Logical Organization of Computer
CHAPTER 2

COMBINATIONAL LOGIC CIRCUITS

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

1. BINARY LOGIC AND GATES

- Digital circuits are hardware components that manipulate binary information. The circuits are implemented using transistors and interconnections in complex semiconductor devices called *integrated circuits*. Each basic circuit is referred to as a *logic gate*.
- Each gate performs a specific logical operation. The outputs of gates are applied to the inputs of other gates to form a digital circuit.
- In order to describe the operational properties of digital circuits, we need to introduce a mathematical notation that specifies the operation of each gate and that can be used to analyze and design circuits known as *Boolean algebras*.

❑ Binary Logic

- Binary logic deals with binary variables, which take on two discrete values, and with the operations of mathematical logic applied to these variables.
- Associated with the binary variables are three basic logical operations called AND, OR, and NOT.
- **1. AND.** This operation is represented by a dot or by the absence of an operator. For example, $Z=X \cdot Y$ or $Z=XY$ is read "Z is equal to X AND Y." The logical operation AND is interpreted to mean that $Z=1$ if and only if $X=1$ and $Y=1$; otherwise $Z=0$.

$$0 \cdot 0 = 0$$
$$0 \cdot 1 = 0$$
$$1 \cdot 0 = 0$$
$$1 \cdot 1 = 1$$

❑ Binary Logic

2. OR. This operation is represented by a plus symbol. For example, $Z = X + Y$ is read "Z is equal to X OR Y," meaning that $Z = 1$ if $X = 1$ or if $Y = 1$, or if both $X = 1$ and $Y = 1$. $Z = 0$ if and only if $X = 0$ and $Y = 0$.

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 1$$

3. NOT. This operation is represented by a bar over the variable. For example, $Z = \overline{X}$ is read "Z is equal to NOT X," meaning that Z is what X is not. In other words, if $X = 1$, then $Z = 0$; but if $X = 0$, then $Z = 1$. The NOT operation is also referred to as the *complement* operation, since it changes a 1 to 0 and a 0 to 1.

❑ Binary Logic

- A *truth table* for an operation is a table of combinations of the binary variables showing the relationship between the values that the variables take on and the values of the result of the operation.
- The truth tables for the operations AND, OR, and NOT are shown below.

❑ Binary Logic

❑ **TABLE 1**
Truth Tables for the Three Basic Logical Operations

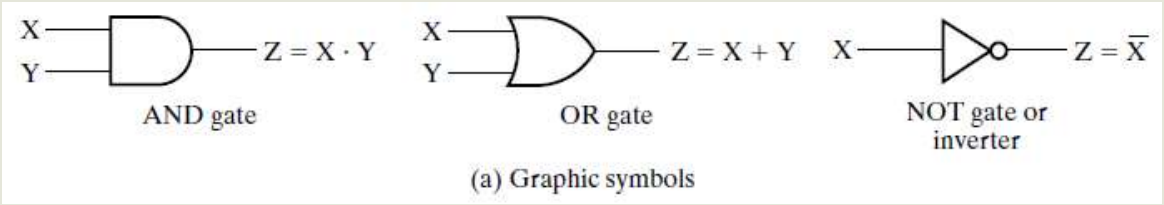
AND			OR			NOT	
X	Y	$Z = X \cdot Y$	X	Y	$Z = X + Y$	X	$Z = \bar{X}$
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

❑ Logic Gates

- Logic gates are electronic circuits that operate on one or more input signals to produce an output signal.
- The input terminals of logic gates accept binary signals within the allowable range and respond at the output terminals with binary signals that fall within a specified range.
- The intermediate regions between the allowed ranges in the figure are crossed only during changes from 1 to 0 or from 0 to 1.
- These changes are called *transitions*, and the intermediate regions are called the *transition regions*.

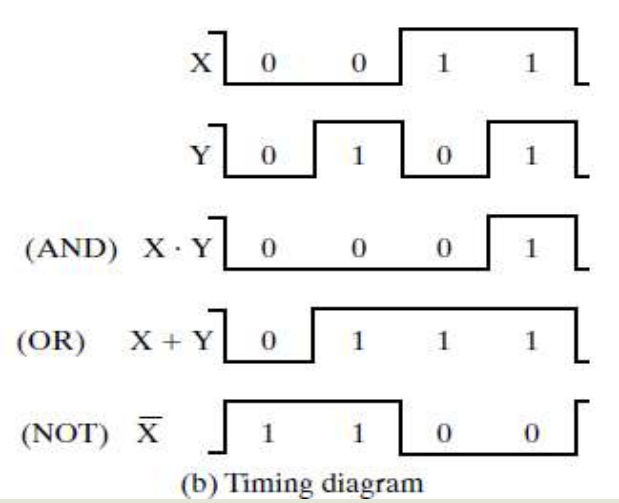
❑ Logic Gates

- The graphics symbols used to designate the three types of gates—AND, OR, and NOT—are shown below.



(a) Graphic symbols

❑ Logic Gates



(b) Timing diagram

2. BOOLEAN ALGEBRA

- A *Boolean expression* is an algebraic expression formed by using binary variables, the constants 0 and 1, the logic operation symbols, and parentheses.
- *Boolean function* can be described by a Boolean equation consisting of a binary variable identifying the function followed by an equals sign and a Boolean expression.

❑ Basic Identities of Boolean Algebra

Annulment Law – A term AND´ed with a "0" equals 0 or OR´ed with a "1" will equal 1

- $A \cdot 0 = 0$ A variable AND'ed with 0 is always equal to 0
- $A + 1 = 1$ A variable OR'ed with 1 is always equal to 1

Identity Law – A term OR´ed with a "0" or AND´ed with a "1" will always equal that term

- $A + 0 = A$ A variable OR'ed with 0 is always equal to the variable
- $A \cdot 1 = A$ A variable AND'ed with 1 is always equal to the variable

❑ Basic Identities of Boolean Algebra

Idempotent Law – An input that is AND´ed or OR´ed with itself is equal to that input

- $A + A = A$ A variable OR'ed with itself is always equal to the variable
- $A \cdot A = A$ A variable AND'ed with itself is always equal to the variable

Complement Law – A term AND´ed with its complement equals "0" and a term OR´ed with its complement equals "1"

- $A \cdot \overline{A} = 0$ A variable AND'ed with its complement is always equal to 0
- $A + \overline{A} = 1$ A variable OR'ed with its complement is always equal to 1

❑ Basic Identities of Boolean Algebra

Commutative Law – The order of application of two separate terms is not important

$A . B = B . A$ The order in which two variables are AND'ed makes no difference

$A + B = B + A$ The order in which two variables are OR'ed makes no difference

Double Negation Law – A term that is inverted twice is equal to the original term

$\overline{\overline{A}} = A$ A double complement of a variable is always equal to the variable

❑ Basic Identities of Boolean Algebra

Distributive Law – This law permits the multiplying or factoring out of an expression.

$A(B + C) = A.B + A.C$ (OR Distributive Law)

$A + (B.C) = (A + B).(A + C)$ (AND Distributive Law)

Absorptive Law – This law enables a reduction in a complicated expression to a simpler one by absorbing like terms.

$A + (A.B) = A$ (OR Absorption Law)

$A(A + B) = A$ (AND Absorption Law)

❑ Basic Identities of Boolean Algebra

Associative Law – This law allows the removal of brackets from an expression and regrouping of the variables.

$A + (B + C) = (A + B) + C = A + B + C$ (OR Associate Law)

$A(B.C) = (A.B)C = A \,.\, B \,.\, C$ (AND Associate Law)

de Morgan´s Theorem – There are two "de Morgan´s" rules or theorems,

(1) Two separate terms NOR´ed together is the same as the two terms inverted (Complement) and AND´ed for example: $\overline{A+B} = \overline{A} \,.\, \overline{B}$

(2) Two separate terms NAND´ed together is the same as the two terms inverted (Complement) and OR´ed for example: $\overline{A.B} = \overline{A} + \overline{B}$

❑ Basic Identities of Boolean Algebra

Truth Tables to Verify DeMorgan's Theorem

(a) X	Y	X + Y	$\overline{X+Y}$	(b) X	Y	$\overline{X}$	$\overline{Y}$	$\overline{X} \cdot \overline{Y}$
0	0	0	1	0	0	1	1	1
0	1	1	0	0	1	1	0	0
1	0	1	0	1	0	0	1	0
1	1	1	0	1	1	0	0	0

❑ Algebraic Manipulation

- Boolean algebra is a useful tool for simplifying digital circuits. Consider, for example, the Boolean function represented by

$$F = \overline{X}YZ + \overline{X}Y\overline{Z} + XZ$$

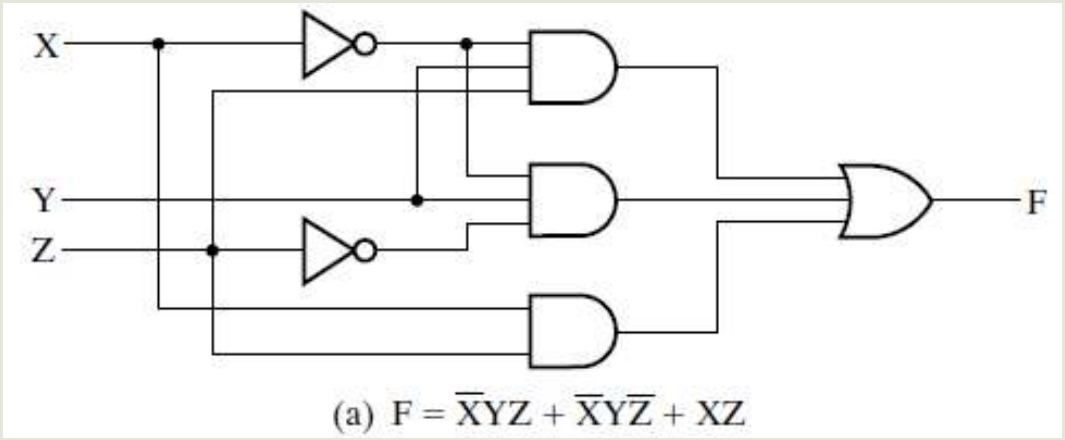
$$\begin{aligned} F &= \overline{X}YZ + \overline{X}Y\overline{Z} + XZ \\ &= \overline{X}Y(Z + \overline{Z}) + XZ \\ &= \overline{X}Y \cdot 1 + XZ \\ &= \overline{X}Y + XZ \end{aligned}$$

Distributive Law

Complement Law

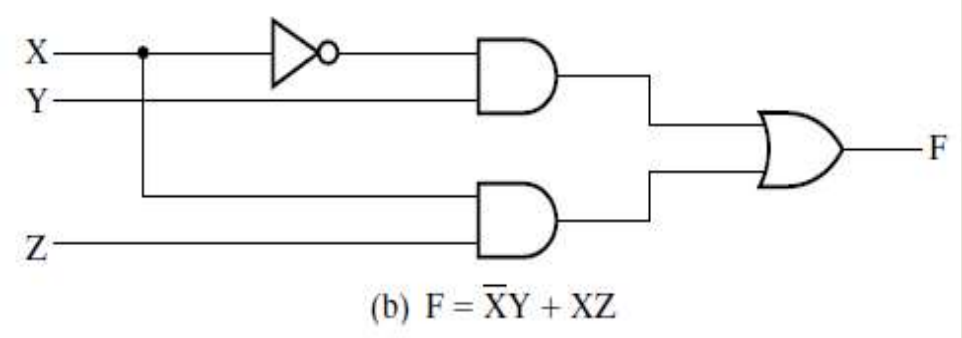
Identity Law

❑ Algebraic Manipulation



(a) $F = \overline{X}YZ + \overline{X}Y\overline{Z} + XZ$

❑ Algebraic Manipulation



(b) $F = \overline{X}Y + XZ$

❑ Algebraic Manipulation

Truth Table for Boolean Function

X	Y	Z	(a) F	(b) F
0	0	0	0	0
0	0	1	0	0
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1

3. STANDARD FORMS

- A Boolean function expressed algebraically can be written in a variety of ways.
- There are, however, specific ways of writing algebraic equations that are considered to be standard forms.
- The standard forms facilitate the simplification procedures for Boolean expressions and, in some cases, may result in more desirable expressions for implementing logic circuits.

3. STANDARD FORMS

The standard forms contain *product terms* and *sum terms*. An example of a product term is $X\overline{Y}Z$. This is a logical product consisting of an AND operation among three literals. An example of a sum term is $X + Y + \overline{Z}$. This is a logical sum consisting of an OR operation among the literals. In Boolean algebra, the words "product" and "sum" do not imply arithmetic operations; instead, they specify the logical operations AND and OR, respectively.

❑ Minterms and Maxterms

- A product term in which all the variables appear exactly once, either complemented or uncomplemented, is called a *minterm*.
- Its characteristic property is that it represents exactly one combination of binary variable values in the truth table. It has the value 1 for that combination and 0 for all others.

❑ Minterms and Maxterms

Minterms for Three Variables

X	Y	Z	Product Term	Symbol	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7
0	0	0	$\overline{X}\overline{Y}\overline{Z}$	m_0	1	0	0	0	0	0	0	0
0	0	1	$\overline{X}\overline{Y}Z$	m_1	0	1	0	0	0	0	0	0
0	1	0	$\overline{X}Y\overline{Z}$	m_2	0	0	1	0	0	0	0	0
0	1	1	$\overline{X}YZ$	m_3	0	0	0	1	0	0	0	0
1	0	0	$X\overline{Y}\overline{Z}$	m_4	0	0	0	0	1	0	0	0
1	0	1	$X\overline{Y}Z$	m_5	0	0	0	0	0	1	0	0
1	1	0	$XY\overline{Z}$	m_6	0	0	0	0	0	0	1	0
1	1	1	XYZ	m_7	0	0	0	0	0	0	0	1

❑ Minterms and Maxterms

- A sum term that contains all the variables in complemented or uncomplemented form is called a *maxterm*.
- Again, it is possible to formulate 2^n maxterms with n variables.
- Each maxterm is a logical sum of the three variables, with each variable being complemented if the corresponding bit of the binary number is 1 and uncomplemented if it is 0.

❑ Minterms and Maxterms

Maxterms for Three Variables

X	Y	Z	Sum Term	Symbol	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7
0	0	0	$X+Y+Z$	M_0	0	1	1	1	1	1	1	1
0	0	1	$X+Y+\overline{Z}$	M_1	1	0	1	1	1	1	1	1
0	1	0	$X+\overline{Y}+Z$	M_2	1	1	0	1	1	1	1	1
0	1	1	$X+\overline{Y}+\overline{Z}$	M_3	1	1	1	0	1	1	1	1
1	0	0	$\overline{X}+Y+Z$	M_4	1	1	1	1	0	1	1	1
1	0	1	$\overline{X}+Y+\overline{Z}$	M_5	1	1	1	1	1	0	1	1
1	1	0	$\overline{X}+\overline{Y}+Z$	M_6	1	1	1	1	1	1	0	1
1	1	1	$\overline{X}+\overline{Y}+\overline{Z}$	M_7	1	1	1	1	1	1	1	0

❑ Minterms and Maxterms

- Consider the Boolean function F in Table

Boolean Functions of Three Variables

(a) X	Y	Z	F	$\overline{F}$
0	0	0	1	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	0	1
1	0	1	1	0
1	1	0	0	1
1	1	1	1	0

$$F = \overline{X}\,\overline{Y}\,\overline{Z} + \overline{X}Y\overline{Z} + X\overline{Y}Z + XYZ = m_0 + m_2 + m_5 + m_7$$

$$\overline{F}(X,Y,Z) = \overline{X}\,\overline{Y}Z + \overline{X}YZ + X\overline{Y}\,\overline{Z} + XY\overline{Z} = m_1 + m_3 + m_4 + m_6$$

❑ Minterms and Maxterms

The following is a summary of the most important properties of minterms:

1. There are 2^n minterms for n Boolean variables. These minterms can be generated from the binary numbers from 0 to $2^n - 1$.
2. Any Boolean function can be expressed as a logical sum of minterms.
3. The complement of a function contains those minterms not included in the original function.
4. A function that includes all the 2^n minterms is equal to logic 1.

❑ Sum of Products (SOP)

- A boolean expression consisting purely of Minterms (product terms) is said to be in canonical sum of products form.
- Example: lets say, we have a boolean function F defined on two variables A and B. So, A and B are the inputs for F and lets say, output of F is true i.e., F = 1 when any one of the input is true or 1. Now we draw the truth table for F.

❑ Sum of Products (SOP)

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

Now we will create a column for the minterm using the variables A and B. If input is 0 we take the complement of the variable and if input is 1 we take the variable as is.

A	B	F	Minterm
0	0	0	A'B'
0	1	1	A'B
1	0	1	AB'
1	1	1	AB

❑ Sum of Products (SOP)

To get the desired canonical SOP expression we will add the **minterms** (product terms) for which the **output is 1**.

F = A'B + AB' + AB

SOP expression implements 2 level AND-OR design in which the 1st level gate is AND gate following the 2nd level gate which is OR gate.

❑ Product of Sums (POS)

- A boolean expression consisting purely of Maxterms (sum terms) is said to be in canonical product of sums form.
- Example: Lets say, we have a boolean function F defined on two variables A and B. So, A and B are the inputs for F and lets say, output of F is true i.e., F = 1 when only one of the input is true or 1. now we draw the truth table for F .

❑ Product of Sums (POS)

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

Now we will create a column for the maxterm using the variables A and B. If input is 1 we take the complement of the variable and if input is 0 we take the variable as is.

A	B	F	Maxterm
0	0	0	A+B
0	1	1	A+B'
1	0	1	A'+B
1	1	0	A'+B'

❑ Product of Sums (POS)

To get the desired canonical POS expression we will multiply the maxterms (sum terms) for which the output is 0.

F = (A+B) . (A'+B')

POS expression implements 2 level OR-AND design in which the 1st level gate is OR gate following the 2nd level gate which is AND gate.

❑ Karnaugh Map Boolean Algebraic Simplification Technique

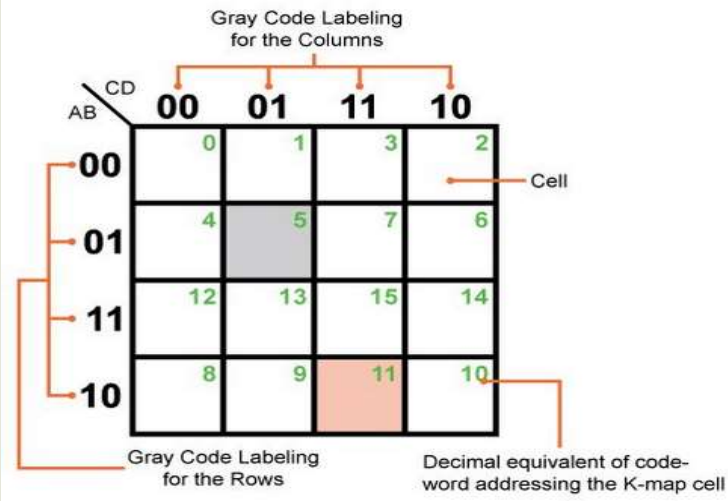
- Karnaugh map (K-map), introduced by Maurice Karnaugh in 1953.
- The K-map method of solving the logical expressions is referred to as the graphical technique of simplifying Boolean expressions. K-maps are also referred to as 2D truth tables .
- K-maps basically deal with the technique of inserting the values of the output variable in cells within a rectangle or square grid according to a definite pattern.
- The number of cells in the K-map is determined by the number of input variables and is mathematically expressed as two raised to the power of the number of input variables, i.e., 2^n, where the number of input variables is n.

❑ Karnaugh Map Boolean Algebraic Simplification Technique

- **Gray Coding**
- Each cell within a K-map has a definite place-value which is obtained by using an encoding technique known as Gray code.
- The specialty of this code is the fact that the adjacent code values differ only by a single bit. That is, if the given code-word is 01, then the previous and the next code-words can be 11 or 00, in any order, but cannot be 10 in any case.
- In K-maps, the rows and the columns of the table use Gray code-labeling which in turn represent the values of the corresponding input variables. This means that each K-map cell can be addressed using a unique Gray Code-Word.

❑ Karnaugh Map Boolean Algebraic Simplification Technique

- These concepts are further emphasized by a typical 16-celled K-map shown in below diagram, which can be used to simplify a logical expression comprising of 4-variables (A, B, C and D mentioned at its top-left corner).



❑ Karnaugh Map Boolean Algebraic Simplification Technique

For example, the grey colored cell of the K-map shown can be addressed using the code-word "0101" which is equivalent to 5 in decimal (shown as the green number in the figure) and corresponds to the input variable combination $\bar{A}B\bar{C}D$ or $A+\bar{B}+C+\bar{D}$, depending on whether the input–output relationship is expressed in SOP (sum of products) form or POS (product of sums) form, respectively.

Similarly, $A\bar{B}CD$ or $\bar{A}+B+\bar{C}+\bar{D}$ refers to the Gray code-word of "1011", equivalent to 11 in decimal (again, shown in green in the figure), which in turn means that we are addressing the pink-colored K-map cell in the figure.

❑ Karnaugh Map Boolean Algebraic Simplification Technique

- With this general idea of K-maps, let us now move on to the procedure employed in designing an optimal (in terms of the number of gates used to realize the logic) digital system. We'll start with a given problem statement.
- **Example 1:**
- **Design a digital system whose output is defined as logically low if the 4-bit input binary number is a multiple of 3; otherwise, the output will be logically high. The output is defined if and only if the input binary number is greater than 2.**

❑ Karnaugh Map Boolean Algebraic Simplification Technique

- **Step 1: Truth Table**
- In the given example:
- Number of input variables = 4, which we will call A, B, C and D.
- Number of output variables = 1, which we will call Y
- where
- Y = Don't Care, if the input number is less than 3 (orange entries in the truth table)
- Y = 0, if the input number is an integral multiple of 3 (green entries in the truth table)
- Y = 1, if the input number is not an integral multiple of 3 (blue entries in the truth table)

❑ Karnaugh Map Boolean Algebraic Simplification Technique

Truth Table

Inputs				Decimal Equivalent	Output
A	B	C	D		Y
0	0	0	0	0	X
0	0	0	1	1	X
0	0	1	0	2	X
0	0	1	1	3	0
0	1	0	0	4	1
0	1	0	1	5	1
0	1	1	0	6	0
0	1	1	1	7	1
1	0	0	0	8	1
1	0	0	1	9	0
1	0	1	0	10	1
1	0	1	1	11	1
1	1	0	0	12	0
1	1	0	1	13	1
1	1	1	0	14	1
1	1	1	1	15	0

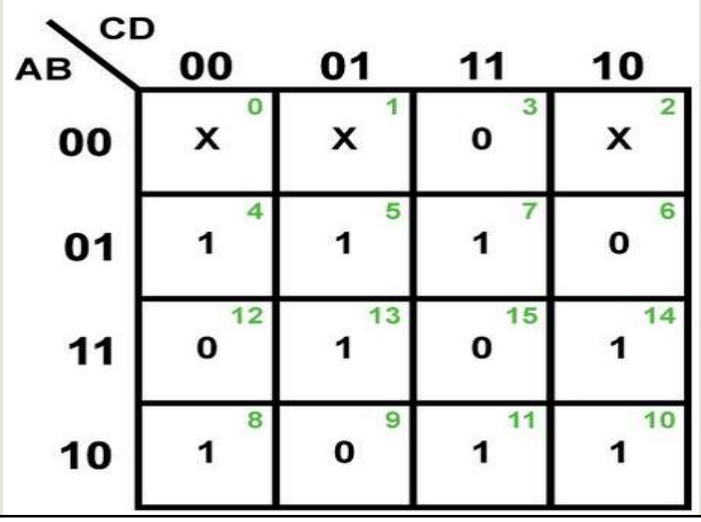
where X indicates Don't Care Condition

❑ Karnaugh Map Boolean Algebraic Simplification Technique

- **Step 2: Select and Populate K-Map**
- From Step 1, we know the number of input variables involved in the logical expression from which size of the K-map required will be decided.
- Further, we also know the number of such K-maps required to design the desired system as the number of output variables would also be known definitely.
- This means that, for the example considered, we require a single (due to one output variable) K-map with 16 cells (as there are four input variables).

❑ Karnaugh Map Boolean Algebraic Simplification Technique

- Next, we have to fill the K-map cells with one for each minterm, zero for each maxterm, and X for Don't Care terms. The procedure is to be repeated for every single output variable.
- Hence for this example, we get the K-map as shown in diagram.



AB \ CD	00	01	11	10
00	X (0)	X (1)	0 (3)	X (2)
01	1 (4)	1 (5)	1 (7)	0 (6)
11	0 (12)	1 (13)	0 (15)	1 (14)
10	1 (8)	0 (9)	1 (11)	1 (10)

❑ Karnaugh Map Boolean Algebraic Simplification Technique

- **Step 3: Form the Groups**
- K-map simplification can also be referred to as the "simplification by grouping" technique as it solely relies on the formation of clusters. That is, the main aim of the entire process is together as many ones (for SOP solution) or zeros (for POS solution) under one roof for each of the output variables in the problem stated.
- However, while doing so we have to strictly abide by certain rules and regulations:

❑ Karnaugh Map Boolean Algebraic Simplification Technique

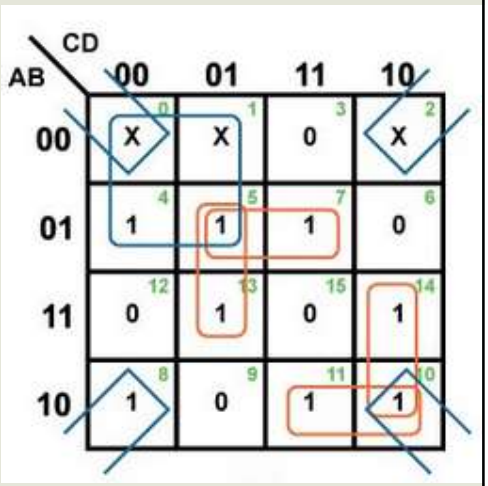
- 1. The process has to be initiated by grouping the bits which lie in adjacent cells such that the group formed contains the maximum number of selected bits. This means that for an *n*-variable K-map with 2^n cells, try to group for 2^n cells first, then for 2^{n-1} cells, next for 2^{n-2} cells, and so on until the "group" contains only 2^0 cells, i.e., isolated bits (if any). Note that the number of cells in the group must be equal to an integer power to 2, i.e., 1, 2, 4, 8. . . .
- 2. The procedure must be applied for all adjacent cells of the K-map, even when they appear to be not adjacent—the top row is considered to be adjacent to the bottom row and the rightmost column is considered to be adjacent to the leftmost column, as if the K-map wraps around from top to bottom and right to left.

❑ Karnaugh Map Boolean Algebraic Simplification Technique

- 3. A bit appearing in one group can be repeated in another group provided that this leads to the increase in the resulting group-size.
- 4. Don't Care conditions are to be considered for the grouping activity if and only if they help in obtaining a larger group. Otherwise, they are to be neglected.

❑ Karnaugh Map Boolean Algebraic Simplification Technique

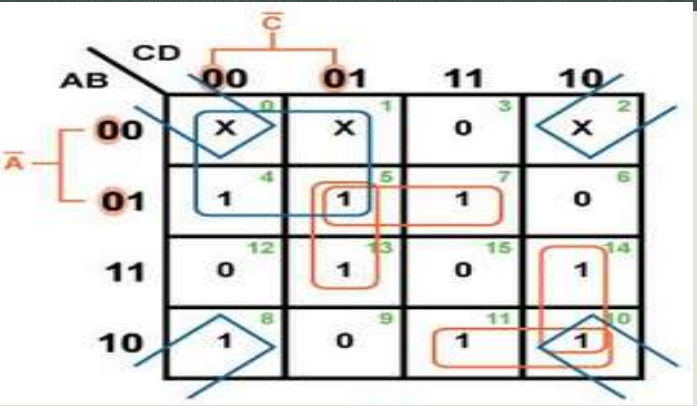
	SOP Form Solution		
Number of groups having 16 cells	0		
Number of groups having 8 cells	0		
Number of groups having 4 cells (Blue Enclosures in Figure 3)	2	Group 1 (Cells 0,2,8,10)	
		Group 2 (Cells 0,1,4,5)	
Number of groups having 2 cells (Orange Enclosures in Figure 3)	4	Group 3 (Cells 5,7)	Group 4 (Cells 5,13)
		Group 5 (Cells 10,11)	Group 6 (Cells 10,14)



❑ Karnaugh Map Boolean Algebraic Simplification Technique

- **Step 4: Simplified Logical Expression**
- For each of the resulting groups, we have to obtain the corresponding logical expression in terms of the input-variables. This can be done by expressing the bits which are common amongst the Gray code-words which represent the cells contained within the considered group.
- Finally, all these group-wise logical expressions need to be combined appropriately to form the simplified Boolean equation for the output variable. The same procedure must be repeated for every output variable of the given problem.

❑ Karnaugh Map Boolean Algebraic Simplification Technique

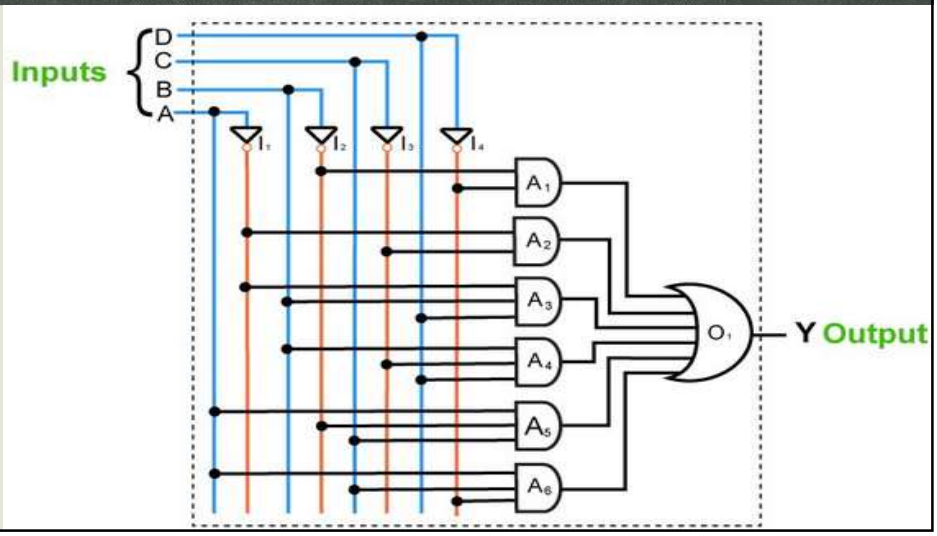


SOP Form Solution	
Groups	**Logical Expression**
Group 1	$\bar{B}\bar{D}$
Group 2	$\bar{A}\bar{C}$
Group 3	$\bar{A}BD$
Group 4	$B\bar{C}D$
Group 5	$A\bar{B}C$
Group 6	$AC\bar{D}$

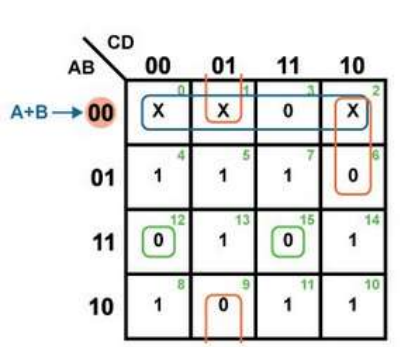
Thus, $Y = \bar{B}\bar{D} + \bar{A}\bar{C} + \bar{A}BD + B\bar{C}D + A\bar{B}C + AC\bar{D}$

❑ Karnaugh Map Boolean Algebraic Simplification Technique

- **Step 5: System Design**



❑ Karnaugh Map Boolean Algebraic Simplification Technique

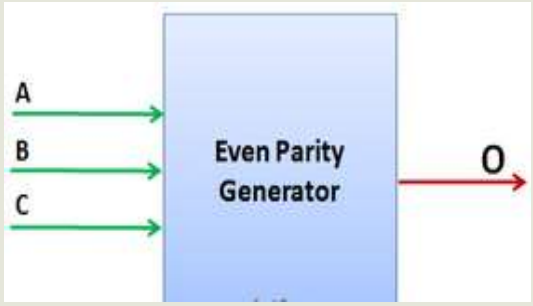


POS Form Solution	
Groups	**Logical Expression**
Group 1	A+B
Group 2	B+C+$\overline{D}$
Group 3	A+$\overline{C}$+D
Group 4	$\overline{A}$+$\overline{B}$+C+D
Group 5	$\overline{A}$+$\overline{B}$+$\overline{C}$+$\overline{D}$

Thus, Y = (A+B) (B+C+$\overline{D}$) (A+$\overline{C}$+D) ($\overline{A}$+$\overline{B}$+C+D) ($\overline{A}$+$\overline{B}$+$\overline{C}$+$\overline{D}$)

❑ Exercise

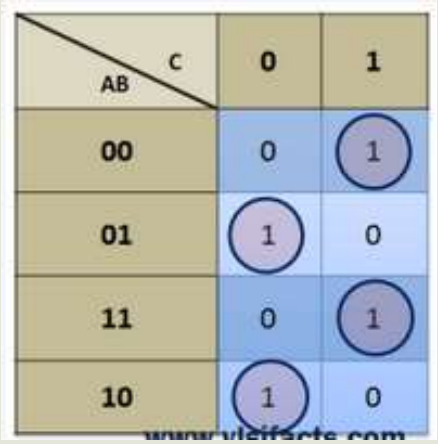
- **Design Problem:** Design a 3 input, 1 output digital logic circuit which will take all the octal digits (0, 1, … 7) as its input and produce the even parity bit for the corresponding octal digit.



❑ Exercise

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

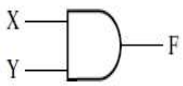
❑ Exercise



$$O = A'B'C + A'BC' + AB'C' + ABC$$

❑ OTHER GATE TYPES

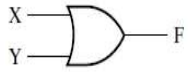
AND



$F = XY$

X	Y	F
0	0	0
0	1	0
1	0	0
1	1	1

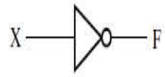
OR



$F = X + Y$

X	Y	F
0	0	0
0	1	1
1	0	1
1	1	1

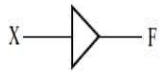
NOT (inverter)



$F = \overline{X}$

X	F
0	1
1	0

Buffer



$F = X$

X	F
0	0
1	1

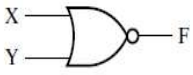
❑ OTHER GATE TYPES

NAND

$F = \overline{X \cdot Y}$

X	Y	F
0	0	1
0	1	1
1	0	1
1	1	0

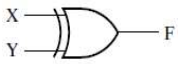
NOR



$F = \overline{X + Y}$

X	Y	F
0	0	1
0	1	0
1	0	0
1	1	0

Exclusive-OR (XOR)

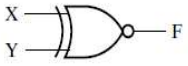


$F = X\overline{Y} + \overline{X}Y$
$= X \oplus Y$

X	Y	F
0	0	0
0	1	1
1	0	1
1	1	0

❑ OTHER GATE TYPES

Exclusive-NOR (XNOR)

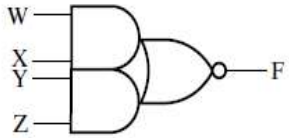


$F = XY + \overline{X}\,\overline{Y}$

$= X \oplus Y$

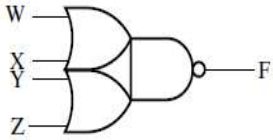
X	Y	F
0	0	1
0	1	0
1	0	0
1	1	1

AND-OR-INVERT (AOI)



$F = \overline{WX + YZ}$

OR-AND -INVERT (OAI)



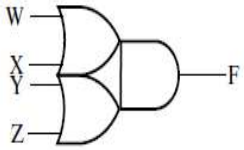
$F = \overline{(W + X)(Y + Z)}$

❑ OTHER GATE TYPES

AND-OR (AO)

$F = WX + YZ$

OR-AND (OA)



$F = (W + X)(Y + Z)$

❑ Exercise

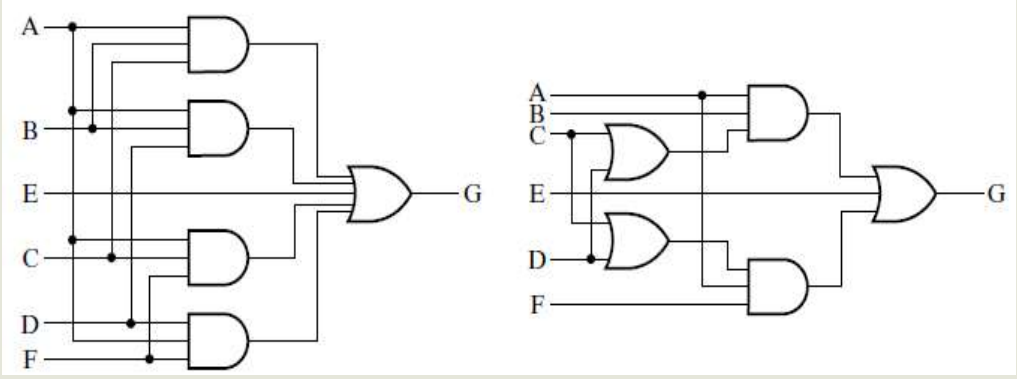
$$G = ABC + ABD + E + ACF + ADF$$

$$G = (AB + AF)(C + D) + E$$

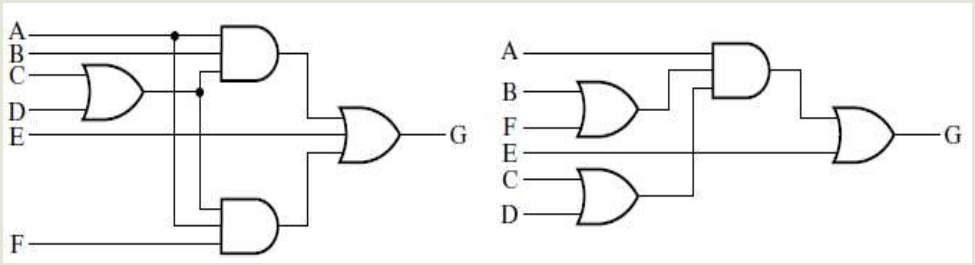
$$G = AB(C + D) + E + A(C + D)F$$

$$G = A(B + F)(C + D) + E$$

❑ Exercise



❑ Exercise



❑ Exercise

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

A.A + A.C + A.B + B.C	– Distributive law
A + A.C + A.B + B.C	– Idempotent AND law (A.A = A)
A(1 + C) + A.B + B.C	– Distributive law
A.1 + A.B + B.C	– Identity OR law (1 + C = 1)
A(1 + B) + B.C	– Distributive law
A.1 + B.C	– Identity OR law (1 + B = 1)
A + (B.C)	– Identity AND law (A.1 = A)

❑ Exercise

~(A * B) * (~A + B) * (~B + B) = ~A

~(A * B) * (~A + B) * 1	Complement law
~(A * B) * (~A + B)	Identity law
(~A + ~B) * (~A + B)	DeMorgan's law
~A + ~B * B	Distributive law
~A + 0	Complement law
~A	Identity law