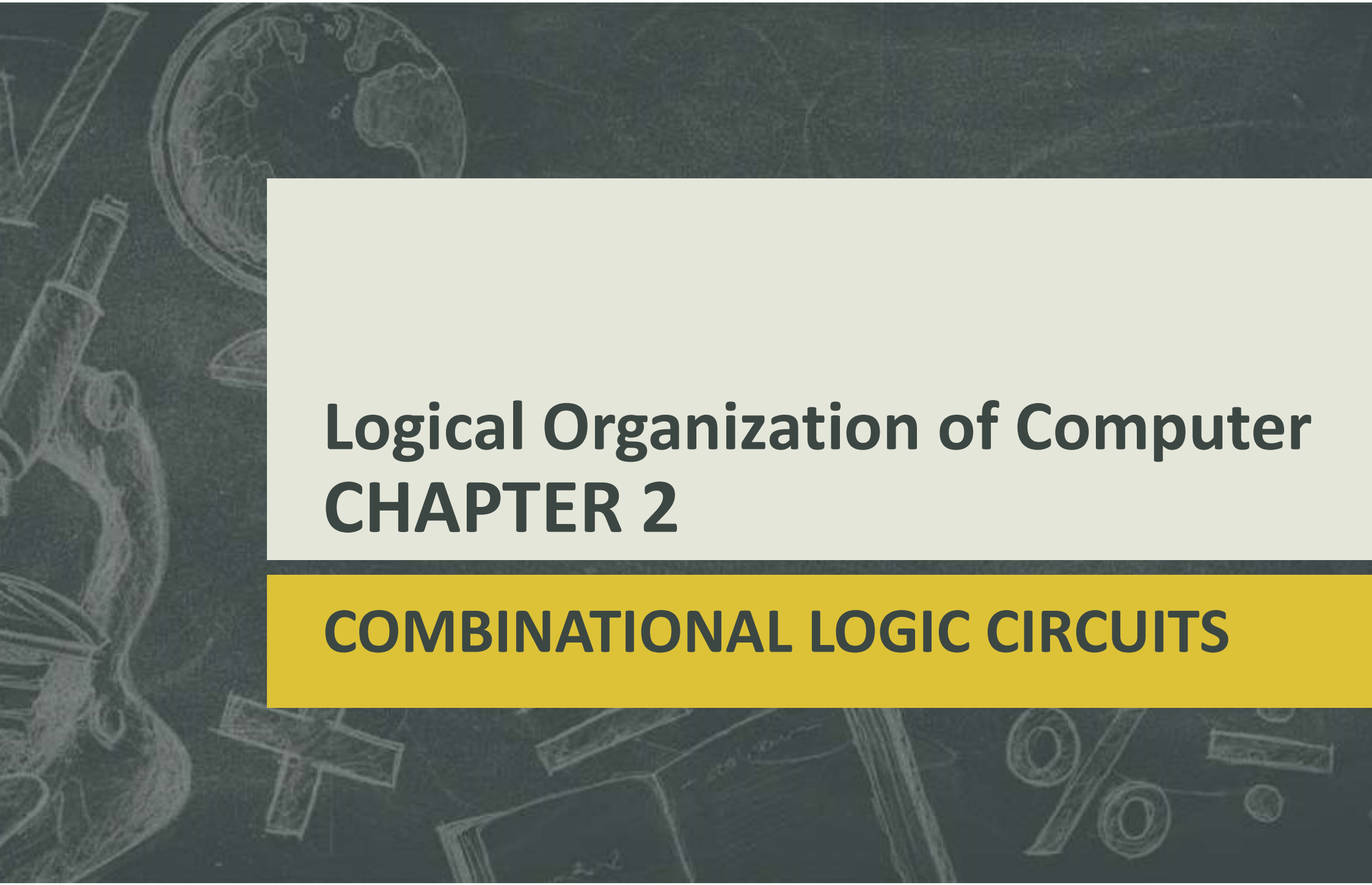


Logical Organization of Computer
CHAPTER 2

COMBINATIONAL LOGIC CIRCUITS

. BINARY LOGIC AND GATES

igital circuits are hardware components that manipulate bir
nformation. The circuits are implemented using transistors
nterconnections in complex semiconductor devices ca
ntegrated circuits. Each basic circuit is referred to as a *logic ga*

ach gate performs a specific logical operation. The outputs
ates are applied to the inputs of other gates to form a dig
rcuit.

n order to describe the operational properties of digital circu
e need to introduce a mathematical notation that specifies
peration of each gate and that can be used to analyze and des
rcuits known as *Boolean algebras*.

Binary Logic

inary logic deals with binary variables, which take on vo discrete values, and with the operations of athematical logic applied to these variables.

ssociated with the binary variables are three basic gical operations called AND, OR, and NOT.

. **AND.** This operation is represented by a dot or by he absence of an operator. For example, $Z=X \cdot Y$ or $=XY$ is read "Z is equal to X AND Y." The logical peration AND is interpreted to mean that $Z=1$ if and nly 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

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

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

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

Binary Logic

truth table for an operation is a table of combinations of t
inary variables showing the relationship between the valu
hat the variables take on and the values of the result of t
peration.

he truth tables for the operations AND, OR, and NOT a
hown 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	$Z = \overline{X}$
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

gic gates are electronic circuits that operate on one or m
put signals to produce an output signal.

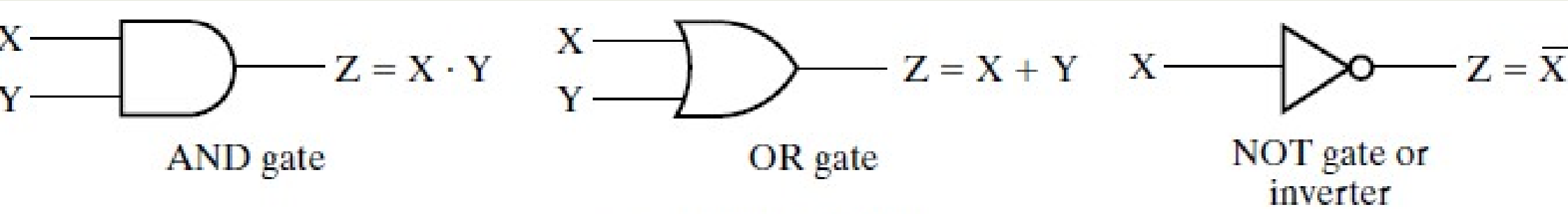
e input terminals of logic gates accept binary signals within
owable range and respond at the output terminals with bir
gnals that fall within a specified range.

e intermediate regions between the allowed ranges in the fig
e crossed only during changes from 1 to 0 or from 0 to 1.

ese changes are called *transitions*, and the intermediate regi
e called the *transition regions*.

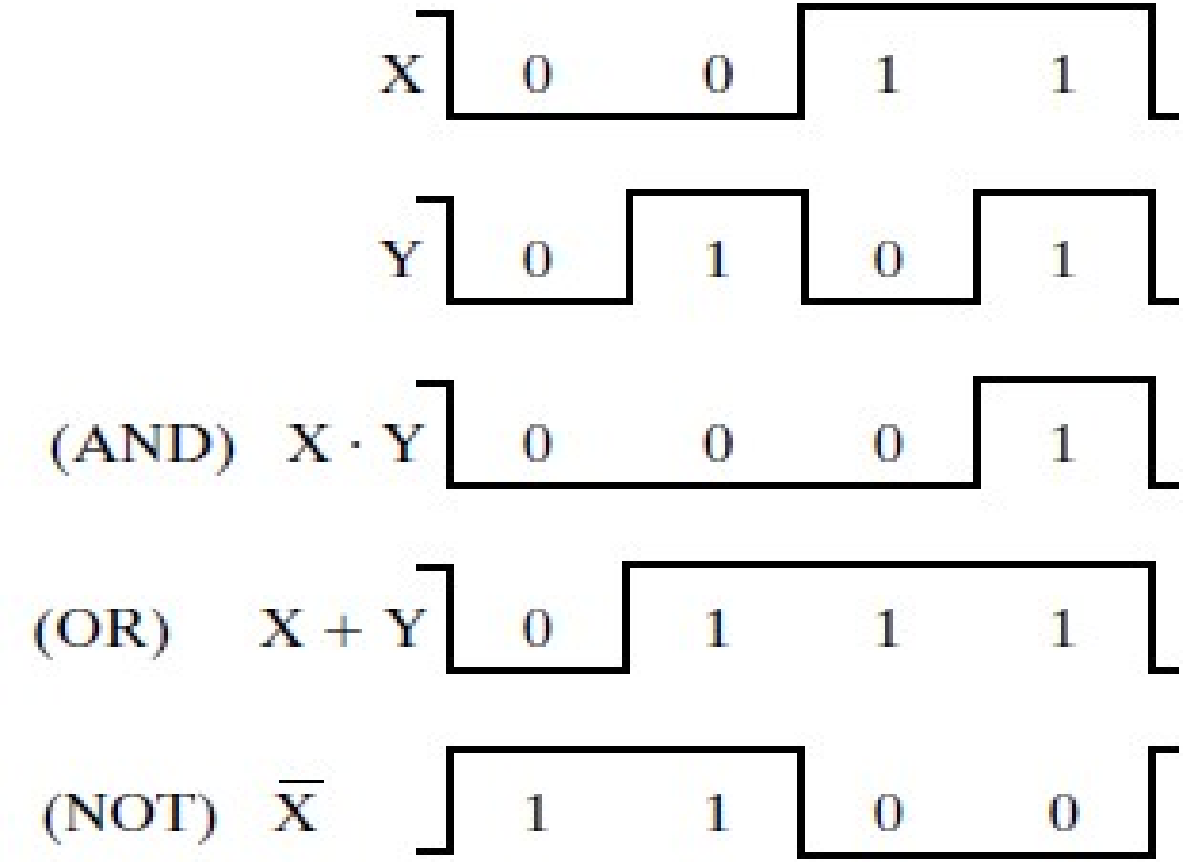
Logic Gates

e graphics symbols used to designate the three types of gate
ND, OR, and NOT—are shown below.



(a) Graphic symbols

Logic Gates



(b) Timing diagram

. BOOLEAN ALGEBRA

Boolean expression is an algebraic expression formed by us
inary variables, the constants 0 and 1, the logic opera
ymbols, and parentheses.

oolean function can be described by a Boolean equa
onsisting of a binary variable identifying the function followe
n 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 . 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 . 1 = A	A variable AND'ed with 1 is always equal to the variable

Basic Identities of Boolean Algebra

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

$A + A = A$	A variable OR'ed with itself is always equal to the variable
$A . A = A$	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"

$A . \overline{A} = 0$	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

$A \cdot B = B \cdot A$ 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)	Since A+AB = A(1+B) = A.1 =A;
A(A + B) = A (AND Absorption Law)	A(A+B) = AA+AB = A+AB = A(1+B) = A.1 =A;

Basic Identities of Boolean Algebra

ssociative Law – This law allows the removal of brackets from an expression
nd 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

Boolean Functions

Boolean function refers to a function having n number of entries o
ariables, so it has 2n number of possible combinations of the give
ariables. Such functions would only assume 0 or 1 in their output.

n example of a Boolean function is, f(p,q,r) = p X q + r. We are implementin
ese functions with the logic gates.

Boolean Functions

We make use of an algebraic expression known as the Boolean Expression to describe the Boolean Function.

The Boolean Expression contains the logic operation symbols, binary variable and the constants 1 and 0. Let us now consider the example given below:

$(W, X, Y, Z) = W + X\bar{Y} + WZY$ Equation No. 1

The left side of this equation here represents the output B. So we can state equation no. 1

$= W + X\bar{Y} + WZY$

Boolean Functions

ruth Table Formation

/e use a truth table to represent a table that has all the combinations of inp
ong with their corresponding results.

F (W, X, Y) = W + XY

Inputs			Output
W	X	Y	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Boolean Functions

ruth Table Formation

A Boolean function may be transferred from an algebraic expression into a logic diagram composed of AND, OR and NOT gates

TABLE 2-2
Truth Tables for $F_1 = xyz'$, $F_2 = x + y'z$, $F_3 = x'y'z + x'yz + xy'$, and $F_4 = xy' + x'z$

x	y	z	F_1	F_2	F_3	F_4
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0

Algebraic Manipulation

oolean algebra is a useful tool for simplifying digital circu
onsider, for example, the Boolean function represented by

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

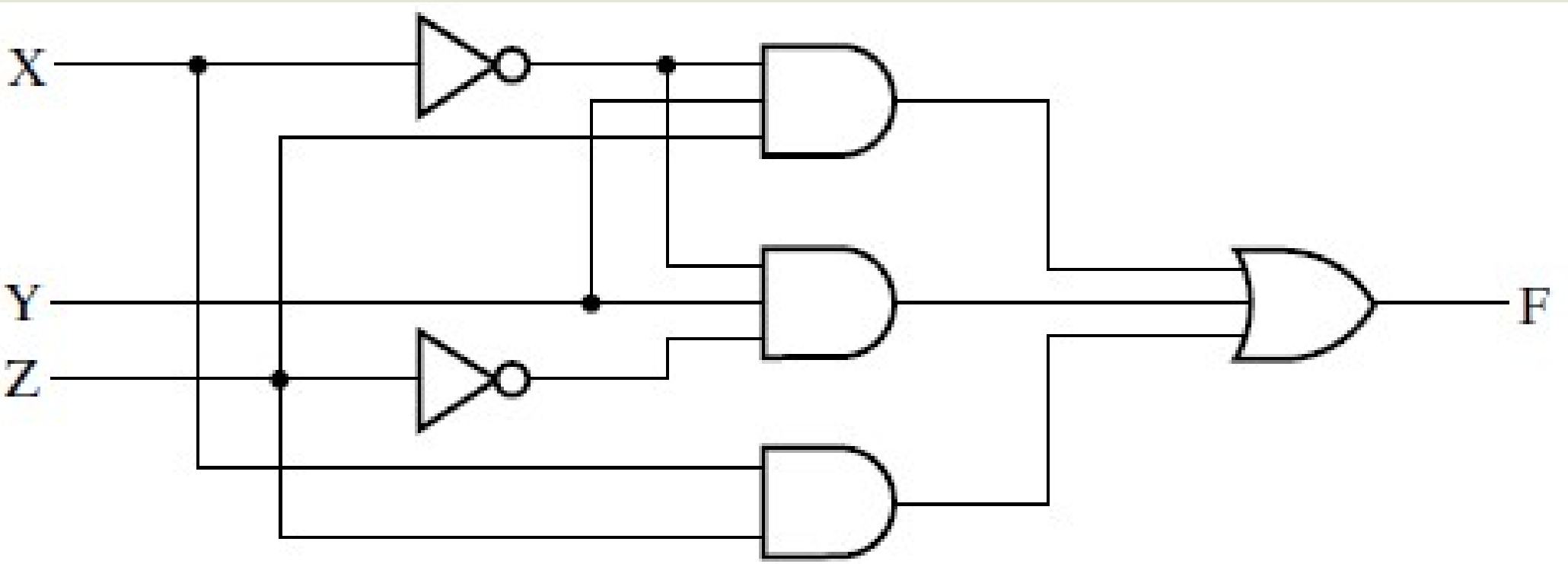
$$\begin{aligned} F &= \overline{X}YZ + \overline{X}Y\overline{Z} + XZ \\ &= \overline{X}Y(Z + \overline{Z}) + XZ \\ &= \overline{X}Y \cdot 1 + XZ \\ &= \overline{X}Y + XZ \end{aligned}$$

Distributive Law

Complement Law

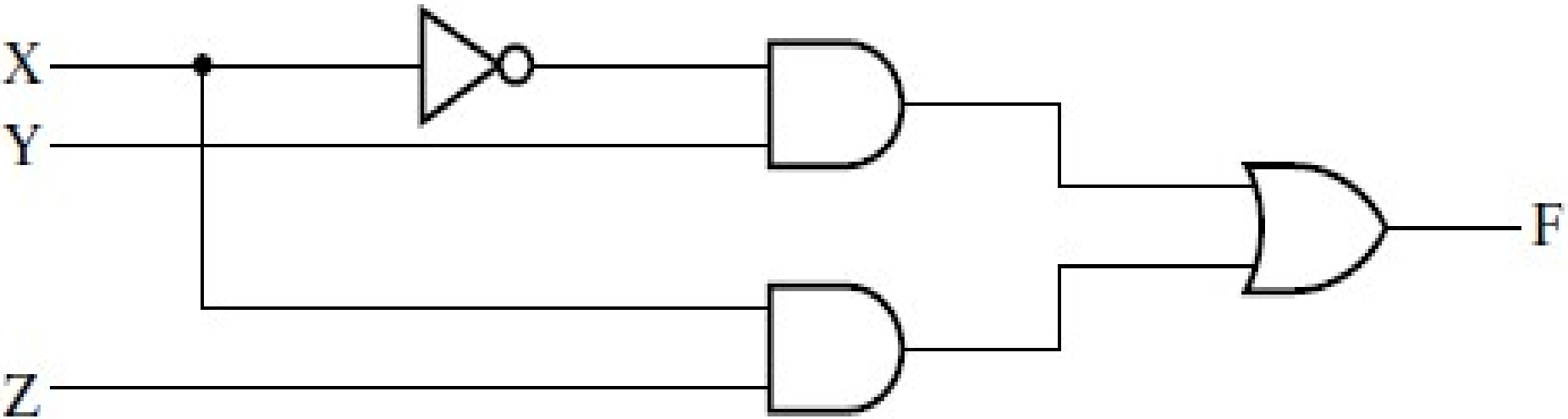
Identity Law

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

Algebraic Manipulation

Minimise the following functions using algebraic method:

ab + ab' + a'b

a'b + a'b' + b‘

b(a+c) + ab' + bc' + c

ab’ +a(b+c)+b(b+c)

[ab’(c+bd)+a’b’]c

a’bc+ab’c’+a’b’c’+abc’+abc

CANONICAL AND STANDARD FORMS

Boolean function expressed algebraically can be written i
ariety of ways.

here are, however, specific ways of writing algebraic equati
hat are considered to be standard forms.

he standard forms facilitate the simplification procedures
oolean expressions and, in some cases, may result in m
esirable expressions for implementing logic circuits.

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.

CANONICAL AND STANDARD FORMS

n Boolean algebra, Boolean function can be expressed
anonical Disjunctive Normal Form known as **minterm** and so
re expressed as Canonical Conjunctive Normal Form kno
s **maxterm** .

Minterms and Maxterms

product term in which all the variables appear exactly o
ther complemented or uncomplemented, is called a *minterm*.

s one characteristic is that it represents exactly one combina
f binary variable values in the truth table. It has the value 1
hat 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
0	0	0	$\overline{X}\,\overline{Y}\,\overline{Z}$	m_0	1	0	0	0	0	0	0
0	0	1	$\overline{X}\,\overline{Y}Z$	m_1	0	1	0	0	0	0	0
0	1	0	$\overline{X}Y\overline{Z}$	m_2	0	0	1	0	0	0	0
0	1	1	$\overline{X}YZ$	m_3	0	0	0	1	0	0	0
1	0	0	$X\overline{Y}\,\overline{Z}$	m_4	0	0	0	0	1	0	0
1	0	1	$X\overline{Y}Z$	m_5	0	0	0	0	0	1	0
1	1	0	$XY\overline{Z}$	m_6	0	0	0	0	0	0	1
1	1	1	XYZ	m_7	0	0	0	0	0	0	0

Minterms and Maxterms

sum term that contains all the variables in complemented ncomplemented form is called a *maxterm*.

gain, it is possible to formulate 2^n maxterms with n variables.

ach maxterm is a logical sum of the three variables, with e ariable being complemented if the corresponding bit of the bir umber 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

onsider the Boolean function F in Table

Boolean Functions of Three Variables

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

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

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

Minterms and Maxterms

				Minterms		*Maxterms*
X	Y	Z		*Product Terms*		*Sum Terms*
0	0	0		$m_0 = \overline{X} \cdot \overline{Y} \cdot \overline{Z} = \min(\overline{X}, \overline{Y}, \overline{Z})$		$M_0 = X + Y + Z = \max(X, Y, Z)$
0	0	1		$m_1 = \overline{X} \cdot \overline{Y} \cdot Z = \min(\overline{X}, \overline{Y}, Z)$		$M_1 = X + Y + \overline{Z} = \max(X, Y, \overline{Z})$
0	1	0		$m_2 = \overline{X} \cdot Y \cdot \overline{Z} = \min(\overline{X}, Y, \overline{Z})$		$M_2 = X + \overline{Y} + Z = \max(X, \overline{Y}, Z)$
0	1	1		$m_3 = \overline{X} \cdot Y \cdot Z = \min(\overline{X}, Y, Z)$		$M_3 = X + \overline{Y} + \overline{Z} = \max(X, \overline{Y}, \overline{Z})$
1	0	0		$m_4 = X \cdot \overline{Y} \cdot \overline{Z} = \min(X, \overline{Y}, \overline{Z})$		$M_4 = \overline{X} + Y + Z = \max(\overline{X}, Y, Z)$
1	0	1		$m_5 = X \cdot \overline{Y} \cdot Z = \min(X, \overline{Y}, Z)$		$M_5 = \overline{X} + Y + \overline{Z} = \max(\overline{X}, Y, \overline{Z})$
1	1	0		$m_6 = X \cdot Y \cdot \overline{Z} = \min(X, Y, \overline{Z})$		$M_6 = \overline{X} + \overline{Y} + Z = \max(\overline{X}, \overline{Y}, Z)$
1	1	1		$m_7 = X \cdot Y \cdot Z = \min(X, Y, Z)$		$M_7 = \overline{X} + \overline{Y} + \overline{Z} = \max(\overline{X}, \overline{Y}, \overline{Z})$

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 gene ated 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 th original function.
4. A function that includes all the 2^n minterms is equal to logic 1.

Minterms and Maxterms

/e perform Sum of minterm also known as Sum of products (SOP) .

/e perform Product of Maxterm also known as Product of sum (POS).

oolean functions expressed as a sum of minterms or product of maxter
re said to be in canonical form.

Sum of Products (SOP)

The sum of products (SOP) expression of a Boolean function can be obtaine from its truth table summing or performing OR operation of the product term corresponding to the combinations containing a function value of 1.

In the product terms the input variables appear either in true (uncomplemented form if it contains the value 1, or in complemented form if it possesses the valu 0.

Now, consider the following truth table in Figure 2.11, for a three-input functio Y. Here the output Y value is 1 for the input conditions of 010, 100, 101, and 11 and their corresponding product terms are A'BC', AB'C', AB'C, and AB respectively

Sum of Products (SOP)

The final sum of products expression (SOP) for the output Y is derived by summing or performing an OR operation of the four product terms as shown below.

$Y = A'BC' + AB'C' + AB'C + ABC'$

Inputs			*Output*	*Product terms*	*Sum t*
A	*B*	*C*	*Y*		
0	0	0	0		A + B
0	0	1	0		A + B
0	1	0	1	A′BC′	
0	1	1	0		A + B′
1	0	0	1	AB′C′	
1	0	1	1	AB′C	
1	1	0	1	ABC′	
1	1	1	0		A′ + B′

Figure 2.11

Sum of Products (SOP)

In general, the procedure of deriving the output expression in SOP form from a truth table can be summarized as below.

1. Form a product term for each input combination in the table, containin an output value of 1.

2. Each product term consists of its input variables in either true form o complemented form. If the input variable is 0, it appears in complemente form and if the input variable is 1, it appears in true form.

3. To obtain the final SOP expression of the output, all the product term are OR operated.

Product of Sums (POS)

The product of sums (POS) expression of a Boolean function can also b obtained from its truth table by a similar procedure.

Here, an AND operation is performed on the sum terms corresponding to th combinations containing a function value of 0.

In the sum terms the input variables appear either in true (uncomplemente form if it contains the value 0, or in complemented form if it possesses the valu 1.

Now, consider the same truth table as shown in Figure 2.11, for a three-inpu function Y. Here the output Y value is 0 for the input conditions of 000, 001, 01 and 111, and their corresponding product terms are A + B + C, A + B + C′, A + B′ C′, and A′ + B′ + C′ respectively.

Product of Sums (POS)

Final product of sums expression (POS) for the output Y is derived by performi an AND operation of the four sum terms as shown below.

$Y = (A + B + C)\,(A + B + C')\,(A + B' + C')\,(A' + B' + C')$

In general, the procedure of deriving the POS form from a truth table can b summarized as below.

1. Form a sum term for each input combination in the table, containing an outp value of 0.

2. Each product term consists of its input variables in either true form complemented form. If the input variable is 1, it appears in complemented for and if the input variable is 0, it appears in true form.

3. To obtain the final POS expression of the output, all the sum terms are AN operated.

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 t 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 o the output variable in cells within a rectangle or square gri according to a definite pattern.

The number of cells in the K-map is determined by the number o input variables and is mathematically expressed as two raised t the power of the number of input variables, i.e., 2^n, where th 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
using an encoding technique known as Gray code.

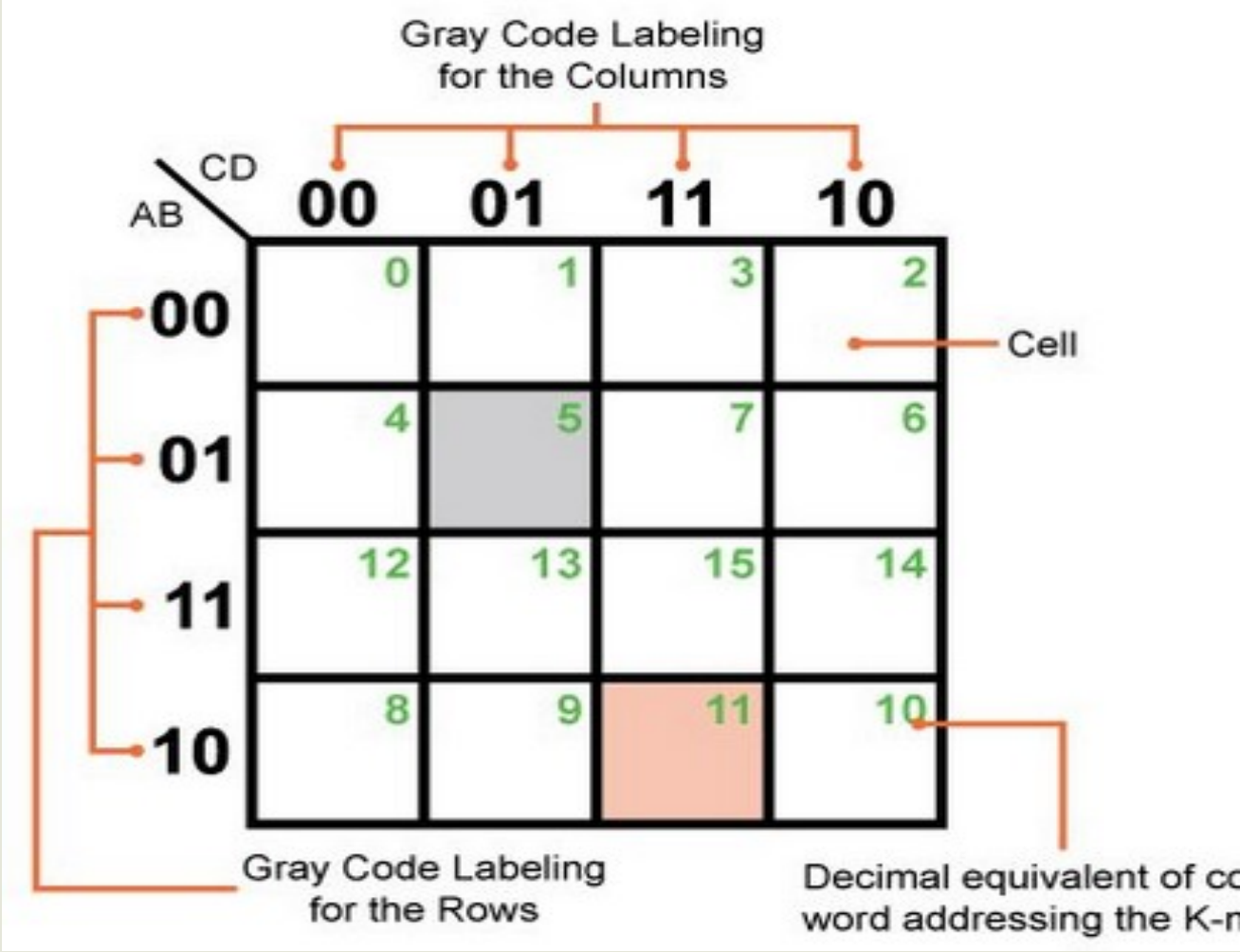
The specialty of this code is the fact that the adjacent code values diff
only by a single bit.

That is, if the given code-word is 01, then the previous and the next cod
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-labeli
which in turn represent the values of the corresponding input variable
This means that each K-map cell can be addressed using a unique Gr
Code-Word.

Karnaugh Map Boolean Algebraic Simplification Technique

hese concepts are urther emphasized by a ypical 16-celled K-map hown in below diagram, which can be used to implify a logical xpression comprising of --variables (A, B, C and D mentioned at its top-left orner).

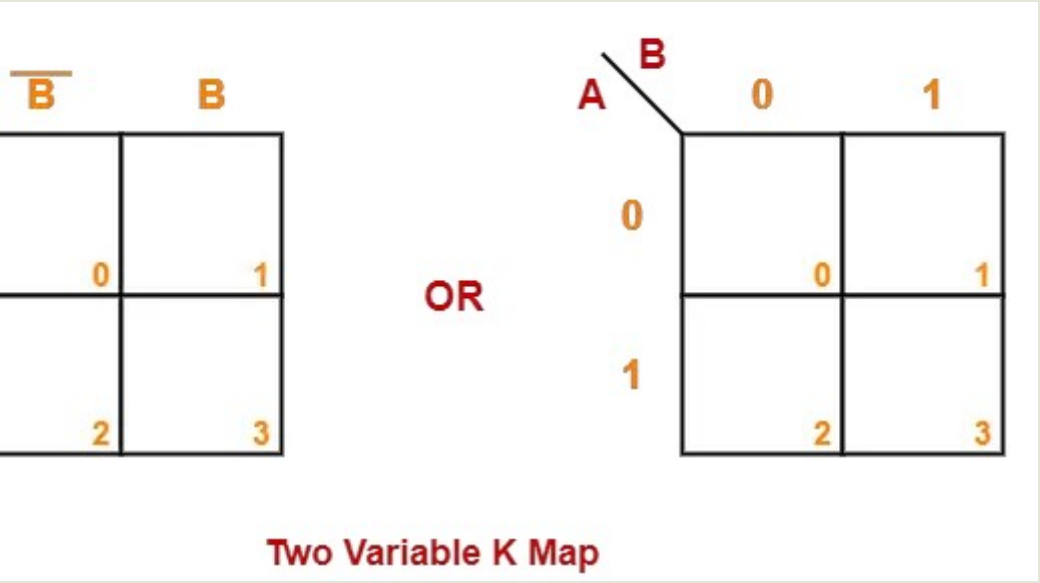


Karnaugh Map Boolean Algebraic Simplification Technique

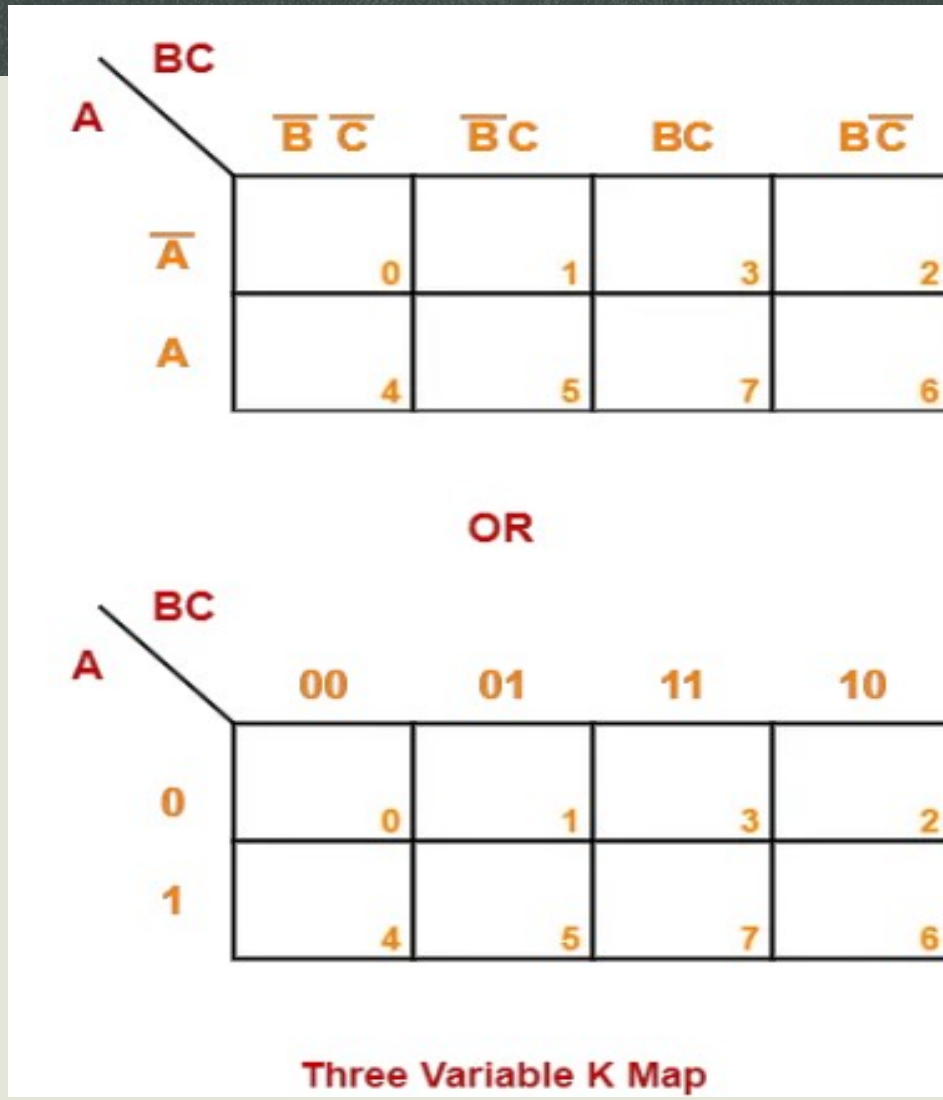
example, the grey colored cell of the K-map shown can be addressed using the code-word "0101" wh
quivalent to 5 in decimal (shown as the green number in the figure) and corresponds to the input
able combination $\bar{A}B\bar{C}D$ or $A+\bar{B}+C+\bar{D}$, depending on whether the input–output relationship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

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

Karnaugh Map Boolean Algebraic Simplification Technique



Two Variable K Map



Three Variable K Map

Karnaugh Map Boolean Algebraic Simplification Technique

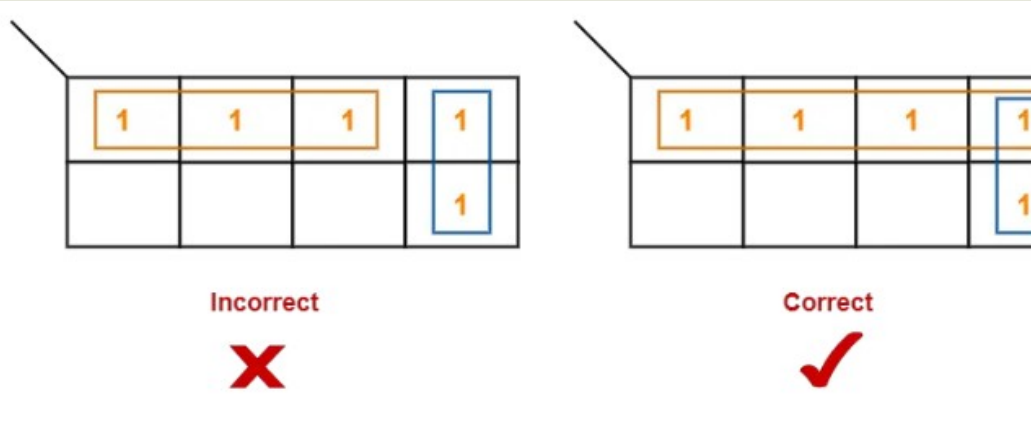
ule-01:

Ve can either group 0's with 0's or 1's with 1's but we can not group 0's and 1's together representing don't care can be grouped with 0's as well as 1's.

ule-02: Groups may overlap each other.

ule-03:

Ve can only create a group whose number of cells can be represented in the power of 2 n other words, a group can only contain 2^n i.e. 1, 2, 4, 8, 16 and so on number of cells.

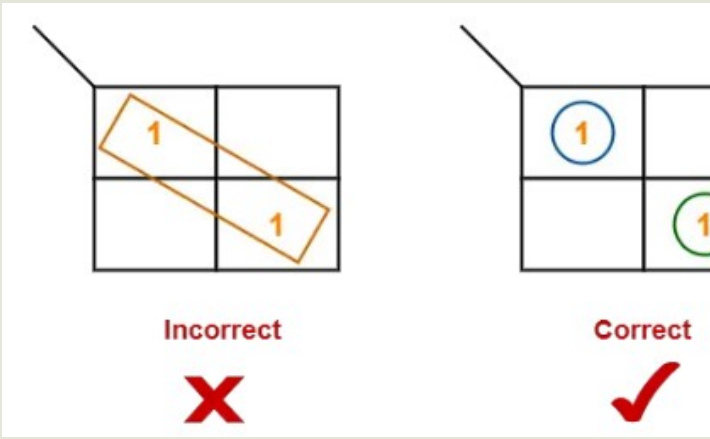


Karnaugh Map Boolean Algebraic Simplification Technique

ule-04:

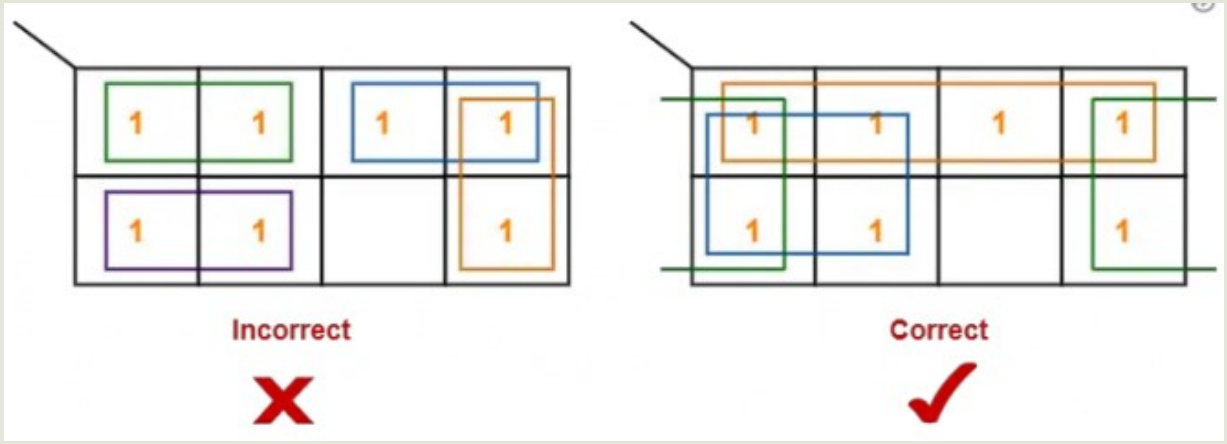
roups can be only either horizontal or vertical.

Ve can not create groups of diagonal or any other shape.



ule-05:

Each group should be as large as possible.



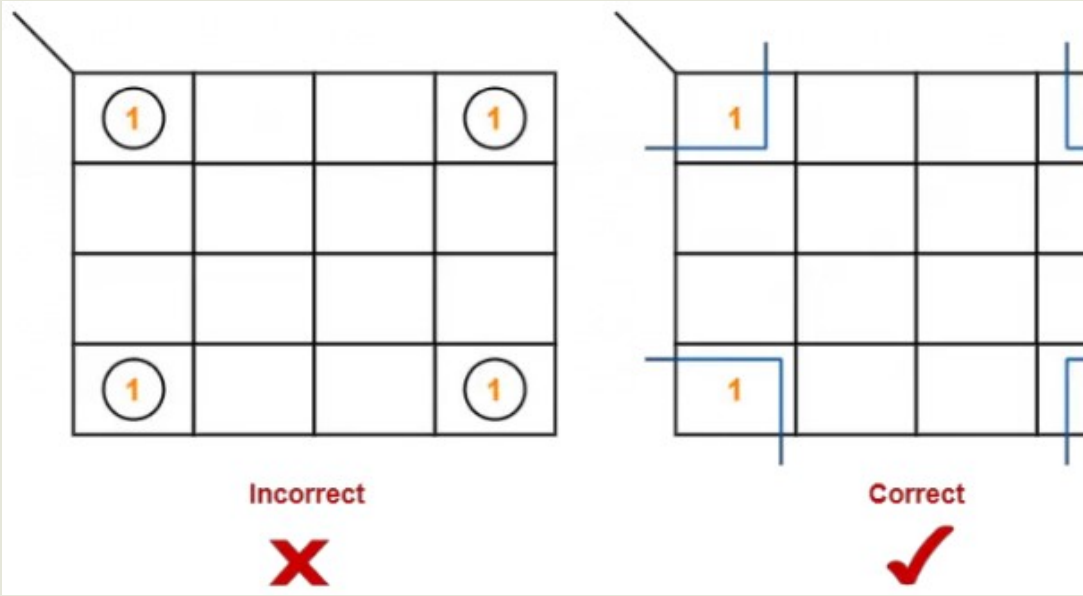
Karnaugh Map Boolean Algebraic Simplification Technique

ule-06:

pposite grouping and corner grouping are allowed.

he example of opposite grouping is shown illustrated in Rule-05.

he example of corner grouping is shown below.



ule-07:

here should be as few groups as possible.

Karnaugh Map Boolean Algebraic Simplification Technique

With this idea of K-maps, let us now move on to the procedur employed in designing an optimal (in terms of the number of gate 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 lo if the 4-bit input binary number is a multiple of 3; otherwise the output will be logically high. The output is defined if an 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

tep 2: Select and Populate K-Map

rom Step 1, we know the number of input variables involved in th
gical expression from which size of the K-map required will be decided.

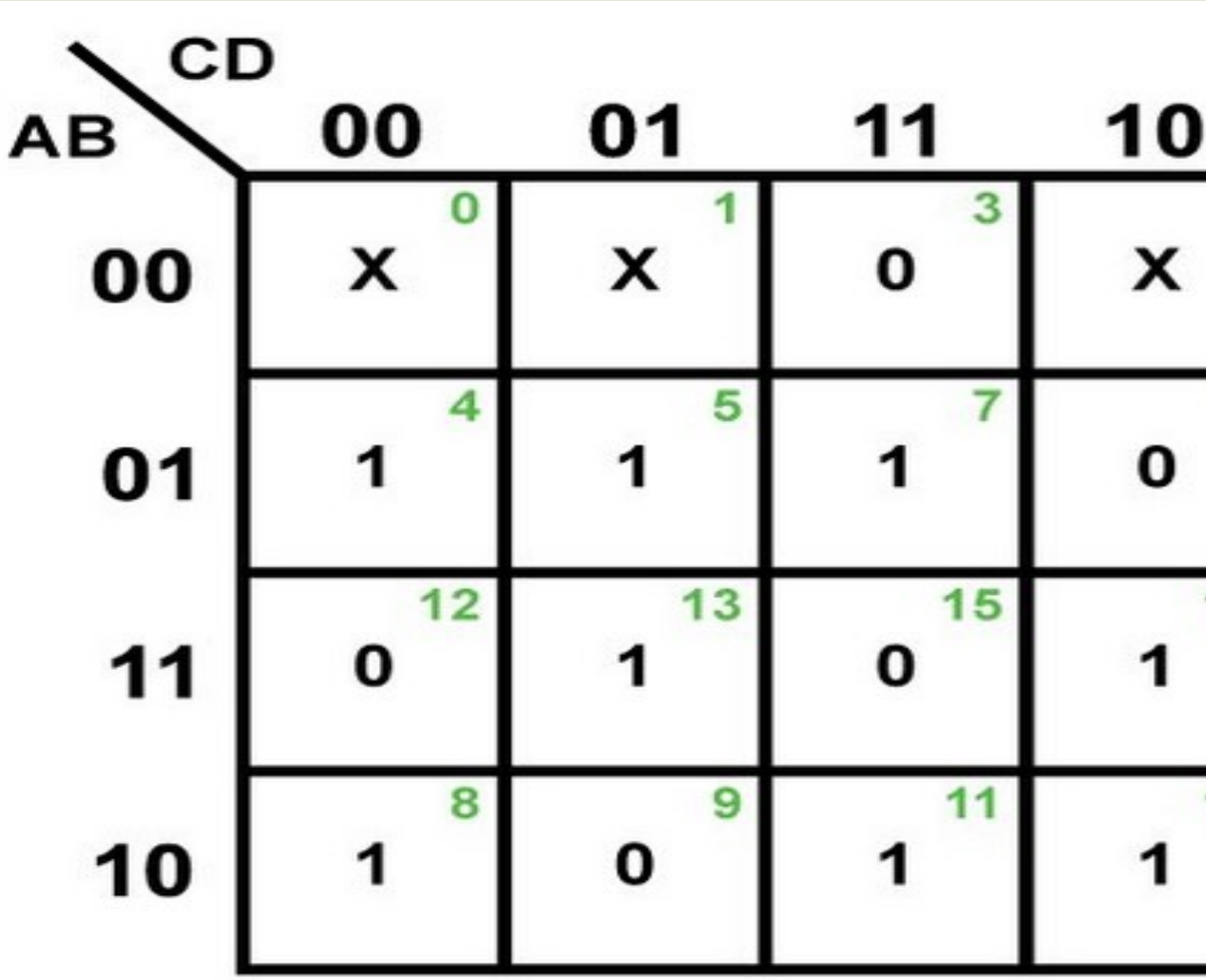
urther, we also know the number of such K-maps required to design th
esired system as the number of output variables would also be know
efinitely.

his means that, for the example considered, we require a single (due t
ne output variable) K-map with 16 cells (as there are four inpu
ariables).

Karnaugh Map Boolean Algebraic Simplification Technique

ext, we have to fill the K-map ells with one for each ninterm, zero for each naxterm, and X for Don't are terms. The procedure is o be repeated for every single utput variable.

ence for this example, we et the K-map as shown in iagram.



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

. The process has to be initiated by grouping the bits which lie i
djacent cells such that the group formed contains the maximun
umber of selected bits.

his means that for an n-variable K-map with 2^n cells, try to group for 2
ells first, then for 2^{n-1} cells, next for 2^{n-2} cells, and so on until th
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 intege
ower to 2, i.e., 1, 2, 4, 8. . . .

Karnaugh Map Boolean Algebraic Simplification Technique

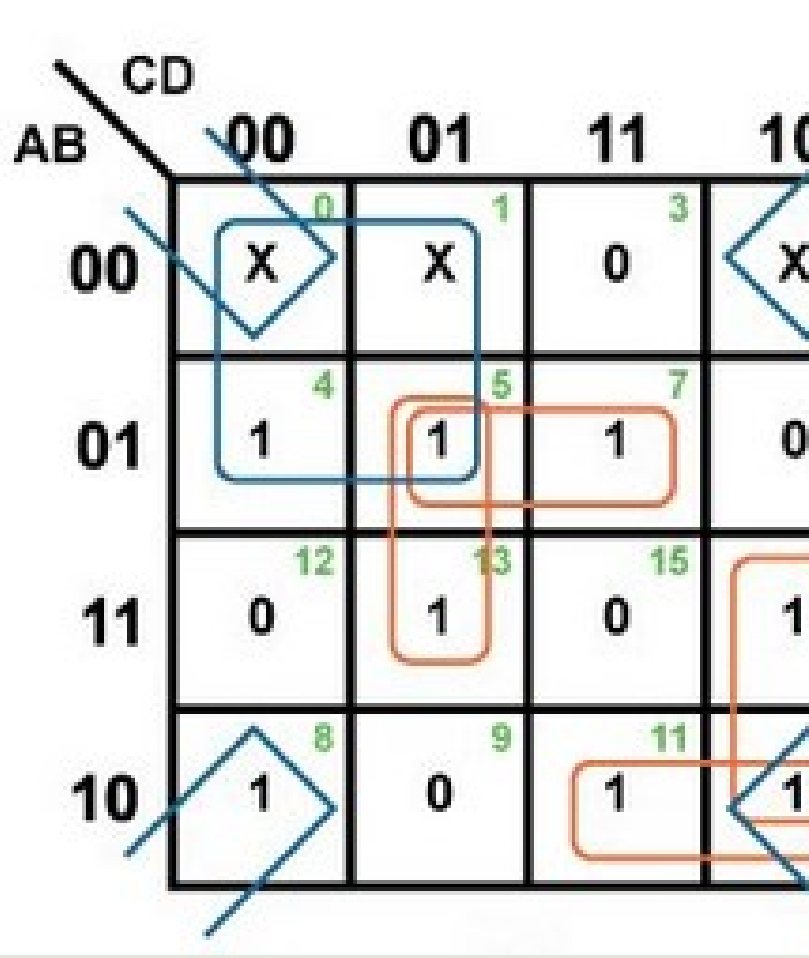
2. The procedure must be applied for all adjacent cells of the K
map, even when they appear to be not adjacent—the top row i
considered to be adjacent to the bottom row and the rightmos
column is considered to be adjacent to the leftmost column, as i
he K-map wraps around from top to bottom and right 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		
mber of groups having 16 cells	0		
mber of groups having 8 cells	0		
mber of groups having 4 cells	2	Group 1 (Cells 0,2,8,10)	
e Enclosures in Figure 3)		Group 2 (Cells 0,1,4,5)	
mber of groups having 2 cells	4	Group 3 (Cells 5,7)	Group 4 (Cells 5,13)
ange Enclosures in Figure 3)		Group 5 (Cells 10,11)	Group 6 (Cells 10,14)



Karnaugh Map Boolean Algebraic Simplification Technique

Step 4: Simplified Logical Expression

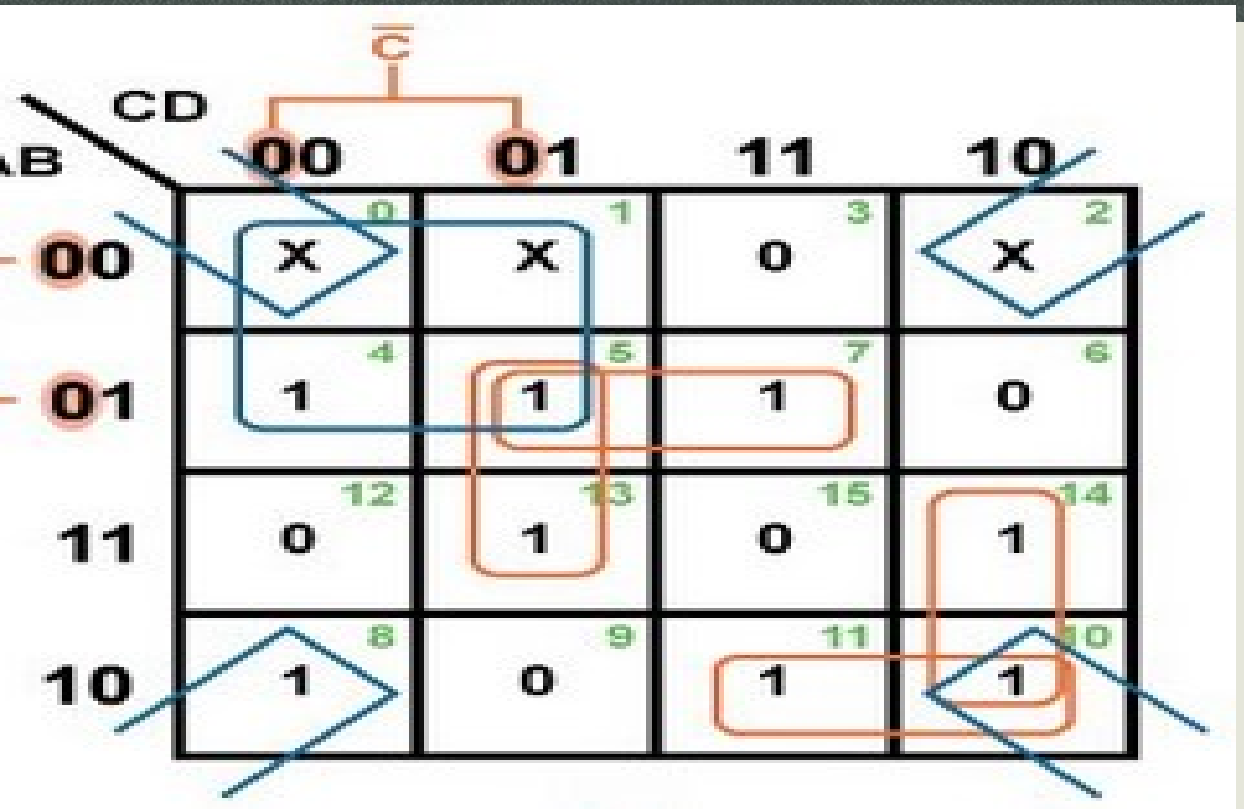
For each of the resulting groups, we have to obtain the correspondi
ogical expression in terms of the input-variables.

This can be done by expressing the bits which are common amongst t
Gray code-words which represent the cells contained within t
considered group.

Finally, all these group-wise logical expressions need to be combin
appropriately to form the simplified Boolean equation for the outp
variable.

The same procedure must be repeated for every output variable of t
given problem.

Karnaugh Map Boolean Algebraic Simplification Technique



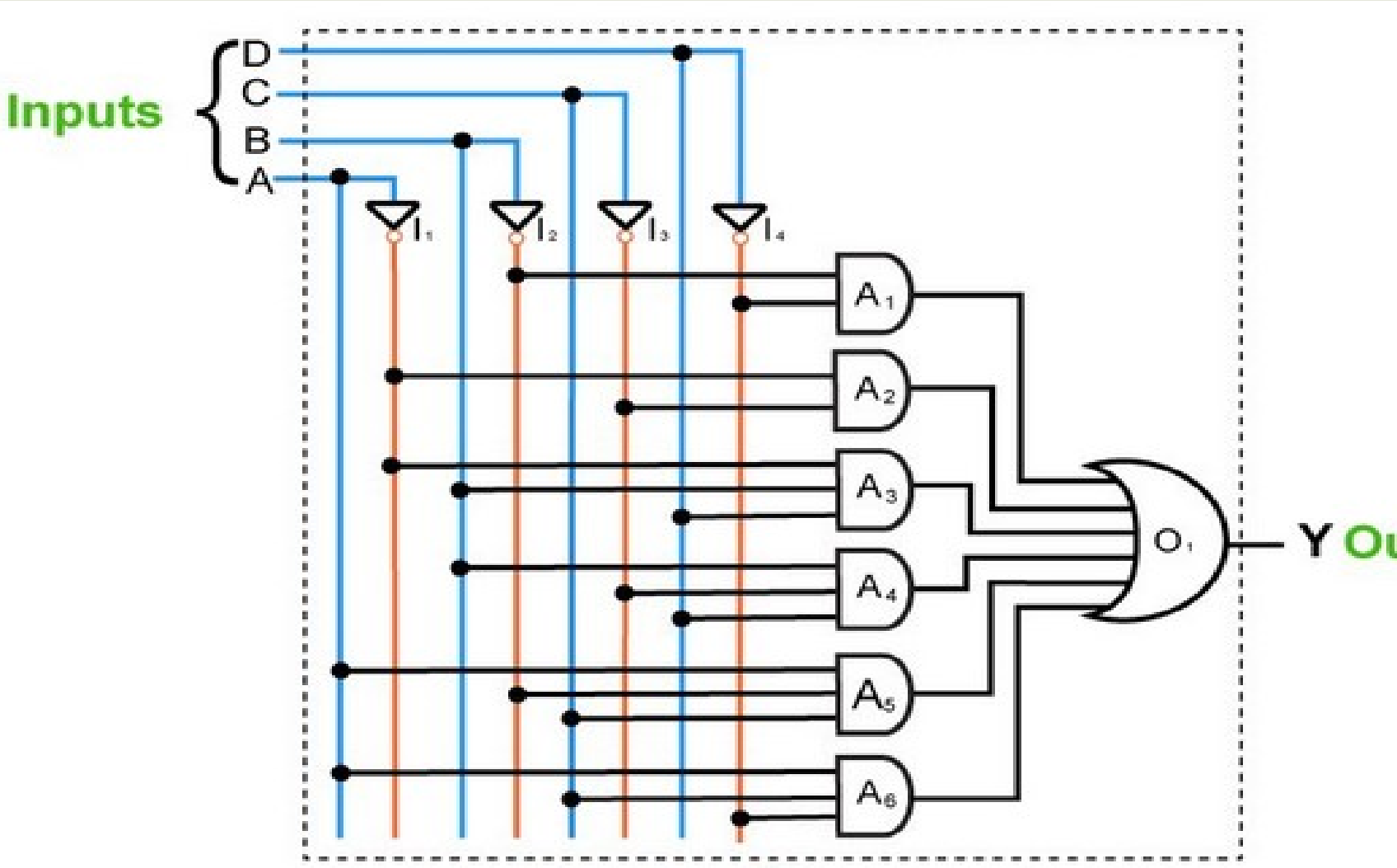
SOP Form Solution	
Groups	**Logical Expre**
Group 1	$\bar{B}\bar{D}$
Group 2	$\bar{A}\bar{C}$
Group 3	$\bar{A}BD$
Group 4	$B\bar{C}D$
Group 5	$A\bar{B}C$
Group 6	$AC\bar{D}$

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

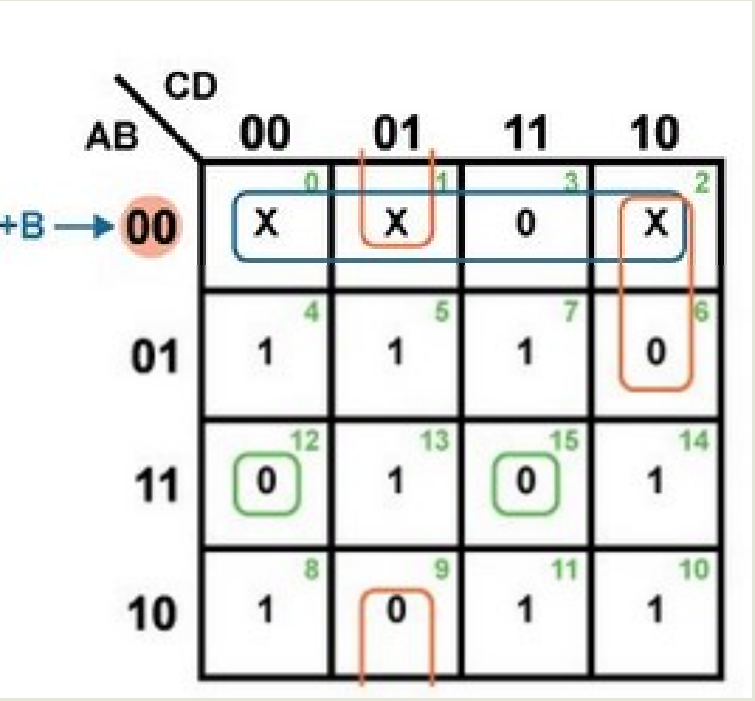
Karnaugh Map Boolean Algebraic Simplification Technique

ep 5: System

ign



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) ($\overline{A}$+$\overline{B}$+$\overline{C}$+$\overline{D}$)	

Don’t Care Conditions

n’t cares in a Karnaugh map, or truth table, may be either **1**s or **0**s, as lor
don’t care what the output is for an input condition we never expect to se

e plot these cells with an asterisk, * or x, among the normal **1**s and **0**s.

hen forming groups of cells, treat the don’t care cell as either a **1** or a **0**, or
ore the don’t cares.

s is helpful if it allows us to form a larger group than would otherwise be
ssible without the don’t cares. There is no requirement to group all or any
e don’t cares.

ly use them in a group if it simplifies the logic.

Exercise

nimize the following boolean functions

F(A, B, C, D) = Σm(0, 2, 8, 10, 14) + Σd(5, 15)

F(A, B, C) = Σm(0, 1, 6, 7) + Σd(3, 4, 5)

F(A, B, C) = Σm(1, 2, 5, 7) + Σd(0, 4, 6)

Exercise

Minimize the following boolean functions

1 F(A, B, C, D) = ACD’ + B’D’

2. F(A, B, C) = A + B’

3. F(A, B, C) = A + B’ + C’

Exercise

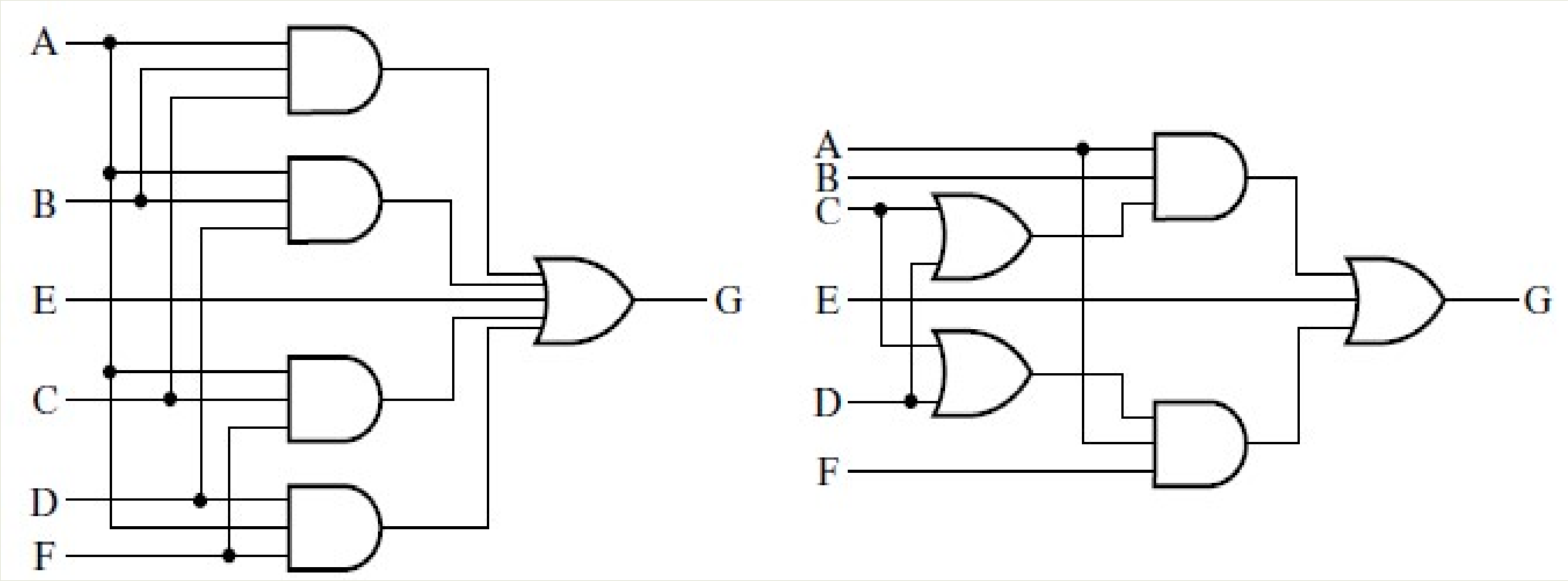
$$G = ABC + ABD + E + ACF + ADF$$

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

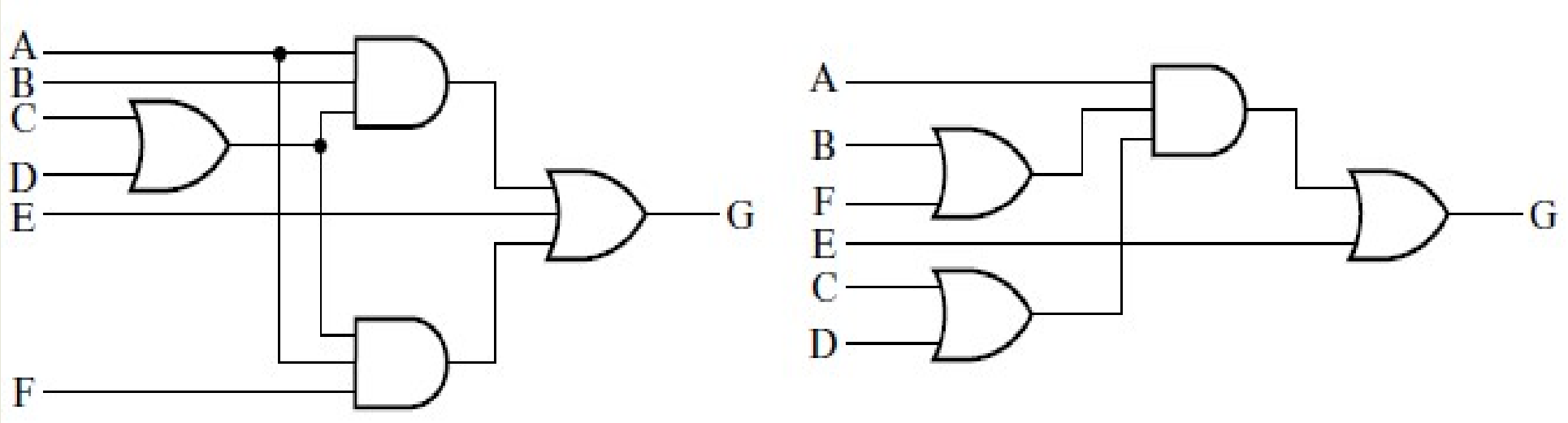
$$G = AB(C + D) + E + A(C + D)F$$

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

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

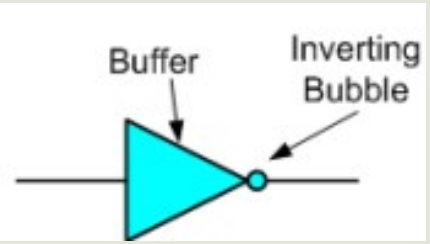
~(A * B) * (~A + B) * (~B + B) = ~A

~(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

Universal Gates

In addition to AND, OR, and NOT gates, other logic gates like NAND and NOR are also used i
the design of digital circuits. The NOT circuit inverts the logic sense of a binary signal.

The small circle (bubble) at the output of the graphic symbol of a NOT gate is formally called
negation indicator and designates the logical complement.



Universal Gates

A universal gate is a gate which can implement any Boolean function without need to use an other gate type. The NAND and NOR gates are universal gates.

In practice, this is advantageous since NAND and NOR gates are economical and easier t fabricate and are the basic gates used in all IC digital logic families.

In fact, an AND gate is typically implemented as a NAND gate followed by an inverter not th other way around. Likewise, an OR gate is typically implemented as a NOR gate followed by a inverter not the other way around.

Universal Gates

NAND Gate

The NAND gate represents the complement of the AND operation. Its name is an abbreviation of NOT AND.

The graphic symbol for the NAND gate consists of an AND symbol with a bubble on the output, denoting that a complement operation is performed on the output of the AND gate.

The truth table and the graphic symbol of NAND gate is shown in the figure.

The truth table clearly shows that the NAND operation is the complement of the AND.

X	Y	NAND
0	0	1
0	1	1
1	0	1
1	1	0

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

Universal Gates

NOR Gate

The NOR gate represents the complement of the OR operation.

Its name is an abbreviation of NOT OR.

The graphic symbol for the NOR gate consists of an OR symbol with a bubble on the output, denoting that a complement operation is performed on the output of the OR gate.

The truth table and the graphic symbol of NOR gate is shown in the figure.

X	Y	NAND
0	0	1
0	1	1
1	0	1
1	1	0

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

Universal Gates

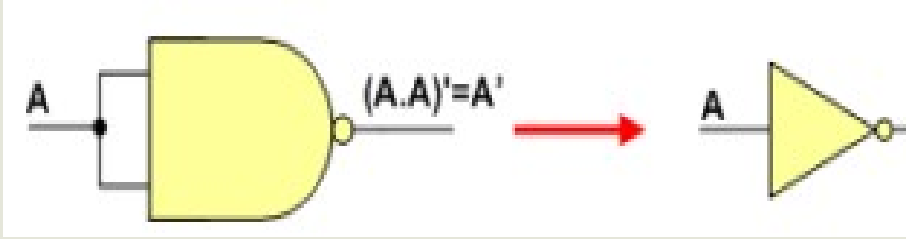
NAND Gate is a Universal Gate

To prove that any Boolean function can be implemented using only NAND gates, we will sho that the AND, OR, and NOT operations can be performed using only these gates.

Implementing an Inverter Using only NAND Gate The figure shows two ways in which a NAN gate can be used as an inverter (NOT gate).

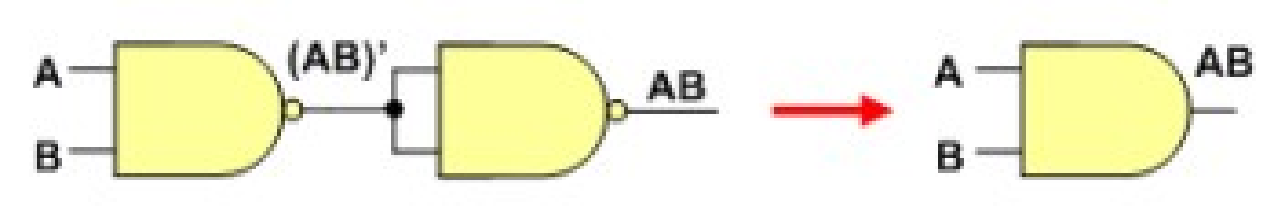
1. All NAND input pins connect to the input signal A gives an output A'.

2. One NAND input pin is connected to the input signal A while all other input pins a connected to logic 1. The output will be A'.

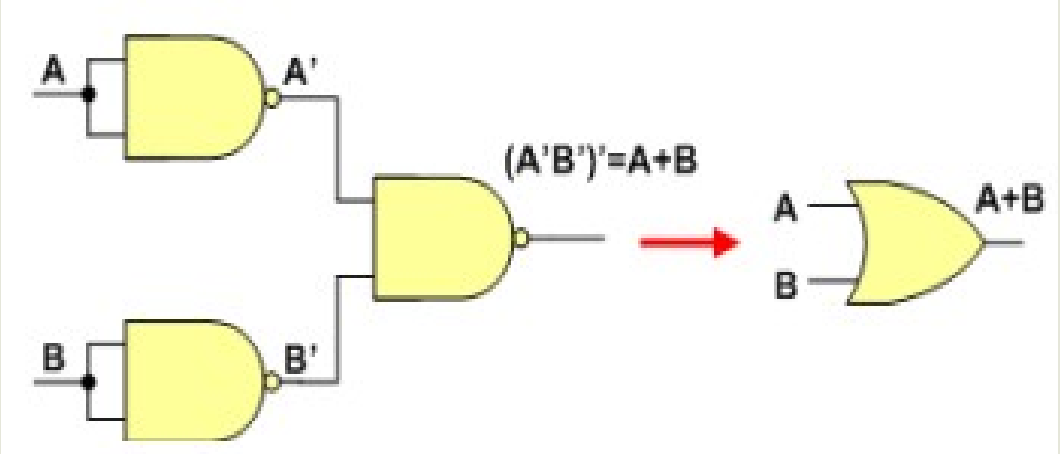


Universal Gates

Implementing AND Using only NAND Gates An AND gate can be replaced by NAND gates a shown in the figure (The AND is replaced by a NAND gate with its output complemented by NAND gate inverter).



Implementing OR Using only NAND Gates An OR gate can be replaced by NAND gates as show in the figure (The OR gate is replaced by a NAND gate with all its inputs complemented by NAN gate inverters).

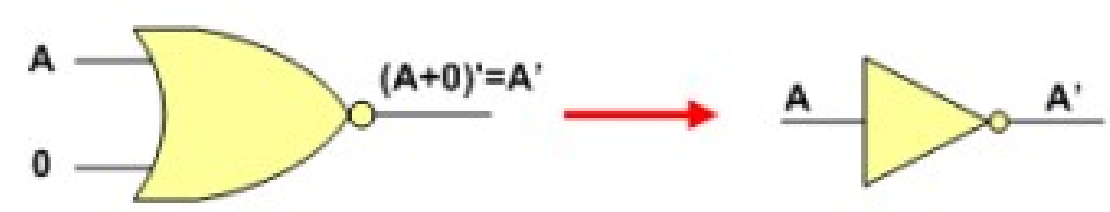


Universal Gates

NOR Gate is a Universal Gate

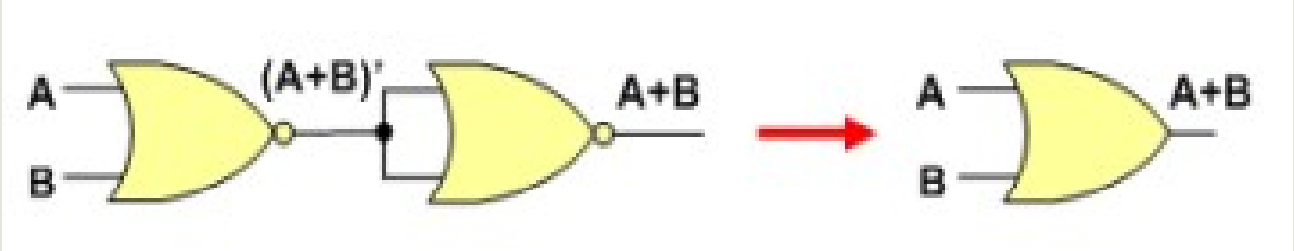
The figure shows two ways in which a NOR gate can be used as an inverter (NOT gate). 1. A
NOR input pins connect to the input signal A gives an output A’

2. One NOR input pin is connected to the input signal A while all other input pins are connecte
to logic 0. The output will be A’.

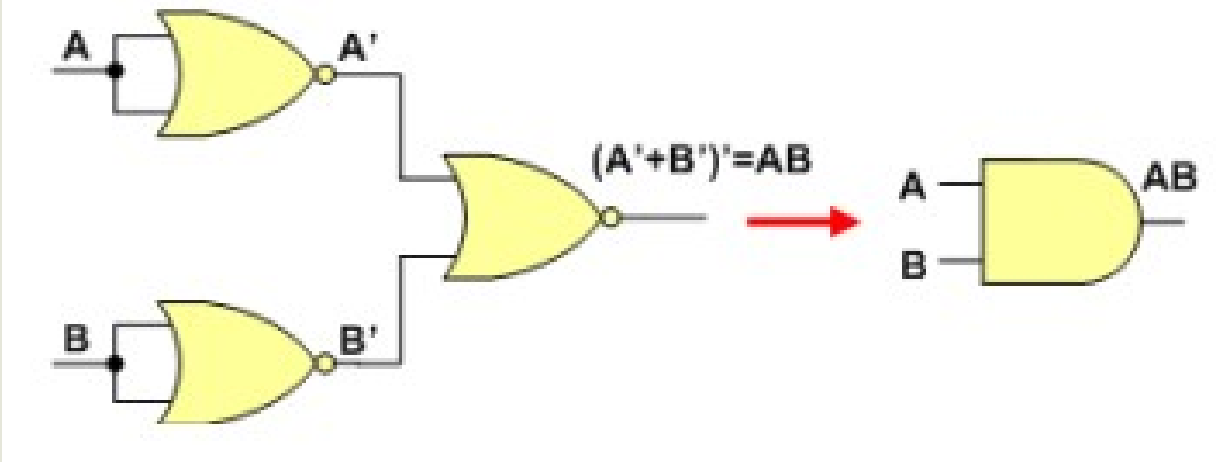


Universal Gates

Implementing OR Using only NOR Gates An OR gate can be replaced by NOR gates as shown the figure (The OR is replaced by a NOR gate with its output complemented by a NOR ga inverter)



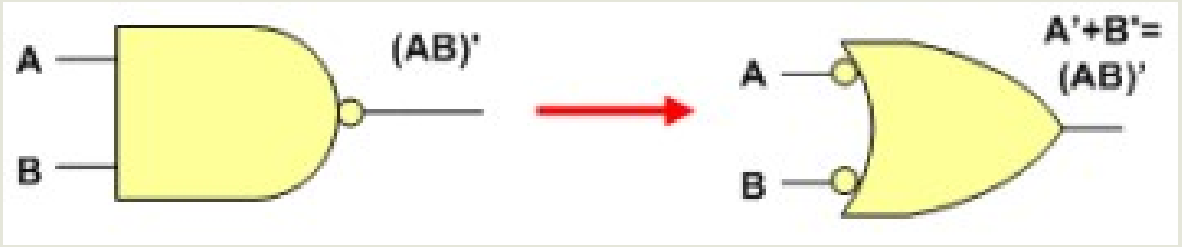
Implementing AND Using only NOR Gates An AND gate can be replaced by NOR gates as show in the figure (The AND gate is replaced by a NOR gate with all its inputs complemented by NO gate inverters)



Universal Gates

Equivalent Gates:

The shown figure summarizes important cases of gate equivalence. Note that bubbles indicate complement operation (inverter). A NAND gate is equivalent to an inverted-input OR gate

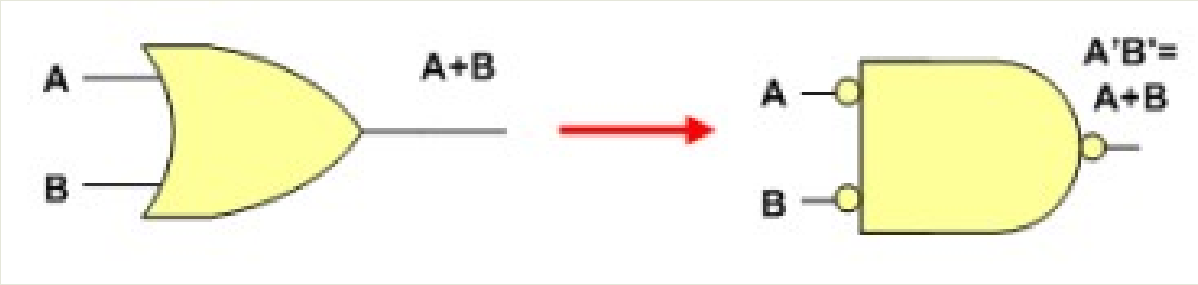


An AND gate is equivalent to an inverted-input NOR gate.

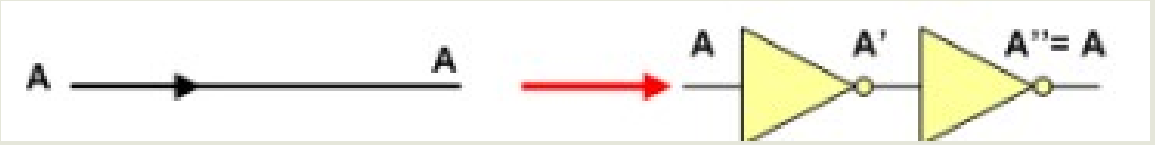
Universal Gates

A NOR gate is equivalent to an inverted-input AND gate.

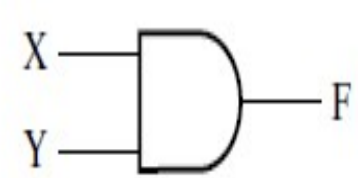
An OR gate is equivalent to an inverted-input NAND gate.



Two NOT gates in series are same as a buffer because they cancel each other as A'' = A.

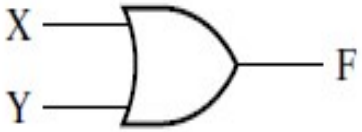


OTHER GATE TYPES



$F = XY$

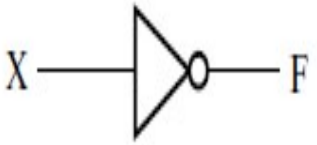
X	Y	F
0	0	0
0	1	0
1	0	0
1	1	1



$F = X + Y$

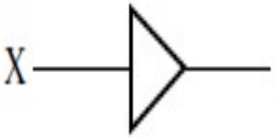
X	Y	F
0	0	0
0	1	1
1	0	1
1	1	1

NOT (inverter)



$F = \overline{X}$

Buffer



$F = X$

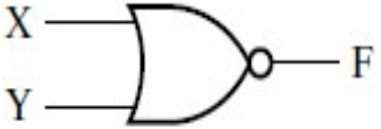
OTHER GATE TYPES

NAND

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

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

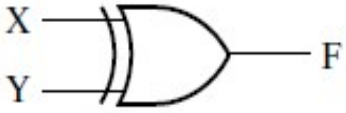
NOR



$F = \overline{X + Y}$

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

Exclusive-OR
(XOR)

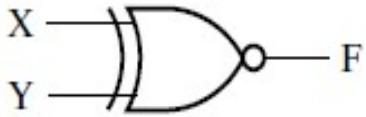


$F = X\overline{Y} + \overline{X}Y$
$= X \oplus Y$

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

OTHER GATE TYPES

Exclusive-NOR
(XNOR)

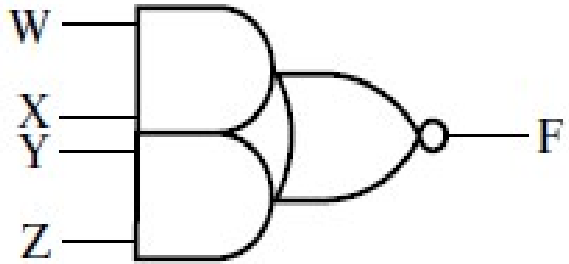


$$F = XY + \overline{X}\,\overline{Y}$$

$$= X \oplus Y$$

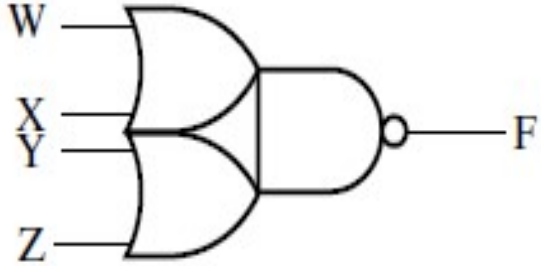
X	Y	F
0	0	1
0	1	0
1	0	0
1	1	1

AND-OR-INVERT
(AOI)



$$F = \overline{WX + YZ}$$

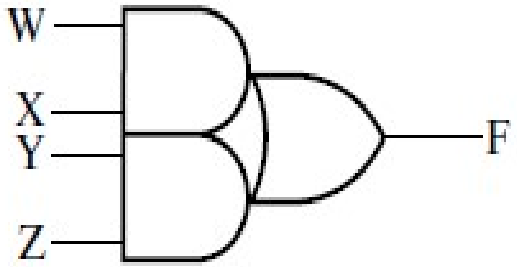
OR-AND -INVERT
(OAI)



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

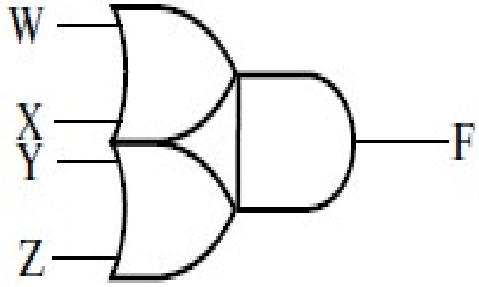
OTHER GATE TYPES

AND-OR
(AO)



$F = WX + YZ$

OR-AND
(OA)



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