UNIT I

BOOLEAN ALGEBRA AND LOGIC GATES

Review of Number Systems – Arithmetic Operations – Binary Codes – Boolean Algebra and theorems – Boolean Functions – Simplification of Boolean Functions using Karnaugh map and Tabulation Methods – Logic gates – NAND and NOR Implementations.

Number Systems

Q1 a) Explain various number system

(Or)

b) Name the number system used in computers

Answer:

- Number system is a basis for counting various items.
- The decimal number system has 10 digits:0,1,2,3,4,5,6,7,8 and 9.
- Modern computers communicate and operate with binary numbers which use only the digits 0 and 1.
- When decimal quantities are represented in the binary form they take more digits.
- For large decimal numbers people have to deal with very large binary strings and therefore they do not like working with binary numbers. This fact gave rise to three new number systems: Octal, Hexadecimal and Binary Coded Decimal (BCD)

Decimal Number System

The decimal number system is a base or radix-10 number system and therefore has 10 different digits or symbols. These are 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9.

For example, the decimal number 3586.265, the integer part (i.e. 3586) can be expressed as
$3586 = 6 \times 10^0 + 8 \times 10^1 + 5 \times 10^2 + 3 \times 10^3 = 6 + 80 + 500 + 3000 = 3586$
and the fractional part can be expressed as
$265 = 2 \times 10^{-1} + 6 \times 10^{-2} + 5 \times 10^{-3} = 0.\ 2 + 0.\ 06 + 0\ .005 = 0.\ 265$

Binary Number System

The binary number system is a radix-2 number system with 0‘and 1‘as the two independent digits.

Octal Number System

The octal number system has a radix of 8 and therefore has eight distinct digits.. The independent digits are 0, 1, 2, 3, 4, 5, 6 and 7.

Hexadecimal Number System

The hexadecimal number system is a radix-16 number system and its 16 basic digits are 0, 1, 2, 3,4, 5, 6, 7, 8, 9, A, B, C, D, E and F. The place values or weights of different digits in a mixed hexa decimal number are 16^0, 16^1, 16^2 and so on (for the integer part) and 16^{-1}, 16^{-2}, 16^{-3} and so on(for the fractional part). The decimal equivalent of A, B, C, D, E and F are 10, 11, 12, 13, 14 and 15respectively, for obvious reasons.

Binary-to-Decimal Conversion

The decimal equivalent of the binary number $(1001.0101)_2$is determined as follows:

The integer part = 1001

The decimal equivalent = $1 \times 2^0 + 0 \times 2^1 + 0 \times 2^2 + 1 \times 2^3 = 1 + 0 + 0 + 8 = 9$

The fractional part = .0101

The decimal equivalent = $0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2{-}4 = 0 + 0.25 + 0 + 0.0625 =$ 0.3125

Therefore, the decimal equivalent of $(1001.0101)_2 = 9.3125$

Octal-to-Decimal Conversion

The decimal equivalent of the octal number $(137.21)_8$is determined as follows:

The integer part = 137

The decimal equivalent = $7 \times 8^0 + 3 \times 8^1 + 1 \times 8^2 = 7 + 24 + 64 = 95$

The fractional part = .21

The decimal equivalent = $2 \times 8^{-1} + 1 \times 8^{-2} = 0.265$

Therefore, the decimal equivalent of $(137.21)_8 = (95.265)_{10}$

The decimal equivalent of the hexadecimal number $(1E0.2A)_{16}$ is determined as follows:

The integer part = 1E0

The decimal equivalent = $0 \times 16^0 + 14 \times 16^1 + 1 \times 16^2 = 0 + 224 + 256 = 480$

The fractional part = 2A

The decimal equivalent = $2 \times 16^{-1} + 10 \times 16^{-2} = 0.164$

Therefore, the decimal equivalent of $(1E0.2A)_{16} = (480.164)_{10}$

Example

Find the decimal equivalent of the following binary numbers expressed in the 2's complement format:

a) 00001110 b) 10001110

Solution:

(a) The MSB bit is _0', which indicates a plus sign.

The magnitude bits are 0001110.

The decimal equivalent = $0 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 + 0 \times 2^4 + 0 \times 2^5 + 0 \times 2^6 = 0 + 2 + 4 + 8 + 0 + 0 + 0 = 14$

Therefore, 00001110 represent **+14**

(b) The MSB bit is _1', which indicates a minus sign

The magnitude bits are therefore given by the 2's complement of 0001110, i.e. 1110010

The decimal equivalent = $0 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 0 \times 2^3 + 1 \times 2^4 + 1 \times 2^5 + 1 \times 2^6$

$= 0 + 2 + 0 + 0 + 16 + 32 + 64 = 114$

Therefore, 10001110 represent **−114**

Decimal-to-Binary Conversion

Example

Find the binary equivalent of $(13.375)_{10}$.

Solution

The integer part = 13

Divisor	Dividend	Remainder
2	13	—
2	6	**1**
2	3	**0**
2	1	**1**
—	0	**1**

The binary equivalent of $(13)_{10}$ is therefore $(1101)_2$

The fractional part = .375

$0.375 \times 2 = 0.75$ with a carry of 0

$0.75 \times 2 = 0.5$ with a carry of 1

$0.5 \times 2 = 0$ with a carry of 1

The binary equivalent of $(0.375)_{10} = (.011)_2$

Therefore, the binary equivalent of $(13.375)_{10} = (1101.011)_2$

Decimal-to-Octal Conversion

Example

Find the octal equivalent of $(73.75)_{10}$

Solution

The integer part = 73

Divisor	Dividend	Remainder
8	73	—
8	9	**1**
8	1	**1**
—	0	**1**

The octal equivalent of $(73)_{10} = (111)_8$

The fractional part = 0.75

0.75 × 8 = 0 with a carry of 6

The octal equivalent of $(0.75)_{10} = (.6)_8$

Therefore, the octal equivalent of $(73.75)_{10} = (111.6)_8$

Decimal-to-Hexadecimal Conversion

Example

Let us determine the hexadecimal equivalent of $(82.25)_{10}$

Solution

The integer part = 82

Divisor	Dividend	Remainder
16	82	—
16	5	**2**
—	0	**5**

The hexadecimal equivalent of $(82)_{10} = (52)_{16}$

The fractional part = 0.25

0.25 × 16 = 0 with a carry of 4

Therefore, the hexadecimal equivalent of $(82.25)_{10} = (52.4)_{16}$

Binary–Octal and Octal–Binary Conversions

Example

Let us find the binary equivalent of $(374.26)_8$ and the octal equivalent of $(1110100.0100111)_2$

Solution

The given octal number = $(374.26)_8$

The binary equivalent = $(011\ 111\ 100.010\ 110)_2 = (011111100.010110)_2$

Any 0s on the extreme left of the integer part and extreme right of the fractional part of the equivalent binary number should be omitted. Therefore,

$(011111100.010110)_2 = (11111100.01011)_2$

The given binary number = $(1110100.0100111)_2$

$(1110100.0100111)_2 = (1\ 110\ 100.010\ 011\ 1)_2$

$= (001\ 110\ 100.010\ 011\ 100)_2 = (164.234)_8$

Hex–Binary and Binary–Hex Conversions

Example

Let us find the binary equivalent of $(17E.F6)_{16}$and the hex equivalent of $(1011001110.011011101)_2$.

Solution

The given hex number = $(17E.F6)_{16}$

The binary equivalent = $(0001\ 0111\ 1110.1111\ 0110)_2$

$= (000101111110.11110110)_2$

$= (101111110.1111011)_2$

The 0s on the extreme left of the integer part and on the extreme right of the fractional part have been omitted.

The given binary number = $(1011001110.011011101)_2$

$= (10\ 1100\ 1110.0110\ 1110\ 1)_2$

The hex equivalent = $(0010\ 1100\ 1110.0110\ 1110\ 1000)_2 = (2CE.6E8)_{16}$

Hex–Octal and Octal–Hex Conversions

Example

Let us find the octal equivalent of $(2F.C4)_{16}$and the hex equivalent of $(762.013)_8$

Solution

The given hexa number = $(2F.C4)_{16}$.

The binary equivalent = $(0010\ 1111.1100\ 0100)_2 = (00101111.11000100)_2$

$(101111.110001)_2 = (101\ 111.110\ 001)_2 = (57.61)_8$.

The given octal number = $(762.013)_8$.

The octal number = $(762.013)_8 = (111\ 110\ 010.000\ 001\ 011)_2$

$= (111110010.000001011)_2$

$= (0001\ 1111\ 0010.0000\ 0101\ 1000)_2 = (1F2.058)_{16}$.

Basic Binary Codes

Q2 a) Explain about basic codes

(Or)

b) Discuss about binary ,BCD,Excess 3 and Gray codes

Binary Coded Decimal

The binary coded decimal (BCD) is a type of binary code used to represent a given decimal number in an equivalent binary form.

The BCD equivalent of a decimal number is written by replacing each decimal digit in the integer and fractional parts with its four-bit binary equivalent. As an example, the BCD equivalent of $(23.15)_{10}$is written as $(0010\ 0011.0001\ 0101)_{BCD}$. The 8421 code is the most popular of all the BCD codes; it is simply referred to as the BCD code.

Decimal	8421 BCD code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Excess-3 Code

The excess-3 code is another important BCD code. It is particularly significant for arithmetic operations as it overcomes the shortcomings encountered while using the 8421 BCD code to add two decimal digits whose sum exceeds 9. The excess-3 code has no such limitation, and it considerably simplifies arithmetic operations.

The excess-3 code for a given decimal number is determined by adding _3' to each decimal digit in the given number and then replacing each digit of the newly found decimal number by its four-bit binary equivalent.

Excess-3 code equivalent of decimal numbers

Decimal number	Excess-3 code	Decimal number	Excess-3 code
0	0011	5	1000
1	0100	6	1001
2	0101	7	1010
3	0110	8	1011
4	0111	9	1100

The excess-3 code for 597 is therefore given by: 1000 1100 1010 = 100011001010.

Gray Code

The Gray code was designed by Frank Gray at Bell Labs and patented in 1953. It is an weighted binary code in which two successive values differ only by 1 bit.

Decimal	Binary	Gray	Decimal	Binary	Gray
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

Alphanumeric Codes

Alphanumeric codes, also called character codes, are binary codes used to represent alphanumeric data. The codes write alphanumeric data, including letters of the alphabet, numbers, mathematical symbols and punctuation marks, in a form that is understandable and processable by a computer. These codes enable us to interface input–output devices such as keyboards, printers, etc.,

Q3 a) State the rules for binary addition and subtraction

(Or)

c) Illustrate the rules for binary addition and subtraction

Basic Rules of Binary Addition and Subtraction

The basic principles of binary addition and subtraction are similar to what we all know so well in the case of the decimal number system.

1. 0 + 0 = 0.
2. 0 + 1 = 1.
3. 1 + 0 = 1.
4. 1 + 1 = 0 with a carry of _1' to the next more significant bit.
5. 1 + 1 + 1 = 1 with a carry of _1' to the next more significant bit.

A	B	Carry- in (C_{in})	**Sum**	**Carry- out (C_o)**	A	B	Carry- in (C_{in})	**Sum**	**Carry- out (C_o)**
0	0	0	**0**	**0**	1	0	0	**1**	**0**
0	0	1	**1**	**0**	1	0	1	**0**	**1**
0	1	0	**1**	**0**	1	1	0	**0**	**1**
0	1	1	**0**	**1**	1	1	1	**1**	**1**

The basic principles of binary subtraction include the following:

1. 0 − 0 = 0
2. 1 − 0 = 1
3. 1 − 1 = 0
4. 0 − 1 = 1 with a borrow of 1 from the next more significant bit

Inputs			Outputs	
Minuend	Subtrahend	Borrow-in	Difference	Borrow-out
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Binary Multiplication

Q4 a) State the rules for binary multiplication and division

(Or)

b) Illustrate the rules for binary multiplication and division

The basic rules of binary multiplication are governed by the way an AND gate functions when the two bits to be multiplied are fed as inputs to the gate.

1. $0 \times 0 = 0$.
2. $0 \times 1 = 0$.
3. $1 \times 0 = 0$.
4. $1 \times 1 = 1$.

Binary Division

While binary multiplication is the process of repeated addition, binary division is the process of repeated subtraction.

1 0 1 0 1 1 1 1	Multiplicand Multiplier
0 0 0 0 0 0 + 1 0 1 0 1 1	Start
1 0 1 0 1 1	Result of first addition
0 1 0 1 0 1 + 1 0 1 0 1 1	1 (Result of addition shifted one bit to right)
1 0 0 0 0 0 0	Result of second addition
0 1 0 0 0 0 0	01 (Result of addition shifted one bit to right)

Fundamental postulates of Boolean algebra

Q5 a) State the Fundamental postulates of Boolean algebra:

(or)

b) State various laws of Boolean algebra

The postulates of a mathematical system forms the basic assumption from which it is possible to deduce the theorems, laws and properties of the system.

The most common postulates used to formulate various structures are,

i) **Closure**:

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.

The result of each operation with operator (+) or (.) is either 1 or 0 and 1, 0 ЄB.

ii) **Identity element**:

A set S is said to have an identity element w.r.t a binary operation * on S, if there exists an element e Є S with the property,

e* x = x * e = x

Eg:

0+ 0 = 0 0+ 1 = 1+ 0 = 1 a) **x+ 0= x**

1 . 1 = 1 1 . 0 = 0 . 1 = 1 b) **x. 1 = x**

iii) **Commutative law**:

A binary operator * on a set S is said to be commutative if,

x * y = y * x for all x, y Є S

Eg: 0+ 1 = 1+ 0 = 1 a) **x+ y= y+ x**

0 . 1 = 1 . 0 = 0 b) **x. y= y. x**

iv) **Distributive law**:

If * and • are two binary operation on a set S, • is said to be distributive over + whenever,

x . (y+ z) = (x. y) + (x. z)

Similarly, + is said to be distributive over • whenever,

x + (y. z) = (x+ y). (x+ z)

v) **Inverse**:

A set S having the identity element e, w.r.t. binary operator * is said to have an inverse, whenever for every x Є S, there exists an element x' Є S such that,

x. x' Є e

a) x+ x' = 1, since 0 + 0' = 0+ 1 and 1+ 1' = 1+ 0 = 1

b) x. x' = 1, since 0 . 0' = 0. 1 and 1. 1' = 1. 0 = 0

POSTULATES	(a)	(b)
Postulate 2 (Identity)	**x + 0 = x**	**x . 1 = x**
Postulate 3 (Commutative)	**x+ y = y+ x**	**x . y = y. x**
Postulate 4 (Distributive)	**x (y+ z) = xy+ xz**	**x+ yz = (x+ y). (x+ z)**
Postulate 5 (Inverse)	**x+x' = 1**	**x. x' = 0**

Basic theorem and properties of Boolean algebra

Q6. a) State and prove the laws of Boolean algebra

(Or)

b) State and prove the theorems of Boolean algebra

Basic Theorems:

The theorems, like the postulates are listed in pairs; each relation is the dual of the one paired with it. The postulates are basic axioms of the algebraic structure and need no proof. The theorems must be proven from the postulates. The proofs of the theorems with one variable are presented below. At the right is listed the number of the postulate that justifies each step of the proof.

1) a) **x+ x = x**

x+ x = (x+ x) . 1 ---------- by postulate 2(b) [x. 1 = x]

= (x+ x). (x+ x') ---------- 5(a) [x+ x' = 1]

= x+ xx' ---------- 4(b) [x+yz = (x+y)(x+z)]

= x+ 0 ---------- 5(b) [x. x' = 0]

= x ---------- 2(a) [x+0 = x]

b) **x. x = x**

x. x = (x. x) + 0 ---------- by postulate 2(a) [x+ 0 = x]

= (x. x) + (x. x') ---------- 5(b) [x. x' = 0]

= x (x+ x') ---------- 4(a) [x (y+z) = (xy)+ (xz)]

= x (1) ---------- 5(a) [x+ x' = 1]

= x ---------- 2(b) [x.1 = x]

2) a) **x+ 1 = 1**

x+ 1 = 1 . (x+ 1) ---------- by postulate 2(b) [x. 1 = x]

= (x+ x'). (x+ 1) ---------- 5(a) [x+ x' = 1]

= x+ x'.1 ---------- 4(b) [x+yz = (x+y)(x+z)]

= x+ x' ---------- 2(b) [x. 1 = x]

= 1 ---------- 5(a) [x+ x'= 1]

b) **x .0 = 0**

3) (x')' = x

From postulate 5, we have x+ x' = 1 and x. x' = 0, which defines the complement of x. The complement of x' is x and is also (x')'.
Therefore, since the complement is unique, (x')' = x.

4) Absorption Theorem:

a) **x+ xy = x**

x+ xy = x. 1 + xy ---------- by postulate 2(b) [x. 1 = x]

= x (1+ y) ---------- 4(a) [x (y+z) = (xy)+ (xz)]

= x (1) ---------- by theorem 2(a) [x+ 1 = x]

= x. ---------- by postulate 2(a) [x. 1 = x]

b) **x. (x+ y) = x**

x. (x+ y) = x. x+ x. y ---------- 4(a) [x (y+z) = (xy)+ (xz)]

= x + x.y ---------- by theorem 1(b) [x. x = x]

= x. ---------- by theorem 4(a) [x+ xy = x]

c) **x+ x'y = x+ y**

x+ x'y = x+ xy+ x'y ---------- by theorem 4(a) [x+ xy = x]

= x+ y (x+ x') ---------- by postulate 4(a) [x (y+z) = (xy)+ (xz)]

= x+ y (1) ---------- 5(a) [x+ x' = 1]

= x+ y ---------- 2(b) [x. 1= x]

d) **x. (x'+y) = xy**

x. (x'+y) = x.x'+ xy ---------- by postulate 4(a) [x (y+z) = (xy)+ (xz)]

= 0+ xy ---------- 5(b) [x. x' = 0]

= xy. ---------- 2(a) [x+ 0= x]

Properties of Boolean algebra:

1. *Commutative property:*

Boolean addition is commutative, given by

x. y = y. x

This means that the order of the AND operation conducted on the variables makes no difference.

2. *Associative property:*

The associative property of addition is given by,

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

The OR operation of several variables results in the same, regardless of the grouping of the variables.

The associative law of multiplication is given by,

(B. C) = (A.B) . C

It makes no difference in what order the variables are grouped during the AND operation of several variables.

3. *Distributive property:*

The Boolean addition is distributive over Boolean multiplication, given by

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

The Boolean addition is distributive over Boolean addition, given by

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

4. *Principle of Duality:*

Principle of Duality theorem says,

- Changing each OR sign to an AND sign
- Changing each AND sign to an OR sign
- Complementing any 0 or 1 appearing in the expression

Dual of relation A+A' = 1 is **A. A' = 0**

5. *DeMorgan's Theorems:*

Two theorems that are an important part of Boolean algebra were proposed by DeMorgan.

The first theorem states that the complement of a product is equal to the sum of the complements.

(AB)' = A'+ B'

The second theorem states that the complement of a sum is equal to the product of the complements.

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

6. *Consensus Theorem:*

In simplification of Boolean expression, an expression of the form **AB+ A'C+ BC**, the term BC is redundant and can be eliminated to form the equivalent expression AB+ A'C. The theorem used for this simplification is known as consensus theorem and is stated as,

AB+ A'C+ BC = AB+ A'C

AB+ A'C+ BC =AB+A'C+BC(A+A')

=AB+A'C+AB+A'C

=AB+A'C

The dual form of consensus theorem is stated as,

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

Boolean Expression:

Boolean expressions are minimized by using Boolean laws and postulates.

Minimization of Boolean Expressions

Q7 a) Simplify the Boolean expression

(Or)

b) Simplify the following Boolean expressions to a minimum number of literals

1. x (x'+y)

= xx'+ xy [x. x'= 0]

= 0 + xy [x+ 0 = x]

= xy.

2. x+ x'y

= x + xy + x'y [x+ xy= x]

= x+ y (x+x')

= x+ y (1) [x+ x' = 1]

= x+ y.

3. (x+ y) (x+ y')

= x.x+ xy'+ xy+ yy'

= x+ xy'+ xy+ 0 [x. x= 0]; [y. y'= 0]

= x (1+ y'+ y)

= x (1) [1+y= 1]

= x.

4. xy + x'z + yz.

= xy + x'z + yz(x+ x') [x+ x'= 1]

= xy + x'z + xyz + x'yz Re-arranging,

= xy + xyz + x'z +x'yz

= xy (1+ z) + x'z (1+y) [1+y= 1]

= xy+ x'z.

5. xy+ yz+ y'z

= xy+ z (y+ y')

= xy+ z (1) [y+ y' = 1]

= xy+ z.

6. (x+ y) (x'+ z) (y+ z)

= (x+ y) (x'+ z) [dual form of consensus theorem,

(A+ B) (A'+ C) (B+ C) = (A+ B) (A'+ C)]

7. x'y+ xy+ x'y'

= y (x'+ x) + x'y' [x (y+ z) = xy+ xz]

= y (1) + x'y' [x+ x' = 1]

= y+ x'y' [x+ x'y' = x+ y']

= y+ x'.

8. x+ xy'+ x'y

= x (1+ y')+ x'y

= x (1) + x'y [1+ x = 1]

= x+ x'y [x+ x'y = x+ y]

= x+ y.

9. AB + (AC)' + AB'C (AB + C)

= AB + (AC)' + AAB'BC + AB'CC

= AB + (AC)' + 0+ AB'CC [B.B' = 0]

= AB + (AC)' + AB'C [C.C = 1]

= AB + A' + C' +AB'C [(AC)' = A' + C']

= AB + A' + C' + AB' [C' + AB'C = C' + AB']

= A' + B+ C'+ AB' [A' + AB = A' + B]

Re- arranging,

= A' + AB'+ B+ C' [A' + AB = A' + B]

= A' + B'+ B+ C' [B'+ B= 1]

= A' +1+ C' [A+ 1= 1]

= 1

10. (x'+ y) (x+ y)

= x'.x+ x'y+ yx+ y.y

= 0+ x'y+ xy+ y [x.x'= 0]; [x. x= x]

= y (x'+ x+ 1)

= y(1) [1+ x = 1]

= y.

11. xy+ xyz+ xy (w+ z)

= xy (1+ z+ w+ z)

= xy (1) [1+ x = 1]

= xy.

12. xy+ xyz+ xyz'+ x'yz

= xy (1+ z+ z')+ x'yz

= xy (1) + x'yz [1+ x = 1]

= xy+ x'yz

= y (x+ x'z) [x+ x'y = x+ y]

= y (x+ z).

13. xyz+ xy'z+ xyz'

= xy (z+ z') + xy'z

= xy+ xy'z [x+ x'= 1]

= x(y+ y'z) [x+ x'y = x+ y]

= x(y+ z)

14. x'y'z'+ x'yz'+ xy'z'+ xyz'

= x'z' (y'+ y)+ xz' (y'+ y)

= x'z'+ xz' [x+ x'= 1]

= z' (x'+ x)

= z' [x+ x'= 1]

15. w'xyz'+ xyz'+ xy'z'+ xy'z

= xyz' (w'+ 1) + xy'z'+ xy'z

= xyz'+ xy'z'+ xy'z [1+ x = 1]

= xz' (y+ y') + xy'z

= xz'+ xy'z [x+ x'= 1]

= x (z'+ y'z)

= x (z'+ y'). [x'+ xy' = x'+ y']

16. w'xy'z+ w'xyz+ wxz

= w'xz (y'+ y)+ wxz

= w'xz (1)+ wxz [x+ x'= 1]

= w'xz+ wxz

= xz (w'+ w)

= xz. [x+ x'= 1]

17. x'y'z'+ x'y'z+ x'yz'+ x'yz+ xy'z'

= x'y' (z'+z) + x'y (z'+z)+ xy'z'

= x' y' (1) + x'y (1)+ xy'z' [x+ x'= 1]

= x'y' + x'y + xy'z'

= x'(y'+y) + xy'z'

= x' (1) + xy'z' [x+ x'= 1]

= x' + xy'z'

= x'+ y'z'. [x'+ xy' = x'+ y']

18. w'y (w'xz)' + w'xy'z' + wx'y

= w'y (w''+ x'+ z') + w'xy'z' + wx'y

= w'y (w+ x'+ z') + w'xy'z' + wx'y [x'' = x]

= w'yw+ w'y x'+ w'y z' + w'xy'z' + wx'y

= 0 + w'x'y+ w'y z' + w'xy'z' + wx'y [x. x'= 0]

Re-arranging,

= w'x'y+ wx'y + w'y z' + w'xy'z'

= x'y (w'+ w) + w'z' (y+ xy')

= x'y (1) + w'z' (y+ xy') [x+ x'= 1]

= x'y+ w'z' (y+x) [x+ x'y = x+ y]

19. xy+ x (y+ z) + y (y+ z)

= xy+ xy+ xz+ yy+ yz

= xy+ xz+ y+ yz [x+ x= x]; [x. x= x]

= xy+ xz+ y [x+ xy= x]

= y+ xz [x+ xy= x]

20. [xy' (z+ wy) + x'y'] z

= [xy'z+ xy'wy+ x'y'] z

= [xy'z+ 0+ x'y'] z [x. x'= 0]

= xy'z. z+ x'y'z

= xy'z+ x'y'z [x. x= x]

= y'z (x+ x')

= y'z (1) [x+ x'= 1]

= y'z.

21. x'yz+ xy'z'+ x'y'z'+ xy'z+ xyz

= yz (x'+x) + xy'z'+ x'y'z'+ xy'z

= yz (1) + y'z' (x+ x') + xy'z [x+ x'=1]

= yz+ y'z' (1) + xy'z [x+ x'=1]

= yz+ y'z'+ xy'z

= yz+ y' (z'+ xz)

= yz+ y' (z'+ x) [x'+ xy = x'+ y]

= yz+ y'z'+ xy'

22. [(xy)'+ x'+ xy]'

= [x'+ y'+ x'+ xy]'

= [x'+ y'+ xy]' [x+ x= x]

= [x'+ y'+ x]' [x'+ xy = x'+ y]

= [y'+ 1]' [x+ x'= 1]

= [1]' [1+ x = 1]

= 0.

23. [xy+ xz]'+ x'y'z

= (xy)'. (xz)'+ x'y'z

= (x'+ y'). (x'+ z')+ x'y'z

= x'x'+ x'z'+ x'y'+ y'z'+ x'y'z

= x'+ x'z'+ x'y'+ y'z'+ x'y'z [x+ x= x]

= x'+ x'z'+ x'y'+ y' [z'+ x'z]

= x'+ x'z'+ x'y'+ y' [z'+ x'] [x'+ xy = x'+ y]

= x'+ x'y'+ y' [z'+ x'] [x+ xy = x]

= x'+ x'y'+ y'z'+ x'y'

= x'+ y'z'+ x'y' [x+ xy = x]

= x'+ y'z'. [x+ xy = x]

24. xy+ xy'(x'z')'

= xy+ xy' (x''+ z'')

= xy+ xy' (x+ z) [x'' = x]

= xy+ xy'x+ xy'z

= xy+ xy'+ xy'z [x. x= x]

= xy+ xy' [1+ z]

= xy+ xy' [1] [1+ x = 1]

= xy+ xy'

= x(y+ y')

= x [1] [x+ x'= 1]

= x.

25. [(xy'+ xyz)'+ x (y+ xy')]'

= [x(y'+yz)'+ x (y+ xy')]'

= [x(y'+z)'+ x (y+ x)]' [x'+ xy = x'+ y]; [x+ x'y = x+ y]

= [x(y'+z)'+ xy+ x.x)]'

= [(xy'+xz)'+ xy+ x)]' [x. x= x]

= [(xy'+xz)'+ x)]' [x+ xy = x]

= [(xy')'. (xz)'+ x]'

= [(x'+y''). (x'+z')+ x]'

= [(x'+y). (x'+z')+ x]' [x'' = x]

= [(x'+ yz')+ x]' [(x+ y) (x+ z)= x+ yz]

= [x'+ yz'+ x]'

= [1+ yz']' [x+ x'= 1]

= [1]' [1+ x = 1]

= 0.

26. [(xy+ z') ((x+ y)'+z)]'

= [(xy+ z') ((x'. y')+z)]'

= [xy. x'y'+ xy. z+ z'. x'y'+ z'. z]'

= [0+ xyz+ x'y'z'+ 0]' [x. x'= 0]

= [xyz+ x'y'z']'

= (xyz)'. (x'y'z')'

= (x'+ y'+ z'). (x''+ y''+ z'')

= (x'+ y'+ z'). (x+ y+ z). [x'' = x]

27. (x+ y) (x'z'+ z) (y'+ xz)'

= (x+ y) (x'z'+ z) (y''. (xz)')

= (x+ y) (x'+ z) (y. (xz)') [x+ x'y = x+ y]; [x'' = x]

= (x+ y) (x'+ z) (y. (x'+z'))

= (x.x'+ xz+ x'y+ yz) (x'y+ yz')

= (0+ xz+ x'y+ yz) (x'y+ yz')

= (xz+ x'y+ yz) (x'y+ yz')

= xz. x'y+ xz. yz'+ x'y. x'y+ x'y. yz'+ yz. x'y+ yz. yz'

= 0+ 0+ x'y+ x'yz'+ x'yz+ 0 [x. x'= 0]; [x. x= x]

= x'y+ x'yz'+ x'yz

= x'y (1+ z'+ z)

= x'y (1) [1+ x = 1]

= x'y.

28. Y= Σm (1, 3, 5, 7)

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

= x'z(y'+y) + xz(y'+y)

= x'z (1)+ xz (1) [x+ x'= 1]

= x'z+ xz

= z(x'+ x)

= z (1) [x+ x'= 1]

= z.

CANONICAL AND STANDARD FORMS:

Minterms and Maxterms:

Q8 a) What do you mean by minterms and maxterms

(Or)

b) What do you mean by standard SOP and POS FORMS?

A binary variable may appear either in its normal form (x) or in its complement form (x'). Now either two binary variables x and y combined with an AND operation. Since each variable may appear in either form, there are four possible combinations:

x'y', x'y, xy' and xy

Each of these four AND terms is called a '***minterm***'.

In a similar fashion, when two binary variables x and y combined with an OR operation, there are four possible combinations:

x'+ y', x'+ y, x+ y' and x+ y

Each of these four OR terms is called a '***maxterm***'.

The minterms and maxterms of a 3- variable function can be represented as in table below.

Variables (xyz)	Minterms(mi)	Maxterms(Mi)
000	x'y'z' = m0	x+ y+ z=M0
001	x'y'z = m1	x+ y+ z'= M1
010	x'yz' = m2	x+ y'+ z= M2
011	x'yz = m3	x+ y'+ z'= M3
100	xy'z' = m4	x'+ y+ z= M4
101	xy'z = m5	x'+ y+ z'= M5
110	xyz' = m6	x'+ y'+ z= M6
111	xyz = m7	x'+ y'+ z'= M7

Sum of Minterm: (Sum of Products)

The logical sum of two or more logical product terms is called sum of products expression. It is logically an OR operation of AND operated variables such as:

1. $Y = AB + BC + AC$

2. $Y = AB + \overline{B}C + A\overline{C}$

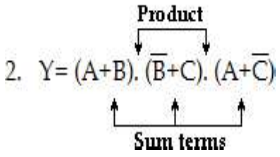
Sum

Product terms

Sum of Maxterm: (Product of Sums)

A product of sums expression is a logical product of two or more logical sum terms. It is basically an AND operation of OR operated variables such as,

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



Canonical Sum of product expression:

If each term in SOP form contains all the literals then the SOP is known as standard (or) canonical SOP form. Each individual term in standard SOP form is called minterm canonical form.

F (A, B, C) = AB'C+ ABC+ ABC'

Steps to convert general SOP to standard SOP form:

1. Find the missing literals in each product term if any.
2. AND each product term having missing literals by ORing the literal and its complement.
3. Expand the term by applying distributive law and reorder the literals in the product term.
4. Reduce the expression by omitting repeated product terms if any.

Q9 a) Obtain the canonical SOP form of the function

(Or)

b) Convert the given expression in standard SOP form

1. Y (A, B) = A+ B

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

= AB+ AB'+ AB+ A'B

= AB+ AB'+ A'B.

2. Y (A, B, C) = A+ ABC

= A. (B+ B'). (C+ C')+ ABC

= (AB+ AB'). (C+ C')+ ABC

= ABC+ ABC'+ AB'C+ AB'C'+ ABC

= ABC+ ABC'+ AB'C+ AB'C'

= m7+ m6+ m5+ m4

3. Y (A, B, C) = A+ BC

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

= (AB+ AB'). (C+ C')+ ABC+ A'BC

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

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

= m7+ m6+ m5+ m4+ m3

= Σm (3, 4, 5, 6, 7).

4. Y (A, B, C) = AC+ AB+ BC

= AC (B+ B') + AB (C+ C')+ BC (A+ A')

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

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

= Σm (3, 5, 6, 7).

5. Y (A, B, C, D) = AB+ ACD

= AB (C+ C') (D+ D') + ACD (B+ B')

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

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

= ABCD+ ABCD'+ ABC'D+ ABC'D'+ AB'CD.

Canonical Product of sum expression:

If each term in POS form contains all literals then the POS is known as standard (or) Canonical POS form. Each individual term in standard POS form is called Maxterm canonical form.

- F (A, B, C) = (A+ B+ C). (A+ B'+ C). (A+ B+ C')
- F (x, y, z) = (x+ y'+ z'). (x'+ y+ z). (x+ y+ z)

Steps to convert general POS to standard POS form:

1. Find the missing literals in each sum term if any.
2. OR each sum term having missing literals by ANDing the literal and its complement.
3. Expand the term by applying distributive law and reorder the literals in the sum term.
4. Reduce the expression by omitting repeated sum terms if any.

Q10 a) Obtain the canonical POS expression of the functions:

(Or)

b) Convert the given expression in standard POS form

1. **Y= A+ B'C**

= (A+ B') (A+ C) [**A+ BC = (A+B) (A+C)**]

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

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

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

= M2. M3. M0

= ΠM (0, 2, 3)

2. **Y= (A+B) (B+C) (A+C)**

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

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

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

= M0. M1. M4. M2

= ΠM (0, 1, 2, 4)

3. **Y= A. (B+ C+ A)**

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

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

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

= M0. M1. M2. M3

= ΠM (0, 1, 2, 3)

4. **Y= (A+B') (B+C) (A+C')**

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

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

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

= M2. M3. M0. M4. M1

= ΠM (0, 1, 2, 3, 4)

5. **Y= xy+ x'z**

= (xy+ x') (xy+ z) Using distributive law, convert the function into OR terms.

= (x+x') (y+x') (x+z) (y+z) [x+ x'=1]

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

= (x'+y+ z.z') (x+z+y.y') (y+z+ x.x')

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

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

= M4. M5. M0. M2

= ΠM (0, 2, 4, 5).

KARNAUGH MAP MINIMIZATION:

Q11a) Explain in detail about karnaugh map

(Or)

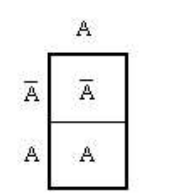
b) Write notes on veitch diagram

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.

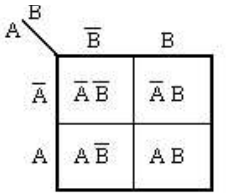
The Karnaugh map technique provides a systematic method for simplifying and manipulation of Boolean expressions. **A** K-map is a diagram made up of squares, with each square representing one minterm of the function that is to be minimized. For *n* variables on a Karnaugh map there are 2n numbers of 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

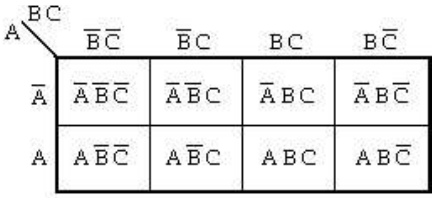
Karnaugh maps can be used for expressions with two, three, four and five variables. The number of cells in a Karnaugh map is equal to the total number of possible input variable combinations as is the number of rows in a truth table. For three variables, the number of cells is 23 = 8. For four variables, the number of cells is 24 = 16.



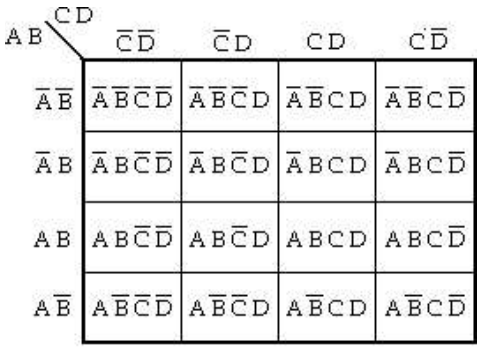
1-Variable map



2-Variable map



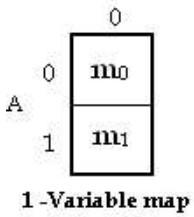
3-Variable map



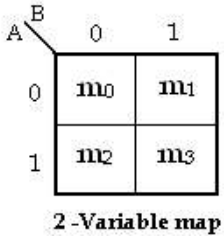
4 -Variable map

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.

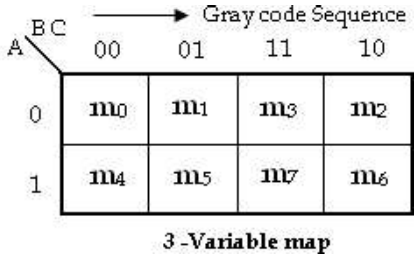
It is important to note that when we move from one cell to the next along any row or from one cell to the next along any column, one and only one variable in the product term changes (to a complement or to an uncomplemented form). Irrespective of number of variables the labels along each row and column must conform to a single change. Hence gray code is used to label the rows and columns of K-map as shown below.



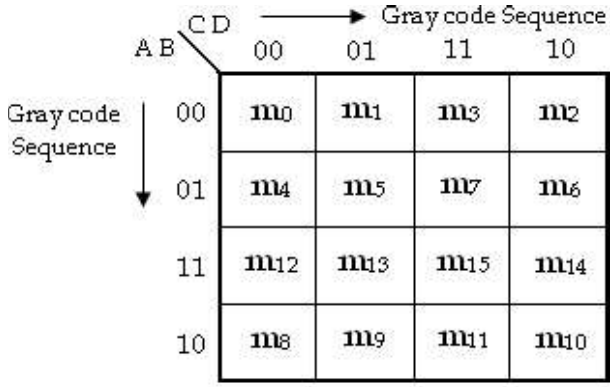
1 -Variable map



2 -Variable map



3 -Variable map



4- Variable map

Simplification of Sum of Products Expressions:

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 makes sure that there is 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.

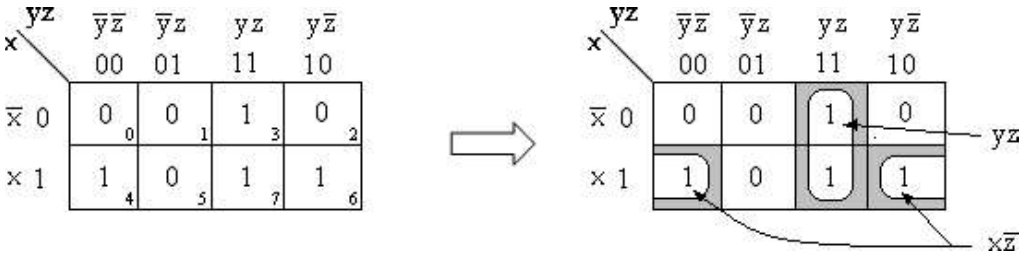
Q12 a) Simplify the Boolean function using 3 variable k map

(Or)

b) Simplify the logical function using 3 variable k map.

1. **F(x, y, z) = Σm (3, 4, 6, 7).**

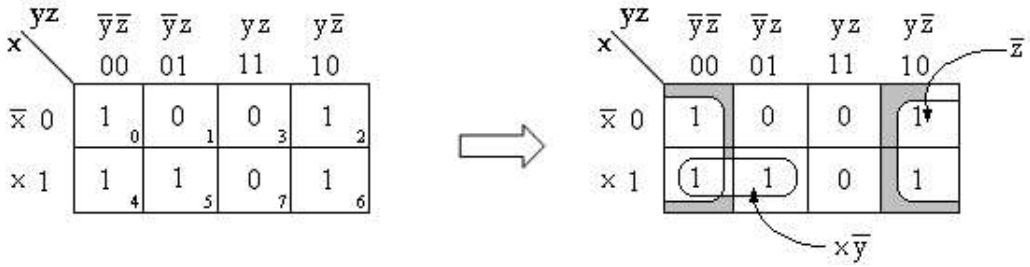
Soln:



F = yz+ xz'

2. **F(x, y, z) = Σm (0, 2, 4, 5, 6).**

Soln:



F = z'+ xy'

Q13 a) Plot Boolean expression on the k map

(Or)

b) Simplify the logical expression using3 variable k-map

F = A'C + A'B + AB'C + BC

Soln:

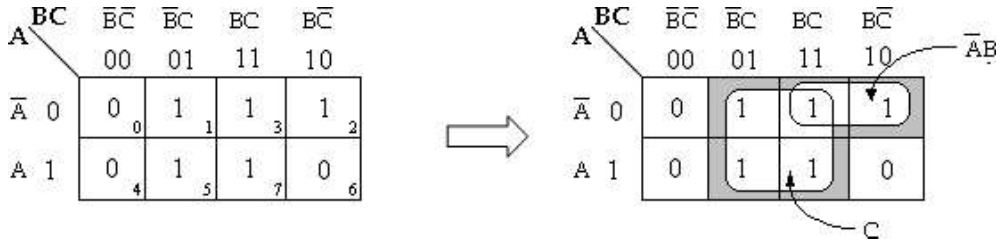
= A'C (B+ B') + A'B (C+ C') + AB'C + BC (A+ A')

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

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

= m3+ m1+ m2+ m5+ m7

= Σ m (1, 2, 3, 5, 7)



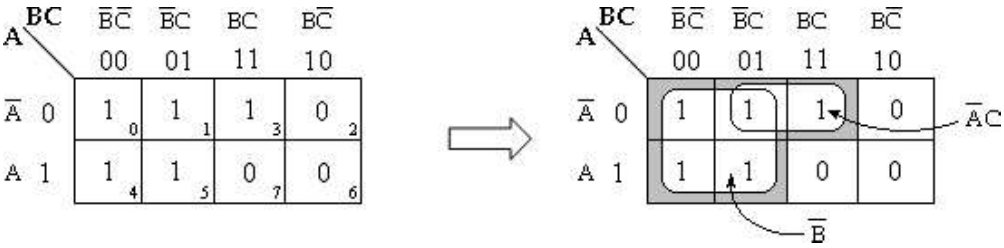
F = C + A'B

AB'C + A'B'C + A'BC + AB'C' + A'B'C'

Soln:

= m5 + m1 + m3 + m4 + m0

= Σ m (0, 1, 3, 4, 5)



F = A'C + B'

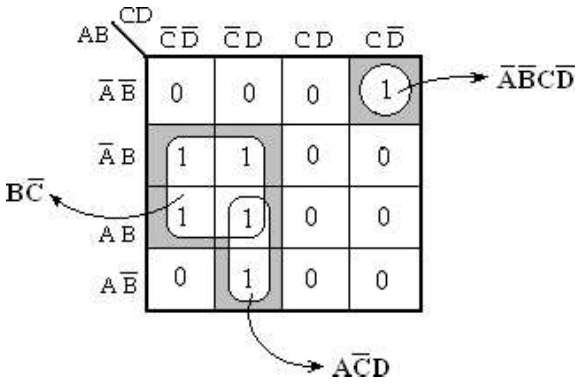
Q14 a) Simplify the Boolean expression using 4 variable k map

(Or)

b) Reduce the switching function using kmap

Y = A'BC'D' + A'BC'D + ABC'D' + ABC'D + AB'C'D + A'B'CD'

Soln:



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

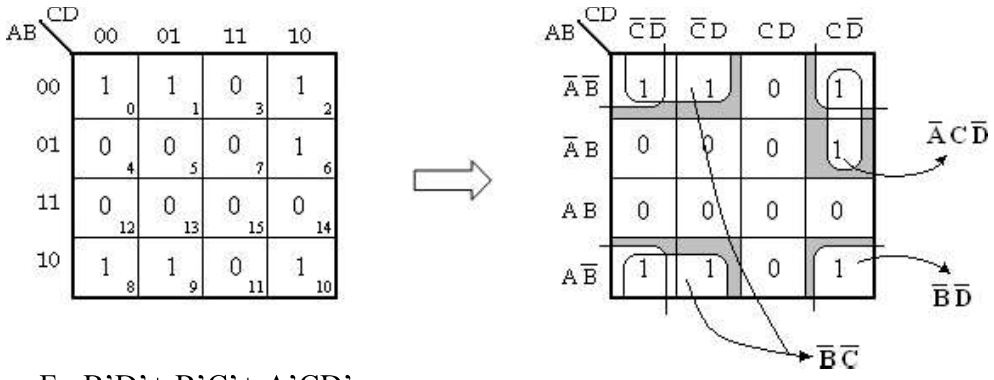
F= A'B'C'+ B'CD'+ A'BCD'+ AB'C'

= 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'

= m1+ m0+ m10+ m2+ m6+ m9+ m8

= Σ m (0, 1, 2, 6, 8, 9, 10)



Therefore, F= B'D'+ B'C'+ A'CD'.

Y= ABCD+ AB'C'D'+ AB'C+ AB

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

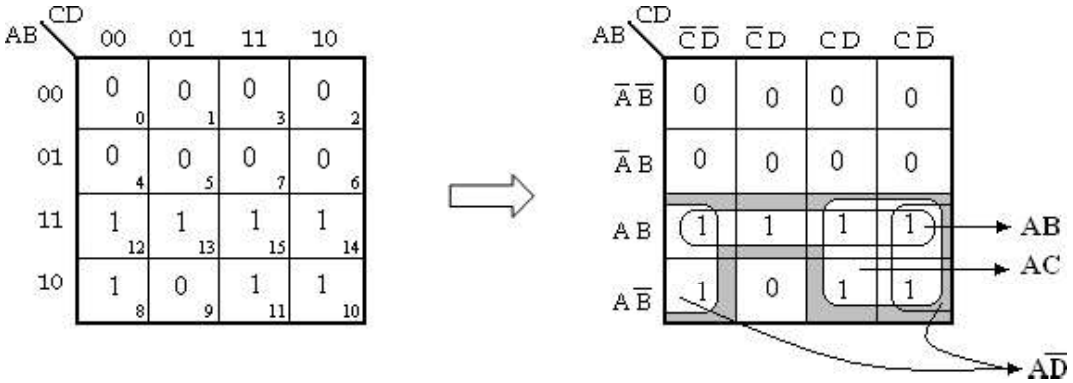
= ABCD+ AB'C'D'+ AB'CD+ AB'CD'+ (ABC+ ABC') (D+ D')

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

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

= m15+ m8+ m11+ m10+ m14+ m13+ m12

= Σ m (8, 10, 11, 12, 13, 14, 15)

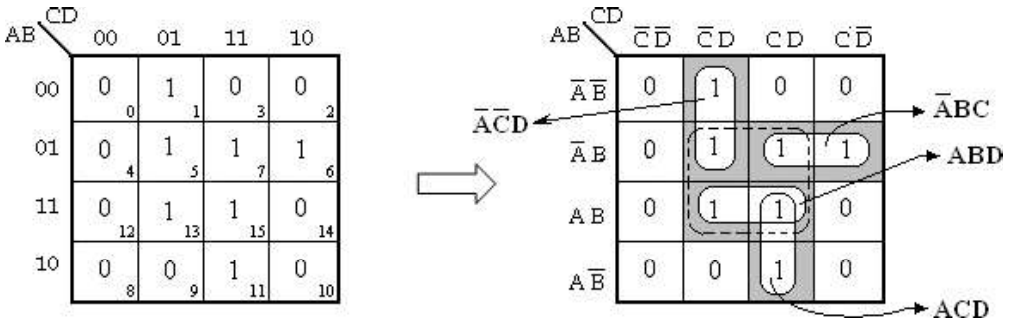


Therefore, Y= AB+ AC+ AD'

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

= m1+ m5+ m7+ m6+ m13+ m15+ m11

= Σ m (1, 5, 6, 7, 11, 13, 15)



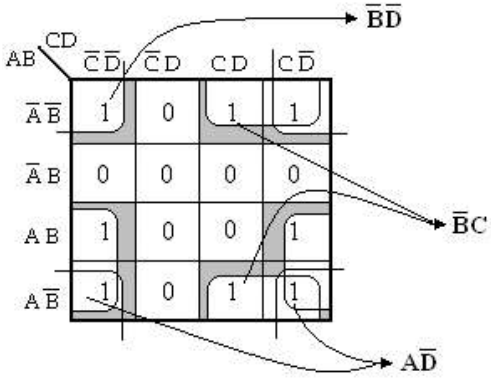
In the above K-map, the cells 5, 7, 13 and 15 can be grouped to form a quad as indicated by the dotted lines. In order to group the remaining 1's, four pairs have to be formed. However, all the four 1's covered by the quad are also covered by the pairs. So, the quad in the above k-map is redundant.

Therefore, the simplified expression will be,

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

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

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



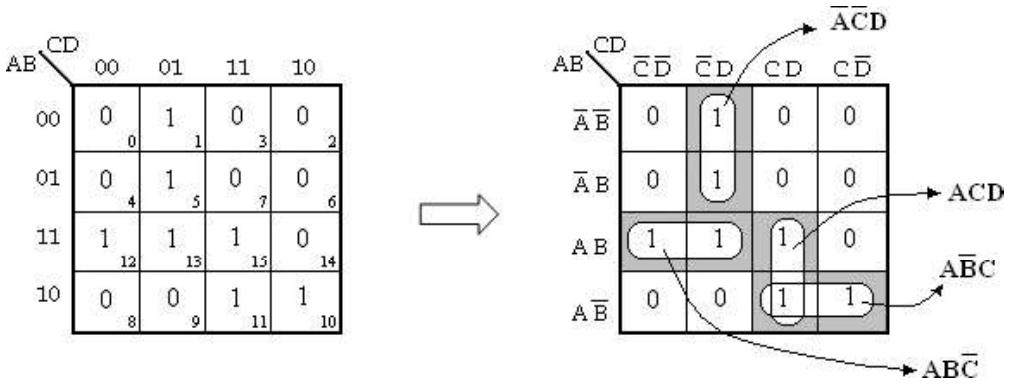
Therefore, Y= AD'+ B'C+ B'D'

Q15 a) Simplify the Boolean function using 4 variable k map

(Or)

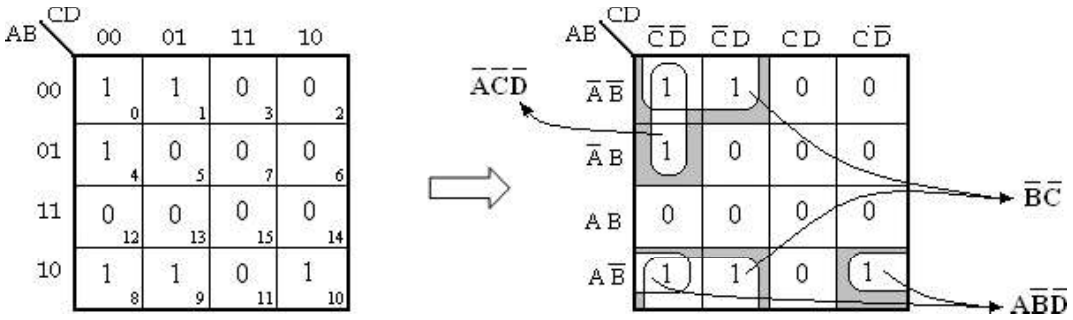
b) Simplify the logical function using 4 variable k map

Y= Σ m (1, 5, 10, 11, 12, 13, 15)



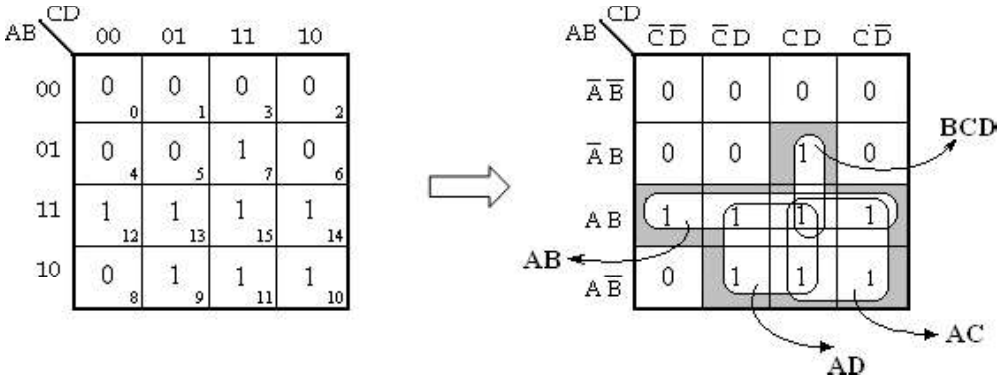
Therefore, Y= A'C'D+ ABC'+ ACD+ AB'C.

F (A, B, C, D) = Σ m (0, 1, 4, 8, 9, 10)



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

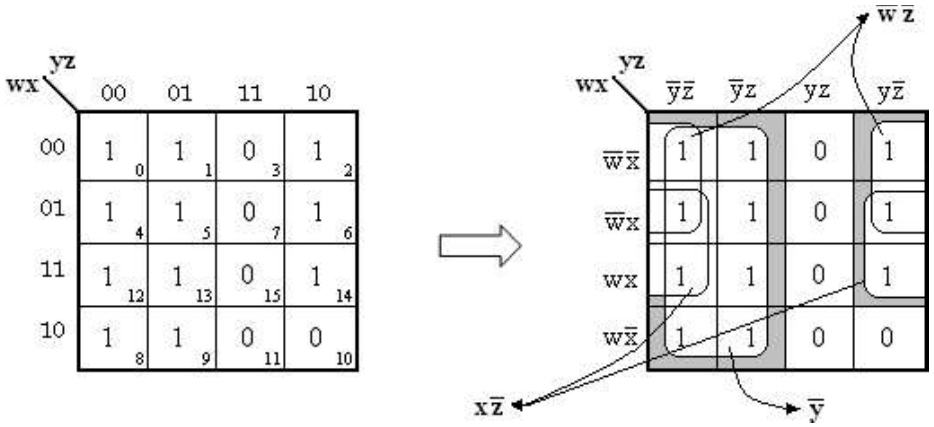
Y (A, B, C, D)= Σ m (7, 9, 10, 11, 12, 13, 14, 15)



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

F (w, x, y, z) = Σm (0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)

Soln:



Therefore, F= y'+ w'z'+ xz'

Simplification of Products of Sum Expressions:

The generalized procedure to simplify Boolean expressions as follows:

1. Plot the K-map and place 1's in those cells corresponding to the 0's in the product of sum expression. Place 1's in the other cells.
2. Check the K-map for adjacent 0's and encircles those 0's which are not adjacent to any other 0's. These are called **isolated 1's**.
3. Check for those 0's which are adjacent to only one other 1 and encircle such **pairs**.
4. Check for **quads** and **octets** of adjacent 0's even if it contains some 0'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 0's that have not yet been grouped.
6. Form the simplified expression by summing product terms of all the groups.

Q16 a) Minimize the following expression in the POS form

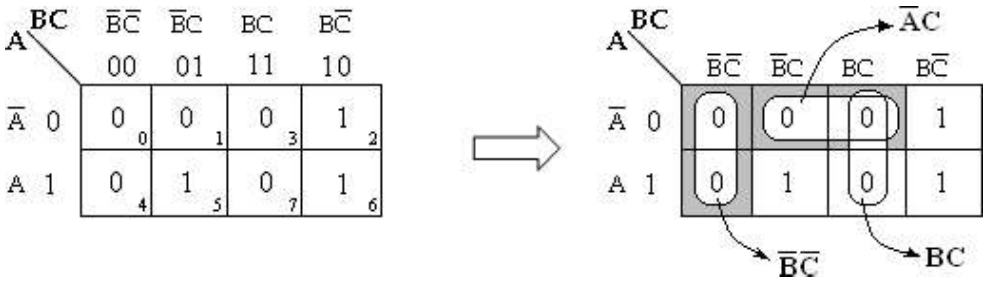
(Or)

b) Simplify the Boolean expression in POS form

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

= M1. M3. M7. M4. M0

= Π M (0, 1, 3, 4, 7)



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

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

= (B'C')' . (A'C)'. (BC)'

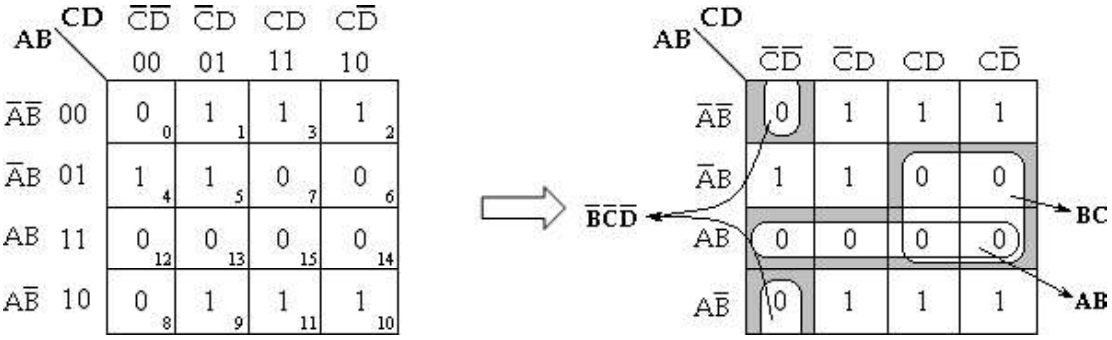
= (B"+ C") . (A"+C') . (B'+ C')

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

= M12. M14. M15. M8. M6. M7. M0. M13

= ΠM (0, 6, 7, 8, 12, 13, 14, 15)



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

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

= (B'C'D')'. (AB)'. (BC)'

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

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

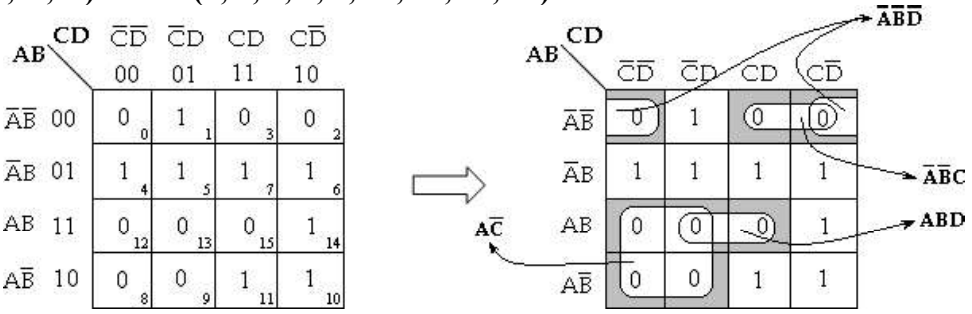
Therefore, Y= (B+ C+ D). (A'+ B'). (B'+ C')

Q16a)Minimize the following function in the POS form

(Or)

b) Simplify the Boolean function in POS form

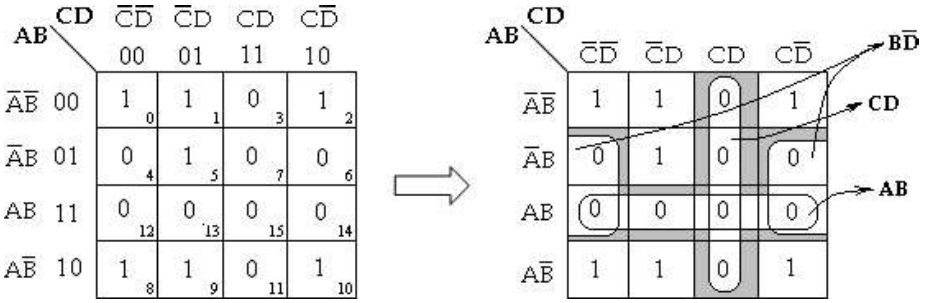
1. F(A, B, C, D) = Π M(0, 2, 3, 8, 9, 12, 13, 14, 15)



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)

2. F(A, B, C, D) = Σm (0, 1, 2, 5, 8, 9, 10)

= ΠM (3, 4, 6, 7, 11, 12, 13, 14, 15)



Y' = BD'+ CD+ AB
Y= Y" = (BD'+ CD+ AB)'
= (BD')'. (CD)'. (AB)'
= (B'+ D"). (C'+ D'). (A'+ B')
= (B'+ D). (C'+ D'). (A'+ B')
Therefore, Y= (B'+ D). (C'+ D'). (A'+ B')

Don't care Conditions:

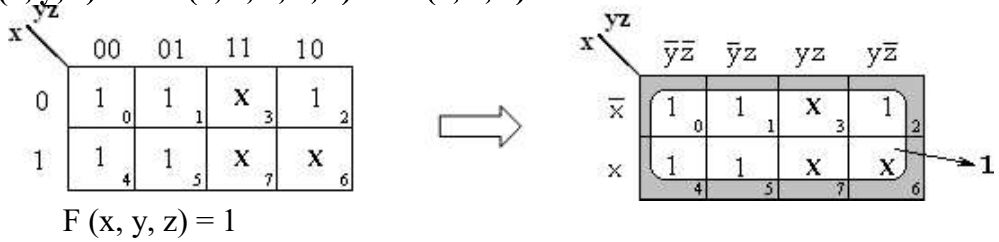
A don't care minterm is a combination of variables whose logical value is not specified. When choosing adjacent squares to simplify the function in a map, the don't care minterms may be assumed to be either 0 or 1. When simplifying the function, we can choose to include each don't care minterm with either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

Q17 a) Simplify the Boolean function using k map

(or)

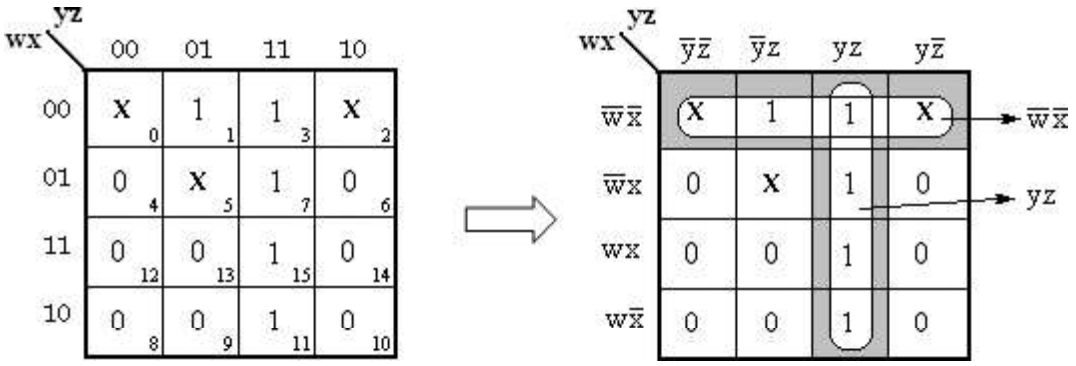
b) Reduce the following function using kmap technique

1. F (x, y, z) = Σm (0, 1, 2, 4, 5)+ Σd (3, 6, 7)



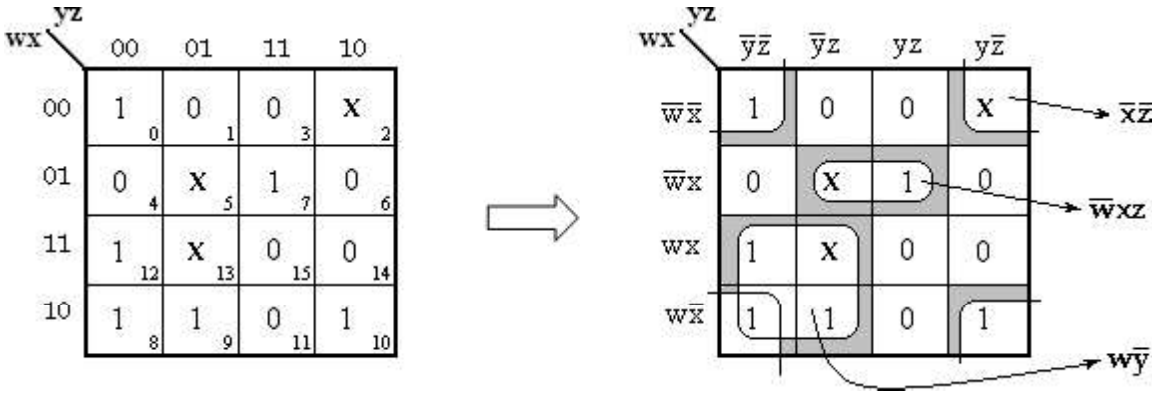
F (x, y, z) = 1

2. F (w, x, y, z) = Σm (1, 3, 7, 11, 15)+ Σd (0, 2, 5)



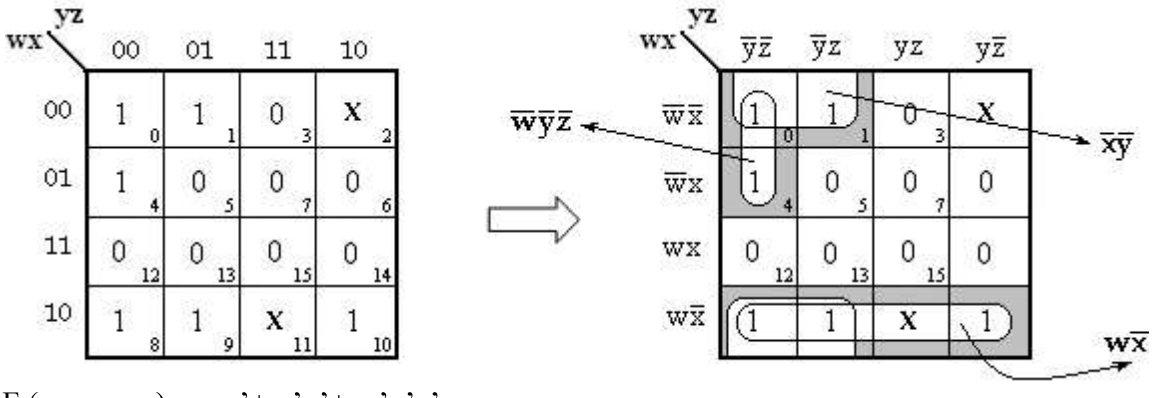
F (w, x, y, z) = w'x'+ yz

3. F (w, x, y, z) = Σm (0, 7, 8, 9, 10, 12)+ Σd (2, 5, 13)



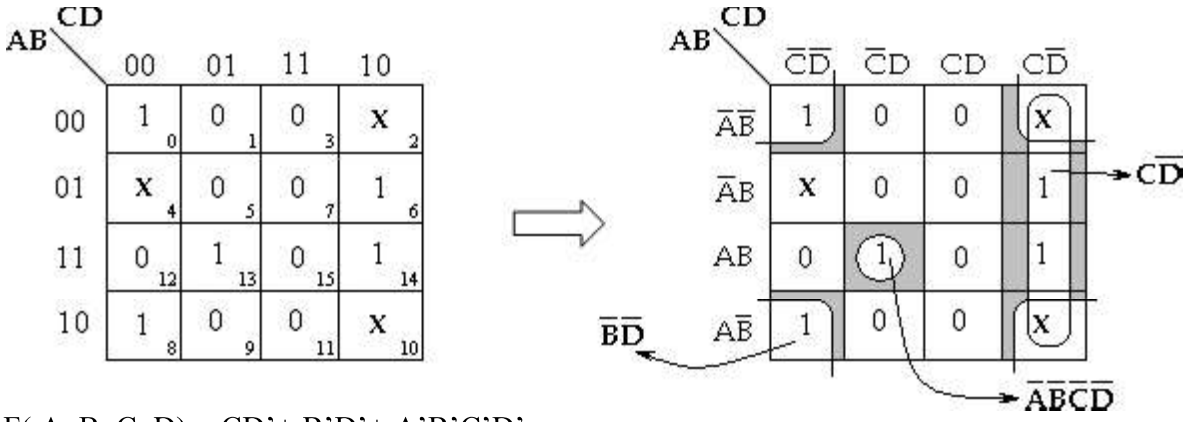
F (w, x, y, z) = w'xz+ wy'+ x'z'.

4. F (w, x, y, z) = Σm (0, 1, 4, 8, 9, 10)+ Σd (2, 11)



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

4. F(A, B, C, D) = Σm (0, 6, 8, 13, 14)+ Σd (2, 4, 10)



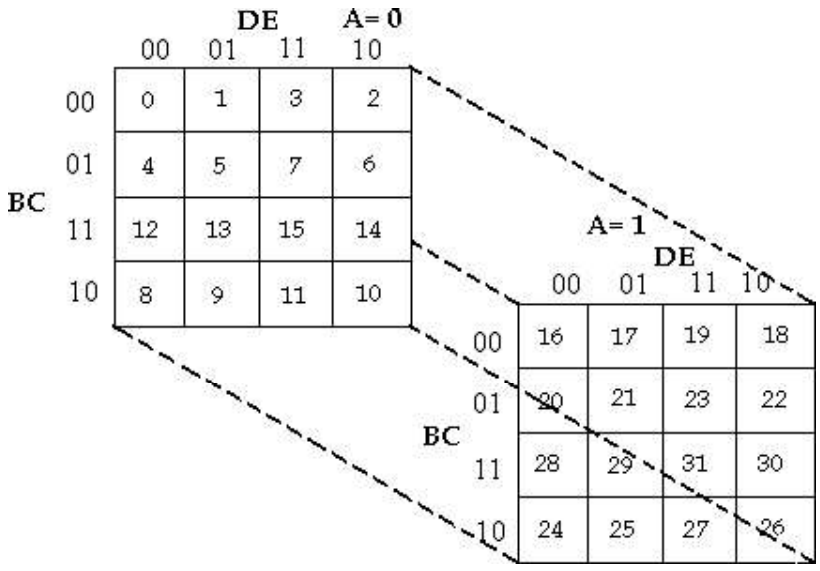
F(A, B, C, D) = CD'+ B'D'+ A'B'C'D'.

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.



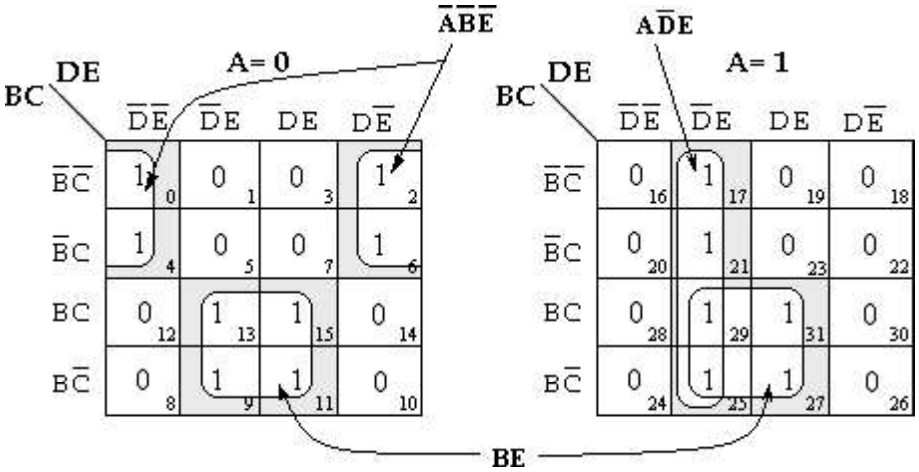
Five- Variable Karnaugh map (Layer Structure)

Q18a) Simplify the Boolean function

(or)

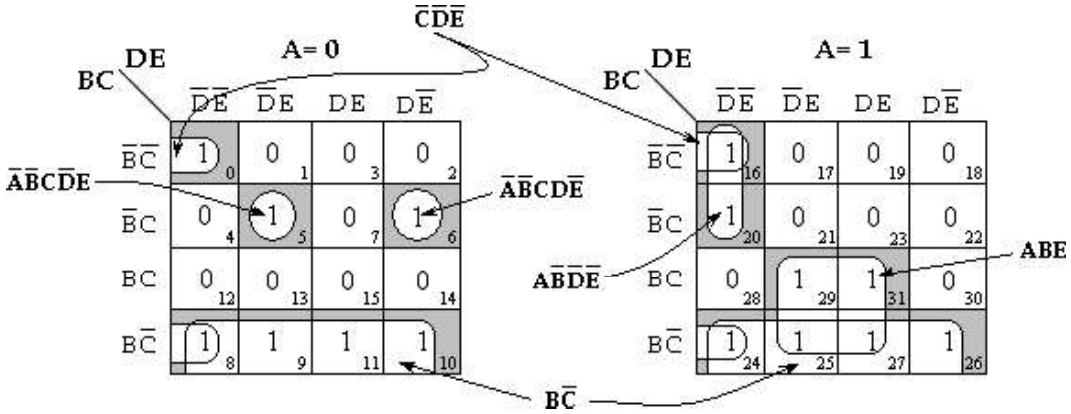
b) Reduce the function using kmap technique

1. F (A, B, C, D, E) = Σm (0, 2, 4, 6, 9, 11, 13, 15, 17, 21, 25, 27, 29, 31)



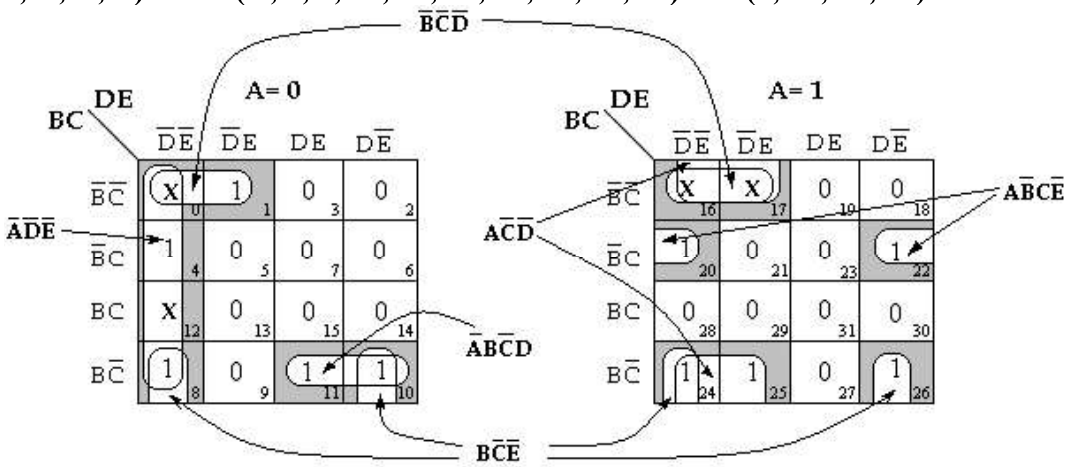
F (A, B, C, D, E) = A'B'E'+ BE+ AD'E

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



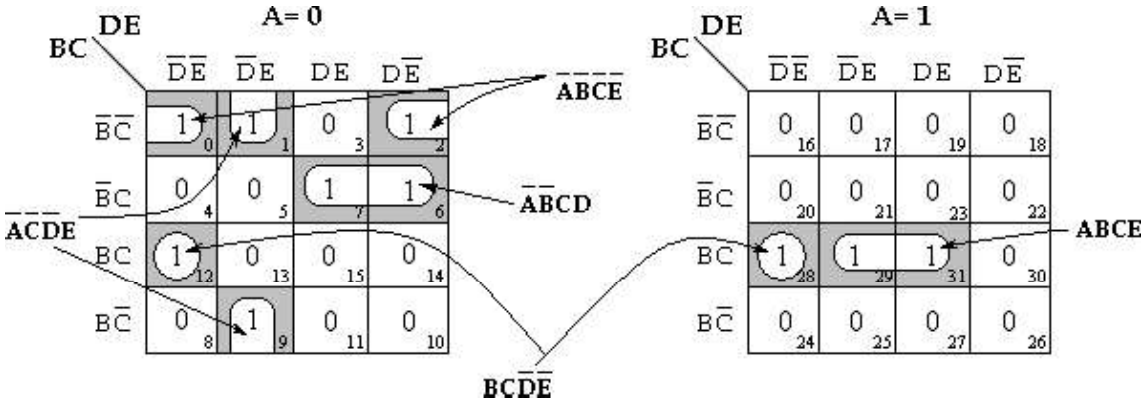
F (A, B, C, D, E) = C'D'E'+ A'B'CD'E+ A'B'CDE'+ AB'D'E'+ ABE+ BC'

3. F (A, B, C, D, E) = Σm (1, 4, 8, 10, 11, 20, 22, 24, 25, 26)+Σd (0, 12, 16, 17)



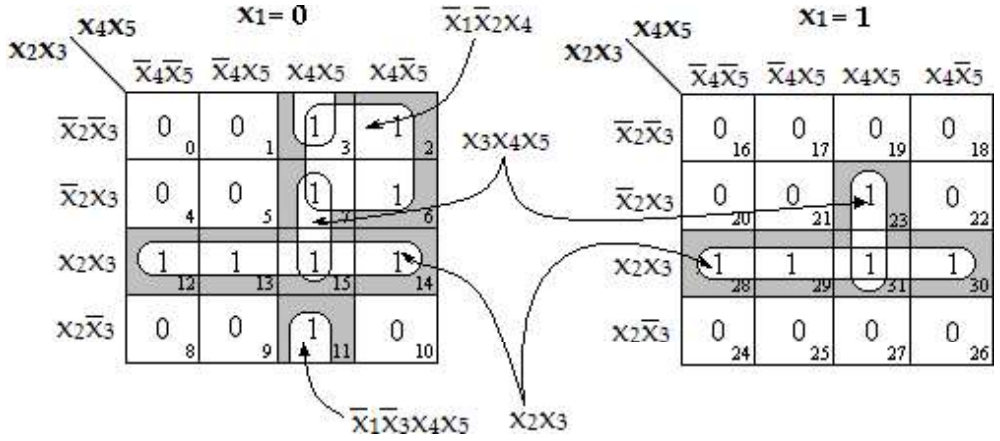
F (A, B, C, D, E) = B'C'D'+ A'D'E'+ BC'E'+ A'BC'D+ AC'D'+ AB'CE'

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



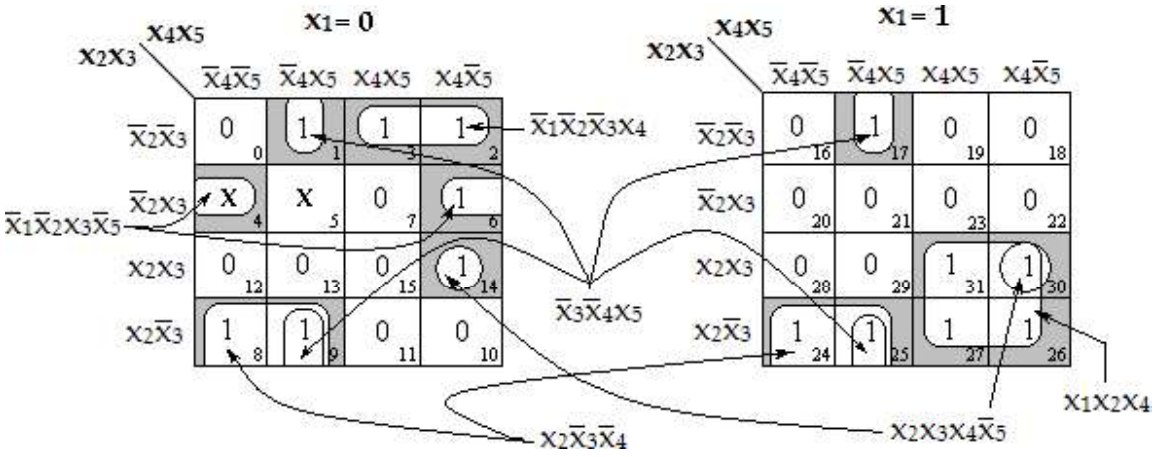
F (A, B, C, D, E) = BCD'E'+ ABCE+ A'B'C'E'+ A'C'D'E+ A'B'CD

5. F (x1, x2, x3, x4, x5) = Σm (2, 3, 6, 7, 11, 12, 13, 14, 15, 23, 28, 29, 30, 31)



F (x1, x2, x3, x4, x5) = x2x3+ x3x4x5+ x1'x2'x4+ x1'x3'x4x5

6. F (x1, x2, x3, x4, x5) = Σm (1, 2, 3, 6, 8, 9, 14, 17, 24, 25, 26, 27, 30, 31)+ Σd (4, 5)



F(x1, x2, x3, x4, x5) = x2x3'x4'+ x2x3x4x5'+ x3'x4'x5+ x1x2x4+ x1'x2'x3x5'+ x1'x2'x3'x4

LOGIC GATES

Q19 a) Write the logic symbol, expression and truthtable for the following logic gates

NOT , AND , OR , NAND ,NOR ,EX- OR ,EX- NOR.

(or)

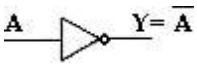
b)Explain in detail about basic logic gates:

Logic gates are electronic circuits that can be used to implement the most elementary logic expressions, also known as Boolean expressions. The logic gate is the most basic building block of combinational logic.

There are three basic logic gates, namely the OR gate, the AND gate and the NOT gate. Other logic gates that are derived from these basic gates are the NAND gate, the NOR gate, the EXCLUSIVE- OR gate and the EXCLUSIVE-NOR gate.

NOT GATE (7404)

SYMBOL



TRUTH TABLE

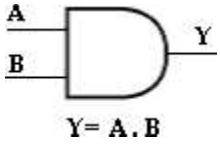
A	Y= $\overline{A}$
0	1
1	0

EXPRESSION

Y=A'

AND GATE (7408)

SYMBOL



TRUTH TABLE

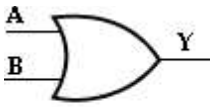
A	B	Y= A.B
0	0	0
0	1	0
1	0	0
1	1	1

EXPRESSION

Y = A . B

OR GATE (7432)

SYMBOL



Y= A+B

TRUTH TABLE

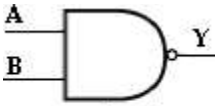
A	B	Y= A+B
0	0	0
0	1	1
1	0	1
1	1	1

EXPRESSION

Y = A + B

NAND GATE (7400)

SYMBOL



Y= $\overline{A.B}$

TRUTH TABLE

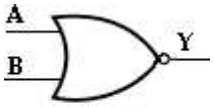
A	B	Y= $\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

EXPRESSION

Y=(A.B)'

NOR GATE (7402)

SYMBOL



Y= $\overline{A+B}$

TRUTH TABLE

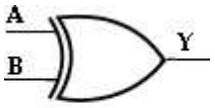
A	B	Y= $\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

EXPRESSION

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

EX- OR GATE (7486)

SYMBOL



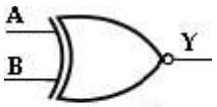
Y= A⊕B

TRUTH TABLE

A	B	Y= A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

EX- NOR GATE

SYMBOL



Y= $\overline{A⊕B}$

(or)

Y= A⊙B

TRUTH TABLE

A	B	Y= A⊙B
0	0	1
0	1	0
1	0	0
1	1	1

UNIVERSAL GATES:

The NAND and NOR gates are known as universal gates, since any logic function can be implemented using NAND or NOR gates. This is illustrated in the following sections.

a) **NAND Gate**:
The NAND gate can be used to generate the NOT function, the AND function, the OR function and the NOR function.

b) **NOR Gate**:
Similar to NAND gate, the NOR gate is also a universal gate, since it can be used to generate the NOT, AND, OR and NAND functions.

Q20 a) Steps to Convert AND/OR/NOT to NAND/NOR:

(or)

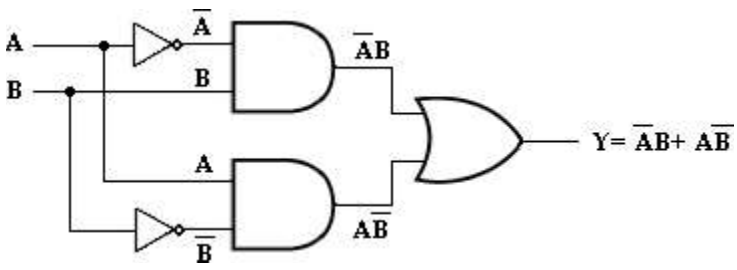
b) Rules for converting AND/OR/NOT to NAND/NOR:

1. Draw AND/OR logic.
2. If NAND hardware has been chosen, add bubbles on the output of each AND gate and bubbles on input side to all OR gates.
   If NOR hardware has been chosen, add bubbles on the output of each OR gate and bubbles on input side to all AND gates.
3. Add or subtract an inverter on each line that received a bubble in step 2.
4. Replace bubbled OR by NAND and bubbled AND by NOR.
5. Eliminate double inversions.

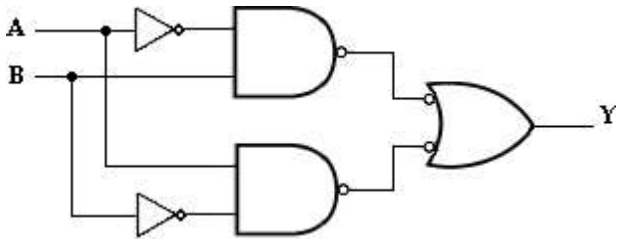
Q21 a) Implement Boolean expression for EX-OR gate using NAND gates.

(or)

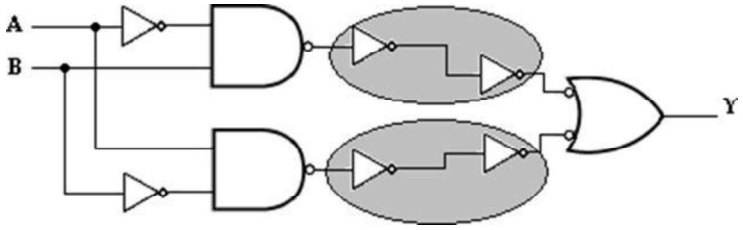
b) Implement the following Boolean function with NAND -NAND logic



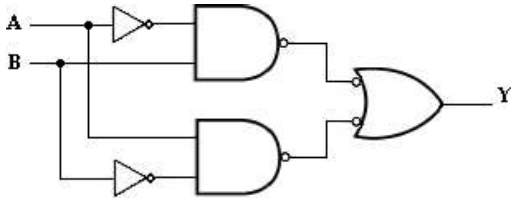
Adding bubbles on the output of each AND gates and on the inputs of each OR



Adding an inverter on each line that received bubble,



Eliminating double inversion,



Replacing inverter and bubbled OR with NAND, we have