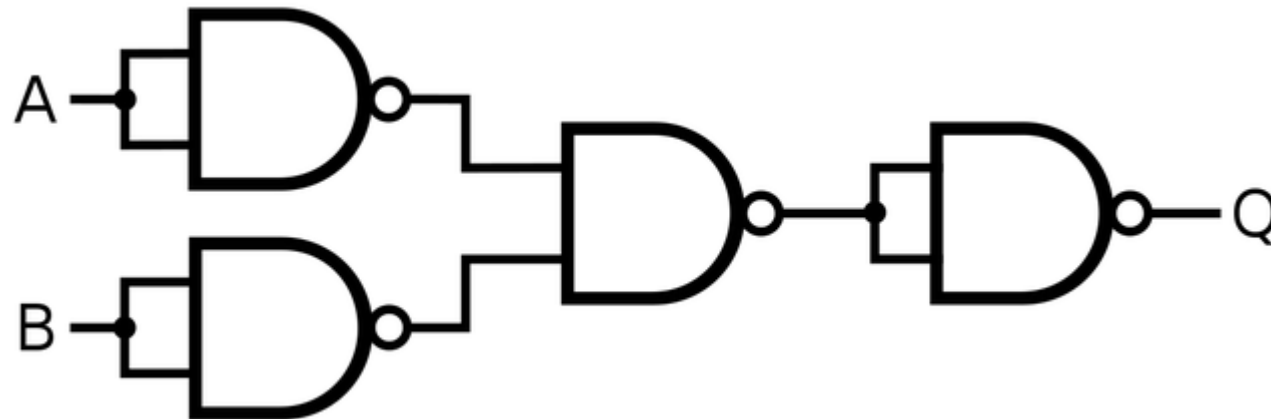


Cont.,

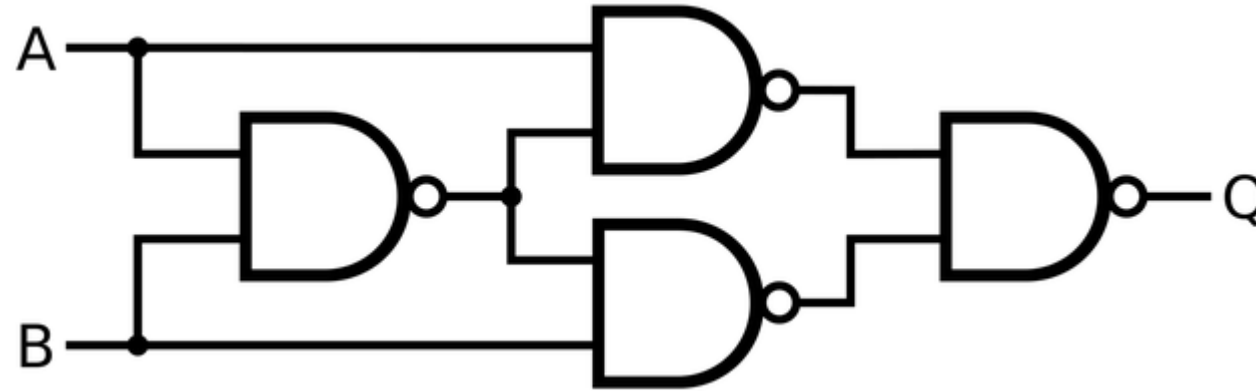
OR Gate using NAND Gate



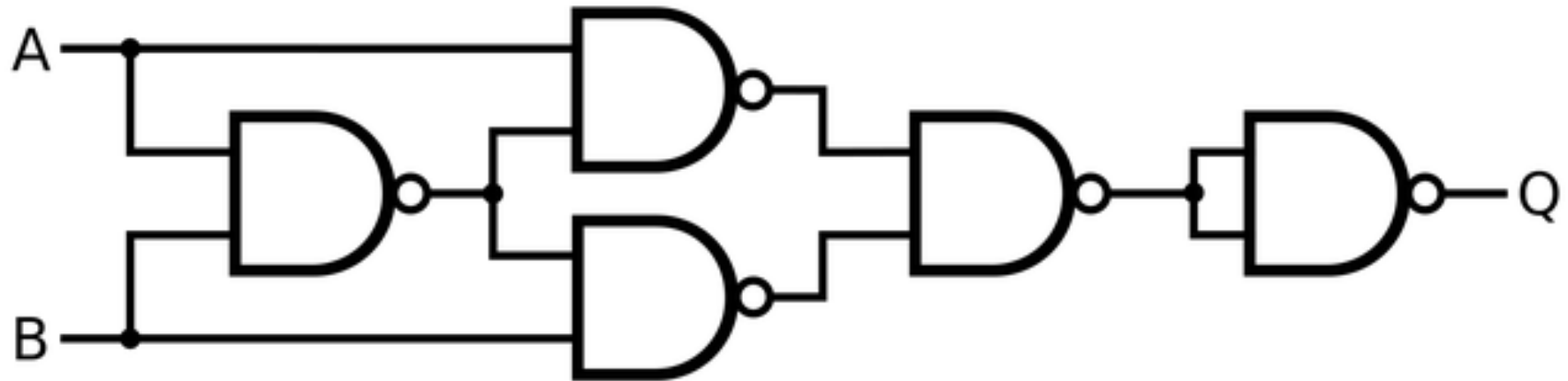
NOR Gate using NAND Gate



# EX-OR Gate using NAND Gate



# EX-NOR Gate using NAND Gate



# Shortcut For Realization of Logic Gates Using NAND Gate (Imp for Quiz)

Number of NAND Gates	Other Gates
1	NOT Gate
2	AND Gate
3	OR Gate
4	EX-OR Gate
5	EX-NOR Gate

# Boolean Algebra

- Category of algebra in which the variable's values are the truth values, **true and false**, ordinarily denoted 1 and 0 respectively.
- It is used to analyze and simplify digital circuits or digital gates.
- It is also called **Binary Algebra** or **logical Algebra**.
- It has been fundamental in the development of digital electronics and is provided for in all modern programming languages.
- It is also used in [set theory](#) and statistics.



# Terminologies

- **Boolean Algebra:** Branch of algebra that deals with logical operations and binary variables.
- **Boolean Variables:** Defined as a variable or a symbol defined as a variable or a symbol, generally an alphabet that represents the logical quantities such as 0 or 1.
- **Boolean Function:** Consists of binary variables, logical operators, constants such as 0 and 1, equal to the operator, and the parenthesis symbols.
- **Literal:** May be a variable or a complement of a variable.
- **Complement:** Defined as the inverse of a variable, which is represented by a bar over the variable.
- **Truth Table:** Table that gives all the possible values of logical variables and the combination of the variables. It is possible to convert the Boolean equation into a truth table. The number of rows in the truth table should be equal to  $2^n$ , where “n” is the number of variables in the equation. For example, if a Boolean equation consists of 3 variables, then the number of rows in the truth table is 8. (i.e.,)  $2^3 = 8$ .

# Terminologies

**Sum term:** A sum term is a literal or logical OR of multiple literals.

Ex:  $X$ ,  $X+Y$ ,  $\bar{A} + B$ ,  $\bar{A} + \bar{B}$ .

**Product term:** A Product term is a literal or logical AND of multiple literals.

Ex:  $X$ ,  $X.Y$ ,  $\bar{A}.B$ ,  $\bar{A}.\bar{B}$

**Sum of Products(SOP):** Logical OR of multiple product terms

Ex:  $X+X.Y$ ,  $\bar{A}.B + \bar{A}.\bar{B}$

**Product of Sum (POS):** Logical AND of multiple Sum terms

Ex:  $(X+\bar{Y})(X+Y)$ ,  $(\bar{A}+B)(\bar{A}+\bar{B})$

**Minterm:** Special case of product term consisting of all the literals, either with complement or without complement that make up a Boolean Expression.

**Maxterm:** Special case of sum term consisting of all the literals, either with complement or without complement that make up a Boolean Expression.

# Terminologies

Canonical SOP: Complete set of minterms that are defined when output is Logic '1' or True.

$$\text{EX: } X\bar{Y} + XY + \bar{X}Y$$

Canonical POS: Complete set of maxterms that are defined when output is Logic '0' or False.

$$\text{Ex: } (X + \bar{Y})(X + Y)$$

# Laws of Boolean Algebra

Name	AND form	OR form
Identity law	$1A = A$	$0 + A = A$
Null law	$0A = 0$	$1 + A = 1$
Idempotent law	$AA = A$	$A + A = A$
Inverse law	$A\bar{A} = 0$	$A + \bar{A} = 1$
Commutative law	$AB = BA$	$A + B = B + A$
Associative law	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Distributive law	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Absorption law	$A(A + B) = A$	$A + AB = A$
De Morgan's law	$\overline{AB} = \bar{A} + \bar{B}$	$\overline{\bar{A} + \bar{B}} = \bar{A}\bar{B}$

# Karnaugh map or a K-map

Pictorial method that is utilised to minimise various Boolean expressions without using the Boolean algebra theorems along with the equation manipulations.

A Karnaugh map can be a special version of the truth table.

Easily minimise various expressions that have 2 to 4 variables using a K-map.

Can easily take two forms, namely, Sum of Product or SOP and Product of Sum or POS

K-map is a table-like representation, but it gives more data than the TRUTH TABLE. Fill a grid of K-map with 1s and 0s, then solve it by creating various groups.

# Solving an Expression Using K-Map

The steps that are used to solve an expression using the K-map method:

1. Select a K-map according to the total number of variables.
2. Identify maxterms or minterms as given in the problem.
3. For SOP, put the 1's in the blocks of the K-map with respect to the minterms (elsewhere 0's).
4. For POS, putting 0's in the blocks of the K-map with respect to the max terms (elsewhere 1's).
5. Make rectangular groups that contain the total terms in the power of two, such as 16,8,4,2,1 ..(highest to lowest group terms based on no of variables) and trying to cover as many numbers of elements in a single group.
6. From the groups that have been created in step 5, find the product terms considering literal wrt row and column wise and if there is any transition of literal value then that literal will not be considered in the product term and adding the other group terms to get in expression in SOP form.

# SOP FORM

## K-map for 3 variables

		BC			
		B'C'	B'C	BC	BC'
A	A'	A'B'C'	A'B'C	A'BC	A'BC'
	A	AB'C'	AB'C	ABC	ABC'

8 Blocks=1

4 Blocks=1 Variable Term

2 Blocks=2 Variable Term

1 Block=3 Variable Term

## K-map for 4 variables

		CD			
		C'D'	C'D	CD	CD'
AB	A'B'	A'B'C'D'	A'B'C'D	A'B'CD	A'B'CD'
	A'B	A'BC'D'	A'BC'D	A'BCD	A'BCD'
	AB	ABC'D'	ABC'D	ABCD	ABCD'
	AB'	AB'C'D'	AB'C'D	AB'CD	AB'CD'

16 Blocks=1

8 Blocks=1 Variable Term

4 Blocks=2 Variable Term

2 Block=3 Variable Term

1 Block=4 Variable Term

## COMBINATIONAL LOGIC:

Introduction, Design procedure, Adders-Half adder, Full adder

- Combinational circuits are used in digital electronics to perform operations on data.
- Adder
- Subtractor
- Multiplier
- Divider

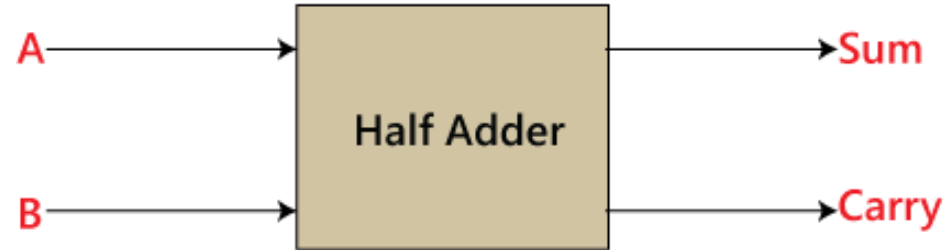


# COMBINATIONAL LOGIC

## Design Procedure

- Determine required number of inputs.
- Outputs from the specifications.
- Derive the truth table for each of the outputs based on their relationships to the input.

# Half Adder



Input(2)- A, B

Output(2)- Sum, Carry

## Truth Table

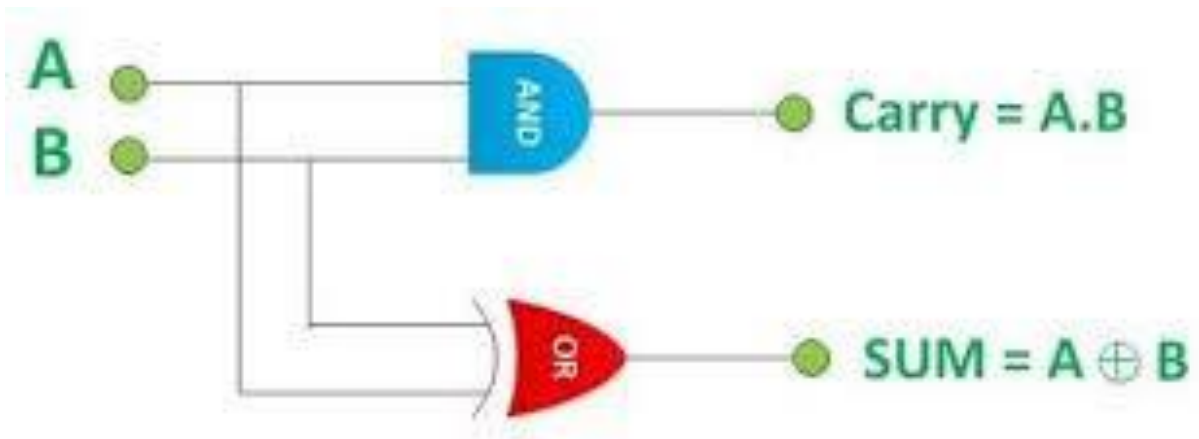
Inputs		Outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- 'A' and 'B' are the input states, and 'sum' and 'carry' are the output states.
- The carry output is 0 in case where both the inputs are not 1.
- The least significant bit of the sum is defined by the 'sum' bit.

# COMBINATIONAL LOGIC

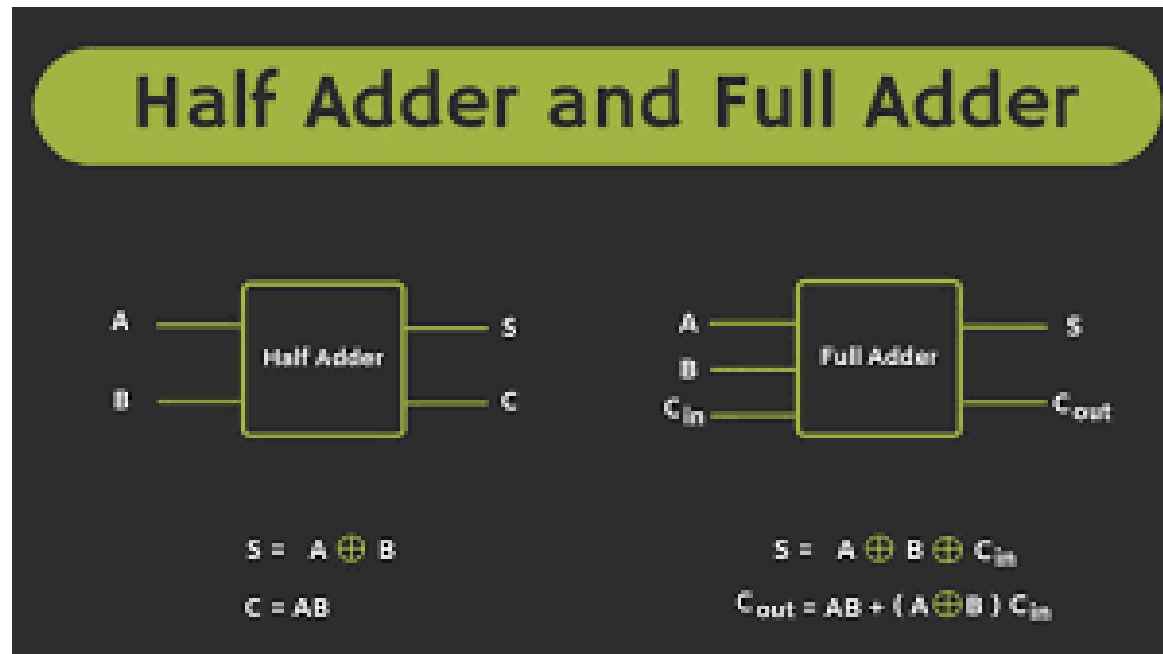
$$\text{Sum} = A'B + AB'$$

$$\text{Carry} = AB$$



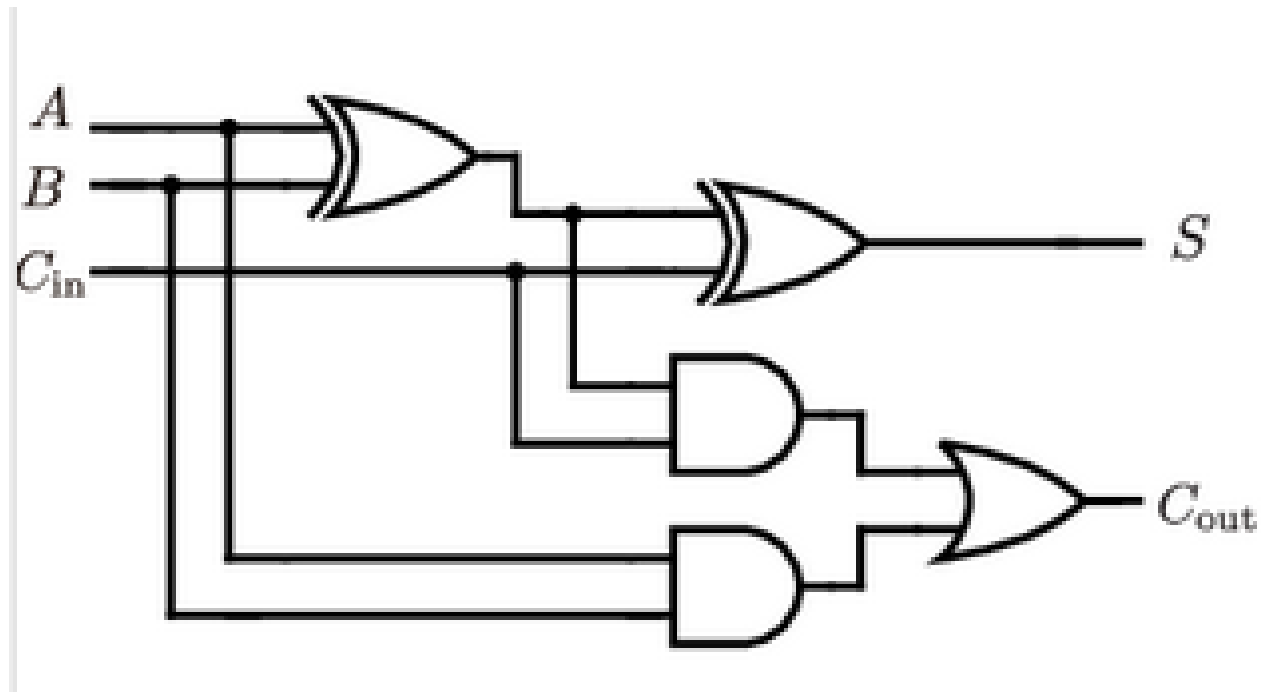
Logic Circuit Of Half Adder

# Full Adder

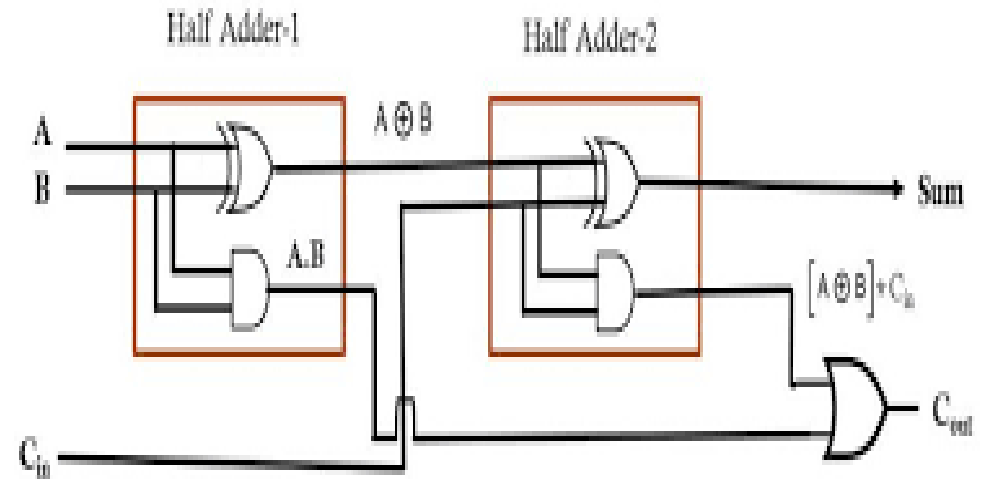
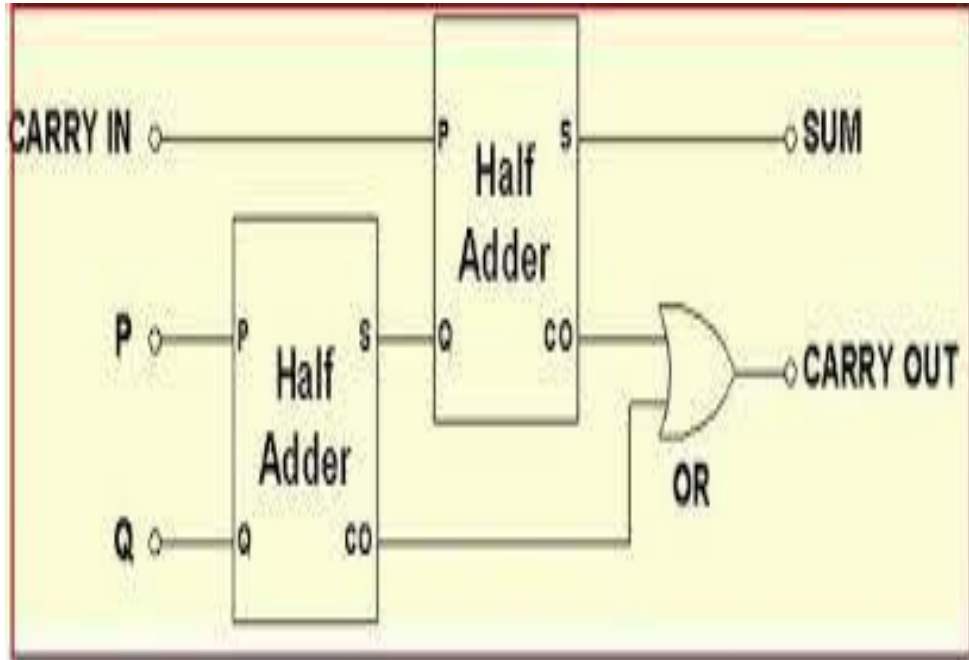


Inputs			Outputs	
$A$	$B$	$C_{in}$	$S$	$C_{out}$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

# Circuit Diagram of Full Adder



## Full Adder using 2 Half Adder



## Full Adder using 2 Half Adder

$$\begin{aligned}\text{SUM} &= \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + AB C_{in} \\ &= C_{in} (\bar{A}\bar{B} + AB) + \bar{C}_{in} (A\bar{B} + \bar{A}B) \\ &= C_{in} [(\bar{A} + B) \cdot (A + \bar{B})] + \bar{C}_{in} (A\bar{B} + \bar{A}B) \\ &= C_{in} (\overline{A\bar{B}} \cdot \overline{\bar{A}B}) + \bar{C}_{in} (A\bar{B} + \bar{A}B) \\ &= C_{in} (\overline{A\bar{B} + \bar{A}B}) + \bar{C}_{in} (A\bar{B} + \bar{A}B) \\ &= C_{in} \oplus (A\bar{B} + \bar{A}B) \\ &= C_{in} \oplus (A \oplus B)\end{aligned}$$

$$\begin{aligned}C_{out} &= AB + C_{in} (A\bar{B} + \bar{A}B) \\ &= AB + A\bar{B}C_{in} + \bar{A}BC_{in} \\ &= AB(C_{in} + 1) + A\bar{B}C_{in} + \bar{A}BC_{in} & \because C_{in} + 1 = 1 \\ &= ABC_{in} + AB + A\bar{B}C_{in} + \bar{A}BC_{in} \\ &= AB + AC_{in}(B + \bar{B}) + \bar{A}BC_{in} \\ &= AB + AC_{in} + \bar{A}BC_{in} \\ &= AB(C_{in} + 1) + AC_{in} + \bar{A}BC_{in} & \because C_{in} + 1 = 1 \\ &= ABC_{in} + AB + AC_{in} + \bar{A}BC_{in} \\ &= AB + AC_{in} + BC_{in}(A + \bar{A}) \\ &= AB + AC_{in} + BC_{in}\end{aligned}$$

## Full Adder using 2 Half Adder

