

## UNIT - 1

### NUMBER SYSTEMS & BOOLEAN ALGEBRA

- Introduction about digital system
- Philosophy of number systems
- Complement representation of negative numbers
- Binary arithmetic

#### INTRODUCTION ABOUT DIGITAL SYSTEM

A Digital system is an interconnection of digital modules and it is a system that manipulates discrete elements of information that is represented internally in the binary form.

Now a day's digital systems are used in wide variety of industrial and consumer products such as automated industrial machinery, pocket calculators, microprocessors, digital computers, digital watches, TV games and signal processing and so on.

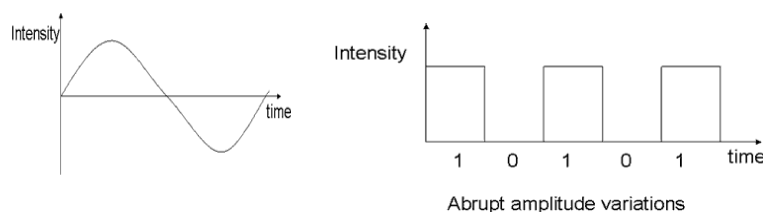
#### Characteristics of Digital systems

- Digital systems manipulate discrete elements of information.
- Discrete elements are nothing but the digits such as 10 decimal digits or 26 letters of alphabets and so on.
- Digital systems use physical quantities called signals to represent discrete elements.
- In digital systems, the signals have two discrete values and are therefore said to be binary.
- A signal in digital system represents one binary digit called a bit. The bit has a value either 0 or 1.

#### Analog systems vs Digital systems

Analog system process information that varies continuously i.e; they process time varying signals that can take on any values across a continuous range of voltage, current or any physical parameter.

Digital systems use digital circuits that can process digital signals which can take either 0 or 1 for binary system.



## Advantages of Digital system over Analog system

### 1. Ease of programmability

The digital systems can be used for different applications by simply changing the program without additional changes in hardware.

### 2. Reduction in cost of hardware

The cost of hardware gets reduced by use of digital components and this has been possible due to advances in IC technology. With ICs the number of components that can be placed in a given area of Silicon are increased which helps in cost reduction.

### 3. High speed

Digital processing of data ensures high speed of operation which is possible due to advances in Digital Signal Processing.

### 4. High Reliability

Digital systems are highly reliable one of the reasons for that is use of error correction codes.

### 5. Design is easy

The design of digital systems which require use of Boolean algebra and other digital techniques is easier compared to analog designing.

### 6. Result can be reproduced easily

Since the output of digital systems unlike analog systems is independent of temperature, noise, humidity and other characteristics of components the reproducibility of results is higher in digital systems than in analog systems.

## Disadvantages of Digital Systems

- Use more energy than analog circuits to accomplish the same tasks, thus producing more heat as well.
- Digital circuits are often fragile, in that if a single piece of digital data is lost or misinterpreted the meaning of large blocks of related data can completely change.
- Digital computer manipulates discrete elements of information by means of a binary code.
- Quantization error during analog signal sampling.

## NUMBER SYSTEM

Number system is a basis for counting various items. Modern computers communicate and operate with binary numbers which use only the digits 0 & 1. Basic number system used by humans is Decimal number system.

For Ex: Let us consider decimal number 18. This number is represented in binary as 10010.

We observe that binary number system takes more digits to represent the decimal number. For large numbers we have to deal with very large binary strings.

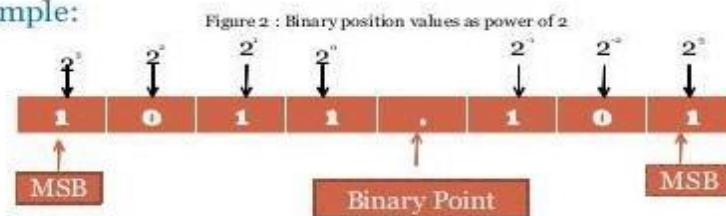
To define any number system we have to specify

- Base of the number system such as 2, 8, 10 or 16.
- The base decides the total number of digits available in that number system.
- First digit in the number system is always zero and last digit in the number system is always base-1.

### Binary number system:

The binary number has a radix of 2. As  $r = 2$ , only two digits are needed, and these are 0 and 1. In binary system weight is expressed as power of 2.

#### • Example:



The left most bit, which has the greatest weight is called the Most Significant Bit (MSB). And the right most bit which has the least weight is called Least Significant Bit (LSB).

For Ex:  $1001.01_2 = [(1) \times 2^3] + [(0) \times 2^2] + [(0) \times 2^1] + [(1) \times 2^0] + [(0) \times 2^{-1}] + [(1) \times 2^{-2}]$

$$1001.01_2 = [1 \times 8] + [0 \times 4] + [0 \times 2] + [1 \times 1] + [0 \times 0.5] + [1 \times 0.25]$$

$$1001.01_2 = 9.25_{10}$$

## Decimal Number system

The decimal system has ten symbols: 0,1,2,3,4,5,6,7,8,9. In other words, it has a base of 10.

## Octal Number System

Digital systems operate only on binary numbers. Since binary numbers are often very long, two shorthand notations, octal and hexadecimal, are used for representing large binary numbers. Octal systems use a base or radix of 8. It uses first eight digits of decimal number system. Thus it has digits from 0 to 7.

## Hexa Decimal Number System

The hexadecimal numbering system has a base of 16. There are 16 symbols. The decimal digits 0 to 9 are used as the first ten digits as in the decimal system, followed by the letters A, B, C, D, E and F, which represent the values 10, 11, 12, 13, 14 and 15 respectively.

Decimal	Binary	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

## Number Base conversions

The human beings use decimal number system while computer uses binary number system. Therefore it is necessary to convert decimal number system into its equivalent binary.

- i) Binary to octal number conversion
- ii) Binary to hexa decimal number conversion

The binary number: 001 010 011 000 100 101 110 111  
The octal number: 1 2 3 0 4 5 6 7

The binary number: 0001 0010 0100 1000 1001 1010 1101 1111  
The hexadecimal number: 1 2 5 8 9 A D F

- iii) Octal to binary Conversion

Each octal number converts to 3 binary digits

Code
0 - 000
1 - 001
2 - 010
3 - 011
4 - 100
5 - 101
6 - 110
7 - 111

To convert  $653_8$  to binary, just substitute code:

6 5 3  
↓ ↓ ↓  
110 101 011

- iv) Hexa to binary conversion  
**0100 1111 1101 0111**

- v) Octal to Decimal conversion

Ex: convert  $4057.06_8$  to octal

$$\begin{aligned} &= 4 \times 8^3 + 0 \times 8^2 + 5 \times 8^1 + 7 \times 8^0 + 0 \times 8^{-1} + 6 \times 8^{-2} \\ &= 2048 + 0 + 40 + 7 + 0 + 0.0937 \end{aligned}$$

$$=2095.0937_{10}$$

vi) Decimal to Octal Conversion

Ex: convert  $378.93_{10}$  to octal

**$378_{10}$  to octal:** Successive division:

$$\begin{array}{r} 8 \overline{) 378} \\ \underline{8 \mid 47} \quad \text{---} \quad 2 \\ \underline{8 \mid 5} \quad \text{---} \quad 7 \quad \uparrow \\ 0 \quad \text{---} \quad 5 \end{array}$$

$$=572_8$$

$0.93_{10}$  to octal :

$$0.93 \times 8 = 7.44$$

$$0.44 \times 8 = 3.52$$

$$0.53 \times 8 = 4.16$$

$$0.16 \times 8 = 1.28$$

$$=0.7341_8$$

$$378.93_{10} = 572.7341_8$$

vii) Hexadecimal to Decimal Conversion

Ex:  $5C7_{16}$  to decimal

$$= (5 \times 16^2) + (C \times 16^1) + (7 \times 16^0)$$

$$= 1280 + 192 + 7$$

$$= 147_{10}$$

viii) Decimal to Hexadecimal Conversion

Ex:  $2598.6751_{10}$

$$\begin{array}{r} 16 \overline{) 2598} \\ \underline{16 \mid 162} \quad -6 \\ 10 \quad \quad -2 \end{array}$$

$$= A26_{(16)}$$

$$0.675_{10} = 0.675 \times 16 \rightarrow 10.8$$

$$= 0.800 \times 16 \rightarrow 12.8 \quad \downarrow$$

$$= 0.800 \times 16 \rightarrow 12.8$$

$$= 0.800 \times 16 \rightarrow 12.8$$

$$= 0.ACCC_{16}$$

$$2598.675_{10} = A26.ACCC_{16}$$

ix) Octal to hexadecimal conversion:

The simplest way is to first convert the given octal no. to binary & then the binary no. to hexadecimal.

Ex:  $756.603_8$

7	5	6	.	6	0	3
111	101	110	.	110	000	011
0001	1110	1110	.	1100	0001	1000
1	E	E	.	C	1	8

x) Hexadecimal to octal conversion:

First convert the given hexadecimal no. to binary & then the binary no. to octal.

Ex:  $B9F.AE_{16}$

B	9	F	.	A	E		
1011	1001	1111	.	1010	1110		
101	110	011	111	.	101	011	100
5	6	3	7	.	5	3	4

$$= 5637.534$$

### Complements:

In digital computers to simplify the subtraction operation & for logical manipulation complements are used. There are two types of complements used in each radix system.

- The radix complement or  $r$ 's complement
- The diminished radix complement or  $(r-1)$ 's complement

### Representation of signed no.s binary arithmetic in computers:

- Two ways of rep signed no.s
  1. Sign Magnitude form
  2. Complement form
- Two complimented forms
  1. 1's complement form
  2. 2's complement form

Advantage of performing subtraction by the complement method is reduction in the hardware. (instead of addition & subtraction only adding ckt's are needed.)  
i.e, subtraction is also performed by adders only.

Instead of subtracting one no. from other the compliment of the subtrahend is added to minuend. In sign magnitude form, an additional bit called the sign bit is placed in front of the no. If the sign bit is 0, the no. is +ve, If it is a 1, the no is \_ve.

Ex:

0	1	0	1	0	0	1
---	---	---	---	---	---	---

Sign bit  = +41 magnitude

A horizontal register with 7 bits. Above the leftmost bit is a blue arrow pointing upwards, indicating an input. The bits from left to right are: 1, 1, 0, 1, 0, 0, 1.

$\equiv -41$

Note: manipulation is necessary to add a +ve no to a -ve no

### Representation of signed no.s using 2's or 1's complement method:

If the no. is +ve, the magnitude is rep in its true binary form & a sign bit 0 is placed in front of the MSB. If the no is \_ve , the magnitude is rep in its 2's or 1's compliment form & a sign bit 1 is placed in front of the MSB.

Ex:

Given no.	Sign mag form	2's comp form	1's comp form
01101	+13	+13	+13
010111	+23	+23	+23
10111	-7	-7	-8
1101010	-42	-22	-21



### Special case in 2's comp representation:

Whenever a signed no. has a 1 in the sign bit & all 0's for the magnitude bits, the decimal equivalent is  $-2^n$ , where n is the no of bits in the magnitude .

Ex: 1000= -8 & 10000=-16

### Characteristics of 2's compliment no.s:

Properties:

1. There is one unique zero
2. 2's comp of 0 is 0
3. The leftmost bit can't be used to express a quantity . it is a 0 no. is +ve.
4. For an n-bit word which includes the sign bit there are  $(2^{n-1}-1)$  +ve integers,  $2^{n-1}$  -ve integers & one 0 , for a total of  $2^n$  unique states.
5. Significant information is contained in the 1's of the +ve no.s & 0's of the -ve no.s
6. A -ve no. may be converted into a +ve no. by finding its 2's comp.

### Signed binary numbers:

Decimal	Sign 2's comp form	Sign 1's comp form	Sign mag form
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0011	0011	0011
+0	0000	0000	0000

-0	--	1111	1000
-1	1111	1110	1001
-2	1110	1101	1010
-3	1101	1100	1011
-4	1100	1011	1100
-5	1011	1010	1101
-6	1010	1001	1110
-7	1001	1000	1111
8	1000	--	--

## Methods of obtaining 2's comp of a no:

- In 3 ways
    1. By obtaining the 1's comp of the given no. (by changing all 0's to 1's & 1's to 0's) & then adding 1.
    2. By subtracting the given n bit no N from  $2^n$
    3. Starting at the LSB , copying down each bit upto & including the first 1 bit encountered , and complimenting the remaining bits.
- Ex: Express -45 in 8 bit 2's comp form

+45 in 8 bit form is 00101101

### I method:

1's comp of 00101101 & the add 1

00101101

11010010

+1

— — — — — — — — — —

11010011

is 2's comp form

### II method:

Subtract the given no. N from  $2^n$

$2^n = 100000000$

Subtract 45 = -00101101

+1

— — — —

11010011

is 2's comp

### III method:

Original no: 00101101

Copy up to First 1 bit 1

Compliment remaining : 1101001

—————

bits

11010011

Ex:

-73.75 in 12 bit 2's comp form

I method

$$\begin{array}{r} 01001001.1100 \\ 10110110.0011 \\ +1 \\ \hline \end{array}$$

10110110.0100 is 2's

II method:

$$2^8 = 100000000.0000$$

$$\text{Sub } 73.75 = -01001001.1100$$

10110110.0100 is 2's comp

III method :

Original no : 01001001.1100

Copy up to 1'st bit 100

Comp the remaining bits: 10110110.0

10110110.0100

### 2's compliment Arithmetic:

- The 2's comp system is used to rep -ve no.s using modulus arithmetic . The word length of a computer is fixed. i.e, if a 4 bit no. is added to another 4 bit no . the result will be only of 4 bits. Carry if any , from the fourth bit will overflow called the Modulus arithmetic.

$$\text{Ex: } 1100 + 1111 = 1011$$

- In the 2's compl subtraction, add the 2's comp of the subtrahend to the minuend . If there is a carry out , ignore it , look at the sign bit I.e, MSB of the sum term .If the MSB is a 0, the result is positive.& it is in true binary form. If the MSB is a 1 ( carry in or no carry at all) the result is negative.& is in its 2's comp form. Take its 2's comp to find its magnitude in binary.

**Ex:** Subtract 14 from 46 using 8 bit 2's comp arithmetic:

$$\begin{array}{rcl} +14 & = & 00001110 \\ -14 & = & 11110010 \quad \text{2's comp} \\ +46 & = & 00101110 \\ -14 & = & +11110010 \quad \text{2's comp form of -14} \\ \hline \end{array}$$

$$\begin{array}{r} \text{---} \\ -32 \quad (1)00100000 \end{array} \quad \text{ignore carry}$$

Ignore carry, The MSB is 0. so the result is +ve. & is in normal binary form. So the result is +00100000=+32.

**EX:** Add -75 to +26 using 8 bit 2's comp arithmetic

$$\begin{array}{r} +75 = 01001011 \\ -75 = 10110101 \quad \text{2's comp} \\ \hline +26 = 00011010 \\ -75 = +10110101 \quad \text{2's comp form of -75} \\ \hline -49 \quad 11001111 \quad \text{No carry} \end{array}$$

No carry, MSB is a 1, result is -ve & is in 2's comp. The magnitude is 2's comp of 11001111. i.e, 00110001 = 49. so result is -49

**Ex:** add -45.75 to +87.5 using 12 bit arithmetic

$$\begin{array}{r} +87.5 = 01010111.1000 \\ -45.75 = +11010010.0100 \end{array}$$

$$\begin{array}{r} \text{---} \\ -41.75 \quad (1)00101001.1100 \text{ ignore carry} \\ \text{MSB is 0, result is +ve. } = +41.75 \end{array}$$

### 1's compliment of n number:

- It is obtained by simply complimenting each bit of the no., & also, 1's comp of a no, is subtracting each bit of the no. from 1. This complemented value rep the -ve of the original no. One of the difficulties of using 1's comp is its rep of zero. Both 00000000 & its 1's comp 11111111 rep zero.
- The 00000000 called +ve zero & 11111111 called -ve zero.

Ex: -99 & -77.25 in 8 bit 1's comp

$$\begin{array}{r} +99 = 01100011 \\ -99 = 10011100 \end{array}$$

$$\begin{array}{r} +77.25 = 01001101.0100 \\ -77.25 = 10110010.1011 \end{array}$$

### 1's compliment arithmetic:

In 1's comp subtraction, add the 1's comp of the subtrahend to the minuend. If there is a carryout, bring the carry around & add it to the LSB called the **end around carry**. Look at the sign bit (MSB). If this is a 0, the result is +ve & is in true binary. If the MSB is a 1 (carry or no carry), the result is -ve & is in its 1's comp form. Take its 1's comp to get the magnitude inn

Ex: Subtract 14 from 25 using 8 bit 1's EX: ADD -25 to +14

$$\begin{array}{rclcl}
 25 & = & 00011001 & +14 & = 00001110 \\
 -45 & = & 11110001 & -25 & = +11100110 \\
 \hline
 +11 & & (1)00001010 & -11 & 11110100 \\
 \hline
 \end{array}$$

+1









$$\begin{array}{rcl}
 & & \hline
 & & 00001011 \\
 \hline
 \end{array}
 \quad \begin{array}{l}
 \text{No carry} \quad \text{MSB} = 1 \\
 \text{result} = -ve = -11_{10}
 \end{array}$$

MSB is a 0 so result is +ve (binary )

$$= +11_{10}$$

## Digital Logic Gates

Boolean functions are expressed in terms of AND, OR, and NOT operations, it is easier to implement a Boolean function with these type of gates.

Name	Graphic symbol	Algebraic function	Truth table															
AND		$F = x \cdot y$	<table><tr><th>x</th><th>y</th><th>F</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>	x	y	F	0	0	0	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = x + y$	<table><tr><th>x</th><th>y</th><th>F</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>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	1
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
Inverter		$F = x'$	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	x	F	0	1	1	0									
x	F																	
0	1																	
1	0																	
Buffer		$F = x$	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	x	F	0	0	1	1									
x	F																	
0	0																	
1	1																	
NAND		$F = (xy)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</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>	x	y	F	0	0	1	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = (x + y)'$	<table><tr><th>x</th><th>y</th><th>F</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>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	0
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	<table><tr><th>x</th><th>y</th><th>F</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>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	<table><tr><th>x</th><th>y</th><th>F</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>1</td></tr></table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

## Properties of XOR Gates

- XOR (also  $\oplus$ ) : the “not-equal” function
- $\text{XOR}(X,Y) = X \oplus Y = X'Y + XY'$
- Identities:
  - $X \oplus 0 = X$
  - $X \oplus 1 = X'$
  - $X \oplus X = 0$
  - $X \oplus X' = 1$
- Properties:
  - $X \oplus Y = Y \oplus X$
  - $(X \oplus Y) \oplus W = X \oplus (Y \oplus W)$

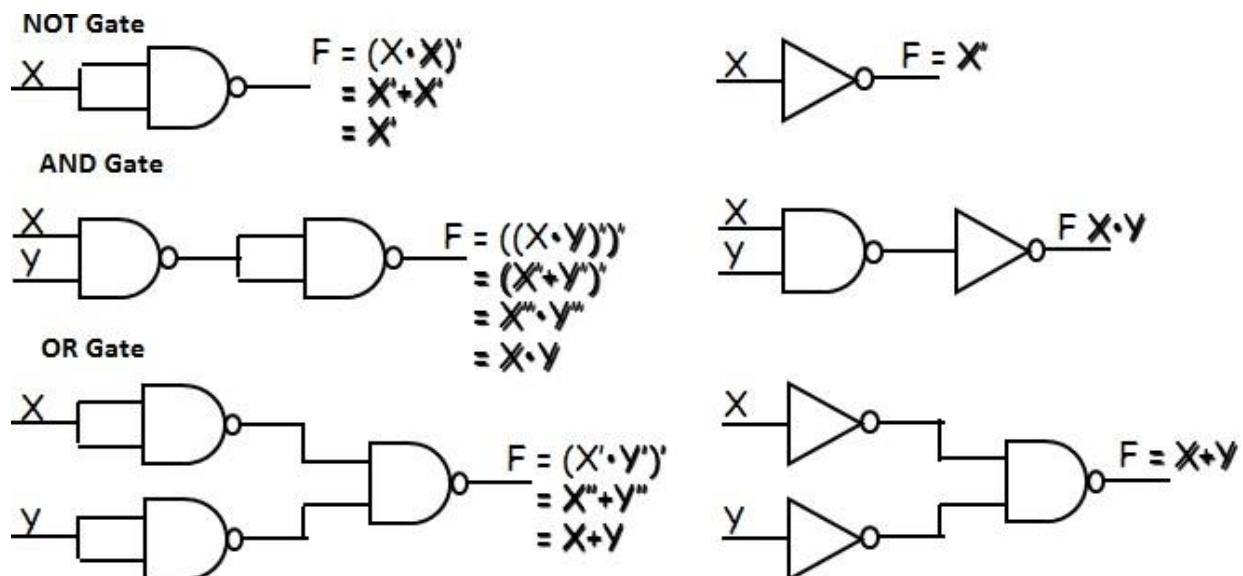
## Universal Logic Gates

NAND and NOR gates are called Universal gates. All fundamental gates (NOT, AND, OR) can be realized by using either only NAND or only NOR gate. A universal gate provides flexibility and offers enormous advantage to logic designers.

### NAND as a Universal Gate

NAND Known as a “universal” gate because ANY digital circuit can be implemented with NAND gates alone.

To prove the above, it suffices to show that AND, OR, and NOT can be implemented using NAND gates only.



**Boolean Algebra:** In 1854, George Boole developed an algebraic system now called Boolean algebra. In 1938, Claude E. Shannon introduced a two-valued Boolean algebra called switching algebra that represented the properties of bistable electrical switching circuits. For the formal definition of Boolean algebra, we shall employ the postulates formulated by E. V. Huntington in 1904.

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

Boolean algebra, like any other deductive mathematical system, may be defined with a set of elements, a set of operators, and a number of unproved axioms or postulates. A *set* of elements is any collection of objects having a common property. If  $S$  is a set and  $x$  and  $y$  are certain objects, then  $x \in S$  denotes that  $x$  is a member of the set  $S$ , and  $y \notin S$  denotes that  $y$  is not an element of  $S$ . A set with a denumerable number of elements is specified by braces:  $A = \{1, 2, 3, 4\}$ , *i.e.* the elements of set  $A$  are the numbers 1, 2, 3, and 4. A *binary operator* defined on a set  $S$  of elements is a rule that assigns to each pair of elements from  $S$  a unique element from  $S$ . Example: In  $a * b = c$ , we say that  $*$  is a binary operator if it specifies a rule for finding  $c$  from the pair  $(a, b)$  and also if  $a, b, c \in S$ .

### Axioms and laws of Boolean algebra

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

	AND Operation	OR Operation	NOT Operation
Axiom1 :	$0.0=0$	$0+0=0$	$\overline{0}=1$
Axiom2:	$0.1=0$	$0+1=1$	$\overline{1}=0$
Axiom3:	$1.0=0$	$1+0=1$	
Axiom4:	$1.1=1$	$1+1=1$	

#### AND Law

Law1:  $A.0=0$  (Null law)  
 Law2:  $A.1=A$  (Identity law)  
 Law3:  $A.A=A$  (Idempotence law)

#### OR Law

Law1:  $A+0=A$   
 Law2:  $A+1=1$   
 Law3:  $A+A=A$  (Idempotence law)

**CLOSURE:** The Boolean system is *closed* with respect to a binary operator if for every pair of Boolean values, it produces a Boolean result. For example, logical AND is closed in the Boolean system because it accepts only Boolean operands and produces only Boolean results.

\_ A set  $S$  is closed with respect to a binary operator if, for every pair of elements of  $S$ , the binary operator specifies a rule for obtaining a unique element of  $S$ .

\_ For example, the set of natural numbers  $N = \{1, 2, 3, 4, \dots, 9\}$  is closed with respect to the binary operator plus (+) by the rule of arithmetic addition, since for any  $a, b \in N$  we obtain a unique  $c \in N$  by the operation  $a + b = c$ .



**ASSOCIATIVE LAW:**

A binary operator  $*$  on a set  $S$  is said to be associative whenever  $(x * y) * z = x * (y * z)$  for all  $x, y, z \in S$ , for all Boolean values  $x, y$  and  $z$ .

**COMMUTATIVE LAW:**

A binary operator  $*$  on a set  $S$  is said to be commutative whenever  $x * y = y * x$  for all  $x, y, z \in S$

**IDENTITY ELEMENT:**

A set  $S$  is said to have an identity element with respect to a binary operation  $*$  on  $S$  if there exists an element  $e \in S$  with the property  $e * x = x * e = x$  for every  $x \in S$

**BASIC IDENTITIES OF BOOLEAN ALGEBRA**

- *Postulate 1(Definition):* A Boolean algebra is a closed algebraic system containing a set  $K$  of two or more elements and the two operators  $\cdot$  and  $+$  which refer to logical AND and logical OR •  $x + 0 = x$
- $x \cdot 0 = 0$
- $x + 1 = 1$
- $x \cdot 1 = x$
- $x + x = x$
- $x \cdot x = x$
- $x + x' = 1$
- $x \cdot x' = 0$
- $x + y = y + x$
- $xy = yx$
- $x + (y + z) = (x + y) + z$
- $x(yz) = (xy)z$
- $x(y + z) = xy + xz$
- $x + yz = (x + y)(x + z)$
- $(x + y)' = x'y'$
- $(xy)' = x' + y'$

- $(x')' = x$

### DeMorgan's Theorem

**(a)**  $(a + b)' = a'b'$

**(b)**  $(ab)' = a' + b'$

### Generalized DeMorgan's Theorem

(a)  $(a + b + \dots z)' = a'b' \dots z'$

(b)  $(a.b \dots z)' = a' + b' + \dots z'$

### Basic Theorems and Properties of Boolean algebra Commutative law

Law1:  $A+B=B+A$

Law2:  $A.B=B.A$

### Associative law

Law1:  $A + (B + C) = (A + B) + C$

Law2:  $A(B.C) = (A.B)C$

### Distributive law

Law1:  $A.(B + C) = AB + AC$

Law2:  $A + BC = (A + B).(A + C)$

### Absorption law

Law1:  $A + AB = A$

Law2:  $A(A + B) = A$

Solution:  $\frac{A(1+B)}{A} =$

Solution:  $\frac{A.A+A.B}{A(1+B)} = A$

### Consensus Theorem

Theorem1.  $AB + A'C + BC = AB + A'C$  Theorem2.  $(A+B).(A'+C).(B+C) = (A+B).(A'+C)$

The BC term is called the consensus term and is redundant. The consensus term is formed from a PAIR OF TERMS in which a variable (A) and its complement (A') are present; the consensus term is formed by multiplying the two terms and leaving out the selected variable and its complement

Consensus Theorem1 Proof:

$$\begin{aligned} AB + A'C + BC &= AB + A'C + (A + A')BC \\ &= AB + A'C + ABC + A'BC \end{aligned}$$

$$=AB(1+C)+A'C(1+B)$$

$$= AB+ A'C$$

## Principle of Duality

Each postulate consists of two expressions statement one expression is transformed into the other by interchanging the operations (+) and (·) as well as the identity elements 0 and 1. Such expressions are known as duals of each other.

If some equivalence is proved, then its dual is also immediately true.

If we prove:  $(x.x)+(x'+x')=1$ , then we have by duality:  $(x+x)·(x'·x')=0$

The Huntington postulates were listed in pairs and designated by part (a) and part (b) in below table.

**Table for Postulates and Theorems of Boolean algebra**

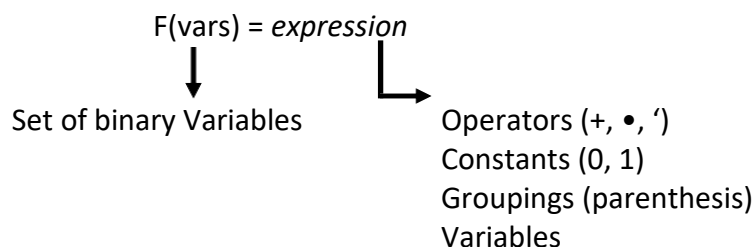
Part-A	Part-B
$A+0=A$	$A·0=0$
$A+1=1$	$A·1=A$
$A+A=A$ (Impotence law)	$A·A=A$ (Impotence law)
$A+\bar{A}1$	$A·\bar{A}0$
$\bar{\bar{A}}A$ (double inversion law)	--
<b>Commutative law:</b> $A+B=B+A$	$A·B=B·A$
<b>Associative law:</b> $A + (B + C) = (A + B) + C$	$A(B·C) = (A·B)C$
<b>Distributive law:</b> $A·(B + C) = AB+ AC$	$A + BC = (A + B)·(A + C)$
<b>Absorption law:</b> $A +AB =A$	$A(A +B) = A$
<b>DeMorgan Theorem:</b> $\overline{(A+B)} = \bar{A}·\bar{B}$	$\overline{(A·B)} = \bar{A} + \bar{B}$
<b>Redundant Literal Rule:</b> $A+ \bar{A}B=A+B$	$A·(\bar{A}+B)=AB$
<b>Consensus Theorem:</b> $AB+ A'C + BC = AB + A'C$	$(A+B)·(A'+C)·(B+C)=(A+B)·(A'+C)$

## Boolean Function

Boolean algebra is an algebra that deals with binary variables and logic operations.

A Boolean function described by an algebraic expression consists of binary variables, the constants 0 and 1, and the logic operation symbols.

For a given value of the binary variables, the function can be equal to either 1 or 0.



Consider an example for the Boolean function

$$F1 = x + y'z$$

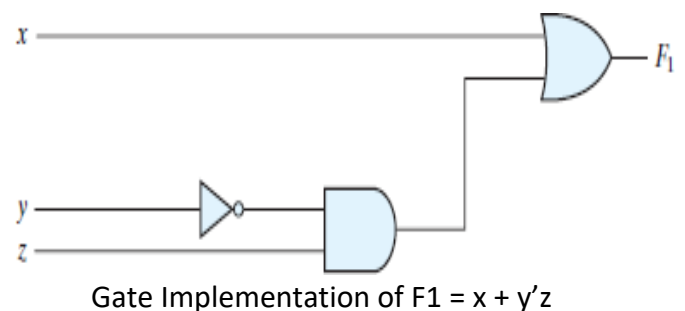
The function  $F_1$  is equal to 1 if  $x$  is equal to 1 or if both  $y'$  and  $z$  are equal to 1.  $F_1$  is equal to 0 otherwise. The complement operation dictates that when  $y' = 1$ ,  $y = 0$ . Therefore,  $F_1 = 1$  if  $x = 1$  or if  $y = 0$  and  $z = 1$ .

A Boolean function expresses the logical relationship between binary variables and is evaluated by determining the binary value of the expression for all possible values of the variables.

A Boolean function can be represented in a truth table. The number of rows in the truth table is  $2^n$ , where  $n$  is the number of variables in the function. The binary combinations for the truth table are obtained from the binary numbers by counting from 0 through  $2^n - 1$ .

Truth Table for  $F_1$

x	y	z	$F_1$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



**Note:**

Q: Let a function  $F()$  depend on  $n$  variables. How many rows are there in the truth table of  $F()$  ?

A:  $2^n$  rows, since there are  $2^n$  possible binary patterns/combinations for the  $n$  variables.

**Truth Tables**

- Enumerates all possible combinations of variable values and the corresponding function value
- Truth tables for some arbitrary functions  
 $F_1(x,y,z)$ ,  $F_2(x,y,z)$ , and  $F_3(x,y,z)$  are shown to the below.

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

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

- Truth table: a unique representation of a Boolean function
- If two functions have identical truth tables, the functions are equivalent (and vice-versa).
- Truth tables can be used to prove equality theorems.
- However, the size of a truth table grows exponentially with the number of variables involved, hence unwieldy. This motivates the use of Boolean Algebra.

### Boolean expressions-NOT unique

Unlike truth tables, expressions representing a Boolean function are NOT unique.

- Example:
  - $F(x,y,z) = x' \bullet y' \bullet z' + x' \bullet y \bullet z' + x \bullet y \bullet z'$
  - $G(x,y,z) = x' \bullet y' \bullet z' + y \bullet z'$
- The corresponding truth tables for F() and G() are to the right. They are identical.
- Thus,  $F() = G()$

x	y	z	F	G
0	0	0	1	1
0	0	1	0	0
0	1	0	1	1
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	1
1	1	1	0	0

### Algebraic Manipulation (Minimization of Boolean function)

- Boolean algebra is a useful tool for simplifying digital circuits.
- Why do it? Simpler can mean cheaper, smaller, faster.
- Example: Simplify  $F = x'yz + x'yz' + xz$ .
 
$$\begin{aligned}
 F &= x'yz + x'yz' + xz \\
 &= x'y(z+z') + xz \\
 &= x'y \bullet 1 + xz
 \end{aligned}$$

$$= x'y + xz$$

- Example: Prove

$$x'y'z' + x'yz' + xyz' = x'z' + yz'$$

- **Proof:**

$$\begin{aligned} x'y'z' + x'yz' + xyz' \\ &= x'y'z' + x'yz' + x'yz' + xyz' \\ &= x'z'(y' + y) + yz'(x' + x) \\ &= x'z' \cdot 1 + yz' \cdot 1 \\ &= x'z' + yz' \end{aligned}$$

### Complement of a Function

- The complement of a function is derived by interchanging ( $\cdot$  and  $+$ ), and (1 and 0), and complementing each variable.
- Otherwise, interchange 1s to 0s in the truth table column showing F.
- The *complement* of a function IS NOT THE SAME as the *dual* of a function.

Example

- Find  $G(x,y,z)$ , the complement of  $F(x,y,z) = xy'z' + x'yz$

$$\begin{aligned} \text{Ans: } G = F' &= (xy'z' + x'yz)' \\ &= (xy'z')' \cdot (x'yz)' && \text{DeMorgan} \\ &= (x' + y + z) \cdot (x + y' + z') && \text{DeMorgan again} \end{aligned}$$

**Note:** The complement of a function can also be derived by finding the function's *dual*, and then complementing all of the literals

### Canonical and Standard Forms

We need to consider formal techniques for the simplification of Boolean functions.

Identical functions will have exactly the same canonical form.

- Minterms and Maxterms
- Sum-of-Minterms and Product-of- Maxterms
- Product and Sum terms
- Sum-of-Products (SOP) and Product-of-Sums (POS)

### Definitions

**Literal:** A variable or its complement

**Product term:** literals connected by  $\cdot$

**Sum term:** literals connected by  $+$

**Minterm:** a product term in which all the variables appear exactly once, either complemented or uncomplemented.

**Maxterm:** a sum term in which all the variables appear exactly once, either complemented or uncomplemented.

**Canonical form:** Boolean functions expressed as a sum of Minterms or product of Maxterms are said to be in canonical form.

### Minterm

- Represents exactly one combination in the truth table.
- Denoted by  $m_j$ , where  $j$  is the decimal equivalent of the minterm's corresponding binary combination ( $b_j$ ).
- A variable in  $m_j$  is complemented if its value in  $b_j$  is 0, otherwise is uncomplemented.

Example: Assume 3 variables (A, B, C), and  $j=3$ . Then,  $b_j = 011$  and its corresponding minterm is denoted by  $m_j = A'BC$

### Maxterm

- Represents exactly one combination in the truth table.
- Denoted by  $M_j$ , where  $j$  is the decimal equivalent of the maxterm's corresponding binary combination ( $b_j$ ).
- A variable in  $M_j$  is complemented if its value in  $b_j$  is 1, otherwise is uncomplemented.

Example: Assume 3 variables (A, B, C), and  $j=3$ . Then,  $b_j = 011$  and its corresponding maxterm is denoted by  $M_j = A+B'+C'$

### Truth Table notation for Minterms and Maxterms

- Minterms and Maxterms are easy to denote using a truth table.

Example: Assume 3 variables  $x, y, z$  (order is fixed)

x	y	z	Minterm	Maxterm
0	0	0	$x'y'z' = m_0$	$x+y+z = M_0$
0	0	1	$x'y'z = m_1$	$x+y+z' = M_1$
0	1	0	$x'yz' = m_2$	$x+y'+z = M_2$
0	1	1	$x'yz = m_3$	$x+y'+z' = M_3$
1	0	0	$xy'z' = m_4$	$x'+y+z = M_4$
1	0	1	$xy'z = m_5$	$x'+y+z' = M_5$
1	1	0	$xyz' = m_6$	$x'+y'+z = M_6$
1	1	1	$xyz = m_7$	$x'+y'+z' = M_7$

### Canonical Forms

- Every function  $F()$  has two canonical forms:
  - Canonical Sum-Of-Products (sum of minterms)
  - Canonical Product-Of-Sums (product of maxterms)

Canonical Sum-Of-Products:

The minterms included are those  $m_j$  such that  $F() = 1$  in row  $j$  of the truth table for  $F()$ .

Canonical Product-Of-Sums:

The maxterms included are those  $M_j$  such that  $F() = 0$  in row  $j$  of the truth table for  $F()$ .

### Example

Consider a Truth table for  $f_1(a,b,c)$  at right

The canonical sum-of-products form for  $f_1$  is

$$f_1(a,b,c) = m_1 + m_2 + m_4 + m_6$$

$$= a'b'c + a'bc' + ab'c' + abc'$$

The canonical product-of-sums form for  $f_1$  is

$$f_1(a,b,c) = M_0 \cdot M_3 \cdot M_5 \cdot M_7$$

$$= (a+b+c) \cdot (a+b'+c') \cdot (a'+b+c') \cdot (a'+b'+c').$$

- Observe that:  $m_j = M_j'$

a	b	c	$f_1$
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	1
1	1	1	0



## Shorthand: $\Sigma$ and $\prod$

- $f_1(a,b,c) = \Sigma m(1,2,4,6)$ , where  $\Sigma$  indicates that this is a sum-of-products form, and  $m(1,2,4,6)$  indicates that the minterms to be included are  $m_1$ ,  $m_2$ ,  $m_4$ , and  $m_6$ .
- $f_1(a,b,c) = \prod M(0,3,5,7)$ , where  $\prod$  indicates that this is a product-of-sums form, and  $M(0,3,5,7)$  indicates that the maxterms to be included are  $M_0$ ,  $M_3$ ,  $M_5$ , and  $M_7$ .
- Since  $m_j = M_j'$  for any  $j$ ,  
 $\Sigma m(1,2,4,6) = \prod M(0,3,5,7) = f_1(a,b,c)$
- 

## Conversion between Canonical Forms

- Replace  $\Sigma$  with  $\prod$  (or *vice versa*) and replace those  $j$ 's that appeared in the original form with those that do not.

- Example:

$$\begin{aligned} f_1(a,b,c) &= a'b'c + a'bc' + ab'c' + abc' \\ &= m_1 + m_2 + m_4 + m_6 \\ &= \Sigma(1,2,4,6) \\ &= \prod(0,3,5,7) \\ &= (a+b+c) \cdot (a+b'+c') \cdot (a'+b+c') \cdot (a'+b'+c') \end{aligned}$$

## Standard Forms

Another way to express Boolean functions is in standard form. In this configuration, the terms that form the function may contain one, two, or any number of literals.

There are two types of standard forms: the sum of products and products of sums.

The sum of products is a Boolean expression containing AND terms, called product terms, with one or more literals each. The sum denotes the ORing of these terms. An example of a function expressed as a sum of products is

$$F1 = y' + xy + x'yz'$$

The expression has three product terms, with one, two, and three literals. Their sum is, in effect, an OR operation.

A product of sums is a Boolean expression containing OR terms, called sum terms. Each term may have any number of literals. The product denotes the ANDing of these terms. An example of a function expressed as a product of sums is

$$F2 = x(y' + z)(x' + y + z')$$

This expression has three sum terms, with one, two, and three literals. The product is an AND operation.

## Conversion of SOP from standard to canonical form

### Example-1.

Express the Boolean function  $F = A + B'C$  as a sum of minterms.

Solution: The function has three variables: A, B, and C. The first term A is missing two variables; therefore,

$$A = A(B + B') = AB + AB'$$

This function is still missing one variable, so

$$\begin{aligned} A &= AB(C + C') + AB'(C + C') \\ &= ABC + ABC' + AB'C + AB'C' \end{aligned}$$

The second term  $B'C$  is missing one variable; hence,

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

Combining all terms, we have

$$\begin{aligned} F &= A + B'C \\ &= ABC + ABC' + AB'C + AB'C' + A'B'C \end{aligned}$$

But  $AB'C$  appears twice, and according to theorem  $(x + x = x)$ , it is possible to remove one of those occurrences. Rearranging the minterms in ascending order, we finally obtain

$$\begin{aligned} F &= A'B'C + AB'C + AB'C' + ABC' + ABC \\ &= m_1 + m_4 + m_5 + m_6 + m_7 \end{aligned}$$

When a Boolean function is in its sum-of-minterms form, it is sometimes convenient to express the function in the following brief notation:

$$F(A, B, C) = \sum m(1, 4, 5, 6, 7)$$

### Example-2.

Express the Boolean function  $F = xy + x'z$  as a product of maxterms.

Solution: First, convert the function into OR terms by using the distributive law:

$$\begin{aligned} F &= xy + x'z = (xy + x')(xy + z) \\ &= (x + x')(y + x')(x + z)(y + z) \\ &= (x' + y)(x + z)(y + z) \end{aligned}$$

The function has three variables: x, y, and z. Each OR term is missing one variable; therefore,

$$\begin{aligned} x' + y &= x' + y + zz' = (x' + y + z)(x' + y + z') \\ x + z &= x + z + yy' = (x + y + z)(x + y' + z) \\ y + z &= y + z + xx' = (x + y + z)(x' + y + z) \end{aligned}$$

Combining all the terms and removing those which appear more than once, we finally obtain

$$\begin{aligned} F &= (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z) \\ F &= M_0 M_2 M_4 M_5 \end{aligned}$$

A convenient way to express this function is as

$$\text{follows: } F(x, y, z) = \pi M(0, 2, 4, 5)$$

The product symbol,  $\pi$ , denotes the ANDing of maxterms; the numbers are the indices of the maxterms of the function.

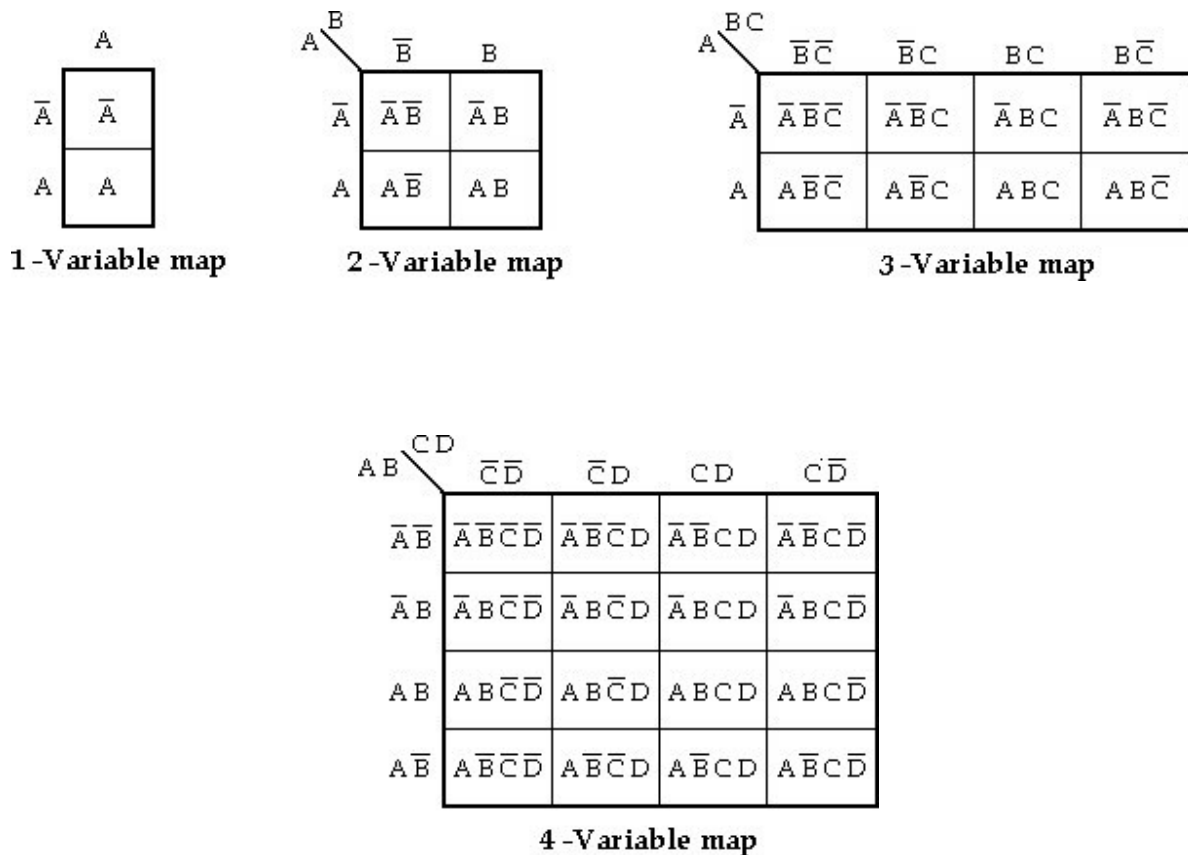
## **KARNAUGH MAP MINIMIZATION:**

The simplification of the functions using Boolean laws and theorems becomes complex with the increase in the number of variables and terms. The map method, first proposed by Veitch and slightly improvised by Karnaugh, provides a simple, straightforward procedure for the simplification of Boolean functions. The method is called **Veitch diagram** or **Karnaugh map**, which may be regarded as a pictorial representation of a truth table.

The Karnaugh map technique provides a systematic method for simplifying and manipulation of Boolean expressions. A K-map is a diagram made up of squares, with each square representing one minterm of the function that is to be minimized. For  $n$  variables on a Karnaugh map there are  $2^n$  numbers of squares. Each square or cell represents one of the minterms. It can be drawn directly from either minterm (sum-of-products) or maxterm (product-of-sums) Boolean expressions.

### **Two- Variable, Three Variable and Four Variable Maps**

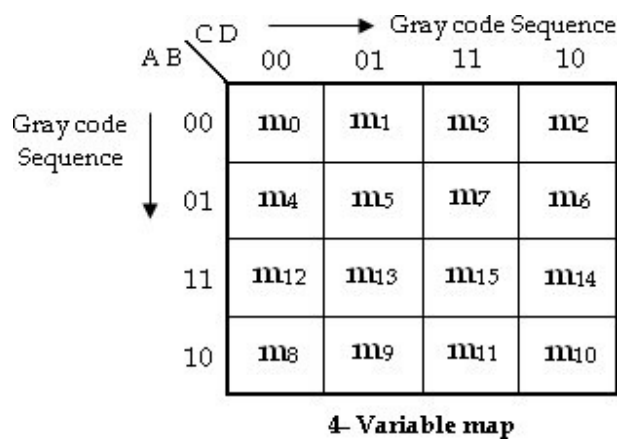
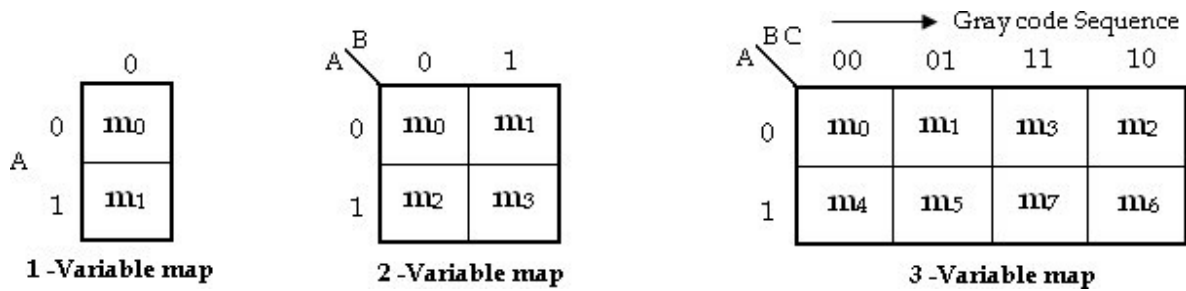
Karnaugh maps can be used for expressions with two, three, four and five variables. The number of cells in a Karnaugh map is equal to the total number of possible input variable combinations as is the number of rows in a truth table. For three variables, the number of cells is  $2^3 = 8$ . For four variables, the number of cells is  $2^4 = 16$ .



Product terms are assigned to the cells of a K-map by labeling each row and each column of a map with a variable, with its complement or with a combination of variables & complements. The below figure shows the way to label the rows & columns of a 1, 2, 3 and 4- variable maps and the product terms corresponding to each cell.

It is important to note that when we move from one cell to the next along any row or from one cell to the next along any column, one and only one variable in the product term changes (to a complement or to an uncomplemented form). Irrespective of number of variables the labels along each row and column must

conform to a single change. Hence gray code is used to label the rows and columns of K-map as shown ow.

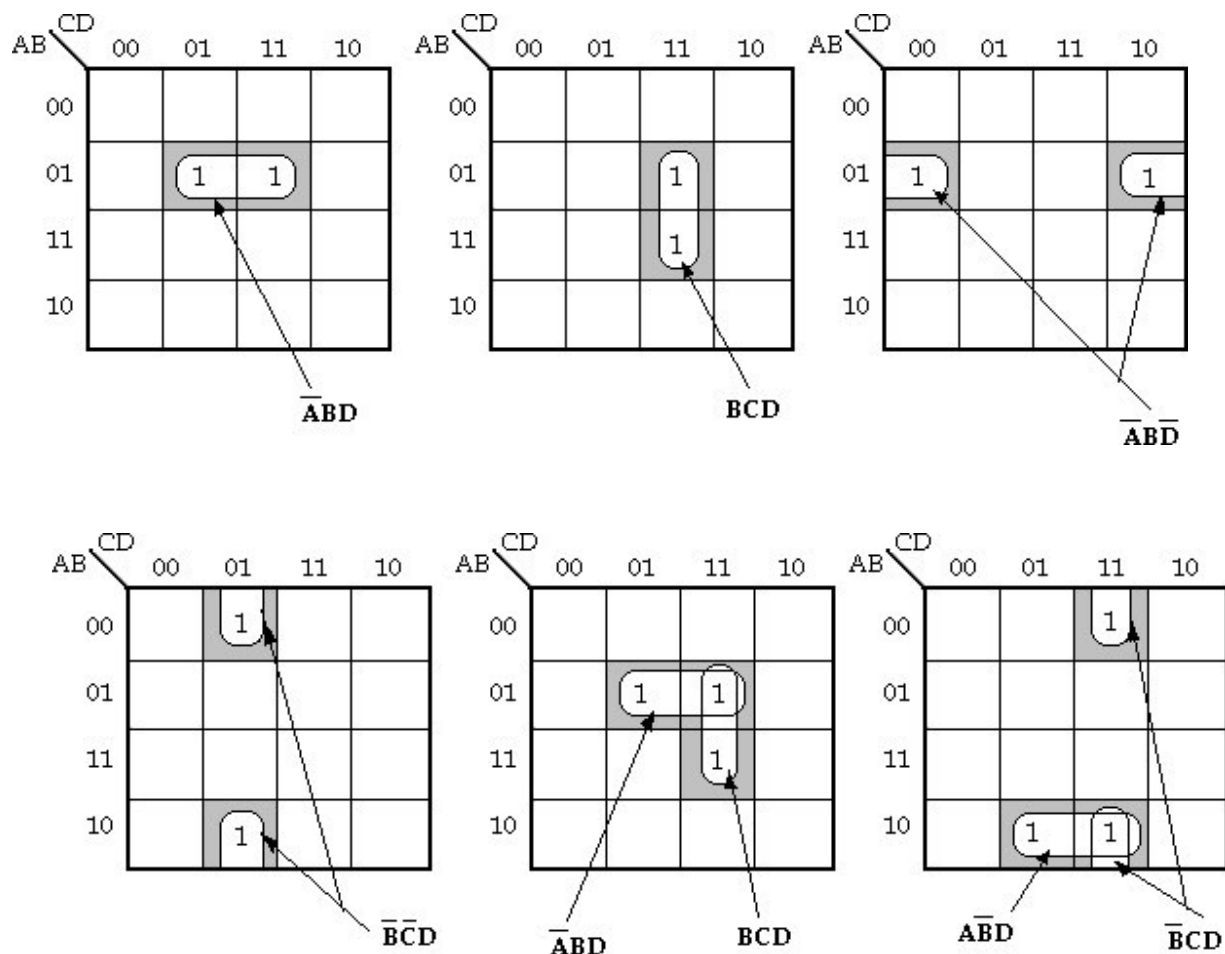


### Grouping cells for Simplification:

The grouping is nothing but combining terms in adjacent cells. The simplification is achieved by grouping adjacent 1's or 0's in groups of  $2^i$ , where  $i = 1, 2, \dots, n$  and  $n$  is the number of variables. When adjacent 1's are grouped then we get result in the sum of product form; otherwise we get result in the product of sum form.

### Grouping Two Adjacent 1's: (Pair)

In a Karnaugh map we can group two adjacent 1's. The resultant group is called Pair.

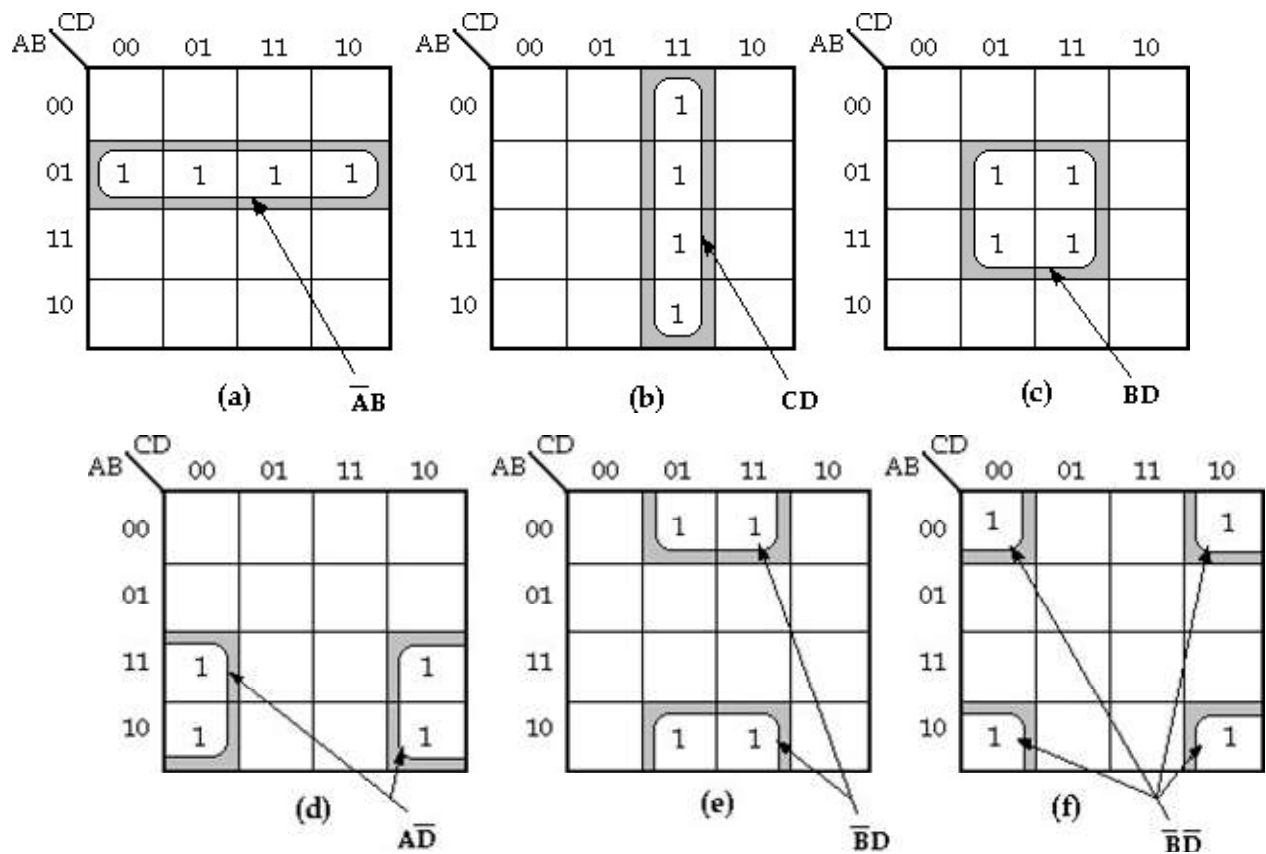


### Examples of Pairs

### Grouping Four Adjacent 1's: (Quad)

In a Karnaugh map we can group four adjacent 1's. The resultant group is called Quad. Fig (a) shows the four 1's are horizontally adjacent and Fig (b) shows

they are vertically adjacent. Fig (c) contains four 1's in a square, and they are considered adjacent to each other.

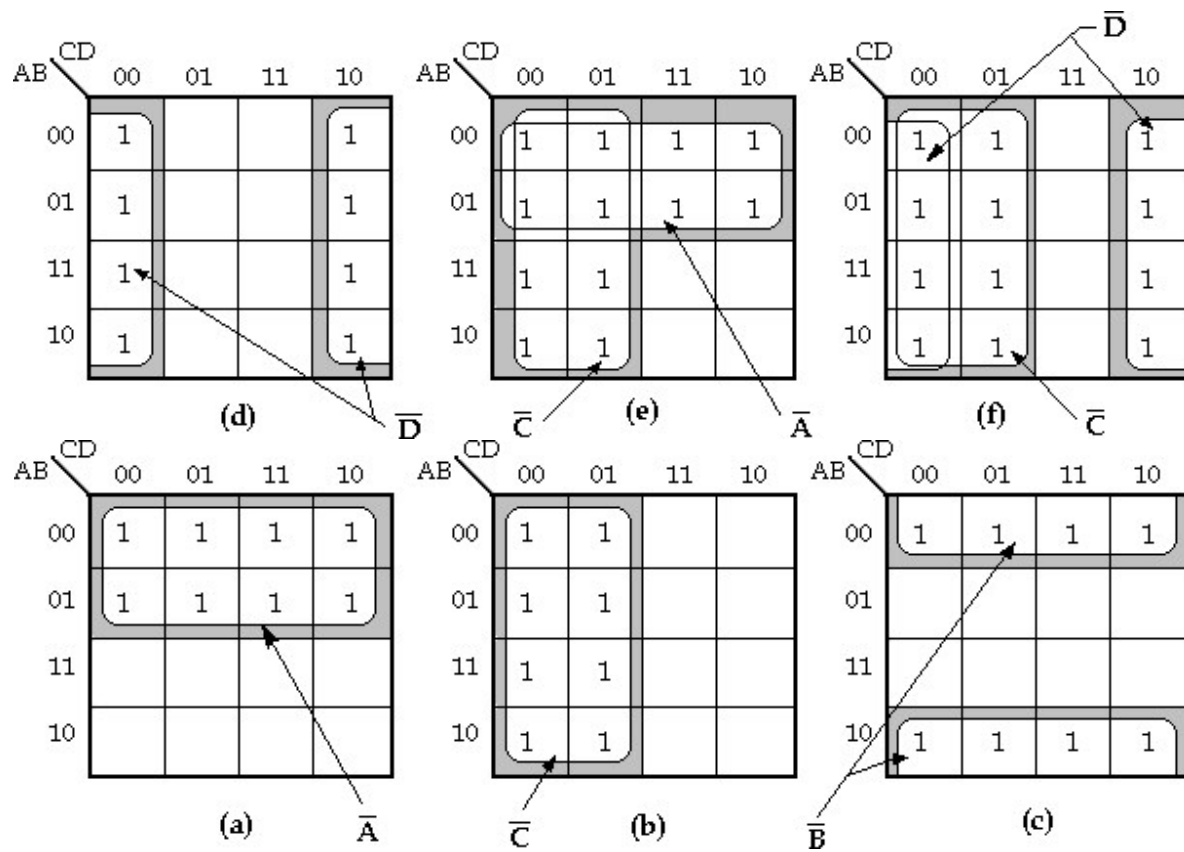


Examples of Quads

The four 1's in fig (d) and fig (e) are also adjacent, as are those in fig (f) because, the top and bottom rows are considered to be adjacent to each other and the leftmost and rightmost columns are also adjacent to each other.

### Grouping Eight Adjacent 1's: (Octet)

In a Karnaugh map we can group eight adjacent 1's. The resultant group is called Octet.



### Simplification of Sum of Products Expressions: (Minimal Sums)

The generalized procedure to simplify Boolean expressions as follows:

1. Plot the K-map and place 1's in those cells corresponding to the 1's in the sum of product expression. Place 0's in the other cells.
2. Check the K-map for adjacent 1's and encircle those 1's which are not adjacent to any other 1's. These are called **isolated 1's**.
3. Check for those 1's which are adjacent to only one other 1 and encircle such **pairs**.
4. Check for **quads** and **octets** of adjacent 1's even if it contains some 1's that have already been encircled. While doing this make sure that there are minimum number of groups.



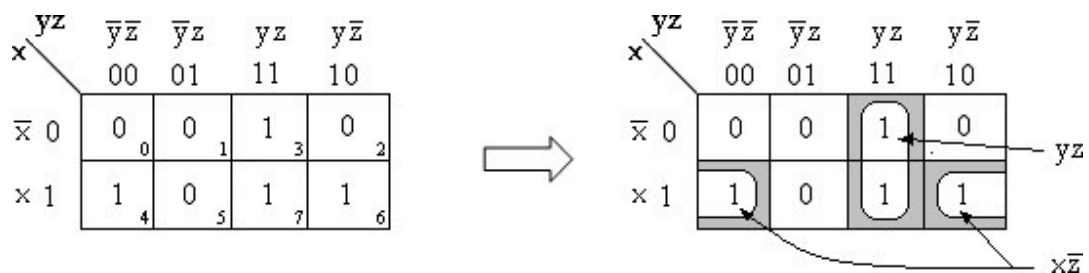
5. Combine any pairs necessary to include any 1's that have not yet been grouped.
6. Form the simplified expression by summing product terms of all the groups.

### Three- Variable Map:

1. Simplify the Boolean expression,

$$F(x, y, z) = \sum m(3, 4, 6, 7).$$

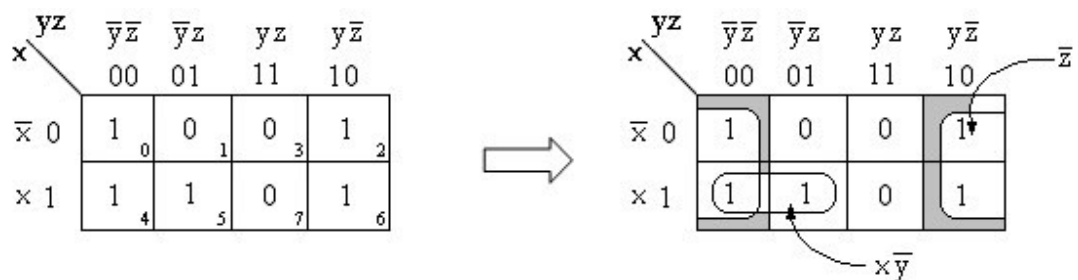
Soln:



$$F = yz + x\bar{z}$$

2.  $F(x, y, z) = \sum m(0, 2, 4, 5, 6).$

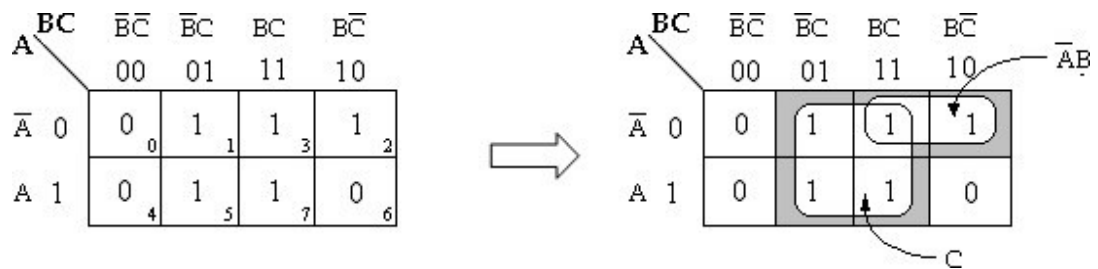
Soln:



$$F = z\bar{y} + x\bar{y}$$

$$3. F = A'C + A'B + AB'C + BC \quad \text{Soln:}$$

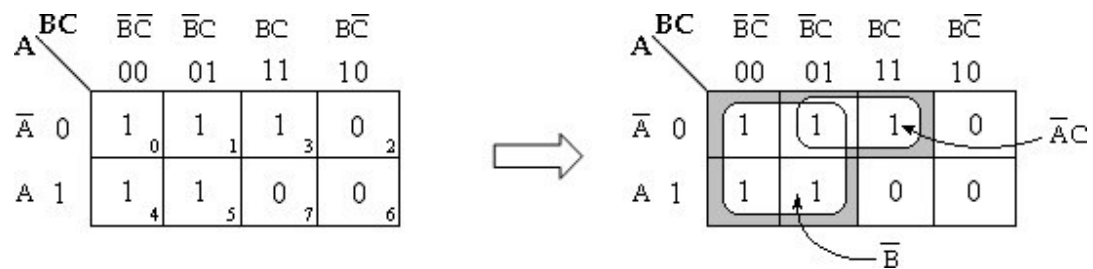
$$\begin{aligned}
&= A'C (B + B') + A'B (C + C') + AB'C + BC (A + A') \\
&= \underline{A'BC} + A'B'C + \underline{A'BC} + A'BC' + AB'C + ABC + \underline{A'BC} \\
&= A'BC + A'B'C + A'BC' + AB'C + ABC \\
&= m_3 + m_1 + m_2 + m_5 + m_7 \\
&= \sum m (1, 2, 3, 5, 7)
\end{aligned}$$



$$F = C + A'B$$

4.  $AB'C + A'B'C + A'BC + AB'C' + A'B'C'$  Soln:

$$\begin{aligned}
&= m_5 + m_1 + m_3 + m_4 + m_0 \\
&= \sum m (0, 1, 3, 4, 5)
\end{aligned}$$

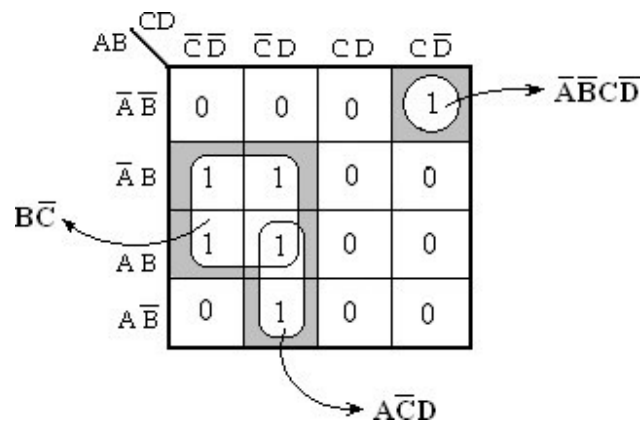


$$F = A'C + B'$$

#### Four - Variable Map:

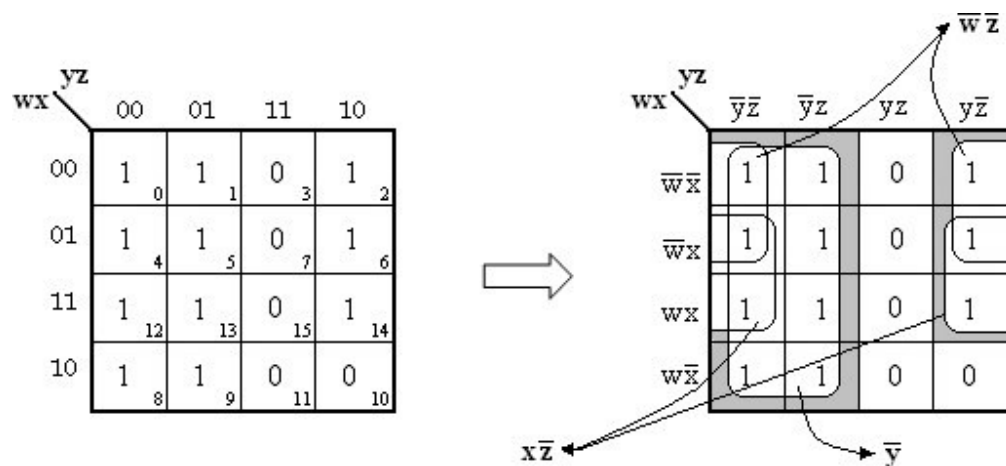
1. Simplify the Boolean expression,

$$Y = A'BC'D' + A'BC'D + ABC'D' + ABC'D + AB'C'D + A'B'CD' \text{ Soln:}$$



Therefore,  $Y = A'B'CD' + AC'D + BC'$

2.  $F(w, x, y, z) = \sum m(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$  Soln:



Therefore,

$$F = y' + w'z' + xz'$$

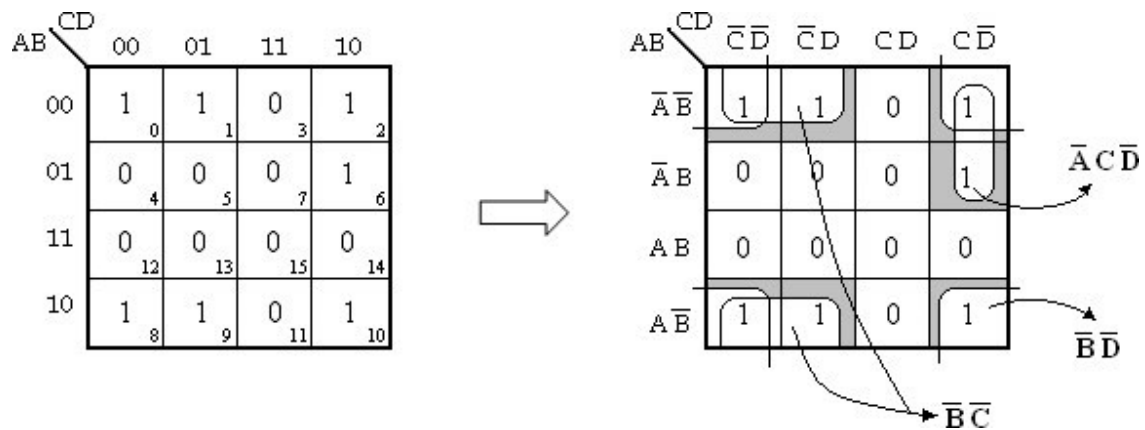
3.  $F = A'B'C' + B'CD' + A'BCD' + AB'C'$

$$= A'B'C' (D + D') + B'CD' (A + A') + A'BCD' + AB'C' (D + D')$$

$$= A'B'C'D + A'B'C'D' + AB'CD' + A'B'CD' + A'BCD' + AB'C'D + AB'C'D'$$

$$= m_1 + m_0 + m_{10} + m_2 + m_6 + m_9 + m_8$$

$$= \sum m(0, 1, 2, 6, 8, 9, 10)$$



Therefore,  $F = B'D' + B'C' +$

$$A'CD'$$

4.  $Y = ABCD + AB'C'D' + AB'C + AB$

$$= ABCD + AB'C'D' + AB'C (D + D') + AB (C + C') (D + D')$$

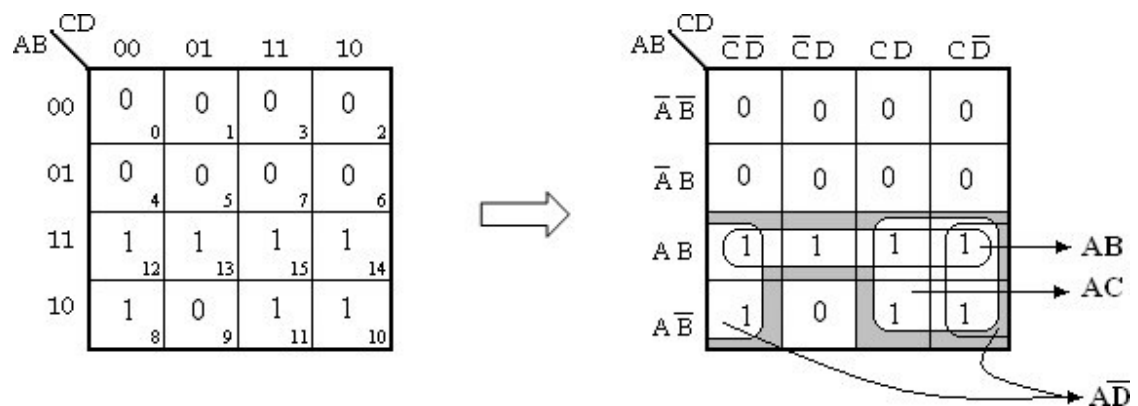
$$= ABCD + AB'C'D' + AB'CD + AB'CD' + (ABC + ABC') (D + D')$$

$$= \underline{ABCD} + AB'C'D' + AB'CD + AB'CD' + \underline{ABCD} + ABCD' + ABC'D + ABC'D'$$

$$= ABCD + AB'C'D' + AB'CD + AB'CD' + ABCD' + ABC'D + ABC'D'$$

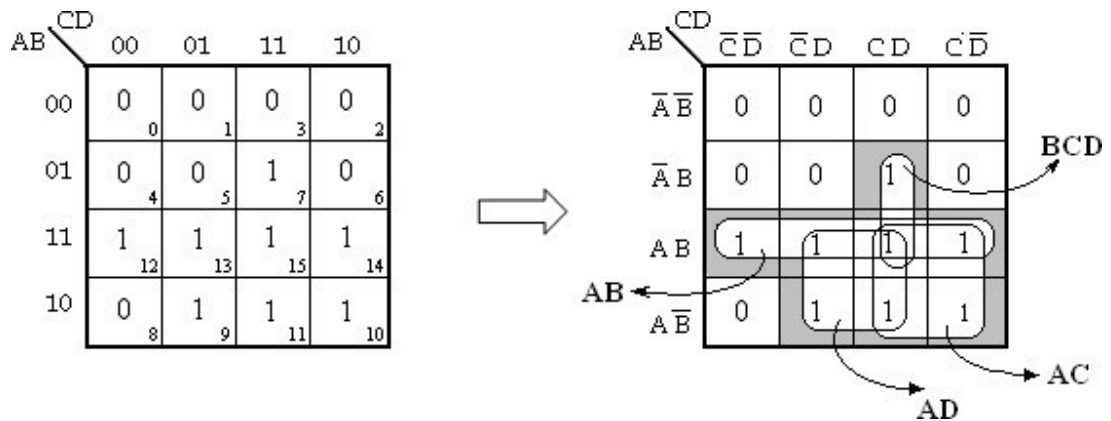
$$= m_{15} + m_8 + m_{11} + m_{10} + m_{14} + m_{13} + m_{12}$$

$$= \sum m(8, 10, 11, 12, 13, 14, 15)$$



Therefore,  $Y = AB + AC + AD$ .

5.  $Y(A, B, C, D) = \sum m(7, 9, 10, 11, 12, 13, 14, 15)$



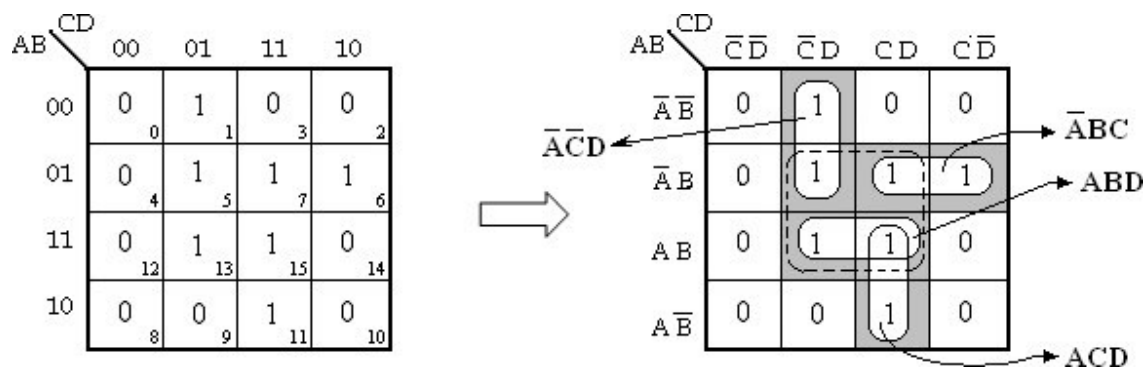
Therefore,

$$Y = AB + AC + AD + BCD.$$

6.  $Y = A'B'C'D + A'BC'D + A'BCD + A'BCD' + ABC'D + ABCD + AB'CD$

$$= m_1 + m_5 + m_7 + m_6 + m_{13} + m_{15} + m_{11}$$

$$= \sum m(1, 5, 6, 7, 11, 13, 15)$$

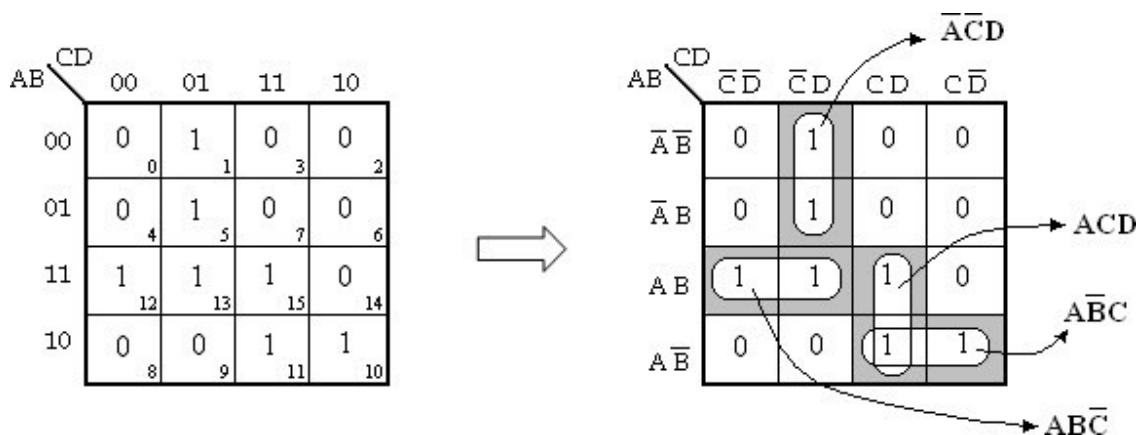


In the above K-map, the cells 5, 7, 13 and 15 can be grouped to form a quad as indicated by the dotted lines. In order to group the remaining 1's, four pairs have to be formed. However, all the four 1's covered by the quad are also covered by the pairs. So, the quad in the above k-map is redundant.

Therefore, the simplified expression will be,

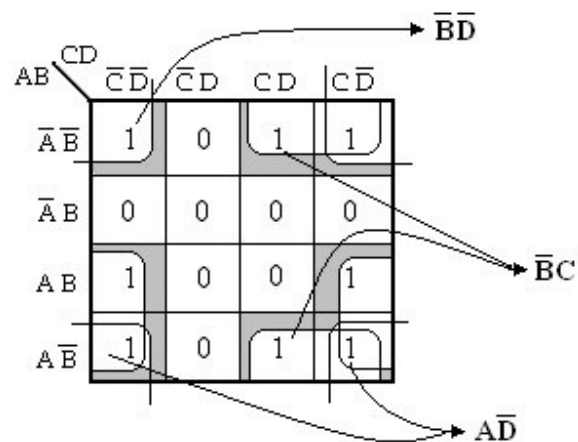
$$Y = A'C'D + A'BC + ABD + ACD.$$

$$7. Y = \sum m(1, 5, 10, 11, 12, 13, 15)$$



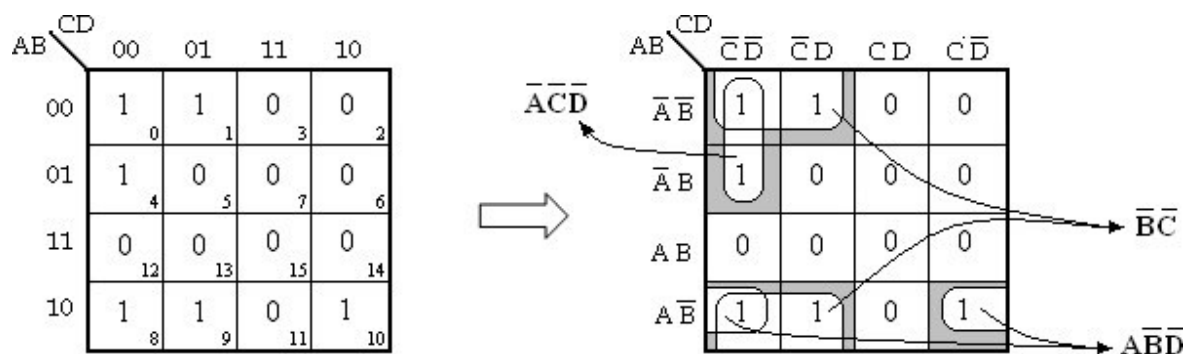
Therefore,  $Y = A'C'D + ABC' + ACD + AB'C$ .

$$8. Y = A'B'CD' + ABCD' + AB'CD' + AB'CD + AB'C'D' + ABC'D' + A'B'CD + A'B'C'D'$$



Therefore,  $Y = A'D + B'C + B'D'$

$$9. F(A, B, C, D) = \sum m(0, 1, 4, 8, 9, 10)$$



Therefore,  $F = A'C'D' + AB'D' + B'C'$

### Simplification of Sum of Products Expressions: (Minimal Sums)

$$1. Y = (A + B + C')(A + B' + C')(A' + B + C)(A' + B + C)(A + B + C)$$

$$= M_1 \cdot M_3 \cdot M_7 \cdot M_4 \cdot M_0$$

$$= \prod M(0, 1, 3, 4, 7)$$

$$= \sum m(2, 5, 6)$$

		BC			
		$\bar{B}\bar{C}$	$\bar{B}C$	$BC$	$B\bar{C}$
A	$\bar{A}$ 0	0 <sub>0</sub>	0 <sub>1</sub>	0 <sub>3</sub>	1 <sub>2</sub>
	A 1	0 <sub>4</sub>	1 <sub>5</sub>	0 <sub>7</sub>	1 <sub>6</sub>



		BC			
		$\bar{B}\bar{C}$	$\bar{B}C$	$BC$	$B\bar{C}$
A	$\bar{A}$ 0	0	0	0	1
	A 1	0	1	0	1

Groupings:  $\bar{A}C$  (top row, columns 3 and 4),  $\bar{B}\bar{C}$  (column 1, rows 0 and 1),  $BC$  (column 4, rows 0 and 1).

$$Y' = B'C' + A'C + BC.$$

$$Y = Y'' = (B'C' + A'C + BC)'$$

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

$$= (B'' + C'') \cdot (A'' + C') \cdot (B' + C') \quad Y =$$

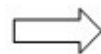
$$(B + C) \cdot (A + C') \cdot (B' + C')$$

$$2. Y = (A' + B' + C + D) (A' + B' + C' + D) (A' + B' + C' + D') (A' + B + C + D) (A + B' + C' + D) (A + B' + C' + D') (A + B + C + D) (A' + B' + C + D')$$

$$= M_{12} \cdot M_{14} \cdot M_{15} \cdot M_8 \cdot M_6 \cdot M_7 \cdot M_0.$$

$$M_{13} = \prod M(0, 6, 7, 8, 12, 13, 14, 15)$$

		CD			
		$\bar{C}\bar{D}$	$\bar{C}D$	$CD$	$C\bar{D}$
AB	$\bar{A}\bar{B}$ 00	0 <sub>0</sub>	1 <sub>1</sub>	1 <sub>3</sub>	1 <sub>2</sub>
	$\bar{A}B$ 01	1 <sub>4</sub>	1 <sub>5</sub>	0 <sub>7</sub>	0 <sub>6</sub>
AB	11	0 <sub>12</sub>	0 <sub>13</sub>	0 <sub>15</sub>	0 <sub>14</sub>
	$A\bar{B}$ 10	0 <sub>8</sub>	1 <sub>9</sub>	1 <sub>11</sub>	1 <sub>10</sub>



		CD			
		$\bar{C}\bar{D}$	$\bar{C}D$	$CD$	$C\bar{D}$
AB	$\bar{A}\bar{B}$	0	1	1	1
	$\bar{A}B$	1	1	0	0
AB	11	0	0	0	0
	$A\bar{B}$	0	1	1	1

Groupings:  $\bar{B}\bar{C}\bar{D}$  (row 0, column 1),  $BC$  (row 1, columns 3 and 4),  $AB$  (row 3, columns 3 and 4).

$$Y' = B'C'D' + AB + BC$$

$$Y = Y'' = (B'C'D' + AB + BC)'$$

$$= (B'C'D')' \cdot (AB)' \cdot (BC)'$$



$$= (B'' + C'' + D''). (A' + B'). (B' + C')$$

$$= (B + C + D). (A' + B'). (B' + C')$$

Therefore,  $Y = (B + C + D). (A' + B'). (B' + C')$

3.  $F(A, B, C, D) = \prod M(0, 2, 3, 8, 9, 12, 13, 14, 15)$

AB \ CD				
	$\bar{C}\bar{D}$	$\bar{C}D$	$CD$	$C\bar{D}$
$\bar{A}\bar{B}$ 00	0 <sub>0</sub>	1 <sub>1</sub>	0 <sub>3</sub>	0 <sub>2</sub>
$\bar{A}B$ 01	1 <sub>4</sub>	1 <sub>5</sub>	1 <sub>7</sub>	1 <sub>6</sub>
$AB$ 11	0 <sub>12</sub>	0 <sub>13</sub>	0 <sub>15</sub>	1 <sub>14</sub>
$A\bar{B}$ 10	0 <sub>8</sub>	0 <sub>9</sub>	1 <sub>11</sub>	1 <sub>10</sub>

AB \ CD				
	$\bar{C}\bar{D}$	$\bar{C}D$	$CD$	$C\bar{D}$
$\bar{A}\bar{B}$	0	1	0	0
$\bar{A}B$	1	1	1	1
$AB$	0	0	0	1
$A\bar{B}$	0	0	1	1

$$Y' = A'B'D' + A'B'C + ABD + AC'$$

$$Y = Y'' = (A'B'D' + A'B'C + ABD + AC')'$$

$$= (A'B'D')'. (A'B'C)'. (ABD)'. (AC')'$$

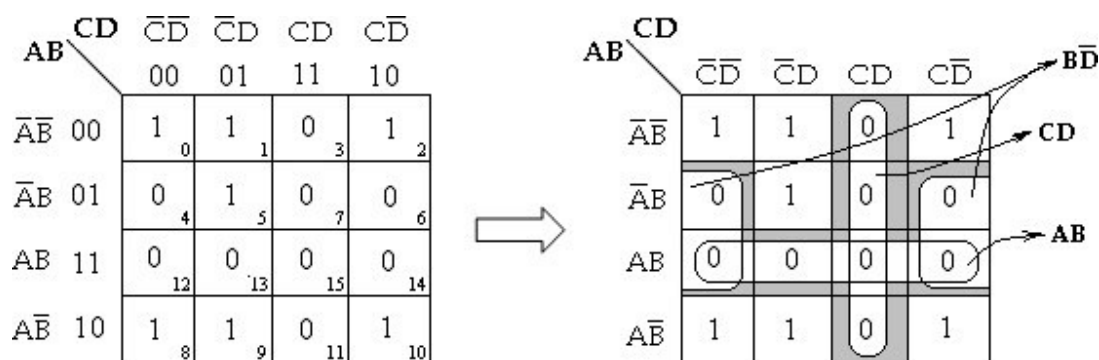
$$= (A'' + B'' + D''). (A'' + B'' + C'). (A' + B' + D'). (A' + C'')$$

$$= (A + B + D). (A + B + C'). (A' + B' + D'). (A' + C)$$

Therefore,  $Y = (A + B + D). (A + B + C'). (A' + B' + D'). (A' + C)$

4.  $F(A, B, C, D) = \sum m(0, 1, 2, 5, 8, 9, 10)$

$$= \prod M(3, 4, 6, 7, 11, 12, 13, 14, 15)$$



$$Y' = BD' + CD + AB$$

$$Y = Y'' = (BD' + CD + AB)'$$

$$= (BD')' \cdot (CD)' \cdot (AB)'$$

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

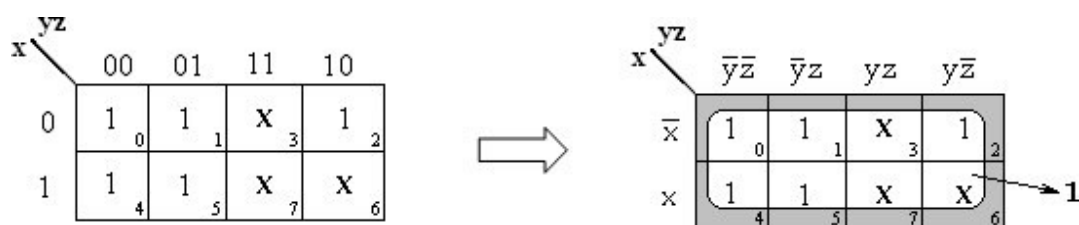
$$= (B' + D) \cdot (C' + D') \cdot (A' + B')$$

$$\text{Therefore, } Y = (B' + D) \cdot (C' + D') \cdot (A' + B')$$

### Don't care Conditions:

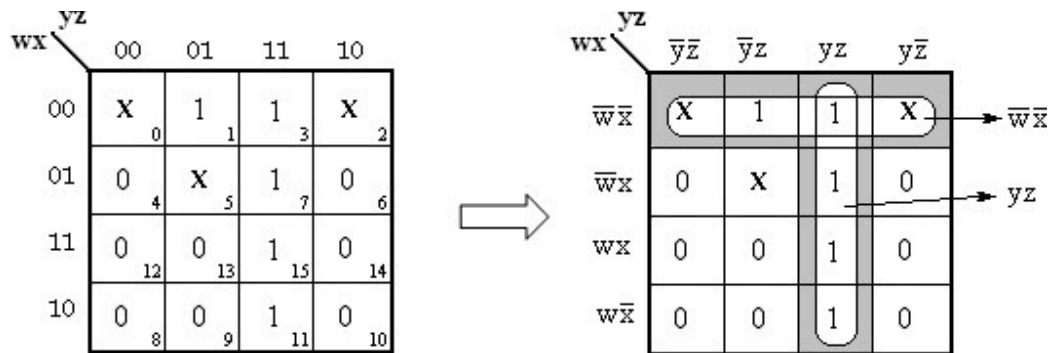
A don't care minterm is a combination of variables whose logical value is not specified. When choosing adjacent squares to simplify the function in a map, the don't care minterms may be assumed to be either 0 or 1. When simplifying the function, we can choose to include each don't care minterm with either the 1's or the 0's, depending on which combination gives the simplest expression.

$$1. F(x, y, z) = \sum m(0, 1, 2, 4, 5) + \sum d(3, 6, 7)$$



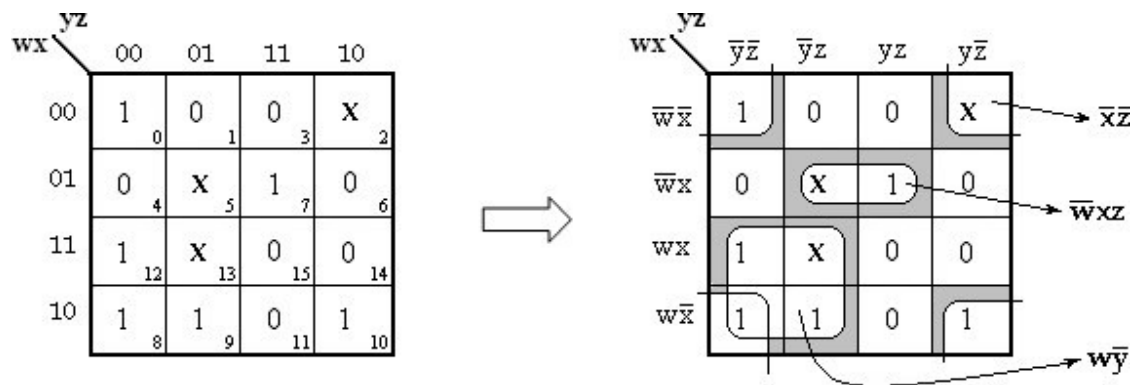
$$F(x, y, z) = 1$$

2.  $F(w, x, y, z) = \sum m(1, 3, 7, 11, 15) + \sum d(0, 2, 5)$



**$F(w, x, y, z) = w'x' + yz$**

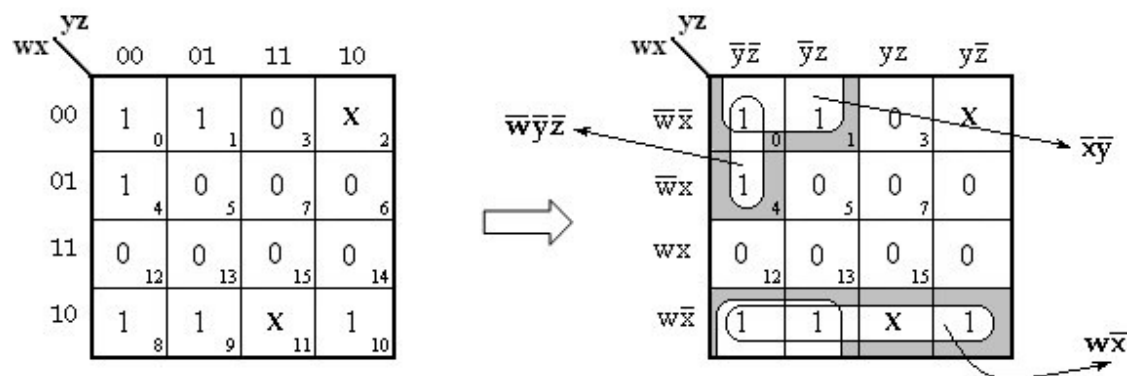
3.  $F(w, x, y, z) = \sum m(0, 7, 8, 9, 10, 12) + \sum d(2, 5, 13)$



**$F(w, x, y, z) = w'xz + w\bar{y} + x'z'$**

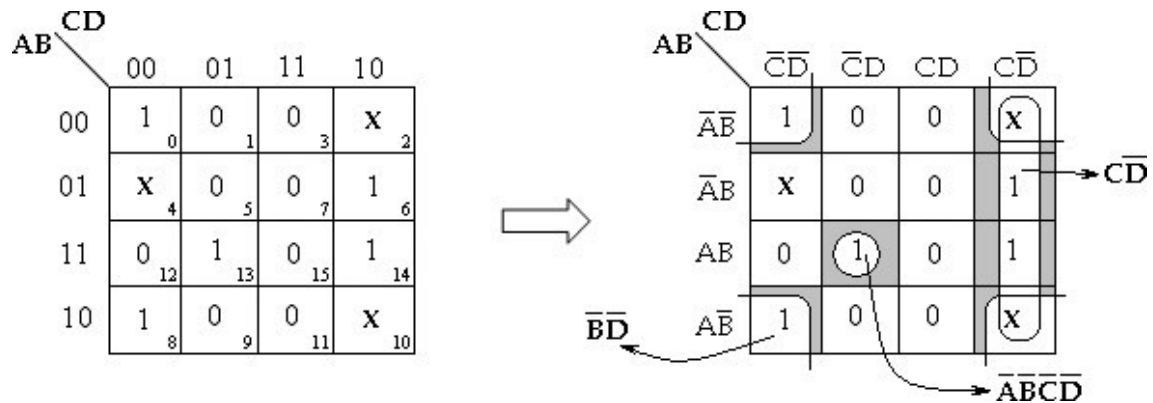
4.  $F(w, x, y, z) = \sum m(0, 1, 4, 8, 9, 10) + \sum d(2, 11)$

**Soln:**



$$F(w, x, y, z) = wx' + x'y' + w'y'z'.$$

$$5. F(A, B, C, D) = \sum m(0, 6, 8, 13, 14) + \sum d(2, 4, 10) \text{ Soln:}$$



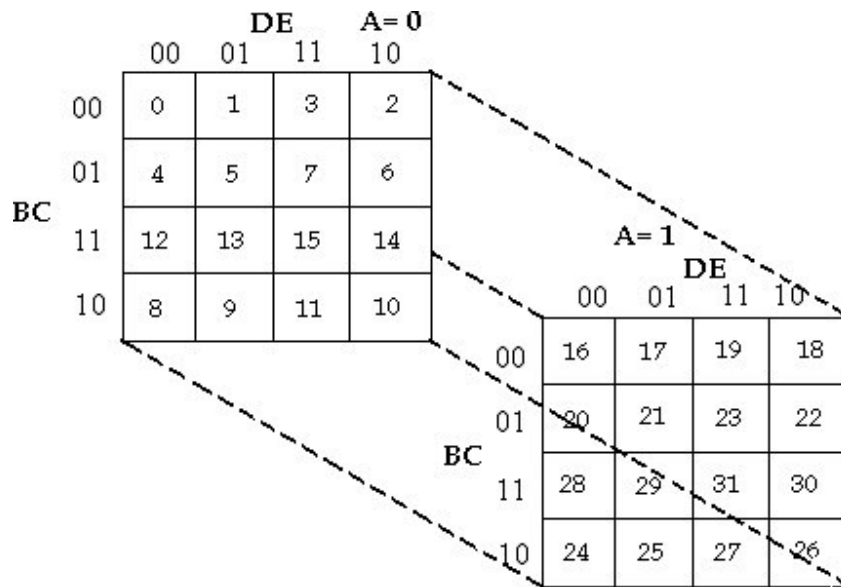
$$F(A, B, C, D) = \bar{C}\bar{D} + B'D + A'B'C'D'.$$

### Five- Variable Maps:

A 5- variable K- map requires 25= 32 cells, but adjacent cells are difficult to identify on a single 32-cell map. Therefore, two 16 cell K-maps are used.

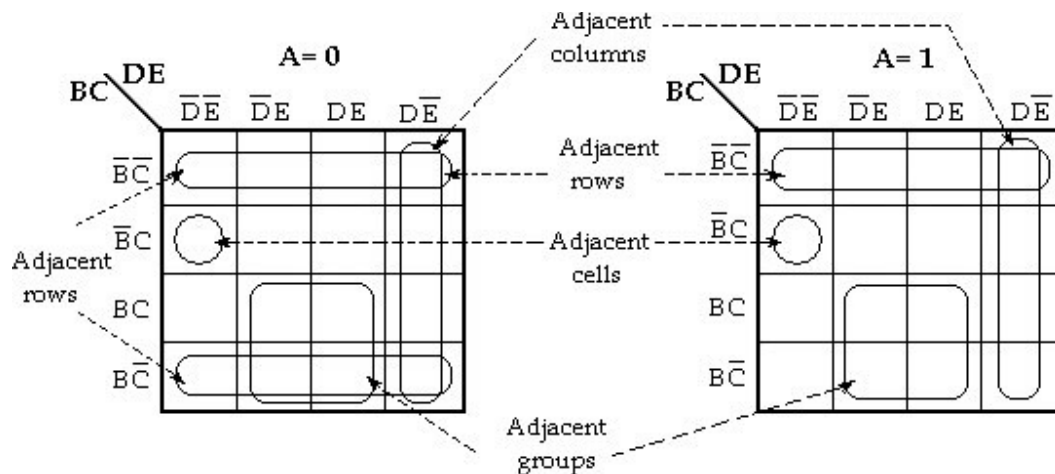
If the variables are A, B, C, D and E, two identical 16- cell maps containing B, C, D and E can be constructed. One map is used for A and other for A'.

In order to identify the adjacent grouping in the 5- variable map, we must imagine the two maps superimposed on one another ie., every cell in one map is adjacent to the corresponding cell in the other map, because only one variable changes between such corresponding cells.



### Five- Variable Karnaugh map (Layer Structure)

Thus, every row on one map is adjacent to the corresponding row (the one occupying the same position) on the other map, as are corresponding columns. Also, the rightmost and leftmost columns within each 16- cell map are adjacent, just as they are in any 16- cell map, as are the top and bottom rows.



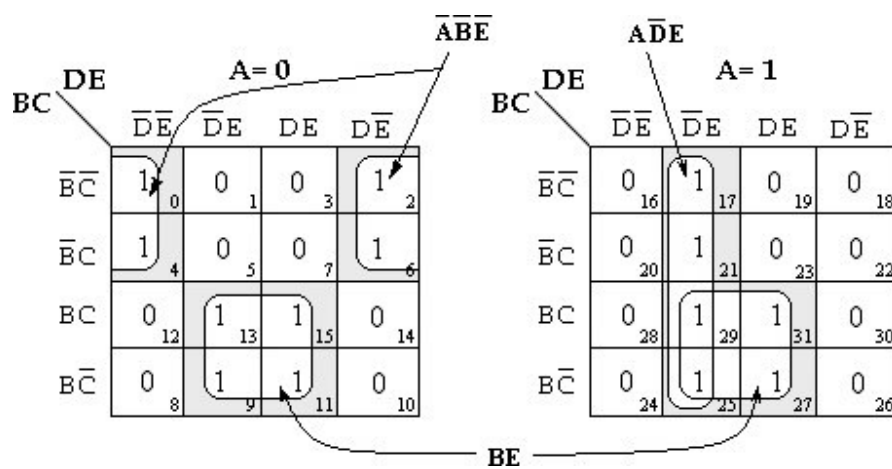
### Typical subcubes on a five-variable map

However, the rightmost column of the map is not adjacent to the leftmost column of the other map.

1. Simplify the Boolean function

$$F(A, B, C, D, E) = \sum m(0, 2, 4, 6, 9, 11, 13, 15, 17, 21, 25, 27, 29, 31)$$

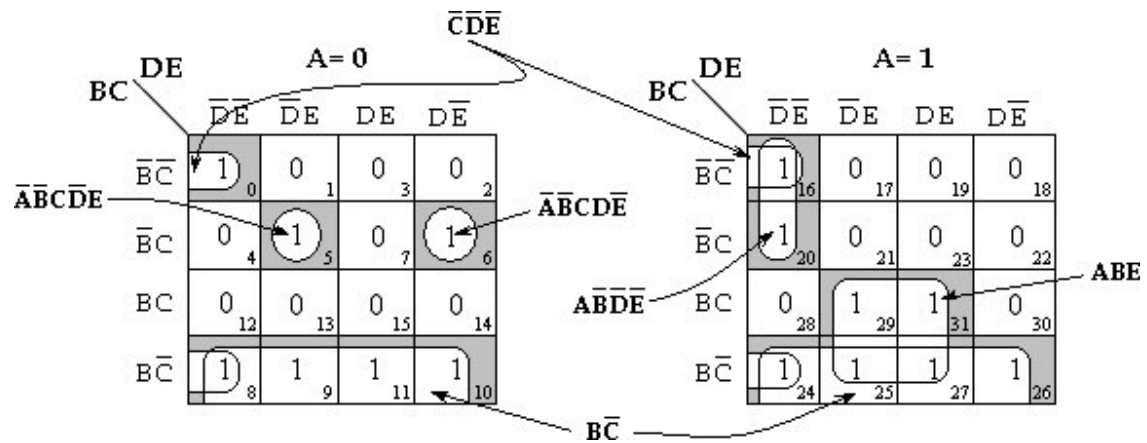
**Soln:**



$$F(A, B, C, D, E) = \overline{A}\overline{B}\overline{E} + BE + A\overline{D}\overline{E}$$

2.  $F(A, B, C, D, E) = \sum m(0, 5, 6, 8, 9, 10, 11, 16, 20, 24, 25, 26, 27, 29, 31)$

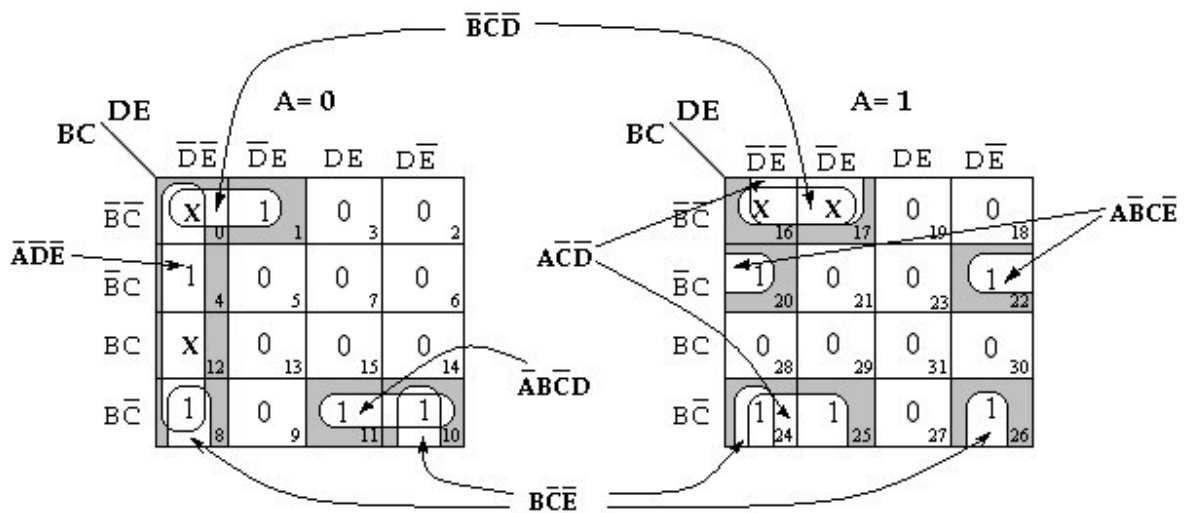
**Soln:**



$$F(A, B, C, D, E) = C'D'E' + A'B'CD'E + A'B'CDE' + AB'D'E' + ABE + BC'$$

3.  $F(A, B, C, D, E) = \sum m(1, 4, 8, 10, 11, 20, 22, 24, 25, 26) + \sum d(0, 12, 16, 17)$

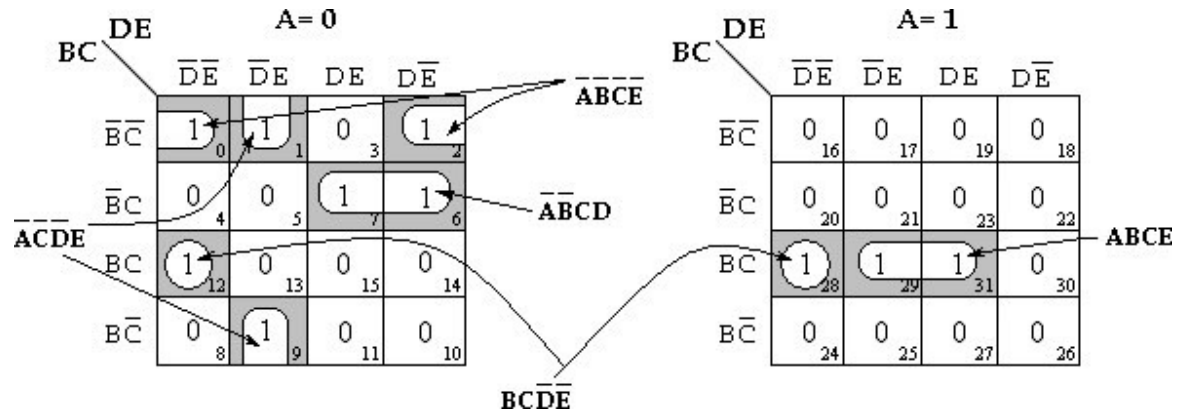
Soln:



$$F(A, B, C, D, E) = B'C'D' + A'D'E' + BC'E' + A'BC'D + AC'D' + AB'CE'$$

4.  $F(A, B, C, D, E) = \sum m(0, 1, 2, 6, 7, 9, 12, 28, 29, 31)$

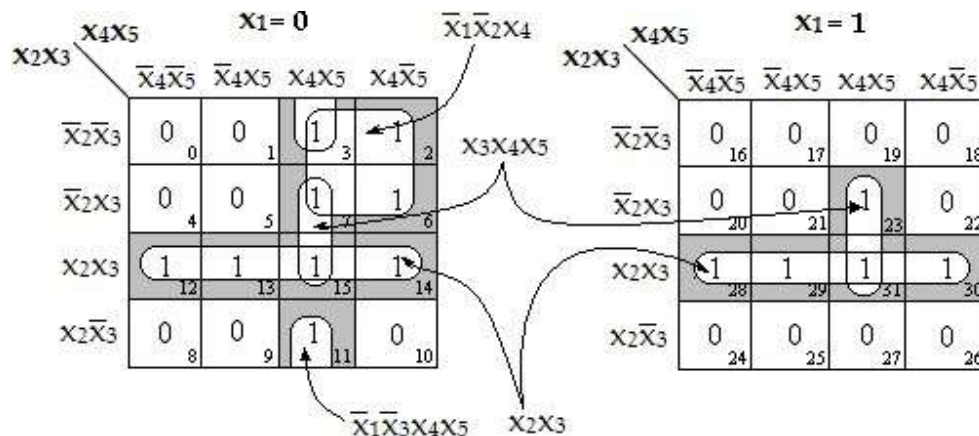
**Soln:**



$F(A, B, C, D, E) = BCD'E' + ABCE + A'B'C'E' + A'C'D'E + A'B'CD$

5.  $F(x_1, x_2, x_3, x_4, x_5) = \sum m(2, 3, 6, 7, 11, 12, 13, 14, 15, 23, 28, 29, 30, 31)$

**Soln:**

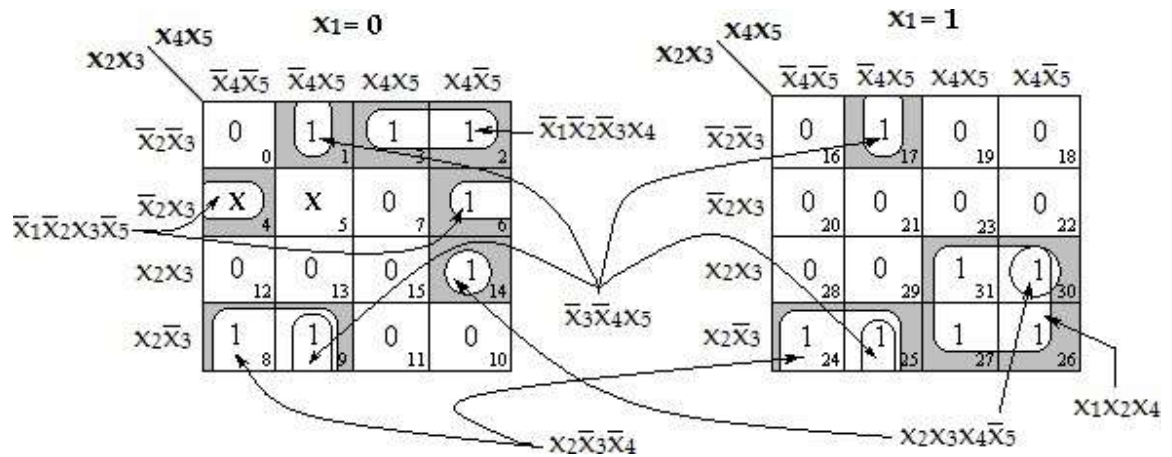


$F(x_1, x_2, x_3, x_4, x_5) = x_2x_3 + x_3x_4x_5 + x_1'x_2'x_4 + x_1'x_3'x_4x_5$



6.  $F(x_1, x_2, x_3, x_4, x_5) = \sum m(1, 2, 3, 6, 8, 9, 14, 17, 24, 25, 26, 27, 30, 31) + \sum d(4, 5)$

**Soln:**



$F(x_1, x_2, x_3, x_4, x_5) = x_2x_3'x_4' + x_2x_3x_4x_5' + x_3'x_4'x_5 + x_1x_2x_4 + x_1'x_2'x_3x_5' + x_1'x_2'x_3'x_4$