UNIT I

BOOLEAN ALGEBRA AND LOGIC GATES

1.1 INTRODUCTION

The number systems are used quite frequently in the field of digital electronics and computers. Number systems of a given radix or base provide the means of quantifying information for processing by digital systems. There are several number systems but the following are the important ones in the field of digital electronics:

- Decimal number system
- Binary number system
- Octal number system
- Hexadecimal number system

1.2 DECIMAL NUMBER SYSTEM

The decimal number system has 10 numerals or symbols. These symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. The decimal system is also called the base-10 system because it has 10 digits. The position weights in decimal number system is shown in **Figure 1.1.**

10^3	10^2	10^1	10^0	•	10^{-1}	10^{-2}	10^{-3}	10^{-4}

Fig. 1.1: Position weights in binary number system

For example, the decimal number 183 represents one hundred, eight tens and three ones. Any number has two parts, one part is integer part and the other part is fractional part. The decimal point is used to separate the integer and fractional parts of the number. The number 8265.14 is equal to,

$$(8\times10^3)+(2\times10^2)+(6\times10^1)+(5\times10^0)\cdot(1\times10^{-1})+(4\times10^{-2})$$

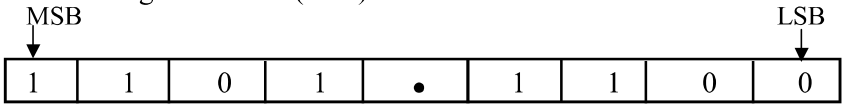
1.3 BINARY NUMBER SYSTEM

The binary number system is a base 2 number system. This number system has two digits 0 and 1. These digits are called BITS. The position of a 1 or 0 in a binary number indicates its weight or value within the number. The weights in a binary number are based in powers of two. The position weights in the binary number system is given in **Figure 1.2**.

2^3	2^2	2^1	2^0	•	2^{-1}	2^{-2}	2^{-3}	2^{-4}

Fig. 1.2: Position weights in binary number system

The bit at the left most position is called Most-Significant Bit (MSB) and the bit at the right most position is called Least-Significant Bit (LSB).



1.4 OCTAL NUMBER SYSTEM

The octal number system is a base 8 number system. It has eight digits 0, 1, 2, 3, 4, 5, 6 and 7. Beyond 7, this number system as 10, 11, 12, ... and so on. The position weights in an octal number system is shown in **Figure 1.3**.

8^3	8^2	8^1	8^0	•	8^{-1}	8^{-2}	8^{-3}

Fig. 1.3: Position weights in an octal number system

1.5 HEXA DECIMAL NUMBER SYSTEM

The hexa decimal number system has a base (radix) of sixteen. It is composed of 16 digits, 0 to 9 and alphabetic characters A, B, C, D, E and F. Hexa decimal numbers are widely used in computer and microprocessor applications. Most digital systems process binary data in groups that are multiples of four bits, making the hexadecimal number very convenient. The digit position in a hexadecimal number system has weights as shown in **Figure 1.4**.

16^3	16^2	16^1	16^0	•	16^{-1}	16^{-2}	16^{-3}

Fig. 1.4: Position Weights in Hexa decimal number

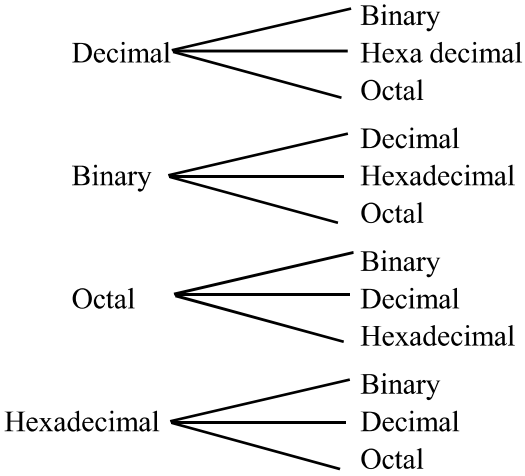
Hexa decimal, Binary and Octal numbers corresponding decimal numbers are given in **Table 1.1**.

Table 1.1: Number Systems

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

1.6 NUMBER BASE CONVERSIONS

Man uses decimal number system while computer uses binary number system. Therefore it is necessary to convert decimal number into its equivalent binary number while feeding the number into the computer. While the computer displaying the result, it is necessary to convert binary number into its equivalent decimal number. When the result is in a large quantity of binary numbers of many bits, like 11001100110000111100, it is inconvenient. Therefore hexa decimal and octal numbers are used as a short hand means of expressing large binary numbers. The following possible conversions can be performed in digital systems:



1.6.1 Decimal to Binary Conversion

Decimal-to-Binary conversion can be done by using repeated division by 2 for integers.

Example 1.1: $15_{10} = ?_2$

2	15	
2	7	– 1
2	3	– 1
	1	– 1

$15_{10} = 1111_2$

Example 1.2: $108_{10} = ?_2$

2	108	
2	54	– 0
2	27	– 0
2	13	– 1
2	6	– 1
2	3	– 0
	1	– 1

$108_{10} = 1101100_2$

Example 1.3: $\mathbf{0.85_{10} = ?_2}$

$0.85 \times 2 = 1.7 = 0.7$ with a carry of 1
$0.7 \times 2 = 1.4 = 0.4$ with a carry of 1
$0.4 \times 2 = 0.8 = 0.8$ with a carry of 0
$0.8 \times 2 = 1.6 = 0.6$ with a carry of 1
$0.6 \times 2 = 1.2 = 0.2$ with a carry of 1
$0.2 \times 2 = 0.4 = 0.4$ with a carry of 0

$0.85_{10} = 0.110110_2$ (approximate)

Example 1.4: $\mathbf{0.3125_{10} = ?_2}$

$0.3125 \times 2 = 0.625 = 0.625$ with a carry of 0
$0.625 \times 2 = 1.25 = 0.25$ with a carry of 1
$0.25 \times 2 = 0.50 = 0.50$ with a carry of 0
$0.50 \times 2 = 1.00 = 0.00$ with a carry of 1

$0.3125_{10} = 0.0101_2$

Example 1.5: $\mathbf{0.625_{10} = ?_2}$

$0.625 \times 2 = 1.25 = 0.25$ with a carry of 1
$0.25 \times 2 = 0.50 = 0.50$ with a carry of 0
$0.50 \times 2 = 1.00 = 0.00$ with a carry of 1

$0.625_{10} = 101_2$

1.6.2 Decimal-to-Hexa Decimal Conversion

The decimal to hexadecimal conversion is explained with the following examples:

Example 1.6: $\mathbf{2479_{10} = ?_{16}}$

16 | 2479
16 | 154 – 15 (F)
9 – 10 (A)

$2479_{10} = 9AF_{16}$

Example 1.7: $\mathbf{3507_{10} = ?_{16}}$

16 | 3507
16 | 219 – 3
13 – 11 (B)

$3507_{10} = DB3_{16}$

Example 1.8: $0.62_{10} = ?_{16}$

$0.62 \times 16 = 9.92 = 0.92$ with a carry of 9
$0.92 \times 16 = 14.72 = 0.72$ with a carry of 14(E)
$0.72 \times 16 = 11.52 = 0.52$ with a carry of 11(B)
$0.52 \times 16 = 8.32 = 0.32$ with a carry of 8

$0.62_{10} = 0.9EB8_{16}$

Example 1.9: $\mathbf{4019.257_{10} = ?_{16}}$

16 | 4019
16 | 251 – 3
15 – 11 (B)

$4019_{10} = FB3_{16}$

$0.257 \times 16 = 4.112 = 0.112$ with a carry of 4
$0.112 \times 16 = 1.792 = 0.792$ with a carry of 1
$0.792 \times 16 = 12.672 = 0.672$ with a carry of 12(C)
$0.672 \times 16 = 10.752 = 0.752$ with a carry of 10(A)

$4019.257_{10} = FB3.41CA_{16}$

1.6.3 Decimal-to-Octal conversion

The decimal to octal conversion can be done by using repeated division by 8.

Example 1.10: $\mathbf{153_{10} = ?_{8}}$

8 | 153
8 | 19 – 1
2 – 3

$153_{10} = 231_{8}$

Example 1.11: $\mathbf{265_{10} = ?_{8}}$

8 | 265
8 | 33 – 1
4 – 1

$265_{10} = 411_{8}$

Example 1.12: $\mathbf{0.55_{10} = ?_{8}}$

$0.55 \times 8 = 4.4 = 0.4$ with a carry of 4
$0.4 \times 8 = 3.2 = 0.2$ with a carry of 3
$0.2 \times 8 = 1.6 = 0.6$ with a carry of 1
$0.6 \times 8 = 4.8 = 0.8$ with a carry of 4
$0.8 \times 8 = 6.4 = 0.4$ with a carry of 6

$0.55_{10} = 0.43146_{8}$ (approximate)

Example 1.13: $2479.64_{10} = ?_8$

$8 \mid 2497$
$8 \mid 312 - 1$
$8 \mid 39 - 0$
$4 - 7$

$2479_{10} = 4701_8$

$0.64 \times 8 = 5.12 = 0.12$ with a carry of 5
$0.12 \times 8 = 0.96 = 0.96$ with a carry of 0
$0.96 \times 8 = 7.68 = 0.68$ with a carry of 7
$0.68 \times 8 = 5.44 = 0.44$ with a carry of 5

$2497.64_{10} = 4701.5075_8$

1.6.4 Binary-to-Decimal Conversion

The positional weight for the binary number system and decimal equivalents are given in **Figure 1.5.**

16	8	4	2	1	•	0.5	0.25	0.125	0.0625	0.03125
2^4	2^3	2^2	2^1	2^0	•	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}

Fig. 1.5: Positional weight and decimal equivalents

Example 1.14: $111011_2 = ?_{10}$

$$(1\times2^5)+(1\times2^4)+(1\times2^3)+(0\times2^2)+(1\times2^1)+(1\times2^0)$$
$$= 32 + 16 + 8 + 0 + 2 + 1 = 59_{10}$$

Example 1.15: $0.1101_2 = ?_{10}$

$$(1\times2^{-1})+(1\times2^{-2})+(0\times2^{-3})+(1\times2^{-4})$$
$$= 0.5 + 0.25 + 0 + 0.0625 = 0.8125_{10}$$

Example 1.16: $10010.011_2 = ?_{10}$

$$(1\times2^4)+(0\times2^3)+(0\times2^2)+(1\times2^1)+(0\times2^0)\cdot(0\times2^{-1})+(1\times2^{-2})+(1\times2^{-3})$$
$$=(16+0+0+2+0)\times(0+0.25+0.125)=18.375_{10}$$

Example 1.17: $01010110.0110_2 = ?_{10}$

$$(0\times2^7)+(1\times2^6)+(0\times2^5)+(1\times2^4)+(0\times2^3)+(1\times2^2)+(1\times2^1)+(0\times2^0)\cdot(0\times2^{-1})$$
$$+(1\times2^{-2})+(1\times2^{-3})+(0\times2^{-4})$$
$$=(0+64+0+16+0+4+2+0)\times(0+0.25+0.125+0) = 86.375_{10}$$

1.6.5 Binary-to-Hexa Decimal Conversion

The binary to hexadecimal conversion is explained with the following examples:

Example 1.18: $111101010110_2 = ?_{16}$

$$\underbrace{1111}_{F}\quad\underbrace{0101}_{5}\quad\underbrace{0110}_{6}$$

$$(111101010110)_2 = (F56)_{16}$$

Example 1.19: $1111110000_2 = ?_{16}$

$$\underbrace{0011}_{3}\quad\underbrace{1111}_{F}\quad\underbrace{0000}_{0}$$

$$(1111110000)_2 = (3F0)_{16}$$

Example 1.20: $1110100011010110_2 = ?_{16}$

$$\underbrace{1110}_{E}\ \underbrace{1000}_{8}\ \underbrace{1101}_{D}\ \underbrace{0110}_{6}$$

$$= (E8D6)_{16}$$

Example 1.21: $0.10110110_2 = ?_{16}$

$$\underbrace{1011}_{B}\quad\underbrace{0110}_{6}$$

$$(0.10110110)_2 = (0.B6)_{16}$$

Example 1.22: $(10111011.1111001)_2 = (?)_{16}$

$$\underbrace{1011}_{B}\quad\underbrace{1011}_{B}\ .\ \underbrace{1111}_{F}\quad\underbrace{0010}_{2}$$

$$= (BB.F2)_{16}$$

Example 1.23: $(10110001101011.1101101)_2 = ?_{16}$

$$\underbrace{0010}_{2}\quad\underbrace{1100}_{C}\quad\underbrace{0110}_{6}\quad\underbrace{1011}_{B}\ .\ \underbrace{1101}_{D}\quad\underbrace{1010}_{A}$$

$$= (2C6B.DA)_{16}$$

1.6.6 Binary-to-Octal Conversion

The binary to octal conversion is performed by forming a 3 bit binary group and converting each 3 bit binary to its octal equivalent.

Example 1.24: $(10101111)_2 = ?_8$

$$\underbrace{010}_{2}\ \underbrace{101}_{5}\ \underbrace{111}_{7} \qquad (10101111)_2 = (257)_8$$

Example 1.25: $(0.0110111)_2 = ?_8$

$$\underbrace{011}_{3}\ \underbrace{011}_{3}\ \underbrace{100}_{1} \qquad (0.110111)_2 = (0.331)_8$$

Example 1.26: $(1011011.01101)_2 = ?_8$

$$\underbrace{001}_{1}\ \underbrace{011}_{3}\ \underbrace{011}_{3}\ .\ \underbrace{011}_{3}\ \underbrace{010}_{2} \qquad (1011011.01101)_2 = (133.32)_8$$

Example 1.27: $(1010011.00101)_2 = ?_8$

$$\underbrace{001}_{1}\ \underbrace{010}_{2}\ \underbrace{011}_{3}\ .\ \underbrace{001}_{1}\ \underbrace{010}_{2} \qquad (1010011.00101)_2 = (123.12)_8$$

1.6.7 Octal-to-Binary Conversion

The octal number is converted into binary number by a group of 3 bits

Example 1.28: $(3574)_8 = ?_2$

$$\begin{array}{cccc} 3 & 5 & 7 & 4 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 011 & 101 & 111 & 100 \end{array} \qquad (3574)_8 = (11101111100)_2$$

Example 1.29: $(0.7460)_8 = ?_2$

$$\begin{array}{cccc} 7 & 4 & 6 & 0 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 111 & 100 & 110 & 000 \end{array} \qquad (0.7460)_8 = (0.111100110)_2$$

Example 1.30: $(34.321)_8 = ?_2$

$$\begin{array}{ccccccc} 3 & 4 & . & 3 & 2 & 1 \\ \downarrow & \downarrow & & \downarrow & \downarrow & \downarrow \\ 011 & 100 & . & 011 & 010 & 001 \end{array} \qquad (34.321)_8 = (11100.011010001)_2$$

1.6.8 Octal-to-Decimal Conversion

The positional weight for the octal number system and their decimal equivalents are given in **Figure 1.6.**

512	64	8	1	•	0.125	0.015625	0.00195
8^3	8^2	8^1	8^0	•	8^{-1}	8^{-2}	8^{-3}

Fig. 1.6: Positional weight and decimal equivalent of octal number system

Example 1.31: $465_8 = ?_{10}$

$$(4\times8^2)+(6\times8^1)+(5\times8^0)$$

$$=(4\times64)+(6\times8)+(5\times1)$$

$$= 256 + 48 + 5$$

$$= 309 \qquad (465)_8 = (309)_{10}$$

Example 1.32: $0.731_8 = ?_{10}$

$$(7\times8^{-1})+(3\times8^{-2})+(1\times8^{-3})$$

$$=(7\times0.125)+(3\times0.015625)+(1\times0.00195)$$

$$= 0.923825_{10} \qquad (0.731)_8 = (0.923825)_{10}$$

Example 1.33: $326.216_8 = ?_{10}$

$$(3\times8^2)+(2\times8^1)+(6\times8^0)\cdot(2\times8^{-1})+(1\times8^{-2})+(6\times8^{-3})$$

$$=(3\times64)+(2\times8)+(6\times1)\cdot\left(2\times\frac{1}{8}\right)+\left(1\times\frac{1}{8^2}\right)+\left(6\times\frac{1}{8^3}\right)$$

$$= 214.27734375_{10}$$

$$(326.216)_8 = 214.27734375_{10}$$

1.6.9 Octal-to-Hexa Decimal Conversion

Example 1.34: $327_8 = ?_{16}$

	3	2	7
Octal to Binary	↓	↓	↓
	011	010	111
	0000	1101	0111
Binary to Hex	0	D	7

$$(327)_8 = (D7)_{16}$$

Example 1.35: $615_8 = ?_{16}$

	6	1	5
Octal to Binary	↓	↓	↓
	110	001	101
Binary to Hex	0001	1000	1101
	1	8	D

$$(615)_8 = (18D)_{16}$$

Example 1.36: $1024.102_8 = ?_{16}$

	1	0	2	4	·	1	0	2	
Octal to Binary	↓	↓	↓	↓		↓	↓	↓	
	001	000	010	100		001	000	010	
Binary to Hex	0010	0001	0100	·		0010	0001	0000	
	2	1	4	·		2	1	0	

$$(1024.102)_8 = (214.210)_{16}$$

1.6.10 Hexadecimal-to-Binary Conversion

The hexadecimal-to-binary conversion is explained with following examples:

Example 1.37: $306_{16} = ?_2$

	3	0	6
	↓	↓	↓
	0011	0000	0110

$$(306)_{16} = (1100000110)_2$$

Example 1.38: $9AF \cdot F_{16} = ?_2$

	9	A	F	·	F
	↓	↓	↓		↓
	1001	1010	1111		1111

$$(9AF \cdot F)_{16} = (100110101111.1111)_2$$

Example 1.39: $7AF4 \cdot BB_{16} = ?_2$

	7	A	F	4	·	B	B
	↓	↓	↓	↓		↓	↓
	0111	1010	1111	0100	·	1011	1011

$$(7AF4 \cdot BB)_{16} = (111101011110100.10111011)_2$$

1.6.11 Hexa Decimal-to-Decimal Conversion

The positional weight for the hexadecimal number system and their decimal equivalents are given in **Figure 1.7.**

4096	256	16	1	•	0.0625	0.0039	0.00024
16^3	16^2	16^1	16^0	•	16^{-1}	16^{-2}	16^{-3}

Fig. 1.7: Positional weight and decimal equivalent of Hex

Example 1.40: $\mathbf{2F59_{16} = ?_{10}}$

$$(2\times16^3)+(F\times16^2)+(5\times16^1)+(9\times16^0)$$
$$=(2\times4096)+(15\times256)+(5\times256)+(5\times16)+9$$
$$=12121_{10} \qquad (2F59)_{16}=(12121)_{10}$$

Example 1.41: $\mathbf{ABCD_{16} = ?_{10}}$

$$(A\times16^3)+(B\times16^2)+(C\times16^1)+(D\times16^0)$$
$$=(10\times4096)+(11\times256)+(12\times16)+(13\times1)$$
$$=40960+2816+192+13$$
$$=43981_{10} \qquad (ABCD)_{16}=(43981)_{10}$$

Example 1.42: $\mathbf{F8E6_{16}\cdot39_{10} = ?_{10}}$

$$(15\times16^3)+(8\times16^2)+(14\times16^1)+(6\times16^0)\cdot(3\times16^{-1})+(9\times16^{-2})$$
$$=(61440+2048+224+6)\cdot(0.1875+0.0352)$$
$$=63718.2227_{10} \qquad (F8E6\cdot39)_{16}=(63718.2227)_{10}$$

1.6.12 Hexa Decimal-to-Octal Conversion

Convert the given hexadecimal number into its equivalent 4 bit binary number and regroup the bits in 3 bit group. Then 3 bit binary number is converted into octal number.

Example 1.43: $\mathbf{25B_{16} = ?_8}$

	2	5	B	
	↓	↓	↓	
	0010	0101	1011	
Binary ⇒	001	001	011	011
Octal ⇒	1	1	3	3

$(25B)_{16}=(1133)_8$

Example 1.44: $A246_{16} = ?_8$

$$\begin{array}{lcccccc} & A & 2 & 4 & 6 & & \\ & \downarrow & \downarrow & \downarrow & \downarrow & & \\ & 1010 & 0010 & 0100 & 0110 & & \\ \text{Binary} \Rightarrow & \underbrace{001} & \underbrace{010} & \underbrace{001} & \underbrace{001} & \underbrace{000} & \underbrace{110} \\ \text{Octal} \Rightarrow & 1 & 2 & 1 & 1 & 0 & 6 \end{array}$$

$(A246)_{16} = (121106)_8$

Example 1.45: $5C2.39_{16} = ?_8$

$$\begin{array}{lcccccccc} & 5 & C & 2 & \cdot & 3 & 9 & & \\ & \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow & & \\ & 0101 & 1100 & 0010 & & 0011 & 1001 & & \\ \text{Binary} \Rightarrow & \underbrace{010} & \underbrace{111} & \underbrace{000} & \underbrace{010} & \cdot & \underbrace{001} & \underbrace{110} & \underbrace{010} \\ \text{Octal} \Rightarrow & 2 & 7 & 0 & 2 & & 1 & 6 & 2 \end{array}$$

$(5C2.39)_{16} = (2702.162)_8$

1.7 COMPLEMENTS

Complements are used in digital computers for simplifying the subtraction operation and for logical manipulation. There are two types of complements:

- r 's complement
- (r – 1)'s complement

For binary numbers, r(base) = 2

- 2's complement
- 1's complement

For decimal numbers, r(base) = 10

- 10's complement
- 9's complement

1.7.1 1's Complement

The 1's complement of a binary number is the number that results when we complement each bit. If the binary number is,

$$A_3\, A_2\, A_1\, A_0 = 1001$$

The 1's complement is

$$\overline{A}_3\, \overline{A}_2\, \overline{A}_1\, \overline{A}_0 = 0110$$

Therefore, the 1's complement of a binary number is formed by changing 1's to 0's and 0's to 1's. The following are some examples:

The 1's complement of 10110001 is 01001110

The 1's complement of 1111 is 0000

1.7.2 2's Complement

The 2's complement is the binary number that results when we add 1 to the 1's complement. i.e.,

2's complement = 1's complement + 1

If the binary number is 1101

1's complement $\Rightarrow$ 0010

1(+)

2's complement $\Rightarrow$ 0011

Some other examples of 2's complements:

Binary Number	*1's Complement*	*2's Complement*
1000 0001	0111 1110	0111 1111
1111 1001	0000 0110	0000 0111

1.7.3 1's Complement Subtraction

Subtraction of binary numbers using 1's complement method allows subtraction only by addition. To subtract a smaller number from a large number ($X - Y$), the 1's complement method as follows:

- Obtain the 1's complement of the smaller number (Y)
- Add this to the large number (X)
- Remove the carry and add it to the result. This carry is called 'end- around-carry'.

Example 1.46: $X - Y = 14 - 10 = 4$

$X = 1110, Y = 1010$

1's complement of $Y = 0101$

Solution: $X =$ 1110

1's complement of $Y =$ 0101 (+)

Sum = 1 0011

End-around carry = 1 (+)

$X - Y =$ 0100

Example 1.47: $(X - Y) = 9 - 3 = 6$

$X =$ 1001

1's complement of $Y =$ 1100 (+)

Sum = 1 0101

End-around carry = 1 (+)

$X - Y =$ 0110

Subtraction of a large number (X) from a smaller (Y), the 1's complement method as follows:

- Obtain the 1's complement of the larger number (X).
- Add this to the smaller number (Y).
- The answer is the 1's complement of the result and is opposite in sign. There is no carry.

Example 1.48: $Y - X = 10 - 14 = -4$

$X =$ 1110

1's complement of $X =$ 0001

$Y =$ 1010

Solution: $Y =$ 1010

1's complement of $X =$ 0001 (+)

Sum = 1011

1's complement of result (1011) is 0100 and is opposite sign. i.e., -0100.

Example 1.49: $Y - X = 3 - 9 = -6$

$X =$ 1001

1's complement of $X =$ 0110

$Y =$ 0011

Solution: $Y =$ 0011

1's complement of $X =$ 0110 (+)

Sum = 1001

1's complement of result (1001) is 0110 and is opposite sign.

i.e., -0110.

1.7.4 2's Complement Subtraction

The subtraction of a small number from a large number by the 2's complement method is as follows:

(i) Determine the 2's complement of the smaller number

(ii) Add this is to the larger number

(iii) Omit the carry.

Example 1.50: $X - Y = 14 - 10 = 4$

$X = 1110$

$Y = 1010$

1's complement of $Y = 0101$

$\underline{\quad 1\ (+)}$

2's complement of $Y = 0110$

Solution: $X = 1110$

2's complement of $Y = 0110\ (+)$

$1\ \ 0100$

The carry (1) is discarded. Therefore the result is (0100).

Example 1.51: $X - Y = 1010100 - 1000011$

1's complement of $Y = 0111100$

$\underline{\quad\quad 1\ (+)}$

2's complement of $Y = 0111101$

Solution: $X = 1010100$

2's complement of $Y = 0111101\ (+)$

$1\ \ 0010001$

Ans: $X - Y = 0010001$

The subtraction of a larger number from a smaller number is as follows:

- Determine the 2's complement of the larger number.
- Add 2's complement to the smaller number.
- There is no carry. The result is in 2's complement form and is negative.
- To get the true answer, take 2's complement of the result and change the sign.

Example 1.52: $Y - X = 10 - 14 = -4$

$Y = 1010$

$X = 1110$

1's complement of $X = 0001$

$\underline{\quad 1\ (+)}$

2's complement of $X = 0010$

Solution: $Y = 1010$

2's complement of $X = 0010\ (+)$

1100

$Y - X = -$ (2's complement of 1100)

1's complement of 1100 = 0011

$$\underline{\quad\quad 1}$$

2's complement of 1100 = 0100

Therefore the true answer is – 0100

Example 1.53: $(Y - X) = 9 - 10 = -1$

$Y = 1001$

$X = 1010$

1's complement of $X = 0101$

$$\underline{\quad 1\ (+)}$$

2's complement of $X = 0110$

Solution: $Y = 1001$

2's complement of $X = 0110\ (+)$

$$\underline{\quad 1111}$$

$(Y - X) = -$ (2's complement of 1111)

1's complement of 1111 = 0000

$$\underline{\quad 1\ (+)}$$

2's complement of 1111 = 0001

$Y - X = -\ 0001$

1.7.5 Comparison of 1's and 2's complements

- ❖ The 1's complement can be easily obtained using an inverter. The 2's complement is obtained as adding '1' to the 1's complement.
- ❖ The 2's complement system requires only one arithmetic operation, but the 1's complement system requires two arithmetic operations.
- ❖ The 1's complement is used in logical manipulation for inversion operation; the 2's complement is used only for arithmetic applications.

1.7.6 9's Complement

The 9's complement of a decimal number is found by subtracting each digit in the number from 9. The examples of 9's complement are:

(i) 9's complement of 71 is, 99

$$\underline{17\ (-)}$$

$$\underline{82}$$

(ii) 9's complement of 444 is, 999

$$\underline{444\ (-)}$$

$$\underline{555}$$

(iii) 9's complement of 1542701 is, 9999999

$$\begin{array}{r} 1542701 \\ \hline 8457298 \end{array}$$

1.7.7 10's Complement

The 10's complement of a decimal number that results add 1 to its 9's complement.

(i) 10's complement of 71 is, 99

$$\begin{array}{r} 17\ (-) \\ \hline 82 \\ 1\ (+) \\ \hline 83 \end{array}$$

(ii) 10's complement of 444 is, 999

$$\begin{array}{r} 444\ (-) \\ \hline 555 \\ 1\ (+) \\ \hline 556 \end{array}$$

(iii) 10's complement of 1542701 is, 9999999

$$\begin{array}{r} 1542701\ (-) \\ \hline 8457298 \\ 1\ (+) \\ \hline 8457299 \end{array}$$

1.7.8 9's Complement Subtraction

The 9's complement subtraction method is same as the 1's complement subtraction method. The 9's complement method for subtraction a smaller number from a larger number is as follows:

- Determine the 9's complement of the smaller number
- Add this is to the larger number
- Remove the carry and add it to the result

Example 1.54: 98 – 18 = 80

9's complement of 18, 99 – 18 = 81

$$\begin{array}{r} 98 \\ 81\ (+) \\ \hline \text{Carry}\quad (1)\ 79 \\ 1\ (+) \\ \hline 80 \end{array}$$

Example 1.55: 55 – 10 = 45

9's complement of 10 is, 99 – 10 = 89

```
    55
    89   (+)
Carry (1) 44
     1 (+)
    45
```

Subtraction of a larger number from a smaller one by the 9's complement method is as follows:

- Determine the 9's complement of the larger number.
- Add this is to the smaller number.
- The answer is the 1's complement of the true result and is opposite in sign.

Example 1.56: 15 – 67 = – 52

9's complement of 67 is, 99 – 67 = 32

```
    15
    32 (+)
    47
```

True result = – (1's complement of 47)

i.e., – (99 – 47) = – 52

Example 1.57: 265 – 347 = – 082

9's complement of 347 is, 999 – 347 = 652

```
   265
   652 (+)
   917
```

True result = – (1's complement of 917)

= – (999 – 917)

= – (082)

1.7.9 10's Complement Subtraction

Subtraction of a smaller number from a larger number by the 10's complement method is as follows:

- Determine the 10's complement of the smaller number.
- Add this is to the larger number.
- The carry is discarded.

Example 1.58: 25 – 15 = 10

1's complement of 15 = 99 – 15 = 84

2's complement of 15 = 84 + 1 = 85

25
85 (+)
(1) 10 **Ans:** 10

Example 1.59: 786 - 412 = 374

1's complement of 412 ⇒ 999
412
587
1 (+)
2's complement of 412 ⇒ 588

786
588 (+)
Carry (1) 374 **Ans:** 374

The 10's complement method for subtraction of a larger number from a smaller number is as follows:

- Determine the 10's complement of larger number.
- Add this to the smaller number.
- The result is in 10's complement form and is negative.
- To get an answer in true form, take the 10's complement and change the sign.

Example 1.60: 31 – 57 = – 26

9's complement of 57 ⇒ 99
57 (–)
42
1 (+)
10's complement of 57 ⇒ 43

31
43 (+)
74

Result = – (10's complement of 74)

9's complement of 74 is 99
74 (–)
25

10's complement of 74 is 25 + 1 = 26. ∴ 31– 57 = – 26.

Example 1.61: 265 – 347 = – 082

9's complement of 347 ⇒ 999

$$\begin{array}{r} 347\ (-) \\ \hline 652 \\ 1\ (+) \\ \hline 653 \end{array}$$

9's complement of 347 ⇒ 653

$$\begin{array}{r} 265 \\ 653\ (+) \\ \hline 918 \end{array}$$

Result = – (10's complement of 918)

9's complement of 918 ⇒ 999

$$\begin{array}{r} 918\ (-) \\ \hline 81 \\ 1\ (+) \\ \hline 82 \end{array}$$

10's complement of 918 ⇒ 82 ∴ 265 – 347 = – 82.

NOTE: The same proceudre is used to find the solution for subtraction with 7's complement, 8's complement, 15's complement and 16's complement.

1.8 SIGNED BINARY NUMBERS

Digital systems must be able to handle both positive and negative numbers. In ordinary arithmetic, a positive number is indicated by a '+' sign and a negative number is indicated by a '–' sign. But in binary numbers, the left most bit (sign bit) denotes the sign.

0 is used for the +ve sign and 1 is used for the –ve sign. Therefore – 001, – 111 are coded as 1001, 1111. These numbers contain a sign bit followed by magnitude bits. Numbers in this form are called signed binary numbers.

When a signed binary number is represented in sign-magnitude form, the left-most bit is the sign bit and the remaining bits are the magnitude bits. For example + 32 is expressed as an 8 bit signed binary number using the sign-magnitude form as,

– 32 is represented as 10100000

Thus in the sign-magnitude form, a negative number has the same magnitude bits as the corresponding positive number, but the sign bit is '1' rather than '0'.

Some other examples are:

(i) – 18 ⇒ 10010010

+ 18 ⇒ 00010010

(ii) – 97 ⇒ 11100001

+ 97 ⇒ 01100001

The sign-magnitude for the decimal numbers (– 15 to + 15) are represented in **Table 1.2.**

In the 1's complement form of signed binary numbers, a negative number is the 1's complement of the corresponding positive number.

For example, – 8 is represented as 10001000

Signed 1's complement $\Rightarrow$ 11110111 (1's complement of + 8)

In the 2's complement form, a negative number is the 2's complement of the corresponding positive number. Signed 2's complement of – 8 is,

11110111

1 (+)

11111000 (2's complement of + 8)

Range of signed numbers

For 8 bit numbers,

Unsigned numbers range = 0 to $2^n - 1$

= 0 to 255

Signed numbers range = $-(2^{n-1})$ to $+(2^{n-1} - 1)$

= – 128 to + 127

16 bit signed numbers range = – 32768 to + 32767

Table 1.2: Sign-Magnitude Numbers

Decimal	*Sign Magnitude*	*Decimal*	*Sign Magnitude*
+ 15	01111	– 15	11111
+ 14	01110	– 14	11110
+ 13	01101	– 13	11101
+ 12	01100	– 12	11100
+ 11	01011	– 11	11011
+ 10	01010	– 10	11010
+ 9	01001	– 9	11001
+ 8	01000	– 8	11000
+ 7	00111	– 7	10111
+ 6	00110	– 6	10110
+ 5	00101	– 5	10101
+ 4	00100	– 4	10100
+ 3	00011	– 3	10011
+ 2	00010	– 2	10010
+ 1	00001	– 1	10001
+ 0	00000	– 0	10000

1.9 BINARY ARITHMETIC

1.9.1 Binary Addition

A + *B*	Sum	Carry
0 + 0	0	0
0 + 1	1	0
1 + 0	1	0
1 + 1	0	1

The examples for binary additon are given below:

Example 1.62:

```
  0011
  1010 (+)
  1101
```

Example 1.63:

```
0101    0111
0011    0101 (+)
1000    1100
```

Example 1.64

57.5 + 0.3125 = 57.8125

```
111001.1000
000000.0101 (+)
111001.1101
```

Example 1.65

31.75 + 19.25 = 51.00

```
011111.110
010011.010 (+)
110011.000
```

Example 1.66

101.01 + 110 + 10.1 = ?

```
  101.01
  110.00
 1011.01
   10.10
 1101.11
```

Example 1.67

101101 + 1110 + 110 = ?

```
  101101
    1110
     110
 -------
 1000001
 -------
```

Note: 1 + 1= 10

1 + 1+ 1 = 11

1 + 1 + 1 + 1 = 100

1.9.2 Binary Subtraction

$A - B$	Difference	Borrow
0 – 0	0	0
0 – 1	1	1
1 – 0	1	0
1 – 1	0	0

Example 1.68:

```
  1011
  0101 (–)
 ------
  0110
 ------
```

Example 1.69:

```
  111.01
  100.10 (–)
 -------
  010.11
 -------
```

Example 1.70:

```
  10101.001
  01110.110 (–)
 ----------
  00110.011
 ----------
```

Note: For binary subtraction, use 1's complement and 2's complement methods also.

1.9.3 Binary Multiplication

The four basic rules for multiplying bits are as follows:

$0 \times 0 = 0$

$0 \times 1 = 0$

$1 \times 0 = 0$

$1 \times 1 = 1$

Example 1.71:
$$\begin{array}{r} 110 \\ \times\ 101 \\ \hline 110 \\ 000 \\ 110 \\ \hline 11110 \end{array}$$

Example 1.72: 110.1×10.1
$$\begin{array}{r} 110.1 \\ \times\ 10.1 \\ \hline 1101 \\ 0000 \\ 1101 \\ \hline 10000.01 \end{array}$$

1.9.4 Binary Division

Example 1.73: $110 \div 11$

$$\begin{array}{r|l} & 10 \\ \hline 11 & 110 \\ & -11 \\ \hline & 000 \end{array}$$

$110 \div 11 = 10$

Example 1.74: $110 \div 10$

$$\begin{array}{r|l} & 11 \\ \hline 10 & 110 \\ & 10 \\ \hline & 10 \\ & 10 \\ \hline & 00 \end{array}$$

$110 \div 10 = 11$

1.9.5 Addition with signed numbers

There are 4 cases that can occur when two signed binary numbers are added.

- Both numbers are +ve
- +ve number with magnitude larger than –ve number
- –ve with magnitude larger than +ve number
- Both numbers are –ve

1. Addition of two positive numbers yields a positive number

$$\begin{array}{rr} 6 & 00000110 \\ 13\ (+) & 00001101 \\ \hline 19 & 00010011 \end{array}$$

2. Addition of a positive number and a smaller negative number yields a positive number,

+ 13		00001101
− 6 (+)		11111010
+ 7	(1)	00000111

↓

discard carry

3. Addition of a positive number and a larger negative number yields a negative number in 2's complement

+ 6		00000110
− 13 (+)		11110011
17	(1)	11111001

↓

discard carry The sum is negative and therefore 2's complement form.

4. Addition of two negative numbers yields a negative number in 2's complement

− 6	11111010
− 13 (+)	11110011
− 19	11101101

The sum is negative and therefore in 2's complement form.

Example 1.75: Perform each of the following computations using signed, 8 bit words in 1's complement and 2's complement binary arithmetic. **(Dec. 2005)**

1. $(+95)_{10} + (-63)_{10}$
2. $(+42)_{10} + (-87)_{10}$
3. $(-13)_{10} + (-59)_{10}$
4. $(+38)_{10} + (-38)_{10}$
5. $(-105)_{10} + (-120)_{10}$

1. $(+95)_{10} + (-63)_{10}$.

In signed binary number, the left most bit is the sign bit and the remaining bits are the magnitude bits.

$(+63)_{10}$ =	**0**	0111 111
$(-63)_{10}$ =	**1**	01111111
	sign	magnitude

1's complement of $(-63)_{10}$ = 1 1000000

2's complement of $(-63)_{10}$ = 1 1000001

1's complement arithmetic

$(+95)_{10} + (-63)_{10}$

$(+95)_{10} \Rightarrow$ 0101 1111

$(-63)_{10} \Rightarrow$ 1100 0000

1 0001 1111

+ 1

0010 0000 = +32

2's complement arithmetic

(+95) + (–63)

0101 1111

1100 0001

0010 0000 = +32

2. $(+42)_{10} + (-87)_{10}$

$(+87)_{10}$ = **0** 1010111

$(-87)_{10}$ = **1** 1010111

1's complement of $(-)_{10}$ = **1** 0101000

2's complement of $(-7)_{10}$ = **1** 0101001

1's complement arithmetic

$(+42)_{10} + (-87)_{10}$

0010 1010

1010 1000

1101 0010

1's complement of result is 1 0101101 = –45

2's complement arithmetic

00101010

10101001

11010011 = –45

Since the two's complement of 1101 0011 is (0010 1100 + 1 = 0010 1101) $(+45)_{10}$, 1101 0011 is $(-45)_{10}$.

3. $(-13)_{10} + (-59)_{10}$.

(+13) ⇒ **0** 000 1101

(–13) ⇒ **1** 000 1101

(+59) ⇒ **0** 0111 011

(–59) ⇒ **1** 0111011

1's complement arithmetic

$(-13)_{10} + (-59)_{10}$

	Sign	
1's complement of $(-13)_{10}$ =	**1**	111 0010
1's complement of $(-59)_{10}$ =	**1**	1000100
	1	0110110
		+1
	1	01100111

1's complement of result is 1 1001000 = –72

2's complement arithmetic

2's complement of (–13) = **1** 111 0011

2's complement of (–59) = **1** 100 0101

1 011 1000 = –72

Since 2's complement of result is

01001000 = +72, 10111000 = –72.

4. $(+38)_{10} + (-38)_{10}$

(+38) = **0** 0100110

(–38) = **1** 0100110

1's complement of (–38) = **1** 1011001

2's complement of (–38) = **1** 1011010

1's complement arithmetic

0 0100110

1 1011001

1 1111111

1's complement of result is 1 000000 = –0

2's complement arithmetic

0 0100110

1 1011010

0 0000000 = 0

5. $(-105)_{10} + (-120)_{10}$

(+105) = **0** 1101001

(–105) = **1** 1101001

(+120) = **0** 1111000

(–120) = **1** 1111000

1's complement arithmetic

(−105) + (−120)

1's complement of (−105) = **1** 0010110

1's complement of (−120) = **1** 0000111

1 **0** 0011101

+1

0 10011110 = 30

2's complement arithmetic

2's complement of (−105) = **1** 0010111

2's complement of (−120) = **1** 0001000

0 0 011111 = 31

Error occurs due to overflow.

1.10 OTHER NUMBER SYSTEMS

Table 1.3 shows the some positional number system and their possible symbols.

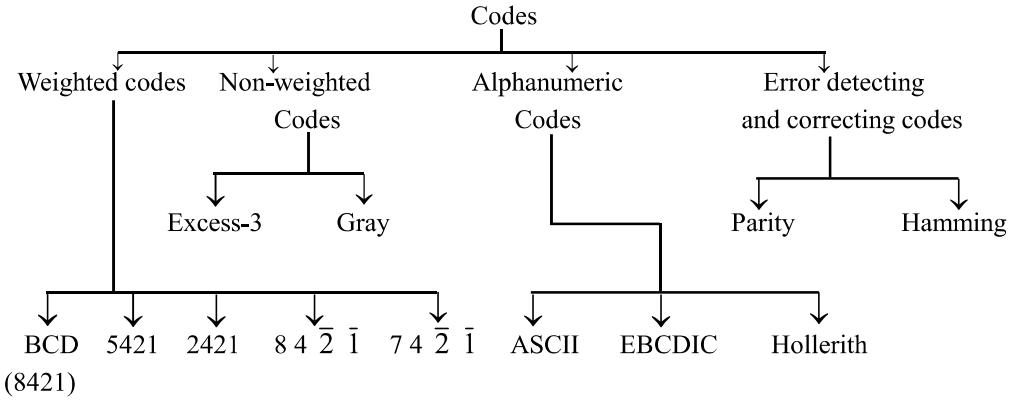
Table 1.3: Number Systems

Number System	*Base*	*Possible Symbols*
Binary	2	0, 1
Ternary	3	0, 1, 2
Quarternary	4	0, 1, 2, 3
Quinary	5	0, 1, 2, 3, 4
Octal	8	0, 1, 2, 3, 4, 5, 6, 7
Decimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Duodecimal	12	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B
Hexadecimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

1.11 BINARY CODES

The digital data is represented, stored and transmitted as groups of binary bits. The group of bits is called as binary code. The binary code represent numbers, alphabets, special characters and control functions. The codes are classified as,

- Weighted codes
- Non-weighted codes
- Error detecting and correcting codes
- Alphanumeric codes



1.11.1 Weighted Codes

Weighted binary codes obey their positional weighting principles. Each digit position of a number represents a specific weight. The bits are multiplied by the weights and the sum of these weighted bits give the equivalent decimal value.

8421 code is the binary-coded-decimal (BCD) code. The other 4 bit weighted binary codes are: 5421, 2421, 8 4 $\bar{2}$ $\bar{1}$, 74 $\bar{2}$ $\bar{1}$ etc. The examples for 5 bit code are 84621, 51111 and 63210 code. The example for 7 bit code is biquinary (5043210) code.

1. BCD (8421) Code

The binary-coded-decimal (BCD) uses the binary number system to specify the decimal numbers 0 to 9. It has 4 bits. It is called $8-4-2-1$ code, i.e., bit 3 has weight $8(2^3)$, bit 2 has weight $4(2^2)$, bit 1 has weight $2(2^1)$ and bit 0 has weight $1(2^0)$. **Table 1.4** shows the 8421 BCD code used to represent the decimal digits.

Table 1.4: BCD Code

Decimal Digit	*BCD Code*
	8 4 2 1
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
10	0001 0000
11	0001 0001
12	0001 0010

2. Other 4 bit codes: Table 1.5 shows the 4 bit binary codes for the decimal digits 0 to 9.

Table 1.5: Binary Codes

Decimal	5 4 2 1	2 4 2 1	8 4 $\bar{2}$ $\bar{1}$	7 4 $\bar{2}$ $\bar{1}$
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1	0 1 1 1	0 1 1 1
2	0 0 1 0	0 0 1 0	0 1 1 0	0 1 1 0
3	0 0 1 1	0 0 1 1	0 1 0 1	0 1 0 1
4	0 1 0 0	0 1 0 0	0 1 0 0	0 1 0 0
5	1 0 0 0	1 0 1 1	1 0 1 1	1 0 1 0
6	1 0 0 1	1 1 0 0	1 0 1 0	1 0 0 1
7	1 0 1 0	1 1 0 1	1 0 0 1	1 0 0 0
8	1 0 1 1	1 1 1 0	1 0 0 0	1 1 1 1
9	1 1 0 0	1 1 1 1	1 1 1 1	1 1 1 0

- Representation of decimal digit 7 in 5421 code:

$$1\,0\,1\,0 \Rightarrow (1\times5)+(0\times4)+(1\times2)+(0\times1)=7$$

- Representation of decimal digit 5 in 2421 code:

$$1\,0\,1\,1 \Rightarrow (1\times2)+(0\times4)+(1\times2)+(1\times1)=5$$

- Representation of decimal digit 2 in 8 4 $\bar{2}$ $\bar{1}$ code:

$$0\,1\,1\,0 \Rightarrow (0\times8)+(1\times4)+(1\times-2)+(0\times-1)=4-2=5$$

- Representation of decimal digit 1 in 7 4 $\bar{2}$ $\bar{1}$ code:

$$0\,1\,1\,1 \Rightarrow (0\times7)+(1\times4)+(1\times-2)+(1\times-1)=4-3=1$$

The some other weighted 4 bit binary codes are:

3 3 2 1 Code
4 2 2 1 Code
5 2 2 1 Code
5 3 1 1 Code
6 3 1 1 Code
7 4 2 1 Code

1.11.2 Non Weighted Codes

Non-weighted codes are codes that are not positionally weighted, i.e, each position within a binary number is not assigned a fixed value. The non-weighted codes are:

- Excess-3 (XS – 3) Code
- Gray Code

(i) ***Excess-3 Codes*:** The excess-3 code is a modified form of binary number. It is obtained by simply adding '3' to a decimal number. For example, to encode the decimal number '4' into an excess 3 code, first add 3. i.e., 4 + 3 = 7. Then it is encoded into binary code 0111. **Table 1.6**

shows excess-3 codes to represent decimal numbers. XS – 3 code is also called as reflective code.

Table 1.6: Excess-3 Codes

Decimal	*BCD Code*	*XS – 3 Code*
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100
10	0001 0000	0100 0011
11	0001 0001	0100 0100
12	0001 0010	0100 0101

(ii) ***Gray Codes*:** In gray code, numbers differ from any adjacent number by a single bit. For example, in going from decimal 3 to 4, the Gray-code number changes from 0010 to 0110. Every number differs by only 1 bit from the preceding number. It is also called as uni**t-**distance code or reflected code. **Table 1.7** shows the gray code representation.

Table 1.7: Gray Codes

Decimal	*Binary Code*	*Gray Code*
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 1
3	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 0
5	0 1 0 1	0 1 1 1
6	0 1 1 0	0 1 0 1
7	0 1 1 1	0 1 0 0
8	1 0 0 0	1 1 0 0
9	1 0 0 1	1 1 0 1
10	1 0 1 0	1 1 1 1
11	1 0 1 1	1 1 1 0
12	1 1 0 0	1 0 1 0
13	1 1 0 1	1 0 1 1
14	1 1 1 0	1 0 0 1
15	1 1 1 1	1 0 0 0

(iii) ***Binary-to-Gray Code Conversion*:** The binary-to-gray code conversion is achieved using following rules:

- The most significant bit (MSB) in the Gray code is the same as the corresponding MSB in the binary number.
- Going from left to right, perform an EX-OR operation between the adjacent pair of binary code bits to get the next Gray code bit.

Example 1.76: Convert the binary number 1001 to Gray code,

Binary : 1 ⊕ 0 ⊕ 0 ⊕ 1

↓ ↓ ↓ ↓

Gray : 1 1 0 1

Gray Code : 1101

Example 1.77: Convert the binary $(10110)_2$ to its gray code.

Binary : 1 ⊕ 0 ⊕ 1 ⊕ 1 ⊕ 0

↓ ↓ ↓ ↓ ↓

Gray : 1 1 1 0 1

Gray Code : 11101

Let the binary number is represented as $B_1, B_2, B_3, \ldots, B_N$ and gray code is $G_1, G_2, G_3, \ldots, G_N$ then

$$G_1 = B_1$$
$$G_2 = B_1 \oplus B_2$$
$$G_3 = B_2 \oplus B_3$$
$$\vdots$$
$$G_N = B_{N-1} \oplus B_N$$

(iv) ***Gray-to-Binary Code Conversion*:** The gray-to-binary code conversion is achieved using following rules:

- The MSB in the binary code is the same as the corresponding bit in the gray code.
- To obtain the next binary digit, perform an EX-OR operation between the bit just written down and the next gray code bit.

Example 1.78: Convert the Gray code 1000 to binary

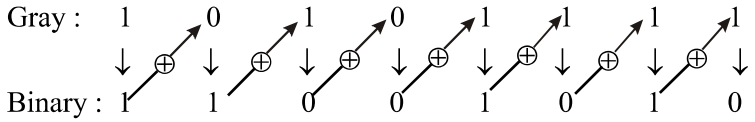
Gray : 1 0 0 0

↓ ⊕ ↓ ⊕ ↓ ⊕ ↓

Binary : 1 1 1 1

Binary Number : 1 1 1 1

Example 1.79: Convert the gray code 10101111 to binary



Binary number is 11001010

1.12 ERROR DETECTION CODES

1.12.1 Parity Bit

In any electronic system involving the transfer of data (in the form of binary digits) then data transmission errors are possible. Any external noise introduced in the physical communication medium may change some of the bits from 0 to 1 or 1 to 0. The purpose of an error-detection code is to detect such bit-reversal errors. The method of **Parity** is widely used as a method of error detection.

A parity bit is an extra bit included with a message to make the total number of 1's transmitted either odd or even.

Even Parity: The value of the parity bit is set such that the total number of 1's in the data word is **even.**

Example : 11001 which has an odd number of 1's. The new total group is thus 110011.

11110 which has an even number of 1's. The new total group is thus 111100.

Odd Parity: The value of the parity bit is set such that the total number of 1's in the data word is **odd**.

Example: 11001 which has an odd number of 1's. The new total group is thus 110010.

11110 which has an even number of 1's. The new total group is thus 111101.

A message of 4 bits and a parity bit are shown in **Table 1.8**.

Error-Detection

Let an even parity bit is generated in the sending end for each message transmission. The message, together with the parity bit is transmitted to its destination. The parity of the received data is checked in the receiving end. If the parity of the received information is not even, it means that atleast one bit has changed value during the transmission. This method detects 1, 3 or any odd combination of errors in each message that is transmitted. An even combination of errors is undetected.

1.12.2 2-out of-5 Code

2-out of-5 code is used for error detection in communications work. It utilizes five bits to represent the ten decimal digits, so it is a form of BCD code. Each code word has exactly two 1's, a convention that facilitates decoding and provides for better error detection than the single-parity-bit method. If more or less than two 1's appear, an error is detected.

Table 1.8: Parity Bit

ODD Parity	
Message	P
0000	1
0001	0
0010	0
0011	1
0100	0
0101	1
0110	1
0111	0
1000	0
1001	1
1010	1
1011	0
1100	1
1101	0
1110	0
1111	1

EVEN Parity	
Message	P
0000	0
0001	1
0010	1
0011	0
0100	1
0101	0
0110	0
0111	1
1000	1
1001	0
1010	0
1011	1
1100	0
1101	1
1110	1
1111	0

1.12.3 63210 Code

This is a BCD code. Like 2-out of-5 code, it is also characterized by having exactly two 1's in each 5 bit groups.

1.12.4 50 - 43210 Code

This code is also called as **Biquinary** (two – five) code. It is used in counters and is composed of a 2 bit group and a 5 bit group, each with a single 1. Its weights are 50 – 43210. The 2 bit group, having weights of five and zero, indicates whether the number represented is less than, equal to, or greater than 5. The 5 bit group indicates the count above or below 5.

During transmission of signals from one location to another, an error may occur. One or more bits may change value. A circuit in the receiving side can detect the presence of more or less than two 1's and if the received combination of bits does not agree with the allowable combination, an error is detected.

1.12.5 Ring Counter Code

This code has ten bits, one for each decimal digit, and a single 1 makes error detection possible. It is easy to decode but wastes bit and requires more circuitry to implement than the 4 bit or 5 bit codes. Its weights are 9876543210. It is used in shift registers and ring counters.

Table 1.9 shows these error detection codes.

Table 1.9: Error Detection Codes

Decimal	2-out of-5	63210	50 – 43210	9876543210
0	00011	00110	01 – 00001	0000000001
1	00101	00011	01 – 00010	0000000010
2	00110	00101	01 – 00100	0000000100
3	01001	01001	01 – 01000	0000001000
4	01010	01010	01 – 10000	0000010000
5	01100	01100	10 – 00001	0000100000
6	10001	10001	10 – 00010	0001000000
7	10010	10010	10 – 00100	0010000000
8	10100	10100	10 – 01000	0100000000
9	11000	11000	10 – 10000	1000000000

1.13 ALPHANUMERIC CODES

Alphanumeric codes are used to represent numbers, letters and other special features using binary bits. The alphanumeric codes are encoding the following:

Decimal digits (0 – 9)

Alphabetic characters (A to Z and a to z)

Mathematical symbols (like, +, –, =, < , >)

Special control characters (like, ESC, NUL, ACK)

The alphanumeric codes are,

- ASCII Code,
- EBCDIC Code,
- Hollerith Code.

1.13.1 ASCII Code

ASCII [American Standard Code for Information Interchange] is a 7 bit code, which is used to represent numbers, letters, characters and other special computer control functions. ASCII is pronouced as “as-kee”.

ASCII is 7 bit code, therefore it has $2^7 = 128$ characters.

Upper case alphabets = 26

Lower case alphabets = 26

Decimal digits (0 – 9) = 10

Special symbols = 33

Control characters = 33

Total = 128

The ASCII characters, ASCII values in binary, decimal and hexadecimal are shown in **Appendix 1**.

1.13.2 EBCDIC Code

EBCDIC [Extended Binary Coded Decimal Interchange Code] is an 8-bit code, pronounced as "eb-see-disk". It differs from ASCII code only in its code grouping for the different alphanumeric characters.

EBCDIC has $2^8 = 256$ characters, in which 117 are unassigned. The remaining 139 characters are as follows:

Upper case alphabets	= 26
Lower case alphabets	= 26
Decimal digits (0 – 9)	= 10
Special symbols	= 27
Control characters	= 50
Total	= 139

The control characters are placed in between 0000 0000 and 0011 1111 and the alphanumeric codes and symbols are placed in between 0100 0000 and 1111 1111.

1.13.3 Hollerith Code

The Hollerith Code is the alphanumeric code used in punched cards. Each card has 80 columns and 12 rows. The rows are numbered 0 through 9, 11 and 12. The columns are numbered 1 through 80, each one containing one character. Each character is uniquely identified by the rows punched in that column.

Thus far, we can express a decimal 8 as follows:

Decimal	8
Binary	1000
Octal	10
Hexadecimal	8
BCD	1000
XS 3	1011
Gray	1100
Biquinary	10 01000
ASCII	0011 1000
EBCDIC	1111 1000
Hollerith	8

Example 1.80:

Consider a decimal number 14 and write the equivalent for this in Binary, BCD, 2421, Gray and Excess 3 codes. **(Dec. 2005)**

Solution:

Binary	1110
BCD	0001 0100
2421	0001 0100
Gray	1001
XS 3	0001 0111

1.14 BOOLEAN POSTULATES AND LAWS

1.14.1 Introduction

In 1854 George Boole introduced a systematic treatment of logic and developed for this purpose an algebraic system now called Boolean Algebra.

In 1938 C.E. Shannon introduced a two-valued Boolean Algebra called Switching Algebra, in which he demonstrated that the properties of bistable electrical switching circuits can be represented by this algebra.

For the formal definition of Boolean Algebra, we shall employ the postulates formulated by E.V. Huntington in 1904.

1.14.2 Postulates or Axioms

The Postulates or Axioms of Boolean Algebra are a set of logical expressions that we accept without proof and upon which we can build a set of useful theorem. Actually the axioms are nothing more than the definitions of 3 basic logic operations: AND, OR and INVERT.

Postulate 1 $\rightarrow$ $0.0 = 0$	Postulate 6 $\rightarrow$ $0 + 1 = 1$
Postulate 2 $\rightarrow$ $0.1 = 0$	Postulate 7 $\rightarrow$ $1 + 0 = 1$
Postulate 3 $\rightarrow$ $1.0 = 0$	Postulate 8 $\rightarrow$ $1 + 1 = 1$
Postulate 4 $\rightarrow$ $1.1 = 1$	Postulate 9 $\rightarrow$ $\bar{1} = 0$
Postulate 5 $\rightarrow$ $0 + 0 = 0$	Postulate 10 $\rightarrow$ $\bar{0} = 1$

1.14.3 Theorems of Boolean Algebra

- The Commutative Properties
- The Associative Properties
- The Idempotent Properties
- The Identity Properties
- The Null Properties
- The Distributive Properties
- The Negation Properties

- The Double Negation Properties
- The Absorption Properties and

  De Morgan's Theorems

(i) ***Commutative Properties***

Theorem 1*a* : $AB = BA$

1*b* : $A + B = B + A$

Theorem 1a: $AB = BA$

"A AND B" is the same as 'B AND A' ; in effect, it makes no difference which input of an AND gate is connected to A and which is connected to B. The truth tables are identical.

A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

=

B	A	BA
0	0	0
0	1	0
1	0	0
1	1	1

Table 1.10(a)

Theorem 1b : $A + B = B + A$

"A OR B" is same as "B OR A"

A	B	$A + B$
0	0	0
0	1	1
1	0	1
1	1	1

=

B	A	$B + A$
0	0	0
0	1	1
1	0	1
1	1	1

Table 1.10(b)

The commulative properties can be extended to any number of variables:

$A + B + C = B + A + C$ $\qquad$ $ABCD = BACD$

$A + C = C + A$ $\qquad$ $BADC = ABDC$

$B + A + C = B + C + A$

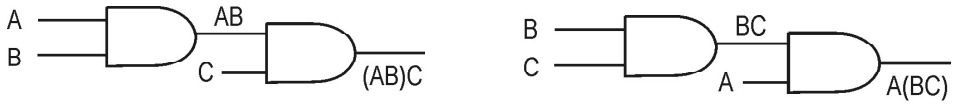
(ii) ***The Associative Properties***

Theorem 2*a*: $(AB)C = A(BC)$

2*b*: $(A + B) + C = A + (B + C)$

Theorem 2*a* : $(AB)C = A(BC)$

A AND B ANDed with C is same as A ANDed with B AND C



A	*B*	*C*	*AB*	*(AB)C*
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

=

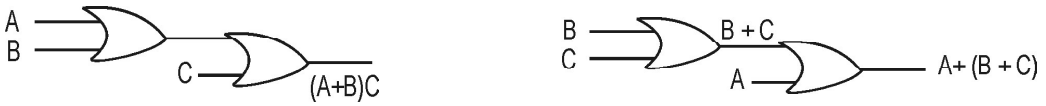
A	*B*	*C*	*BC*	*A(BC)*
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Fig. 1.8: Associative Law

Note that $(AB)C = A(BC) = ABC$

Theorem 2b: $(A + B) + C = A + (B + C)$

A OR *B* *O*Red with *C* is some as *A* *O*Red with *B* OR *C*



A	*B*	*C*	*A + B*	*(A+B) + C*
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

=

A	*B*	*C*	*B + C*	*A+(B+C)*
0	0	0	0	0
0	0	0	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	0	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Fig. 1.9: Associative Law

Note that, $(A + B) + C = A + (B + C) = A + B + C$

The associative properties can be extended to any number variables:

$$A(BCD) = (AB)(CD) = (ABC)D = ABCD$$

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

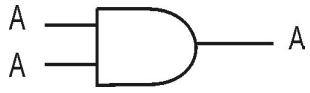
$$= A + B + C + D$$

(iii) ***The Idempotent Properties***

Theorem 3a : $AA = A$

Theorem 3b : $A + A = A$

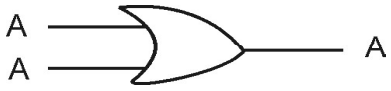
$AA = A$



If $A = 0$, then $AA = 0.0 = 0 = A$

If $A = 1$, then $AA = 1.1 = 1 = A$

$A + A = A$



If $A = 0$, then $A+A = 0 + 0 = 0 = A$

If $A = 1$, then $A+A = 1+1 = 1 = A$

Note: $A\,A\,A = A$

$A + A + A = A$

(iv) ***Identity Properties***

Theorem 4a : $A \cdot 1 = A$

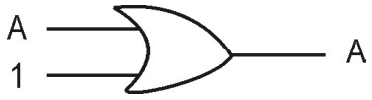
Theorem 4b : $A + 1 = 1$

$A \cdot 1 = A$

If $A = 0$, then $A \cdot 1 = 0.1 = 0 = A$

If $A = 1$, then $A \cdot 1 = 1.1 = 1 = A$

$A+1=1$



If $A = 0$, then $A+1 = 1+1 = 1$

If $A = 1$, then $A+1 = 1+1 = 1$

Note: $1 \cdot A = A$

$1 + A = 1$

$(AB+C) \cdot 1 = AB+C$

$AB+CD+1 = 1$

(v) ***The Null Properties***

Theorem 5a: $A \cdot 0 = 0$

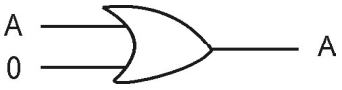
Theorem 5b: $A + 0 = A$

$\boldsymbol{A \cdot 0 = 0}$

If $A = 0$, then $A \cdot 0 = 0.0 = 0$

If $A = 1$, then $A \cdot 0 = 1.0 = 0$

$\boldsymbol{A + 0 = A}$



If $A = 0$, then $A + 0 = 0 + 0 = 0 = A$

If $A = 1$, then $A + 0 = 1 + 0 = 1 = A$

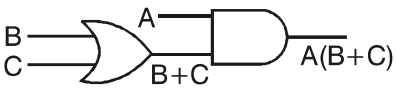
Note:

$(AB + CD) \cdot 0 = 0$

$ABC + D + 0 = ABC + D$

(vi) ***The Distributive Property***

Theorem 6: $A(B + C) = AB + AC$



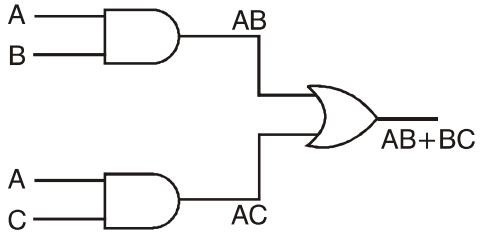


Fig. 1.10: Distributive Law

A	*B*	*C*	*(B + C)*	*A(B + C)*
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

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

=

A	*B*	*C*	*AB*	*AC*	*AB + AC*
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

$AB + AC = AB + AC$

$$AB(C+DE)=ABC+ABDE$$

$$(A+B)(C+D)=(A+B)C+(A+B)D$$
$$=AC+BC+AD+BD$$

Note: The distributive property is often used in reverse called as **factoring**.

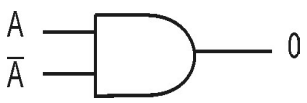
$$AB+AC=A(B+C)$$

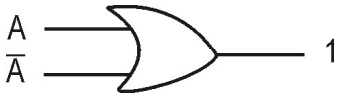
$$XYW+XYZ=XY(W+Z)$$

(vii) ***The Negative Properties***

Theorem 7a : $A\overline{A}=0$

Theorem 7b : $A+\overline{A}=1$





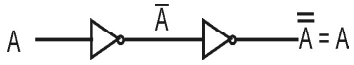
If $A=0, 0.1=0$ If $A=0, 0+1=1$

If $A=1, 1.0=0$ If $A=1, 1+0=1$

Note: $ABC+ABC=1$

(viii) ***The Double Negation Property***

Theorem 8 : $\overline{\overline{A}}=A$



If $A=0, \overline{A}=1, \overline{\overline{A}}=0=A$

If $A=1, \overline{A}=0, \overline{\overline{A}}=1=A$

Note: $\overline{\overline{A + BC}} = A + BC.$

(ix) ***The Absorption Properties***

Theorem 9a: $A + AB = A$

Theorem 9b: $A(A + B) = A$

Theorem 9c: $A+\overline{A}B=A+B$

9a) ***A + AB = A*** **Example**

$$A(1+B)=A\cdot 1=A$$

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

9b) $A(A+B)=A$

$$AA+AB=A+AB=A$$

$$XY\left(X\overline{Y}+WYZ\right)=X\overline{Y}$$

9c) $A+\overline{A}B=A+B$

$$A+\overline{A}B=A+AB+\overline{A}B$$

$$=A+\left(A+\overline{A}\right)B$$

$$=A+\left(1\cdot B\right)$$

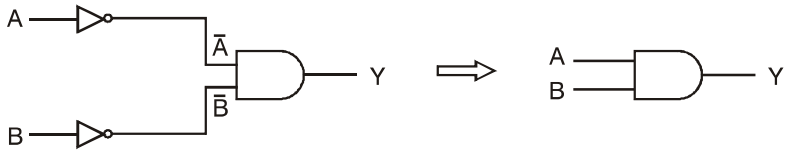
$$=A+B$$

$$\overline{A}=AB=\overline{A}+B$$

$$X+Y+\left(\overline{X+Y}\right)Z$$

$$=X+Y+Z$$

Bubbled AND Gate



AND gate with inverted inputs

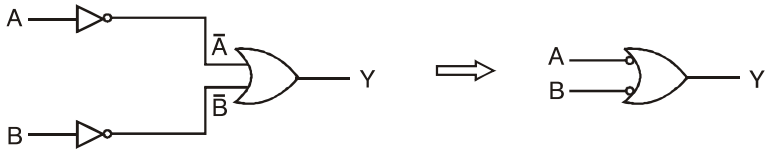
$Y=\overline{A}\overline{B}$

Bubbled AND gate - Truth Table

A	*B*	*Y*
0	0	1
0	1	0
1	0	0
1	1	0

Fig. 1.11: Bubbled AND gate

Bubbled OR Gate



OR gate with inverted inputs

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

Bubbled OR gate

Bubbled OR gate – Truth Table

A	*B*	*Y*
0	0	1
0	1	1
1	0	1
1	1	0

Fig. 1.12 : Bubbled OR gate

1.15 DE MORGAN'S THEOREMS

Theorem (1) : $\overline{A\,B}=\overline{A}+\overline{B}$

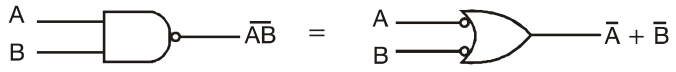
Theorem (2) : $\overline{A+B}=\overline{A}\,\overline{B}$

De Morgan's First Theorem

$\overline{A\,B}=\overline{A}+\overline{B}$

"The complement of a product equals the sum of the complements."

A NAND gate performs the same operation as a bubbled OR gate.



NAND

A	B	AB	$\overline{A\,B}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Bubbled OR

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

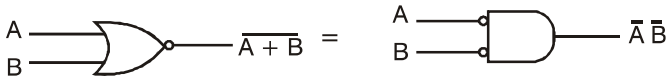
Fig. 1.13: De Morgan's Theorem 1

De Morgan's Second Theorem

$$\overline{A+B}=\overline{A}\,\overline{B}$$

"The complement of sum equals the product of the complements".

A NOR gate performs the same operation as Bubbled AND gate



NOR

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

Bubbled AND

A	B	$\overline{A}$	$\overline{B}$	$\overline{A}\,\overline{B}$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

Fig. 1.14: De Morgan's Theorem 2

Note: $A\,B \neq \overline{A}\,\overline{B}$; $A + B \neq \overline{A} + \overline{B}$

De Morgan's two theorems can be regarded as a single theorem by observing the following rule:

"Change the logic operation covered by the inversion bar, remove the inversion bar, and complement each variable that was originally covered by the bar."

(i) $\overline{A\,B\,C} = \overline{A} + \overline{B} + \overline{C}$ and $\overline{A + B + C} = \overline{A}\,\overline{B}\,\overline{C}$

(ii) $\overline{\overline{A}B} = \overline{\overline{A}} + \overline{B} = A + \overline{B}$

(iii) $\overline{A \cdot \overline{B} \cdot \overline{C}} = \overline{A} + \overline{\overline{B}} + \overline{\overline{C}} = \overline{A} + B + C$

(iv) $\overline{A\overline{B} + C} = \left(\overline{A} + \overline{\overline{B}}\right)\overline{C} = \left(\overline{A} + B\right)\overline{C}$

(v) $\overline{\left(A + \overline{B}\right)\overline{C}D + E} = \left[\overline{\left(A + \overline{B}\right)\overline{C}D}\right]\overline{E}$

$$= \left[\left(\overline{\overline{A + \overline{B}}}\right) + \overline{\overline{C}D}\right]\overline{E}$$

$$= \left[\left(A + \overline{B}\right) + \overline{\overline{C}} + \overline{D}\right]\overline{E}$$

$$= \left[A + \overline{B} + C + \overline{D}\right]\overline{E}$$

(vi) $\overline{A\,B\,C + \overline{B\,C}} = \left(\overline{A\,B\,C}\right)\overline{\overline{B\,C}}$

$$= \left(\overline{ABC}\right)(BC)$$

$$= \left(\overline{A} + \overline{B} + \overline{C}\right)B\,C$$

$$= \overline{A}B\,C + \overline{B}B\,C + B\,C\overline{C}$$

$$= \overline{A}B\,C + 0 + 0$$

$$= \overline{A}B\,C$$

The Boolean theorems are given in **Table 1.11.**

TABLE 1.11: Boolean Theorems

1. Commutative	1*a*	$AB = BA$	1*b*	$A + B = B + A$
2. Associative	2*a*	$(AB)C = A(BC)$	2*b*	$(A + B) + C = A + (B + C)$
3. Idempotent	3*a*	$AA = A$	3*b*	$A + A = A$
4. Identity	4*a*	$A \cdot 1 = A$	4*b*	$A + 1 = 1$
5. Null	5*a*	$A \cdot 0 = 0$	5*b*	$A + 0 = A$
6. Distributive	6	$A(B + C) = AB + AC$		
7. Negation	7*a*	$A\overline{A} = 0$	7*b*	$A + \overline{A} = 1$
8. Double Negation	8	$\overline{\overline{A}} = A$		
9. Absorption	9*a* 9*c*	$A + AB = A$ $A + \overline{A}B = A + B$	9*b*	$A(A + B) = A$
10. De Morgan		$\overline{AB} = \overline{A} + \overline{B}$		$\overline{A + B} = \overline{A}\,\overline{B}$

1.16 BOOLEAN RULES FOR SIMPLIFICATION

Boolean algebra finds its most practical use in the simplification of logic circuits. To translate a logic circuit's function into symbolic (Boolean) form, and apply certain algebraic rules to the resulting equation to reduce the number of terms and/or arithmetic operations, the simplified equation may be translated back into circuit form a logic circuit performing the same function with fewer components. If equivalent function may be achieved with fewer component, the result will be increased reliability and decreased cost of manufacture.

To this end, there are several rules of Boolean algebra presented in this section for use in reducing expressions to their simplest forms. The identities and properties already reviewed in this chapter are very useful in Boolean simplification, and for the most part bear similarity to many identities and properties of "normal" algebra. However, the rules shown in this section are all unique to Boolean mathematics.

Rule 1:

$$A + AB = A$$

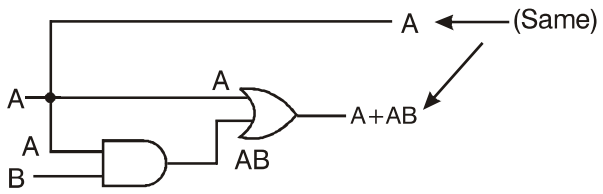


Fig. 1.15: Logic diagram for *A* + *AB* = *A*

This rule may be proven symbolically by factoring an "*A*" out of the two terms, then applying the rules of $A + 1 = 1$ and $1A = A$ to achieve the final result:

$A + AB$

↓ Factoring A out of both terms

$A(1 + B)$

↓ Applying identity $B + 1 = 1$

$A(1)$

↓ Applying identity $1A = A$

A

Rule 2: **(Dec. 2005)**

$A + \overline{A}B = A + B$

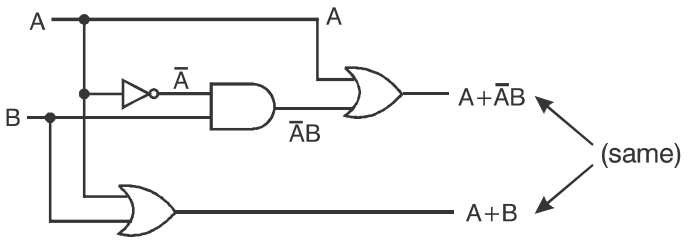


Fig. 1.16: Logic diagram for $A + \overline{A}B = A + B$

$A + \overline{A}B$

↓ Applying the previous rule to expand A term $A + AB = A$

$A + AB + \overline{A}B$

↓ Factoring B out of 2nd and 3rd terms

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

↓ Applying negation $A + \overline{A} = 1$

$A + B(1)$

↓ Applying identity $1B = B$

$A + B$

Rule 3:

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

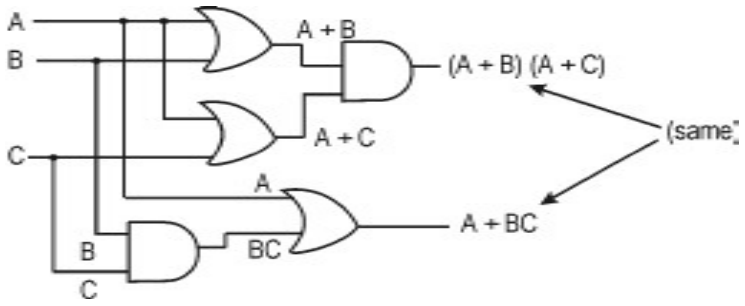


Fig. 1.17 : Logic diagram (*A* + *B*) (*A* + *C*) = *A* + *BC*

And, the corresponding proof:

$(A + B)(A + C)$

↓ Distributing terms

$AA + AC + AB + BC$

↓ Applying identity $AA = A$

$A + AC + AB + BC$

↓ Applying rule $A + AB = A$ to the $A + AC$ term

$A + AB + BC$

↓ Applying rule $A + AB = A$ to the $A + AB$ term

$A + BC$

Example 1.81: Simplify the expression $AB + BC(B + C)$

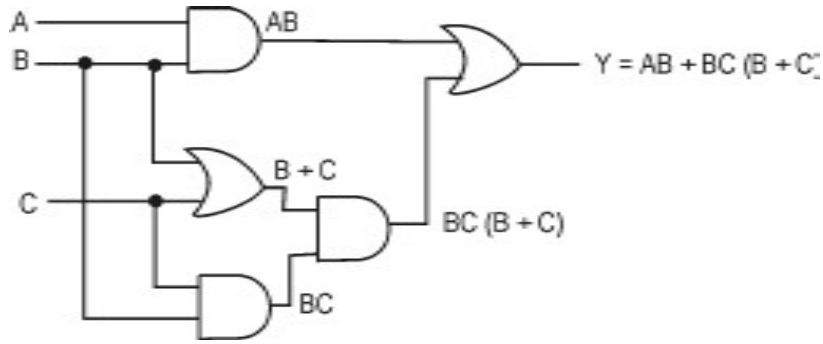


Fig. 1.18: Logic diagram

$AB + BC(B + C)$

↓ Distributing terms

$AB + BBC + BCC$

↓ Applying identity $AA = A$ to 2nd and 3rd terms

$AB + BC + BC$

↓ Applying identity $A + A = A$ to 2nd and 3rd terms

$AB + BC$

↓ Factoring B out of terms

$B(A + C)$

The final expression, $B(A + C)$, is much simpler than the original, yet performs the same function. The truth tables for these two expressions should be identical.

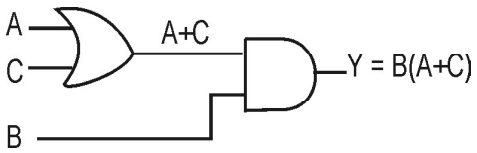


Fig. 1.19 : Simplified function diagram

1.17 MINIMIZATION OF BOOLEAN EXPRESSIONS

Example 1.82: $W = ABC + CAB + BAC$

$= ABC + ABC + ABC$ (Theorem 1*a*)

$= ABC + ABC$ (Theorem 3*b*)

$= ABC$

Example 1.83: $X = A\,B\,C + A\,B\overline{C}$

$= A\,B\left(C + \overline{C}\right)$ (Theorem 6)

$= A\,B \cdot 1$ (Theorem 7*b*)

$= AB$ (Theorem 4*a*)

Example 1.84: $W = X\left(\overline{X}YZ + \overline{X}Y\overline{Z}\right)$

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

$= 0 \cdot YZ + 0 \cdot Y\overline{Z}$

$= 0 + 0$

$= 0$

Example 1.85: $Z = A\overline{B}\overline{C} + A\overline{B}C + \overline{A}B\,C + \overline{A}\overline{B}C$

$= A\overline{B}\left(\overline{C} + C\right) + \overline{A}B\,C + \overline{A}\overline{B}C$ (Theorem 6)

$= A\overline{B} \cdot 1 + \overline{A}C\left(B + \overline{B}\right)$ (Theorem 7*b*)

$= A\overline{B} \cdot 1 + \overline{A}C \cdot 1$

$= A\overline{B} + \overline{A}C$ (Theorem 4*a*)

Example 1.86: $W = \left(X_1 + X_2\right)\left(\overline{X}_1 + X_1X_2\right) + \left(\overline{X}_2 + X_1\overline{X}_2\right)$

$= X_1\overline{X}_1 + X_1X_1X_2 + X_2\overline{X}_1 + X_2X_1X_2 + \overline{X}_2 + X_1\overline{X}_2$ (Theorem 6)

$= 0 + X_1X_2 + X_2\overline{X}_1 + X_1X_1 + \overline{X}_2 + X_1\overline{X}_2$ (Theorem 7*a*, 3*a*)

$= X_2X_1 + X_2\overline{X}_1 + X_1\overline{X}_2 + X_1X_2 + \overline{X}_2$

$= X_2\left(X_1 + \overline{X}_1\right) + X_1\left(X_2 + \overline{X}_2\right) + \overline{X}_2$

$= X_2 \cdot 1 + X_1 \cdot 1 + \overline{X}_2$ (Theorem 7*b*)

$= X_2 + X_1 + \overline{X}_2$

$= X_1 + \left(X_2 + \overline{X}_2\right)$ (Theorem 7*b*)

$= X_1 + 1$ (Theorem 4*b*)

$= 1$

Example 1.87: $A = WXY\left(W\overline{X} + W\overline{Y}\right) + W\overline{X}Y\left(\overline{W}\,\overline{X} + X\overline{Y}\right)$

$= WXYW\overline{X} + WXYW\overline{Y} + W\overline{X}Y\overline{W}\,\overline{X} + W\overline{X}YX\overline{Y}$

$= WYWX\overline{X} + WXWY\overline{Y} + W\overline{W}\,\overline{X}Y\overline{X} + W\overline{X}XY\overline{Y}$ (Theorem 7*a*)

$= WYW \cdot 0 + WXW \cdot 0 + 0 \cdot \overline{X}Y\overline{X} + W\overline{X}X \cdot 0$ (Theorem 5*a*)

$= 0 + 0 + 0 + 0$ (Theorem 3*b*)

$= 0$

Example 1.88: $Y = ABC + AB + A$

$= A(BC + B + 1)$ (Theorem 4b)

$= A \cdot 1$ (Theorem 4a)

$= A$

Example 1.89: $W = X\overline{Y} + \overline{Y}ZX$

$= X\overline{Y} + X\overline{Y}Z$ (Theorem 1*a*)

$= X\overline{Y}$ (Theorem 9*a*)

Example 1.90: $W = \overline{A} + \overline{BC}$

$= \overline{\overline{A}}\;\overline{\overline{B}}\;\overline{\overline{C}}$ (De Morgan's Theorem 10*b*)

$= ABC$

Example 1.91: $W = \overline{\overline{A}\,\overline{B}\left(\overline{C} + \overline{D}\right)}$

$= \overline{\overline{A}}\,\overline{\overline{B}} + \overline{\left(\overline{C} + \overline{D}\right)}$ (Theorem 10*a*)

$= \overline{\overline{A}} + \overline{\overline{B}} + \overline{\overline{C}}\;\overline{\overline{D}}$ (Theorem 10*a*, 10*b*)

$= A + B + CD$

Example 1.92: $W = \overline{\overline{A}\left(C + \overline{D}\right)} + \overline{\overline{C}\left(A + \overline{B}\right)}$

$= \overline{\overline{A}} + \overline{\left(C + \overline{D}\right)} + \overline{\overline{C}}\;\overline{\left(A + \overline{B}\right)}$ (Theorem 10*a*, 10*b*)

$= A + \overline{C}\;\overline{\overline{D}} + C + \overline{A}\;\overline{\overline{B}}$

$= A + \overline{C}D + C + \overline{A}B$ (Theorem 8)

$= A + \overline{A}B + C + \overline{C}D$ (Theorem 1*b*)

$= \text{A} + \text{B} + \text{C} + \text{D}$ (Theorem 9*c*)

Example 1.93: $W = \overline{\left(\overline{A} + \overline{B}\right)\left(\overline{C} + AB\right)}$

$= \overline{\overline{A}}\ \overline{\overline{B}} + \overline{\overline{C}}\,\overline{AB}$

$= \overline{A}\,\overline{B} + C\ \overline{AB}$

Example 1.94: $Y = A + AB + ABC + ABCD$

$= A(1 + B + BC + BCD)$

$= A \cdot 1 = A$

Example 1.95: $W = A\left(A + AB\right)\left(A + ABC\right)\left(A + ABCD\right)$

$= \left(A \cdot (A)\right)\left[\left(A\left(1 + BC\right)\right)\right]\left[A\left(1 + BCD\right)\right]$

$= AA\left(A \cdot 1\right)\left(A \cdot 1\right)$

$= A\,A\,A\,A = A$

Example 1.96: $\overline{\left(A_1\,\overline{A}_2\,A_3\right)\overline{\left(A_1\,\overline{A}_2\right)} + A_2 A_3}$

$= \overline{\left(A_1\,\overline{A}_2\,A_3\right)} + \overline{\overline{\left(A_1 + \overline{A}_2\right)}} + \overline{A_2\,A_3}$

$= \left(\overline{A}_1\,A_2\,\overline{A}_3\right)\left(A_1 + \overline{A}_2\right) + \overline{A_2\,A_3}$

$= \overline{A}_1\,A_1\,A_2\,\overline{A}_3 + \overline{A}_1\,A_2\,\overline{A}_2\,\overline{A}_3 + \overline{A_2\,A_3}$

$= 0 + 0 + \overline{A_2\,A_3} = \overline{A_2\,A_3}$

Example 1.97: $Y = \left(\overline{A} + B\right)\left(A + B\right)$

$= \overline{A}A + \overline{A}B + BA + BB$

$= \overline{A}B + AB + B$

$= \left(\overline{A} + A\right)B + B$

$= 1 \cdot B + B = B + B = B.$

Example 1.98: $Y=\overline{A}\overline{B}\overline{C}+\overline{A}B\overline{C}+A\overline{B}\overline{C}+A\,B\overline{C}$

$=\left(\overline{A}\overline{B}+\overline{A}B+A\overline{B}+A\,B\right)\overline{C}$

$=\left(\overline{A}\left(\overline{B}+B\right)+A\left(\overline{B}+B\right)\right)\overline{C}$

$=\left(\overline{A}\cdot 1+A\cdot 1\right)\overline{C}$

$=\left(\overline{A}+A\right)\overline{C}$

$=1\cdot\overline{C}=\overline{C}\,.$

Example 1.99: $Y = AB + A(B + C) + B(B + C)$

$= AB + AB + AC + BB + BC$

$= AB + AB + AC + B + BC$ $(BB = B)$

$= AB + AC + B + BC$ $(AB + AB = AB)$

$= AB + AC + B$ $(B + BC = B)$

$= AC + B$ $(B + BA = B)$

Example 1.100: $Y=\left(A\overline{B}\left(C+B\,D\right)+\overline{A}\overline{B}\right)C$

$=\left(A\overline{B}C+A\overline{B}B\,D+\overline{A}\overline{B}\right)C$

$=\left(A\overline{B}C+A\cdot 0\cdot D+\overline{A}\overline{B}\right)C$ $\left(\overline{B}B=0\right)$

$=\left(A\overline{B}C+0+\overline{A}\overline{B}\right)C$ $\left(A\cdot 0\cdot D=0\right)$

$=\left(A\overline{B}C+\overline{A}\overline{B}\right)C$

$=A\overline{B}C\,C+\overline{A}\overline{B}C$

$=A\overline{B}C+\overline{A}\overline{B}C$ $(CC = C)$

$=\overline{B}C\left(A+\overline{A}\right)=\overline{B}C\cdot 1$ $\left(A+\overline{A}=1\right)$

$=\overline{B}C$

Example 1.101: $Y=\overline{A}B\,C+A\overline{B}\,\overline{C}+\overline{A}\,\overline{B}\,\overline{C}+A\overline{B}C+A\,B\,C$

$=B\,C\left(A+\overline{A}\right)+\overline{A}\,\overline{B}\,\overline{C}+A\overline{B}\,\overline{C}+A\overline{B}C$

$=B\,C+A\overline{B}\left(\overline{C}+C\right)+\overline{A}\,\overline{B}\,\overline{C}$ $\left(A+\overline{A}=1\right)$

$$= BC + A\overline{B} + \overline{A}\overline{B}\overline{C} \qquad (C + \overline{C} = 1)$$

$$= BC + \overline{B}\left(A + \overline{A}\overline{C}\right)$$

$$= BC + \overline{B}\left(A + \overline{C}\right) \qquad \left(A + \overline{A}\overline{C} = A + \overline{C}\right)$$

$$= BC + A\overline{B} + \overline{B}\overline{C}$$

Example 1.102: $Y = \overline{AB + AC} + \overline{A}\overline{B}C$

$$= \left(\overline{AB}\right)\left(\overline{AC}\right) + \overline{A}\overline{B}C$$

$$= \left(\overline{A} + \overline{B}\right)\left(\overline{A} + \overline{C}\right) + \overline{A}\overline{B}C$$

$$= \overline{A}\overline{A} + \overline{A}\overline{C} + \overline{A}\overline{B} + \overline{B}\overline{C} + \overline{A}\overline{B}C$$

$$= \overline{A} + \overline{A}\overline{C} + \overline{A}\overline{B} + \overline{B}\overline{C} + \overline{A}\overline{B}C \qquad (\overline{A}\overline{A} = \overline{A})$$

$$= \overline{A} + \overline{A}\overline{C} + \overline{A}\overline{B} + \overline{B}\overline{C} \qquad (\overline{A}\overline{B} + \overline{A}\overline{B}C = \overline{A}\overline{B})$$

$$= \overline{A} + \overline{A}\overline{B} + \overline{B}\overline{C} \qquad \left(\overline{A} + \overline{A}\overline{C} = \overline{A}\left(1 + \overline{C}\right) = \overline{A}\right)$$

$$= \overline{A} + \overline{B}\overline{C} \qquad \left(\overline{A} + \overline{A}\overline{B} = \overline{A}\left(1 + \overline{B}\right) = \overline{A}\right)$$

Example 1.103: $Y = AB + \overline{AC} + A\overline{B}C\left(AB + C\right)$

$$= AB + \overline{AC} + A\overline{B}C \cdot AB + A\overline{B}CC$$

$$= AB + \overline{AC} + A\overline{B}CC \qquad \left(AACB\overline{B} = AAC \cdot 0 = 0\right)$$

$$= AB + \overline{AC} + A\overline{B}C \qquad \left(C \cdot C = C\right)$$

$$= AB + \overline{A} + \overline{C} + A\overline{B}C$$

$$= AB + \overline{A} + \overline{C} + \overline{B}C \qquad \left(A + \overline{A}B = A + B\right)$$

$$= \overline{A} + AB + \overline{C} + C\overline{B}$$

$$= \overline{A} + B + \overline{C} + \overline{B}$$

$$= \overline{A} + \overline{C} + 1 \qquad \left(B + \overline{B} = 1\right)$$

$$= 1$$

Example 1.104: $Y = \left(\overline{A} + B\right)\left(A + B\right)$

$$= \overline{A}A + \overline{A}B + AB + BB$$

$$= \overline{A}B + AB + BB \qquad \left(A\overline{A} = 0\right)$$

$$= \overline{A}B + AB + B \qquad \left(BB = B\right)$$

$$= \left(\overline{A} + A + 1\right)B \qquad \left(\overline{A} + A + 1 = 1\right)$$

$$= B$$

Example 1.105: $Y = AB + A(B+C) + B(B+C)$

$= AB + AB + AC + BB + BC$

$= AB + AB + AC + B + BC$ $(BB = B)$

$= AB + AC + B + BC$ $(AB + AB = AB)$

$= AB + AC + B$ $(B + BC = B)$

$= AB + B + AC$

$= B + AC$ $(AB + B = B)$

Example 1.106: $Y = \overline{AB + AC} + \overline{A}\,\overline{B}C$

$= (\overline{AB}) \cdot (\overline{AC}) + \overline{A}\,\overline{B}C$

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

$= \overline{A}\,\overline{A} + \overline{A}\,\overline{C} + \overline{A}\,\overline{B} + \overline{B}\,\overline{C} + \overline{A}\,\overline{B}C$

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

$= \overline{A} + \overline{B}\,\overline{C} + \overline{A}\,\overline{C} + \overline{A}\,\overline{B}$ $(\overline{A}\,\overline{B} + \overline{A}\,\overline{B}C = \overline{A}\,\overline{B}(1+C) = \overline{A}\,\overline{B})$

$= \overline{A} + \overline{B}\,\overline{C} + \overline{A}\,\overline{B}$ $(\overline{A} + \overline{A}\,\overline{C} = \overline{A}(1+C) = \overline{A})$

$= \overline{A} + \overline{B}\,\overline{C}$ $(\overline{A} + \overline{A}\,\overline{B} = \overline{A}(1+\overline{B}) = \overline{A})$

Example 1.107: $Y = \overline{\overline{\text{ABC.D}}}$

$= \overline{ABC} + \overline{D}$

$= (\overline{AB})C + \overline{D}$

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

$= \overline{A}C + \overline{B}C + \overline{D}$

Example 1.108: Simplify the following Boolean expression to a minimum number of literals:

(i) $\overline{\text{AC}} + \text{ABC} + \text{A}\overline{\text{C}}$ **(Dec 2011)**

$= \overline{\text{A}} + \overline{\text{C}} + \text{ABC} + \text{A}\overline{\text{C}}$ (Theorem 10)

$= (\overline{\text{A}} + \text{A}\overline{\text{C}}) + (\overline{\text{C}} + \text{ABC})$ (Theorem 9C)

$= (\overline{\text{A}} + \overline{\text{C}}) + (\overline{\text{C}} + \text{AB})$

$= (\overline{\text{A}} + \text{AB}) + \overline{\text{C}} + \overline{\text{C}}$

$= \overline{\text{A}} + \text{B} + \overline{\text{C}} + \overline{\text{C}}$

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

$= 1 + \overline{B} = 1$

(ii) $\mathbf{XYZ + \overline{X}Y + XY\overline{Z}}$

$= y\left(xz + \overline{x} + x\overline{z}\right)$

$= y\left(xz + \overline{x} + \overline{z}\right)$

$= y\left(x + \overline{x} + \overline{z}\right)$

$= y\left(1 + \overline{z}\right)$

$= y(1)$

$= y$

(iii) $\mathbf{XY + YZ + X\overline{Y}Z}$

$= xy + z\left(y + \overline{y}x\right)$

$= xy + z\left(y + x\right)$

$= xy + yz + xz$

$= xy + yz + zx$

(iv) $\mathbf{A\overline{B} + ABD + AB\overline{D} + \overline{A}\,\overline{C}\,\overline{D} + \overline{A}B\overline{C}}$

$= A\overline{B} + AB + \overline{A}\,\overline{C}\left(\overline{D} + B\right)$

$= A + \overline{A}\left(\overline{C}\,\overline{D} + B\overline{C}\right)$

$= A + \overline{A}\,\overline{C}\,\overline{D} + \overline{A}B\overline{C}$

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

(v) $\mathbf{BD + BC\overline{D} + A\overline{BCD}}$

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

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

$= BD + BC + A\overline{B} + A\overline{C} + A\overline{D}$

1.18 DUALITY

The dual is formed by replacing

AND with OR

OR with AND

0 with 1

1 with 0 in a Boolean expression.

The variables and components are left unchanged. This rule for forming the dual as,

$$\left[f\left(X_1, X_2, \ldots, X_N, 0, 1, +, \cdot\right)\right]^D = f\left(X_1, X_2, \ldots, X_N, 1, 0, \cdot, +\right)$$

If $F = A\,B\,\overline{C} + \overline{A}\,\overline{B}\,C$, then the dual expression is $F^D = \left(A + B + \overline{C}\right)\left(\overline{A} + \overline{B} + C\right)$

Table 1.12 lists the postulates and theorems related to duality theorem.

TABLE 1.12: Duality Theorem

Expression	***Dual***
$X + 0 = X$	$X \cdot 1 = X$
$X + \overline{X} = 1$	$X \cdot \overline{X} = 0$
$X + X = X$	$X \cdot X = X$
$X + 1 = 1$	$X \cdot 0 = 0$
$X + Y = Y + X$	$XY = YX$
$X + \left(Y + Z\right) = \left(X + Y\right) + Z$	$X\left(YZ\right) = \left(XY\right)Z$
$X\left(Y + Z\right) = XY + XZ$	$X + YZ = \left(X + Y\right)\left(X + Z\right)$
$\left(\overline{X + Y}\right) = \overline{X}\,\overline{Y}$	$\left(\overline{XY}\right) = \overline{X} + \overline{Y}$
$X + XY = X$	$X(X + Y) = X$

Proof:

1. $X + X = X$

$$\begin{aligned} X + X &= \left(X + X\right) \cdot 1 \\ &= \left(X + X\right)\left(X + \overline{X}\right) \\ &= X + X\,\overline{X} \\ &= X + 0 \\ &= X \end{aligned}$$

$X \cdot X = X$

$$\begin{aligned} X \cdot X &= XX + 0 \\ &= XX + X\,\overline{X} \\ &= X\left(X + \overline{X}\right) \\ &= X \cdot 1 \\ &= X \end{aligned}$$

2. $X + 1 = 1$

$$\begin{aligned} X + 1 &= 1 \cdot \left(X + 1\right) \\ &= \left(X + \overline{X}\right)\left(X + 1\right) \\ &= X + \overline{X} \cdot 1 \\ &= X + \overline{X} \\ &= 1 \end{aligned}$$

$X \cdot 0 = 0$

3. $\overline{ABC + DEF} = \left(\overline{A} + \overline{B} + \overline{C}\right)\left(\overline{D} + \overline{E} + \overline{F}\right)$

$$\begin{aligned} \overline{ABC} + \overline{DEF} &= \overline{ABC} \cdot \overline{DEF} \\ &= \left(\overline{A} + \overline{B} + \overline{C}\right)\left(\overline{D} + \overline{E} + \overline{F}\right) \end{aligned}$$

1.19 BOOLEAN FUNCTIONS

A Boolean function is an expression formed with binary variables, the two binary operators OR and AND, the unary operator NOT, parentheses and an equal sign. For a given value of the variables, the function can be either 0 or 1.

Consider, for example, the Boolean function

$$F_1 = xy\bar{z}$$

$F_1 = 1$ if $x=1, y=1$ and $\bar{z}=1$

$F_1 = 0$; otherwise

Boolean function also be represented in a Truth Table. To represent a function in a truth table, we need a list of the 2^n combinations of 1's and 0's of the 'n' binary variables and a column showing the combinations for which the function is equal to 1 or 0.

TABLE 1.13 : Boolean Functions

X Y Z	$F_1 = XY\bar{Z}$	$F_2 = X + \bar{Y}Z$	$F_3 = \bar{X}\bar{Y}Z + \bar{X}YZ + X\bar{Y}$	$F_4 = X\bar{Y} + \bar{X}Z$
0 0 0	0	0	0	0
0 0 1	0	1	1	1
0 1 0	0	0	0	0
0 1 1	0	0	1	1
1 0 0	0	1	1	1
1 0 1	0	1	1	1
1 1 0	1	1	0	0
1 1 1	0	1	0	0

When $X = 1$, $Y=1$, $Z=1$

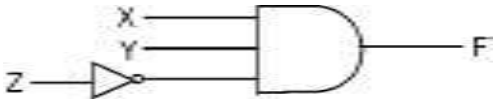
$F_1 = 1 \cdot 1 \cdot 0 = 0$

$F_2 = 1 + 0.1 = 1$

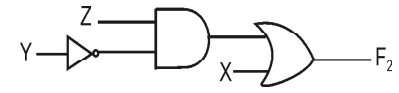
$F_3 = 0.0.1 + 0.1.1 + 1.0 = 0$

$F_4 = 1.0 + 0.1 = 0$

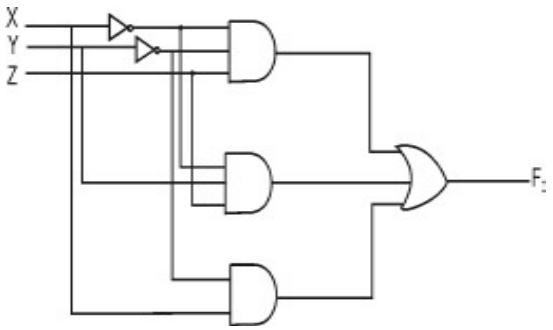
A Boolean function may be transformed from an algebraic expression into a logic diagram composed of AND, OR and NOT gates



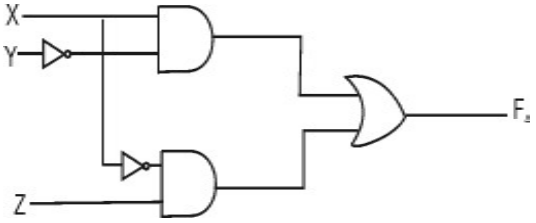
(a) $F_1 = XY\bar{Z}$



(b) $F_2 = X + \overline{Y}Z$



(c) $F_3 = \overline{X}\,\overline{Y}Z + \overline{X}YZ + X\overline{Y}$



(d) $F_4 = X\overline{Y} + \overline{X}Z$

Fig. 1.20 : Implementation of Boolean functions with gates

To implement a Boolean function with a less number of gates we have to minimize literals and the number of terms. Usually, literals (Boolean variables in complemented or uncomplemented form) and terms are arranged in one of the two standard forms of switching equations:

- Sum of Product form (SOP)
- Product of Sum form (POS)

1.20 PRODUCT-OF-SUMS METHODS

The logical product of those fundamental sums that produce output 0's in the truth table. The corresponding logic circuit is an OR-AND circuit or the equivalent NOR-NOR circuit.

TABLE 1.14 : POS table

A	*B*	*C*	*Y*
0	0	0	$0 \rightarrow A+B+C$
0	0	1	1
0	1	0	1
0	1	1	$0 \rightarrow A+\overline{B}+\overline{C}$
1	0	0	1
1	0	1	1
1	1	0	$0 \rightarrow \overline{A}+\overline{B}+C$
1	1	1	1

$Y = 0$, when $A = 0, B = 1$ and $C = 1$. So this particular combination makes the output of an OR gate equal to 0, when $\overline{B}=0$ and $\overline{C}=0$. Thus $A+\overline{B}+\overline{C}$ is on sum term. Similarly other two sum terms are $A + B + C$ and $\overline{A}+\overline{B}+C$. Thus the standard product of sum form is,

$$F=(A+B+C)(A+\overline{B}+\overline{C})(\overline{A}+\overline{B}+C)$$

Figure 1.21 Shows the OR-AND logic circuit and **Figure 1.22** shows the NOR-NOR logic circuit for this expression.

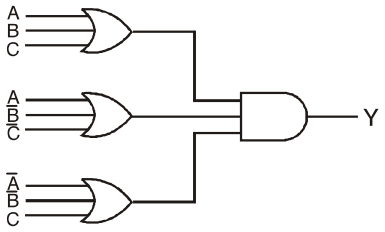


Fig. 1.21: OR-AND circuit

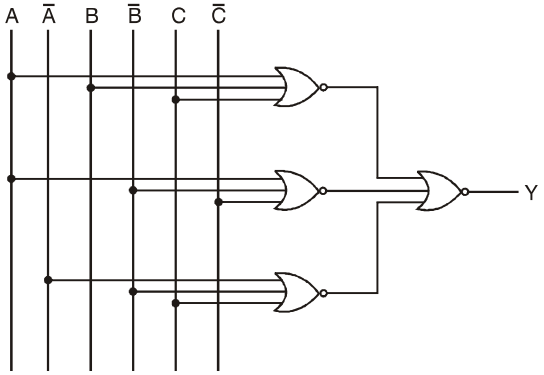


Fig. 1.22: NOR-NOR circuit

1.21 SUM-OF-PRODUCTS METHOD

The logical sum of those fundamental products that produce output 1's in the truth table. The corresponding logic circuit is an AND-OR circuit or the equivalent NAND-NAND circuit.

For example, $F = A\overline{B}C + ABC + \overline{A}B\overline{C}$, is a standard sum of products of A, B and C are the only variables pertaining to the logic. Note that this expression is not in simplest form because we can write,

$$F = A\overline{B}C + ABC + \overline{A}B\overline{C}$$

$$= AC\left(\overline{B} + B\right) + \overline{A}B\overline{C}$$

$$= AC + \overline{A}B\overline{C}$$

This simplified expression is a sum of products; but not a **standard** sum of products.

The logic expression corresponding to a given truth table can be written in a standard sum-of-products form of writing one product term for each input combination that produces an output of 1. These product terms are ORed together to create the standard sum of products.

TABLE 1.15 : SOP table

A	*B*	*C*	*F*	
0	0	0	0	
0	0	1	1	$\leftarrow \overline{A}\overline{B}C$
0	1	0	0	
0	1	1	0	
1	0	0	1	$\leftarrow A\overline{B}\overline{C}$
1	0	1	1	$\leftarrow A\overline{B}C$
1	1	0	0	
1	1	1	0	

We note that F is 1 when $A = 0, B = 0$ and $C = 1$, so this particular combination makes the output of an AND gate equal to 1 when $\overline{A}$ and $\overline{B}$ roman and C are all equal to 1. Thus $\overline{A}\overline{B}C$ is one product term. Similarly other two product terms are $A\overline{B}\overline{C}$ and $A\overline{B}C$. Thus the standard sum-of-products form is,

$$F = \overline{A}\overline{B}C + A\overline{B}\overline{C} + A\overline{B}C$$

Consider the truth table given in **Table 1.16.**

TABLE 1.16: Design Truth Table

A	*B*	*C*	*Y*	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\rightarrow \overline{A}BC$
1	0	0	0	
1	0	1	1	$\rightarrow A\overline{B}C$
1	1	0	1	$\rightarrow AB\overline{C}$
1	1	1	1	$\rightarrow ABC$

$$Y = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

Figure 1.23 shows the AND-OR logic circuit and **Figure 1.24** shows the NAND-NAND circuit for the expression.

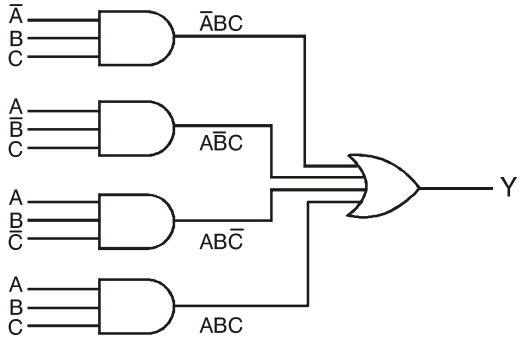


Fig. 1.23: AND-OR circuit

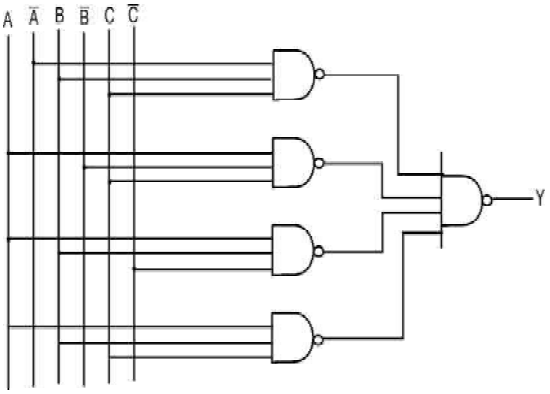


Fig. 1.24: NAND-NAND circuit

Example 1.109: Suppose a truth table has a low output for the first three input conditions: 000, 001 and 010. If all other outputs are high, what is the product-of-sum (POS) form?

Solution: $Y=(A+B+C)(A+B+\overline{C})(A+\overline{B}+C)$

Example 1.110: Suppose a 3 variable truth table has a high output for these input conditions: 000, 010, 100 and 110. What is the sum-of-product (SOP) form?

Solution: $Y=\overline{A}\,\overline{B}\,\overline{C}+\overline{A}B\overline{C}+A\overline{B}\,\overline{C}+AB\overline{C}$

Example 1.111: Convert the Boolean expressions to SOP form.

(a) $AB + B(CD + EF) = AB + BCD + BEF$

(b) $(A + B)\,(B + C + D) = AB + AC + AD + BB + BC + BD$

(c) $\overline{(\overline{A+B})+C}=\overline{(\overline{A+B})}\,\overline{C}$

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

$= A\overline{C} + B\overline{C}$

1.22 MINTERMS

The 'n' variables forming an AND term, with each variable being primed or unprimed, provide 2^n possible combinations, called Minterms or Standard Products.

Consider two binary variables x and y combined with an AND operation. Since each variable may appear in either form, there are 4 possible combinations:

$$\overline{x}\,\overline{y}, \overline{x}y, x\overline{y} \text{ and } xy$$

Each of these four AND terms represents one of the distinct areas in the Venn diagram and is called a Minterm.

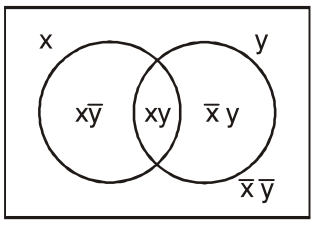


Fig. 1.25 : Venn Diagram for two variables

The 2^n difference minterms may be determined by a method similar to one shown in **Table 1.17** for 3 variables. The binary numbers from 0 to $2^n - 1$ are listed under the 'n' variables. Each Minterm

is obtained from an AND term of the '*n*' variables, with each variable being primed, if the corresponding bit of the binary number is a '0' and unprimed if a '1'.

Symbol for Minterm $\Rightarrow$ M_j

TABLE 1.17 : Minterms and Maxterms

			Minterms		Maxterms	
X	*Y*	*Z*	Term	Symbol	Term	Symbol
0	0	0	$\bar{x}\bar{y}\bar{z}$	m_0	$x+y+z$	M_0
0	0	1	$\bar{x}\bar{y}z$	m_1	$x+y+\bar{z}$	M_1
0	1	0	$\bar{x}y\bar{z}$	m_2	$x+\bar{y}+z$	M_2
0	1	1	$\bar{x}yz$	m_3	$x+\bar{y}+\bar{z}$	M_3
1	0	0	$x\bar{y}\bar{z}$	m_4	$\bar{x}+y+z$	M_4
1	0	1	$x\bar{y}z$	m_5	$\bar{x}+y+\bar{z}$	M_5
1	1	0	$xy\bar{z}$	m_6	$\bar{x}+\bar{y}+z$	M_6
1	1	1	xyz	m_7	$\bar{x}+\bar{y}+\bar{z}$	M_7

1.23 MAXTERMS

The '*n*' variables forming an OR term, with each variable being primed or unprimed, provide in 2^n possible combinations, called Maxterms or Standard sums.

The eight maxterms for 3 variables, together with their symbolic designation are listed in Table. Each maxterm is obtained from an OR term of the '*n*' variables, with each variable being unprimed if the corresponding bit is a 0 and primed if a 1.

Each Maxterm is the complement of its corresponding Minterm and Vice versa.

A Boolean function may be expressed algebraically from truth table **(Table 1.18)** by forming a minterm for each combination of the variables that produces a 1 in the function and then taking the OR of all those terms for example the function f_1 in the table is determined by expressing the combinations 001, 100 and 111 as $\bar{x}\bar{y}z, x\bar{y}\bar{z}$ and xyz.

$$f_1 = \bar{x}\bar{y}z + x\bar{y}\bar{z} + xyz = m_1 + m_4 + m_7 = 1$$

$$f_2 = \bar{x}yz + x\bar{y}z + xy\bar{z} + xyz = m_3 + m_5 + m_6 + m_7$$

"Any Boolean function can be expressed as a sum of Minterms (by 'SUM' is meant the ORing of terms)".

The complement of f_1 is

$$\overline{f}_1 = \left(\bar{x}\bar{y}\bar{z} + \bar{x}y\bar{z} + \bar{x}yz + x\bar{y}z + xy\bar{z}\right)$$

TABLE 1.18 : Truth Table for f_1 and f_2

x	y	z	Function f_1	Function f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

The complement of $\overline{f}_1$ gives the function f_1,

$$= \overline{\left(\bar{x}\bar{y}\bar{z} + \bar{x}y\bar{z} + \bar{x}yz + x\bar{y}z + xy\bar{z}\right)}$$

$$f_1 = \left(x+y+z\right)\left(x+\bar{y}+z\right) + \left(x+\bar{y}+\bar{z}\right) + \left(\bar{x}+y+\bar{z}\right) + \left(\bar{x}+\bar{y}+z\right)$$

$$= M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6$$

Similarly, it is possible to read the expression for f_2 from the table,

$$f_2 = \left(x+y+z\right)\left(x+y+\bar{z}\right)\left(x+\bar{y}+z\right) + \left(\bar{x}+y+z\right)$$

$$= M_0 \cdot M_1 \cdot M_2 \cdot M_4$$

"Any Boolean function can be expressed as a product of Maxterms (by 'product' is the meant of ANDing of terms)."

1.24 CANONICAL FORM

Boolean functions expressed as a sum of minterms $\left(\sum m\right)$ or product of maxterms (πM) are said to be in canonical forms.

1.24.1 Sum of Minterms

The sum of minterms of a Boolean function is obtained in two ways:

- from truth table
- from algebraic expression.

The following example clarifies these procedure:

Example 1.112: Express the Boolean function $F = A + \overline{B}C$ in a sum of minterms.

Method 1: The truth table for the function $F = A + \overline{B}C$ is shown in **Table 1.19.**

TABLE 1.19 : Truth table for $F = A + \overline{B}C$

A	B	C	$\overline{B}$	$\overline{B}C$	$F = A + \overline{B}C$
0	0	0	1	0	0
0	0	1	1	1	1
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	0	0	1
1	1	1	0	0	1

The combination of variables that produces '1' in the function are the minterms and then taking the OR of all these minterms is the sum of minterms.

$$F(A, B, C) = m_1 + m_4 + m_5 + m_6 + m_7$$
$$= \sum m(1,4,5,6,7)$$

Method 2: Explaining the expression into a sum of AND terms. Each term is then inspected to see if it contains all the variables. If it misses one or more variables, it is ANDed with an expression such as $(x + \overline{x})$, where x is one of the missing variables.

The Boolean function, $F = A + \overline{B}C$

The function has three variables. The first term A is missing two variables and the second term $\overline{B}C$ is missing one variable. Therefore, the first term

$$A = A(B + \overline{B})$$
$$= AB + A\overline{B}$$

This is still missing one variable, therefore

$$A = AB(C + \overline{C}) + A\overline{B}(C + \overline{C})$$
$$= ABC + AB\overline{C} + A\overline{B}C + A\overline{B}\overline{C}$$

The second term, $B\overline{C} = \overline{B}C(A + \overline{A})$

$$= A\overline{B}C = \overline{A}\overline{B}C$$

Combining all terms, we obtain,

$$F = A + \overline{B}C$$
$$= ABC + AB\overline{C} + A\overline{B}C + A\overline{B}\overline{C} + A\overline{B}C + \overline{A}\overline{B}C$$
$$= ABC + AB\overline{C} + A\overline{B}C + A\overline{B}\overline{C} + \overline{A}\overline{B}C$$
$$= \overline{A}\overline{B}C + A\overline{B}\overline{C} + A\overline{B}C + AB\overline{C} + ABC$$
$$= m_1 + m_4 + m_5 + m_6 + m_7$$

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

1.24.2 Product of Maxterms

The product of maxterms of a Boolean function is obtained

- from truth table
- from expression

Example 1.113: Express the Boolean function $F = XY + \overline{X}Z$ in a product of maxterm form.

Method 1:

TABLE 1.20 : Truth table for $F = XY + \overline{X}Z$

X	Y	Z	XY	$\overline{X}Z$	$F = XY + \overline{X}Z$
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	1	0	1
1	1	1	1	0	1

Form a maxterm for each combination of the variables that produces a 0 in the function and then form the AND of all those maxterms. Therefore the product of Maxterms,

$$F = M_0 \cdot M_2 \cdot M_4 \cdot M_5$$

$$F = (X, Y, Z) = \pi M(0, 2, 4, 5)$$

or

$$= \pi(0, 2, 4, 5)$$

Method 2: Expanding the expression into a product of OR terms using the distributive law, $X + YZ = (X + Y)(X + Z)$. Each term is then inspected to see if it contains all the variables. If it misses one or more variables, it is *O*Red with an expression such as $(x\overline{x})$, where x is one of the missing variables.

$$\begin{aligned} F &= XY + \overline{X}Z \\ &= (XY + \overline{X})(XY + Z) \\ &= (X + \overline{X})(Y + \overline{X})(X + Z)(Y + Z) \\ &= (\overline{X} + Y)(X + Z)(Y + Z) \end{aligned}$$

The function has 3 variables: X, Y and Z. Each OR term is missing one variable; therefore,

I term, $(\overline{X} + Y) = \overline{X} + Y + Z\overline{Z} = (\overline{X} + Y + Z)(\overline{X} + Y + \overline{Z})$

II term, $(X + Y) = X + Y + Y\overline{Y} = (X + Y + Z)(X + \overline{Y} + Z)$

III term, $(Y + Z) = Y + Z + X\overline{X} = (X + Y + Z)(\overline{X} + Y + Z)$

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

$$F=(X+Y+Z)(X+\overline{Y}+Z)(\overline{X}+Y+Z)(\overline{X}+Y+\overline{Z})$$
$$=M_0\cdot M_2\cdot M_4\cdot M_5$$
$$=\pi M(0,2,4,5)$$
$$F(X,Y,Z)=\pi(0,2,4,5)$$

Example 1.114: Express the Boolean function

$F=(A+\overline{B}+C)(\overline{B}+C+\overline{D})(A+\overline{B}+\overline{C}+D)$ in canonical POS form.

Solution: The first term is missing variable '*D*'

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

The second term is missing variable '*A*'

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

The third term is already in standard form.

The canonical form, $F=(A+\overline{B}+C+D)(A+\overline{B}+C+\overline{D})$

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

Example 1.115: Convert the Boolean function into canonical SOP form,

$$F=A\overline{B}C+\overline{A}\overline{B}+AB\overline{C}D$$

Solution: First term is missing variable '*D*'.

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

Second term is missing two variables *C* and *D*.

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

then, $(\overline{A}\overline{B}C+\overline{A}\overline{B}\overline{C})(D+\overline{D})=\overline{A}\overline{B}CD+\overline{A}\overline{B}C\overline{D}+\overline{A}\overline{B}\overline{C}D+\overline{A}\overline{B}\overline{C}\overline{D}$

Third term is already in standard form,

$$\therefore F=A\overline{B}CD+A\overline{B}C\overline{D}+\overline{A}\overline{B}CD+\overline{A}\overline{B}C\overline{D}+\overline{A}\overline{B}\overline{C}D+\overline{A}\overline{B}\overline{C}\overline{D}+AB\overline{C}D$$

Example 1.116: Obtain the canonical sum of product (SOP) from the function,

$$F=A+B$$

Soluton: $F=A+B$

$$=A(B+\overline{B})+B(A+\overline{A})$$

$$= AB + A\overline{B} + AB + \overline{A}B$$

$$= AB + A\overline{B} + \overline{A}B$$

Example 1.117: Obtain the canonical SOP of the function $F = AB + ACD$.

Solution: First term is missing two variables C and D

$$AB = AB\left(C + \overline{C}\right)\left(D + \overline{D}\right)$$

$$= \left(ABC + AB\overline{C}\right)\left(D + \overline{D}\right)$$

$$= ABCD + ABC\overline{D} + AB\overline{C}D + AB\overline{C}\,\overline{D}$$

Second term is missing one variable 'B'

$$ACD = ACD\left(B + \overline{B}\right)$$

$$= ABCD + A\overline{B}CD$$

Canonical form,

$$F = ABCD + ABC\overline{D} + AB\overline{C}D + AB\overline{C}\,\overline{D} + A\overline{B}CD$$

Example 1.118: Obtain the canonical SOP of the function,

$$F = A + BC$$

Solution:

$$F = A\left(B + \overline{B}\right)\left(C + \overline{C}\right) + BC\left(A + \overline{A}\right)$$

$$= \left(AB + A\overline{B}\right)\left(C + \overline{C}\right) + ABC + \overline{A}BC$$

$$= ABC + AB\overline{C} + A\overline{B}C + A\overline{B}\,\overline{C} + ABC + \overline{A}BC$$

$$= ABC + AB\overline{C} + A\overline{B}C + A\overline{B}\,\overline{C} + \overline{A}BC$$

This result can be checked with the truth table.

TABLE 1.21 : Truth Table for F = A + BC

A	*B*	*C*	*BC*	*F = A + BC*	
0	0	0	0	0	
0	0	1	0	0	
0	1	0	0	0	
0	1	1	1	1	$\overline{A}BC$
1	0	0	0	1	$A\overline{B}\,\overline{C}$
1	0	1	0	1	$A\overline{B}C$
1	1	0	0	1	$AB\overline{C}$
1	1	1	1	1	ABC

$$F = \overline{A}BC + A\overline{B}\overline{C} + A\overline{B}C + AB\overline{C} + ABC$$
$$= \sum m(3, 4, 5, 6, 7)$$

Example 1.119: Obtain the canonical POS form $F = (A + \overline{B})(B + C)$

Solution: The first term has a missing variable 'C'

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

The second term has a missing variable 'A'

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

$\therefore$ Canonical form is $F = (A + \overline{B} + C)(A + \overline{B} + \overline{C})(A + B + C)(\overline{A} + B + C)$

Example 1.120: Express the function $F = A + \overline{B}C$ in

(a) Canonical SOP and (b) Canonical POS form

Solution: Canonical SOP form:

$$F = A + \overline{B}C$$
$$= A(B + \overline{B})(C + \overline{C}) + \overline{B}C(A + \overline{A})$$
$$= (AB + A\overline{B})(C + \overline{C}) + A\overline{B}C + \overline{A}\overline{B}C$$
$$= ABC + AB\overline{C} + A\overline{B}C + A\overline{B}\overline{C} + A\overline{B}C + \overline{A}\overline{B}C$$
$$= ABC + AB\overline{C} + A\overline{B}C + A\overline{B}\overline{C} + \overline{A}\overline{B}C$$
$$= m_7 + m_6 + m_5 + m_4 + m_1$$
$$F = \sum(1, 4, 5, 6, 7)$$

Canonical POS form:

$$F = A + \overline{B}C$$
$$= (A + \overline{B})(A + C)$$
$$= (A + \overline{B} + C\overline{C})(A + C + B\overline{B})$$
$$= (A + \overline{B} + C)(A + \overline{B} + \overline{C})(A + B + C)(A + \overline{B} + C)$$
$$= (A + \overline{B} + C)(A + \overline{B} + \overline{C})(A + B + C)$$
$$= M_2 \cdot M_3 \cdot M_0$$
$$F = \pi(0, 2, 3)$$

1.25 CONVERSION BETWEEN CANONICAL FORMS

The binary values of the product terms in given canonical SOP expression are not present in the equivalent canonical POS expression. Therefore to convert from canonical SOP to canonical POS, the following steps are taken:

Step 1: Evaluate each product term in the SOP expression. i.e., determine the binary numbers that represent the product terms.

Step 2: Determine all of the binary numbers not included in the evaluation of step 1.

Step 3: Write the equivalent sum term for each binary number from step 2 and express in POS form.

Using a similar procedure, we can convert POS to SOP form.

Example 1.121: Convert the following SOP expression to an equivalent POS expression:

$$\overline{A}\,\overline{B}\,\overline{C}+\overline{A}B\overline{C}+\overline{A}BC+A\overline{B}C+ABC$$

Solution:

Step 1: The evaluation is: 000 + 010 + 011 + 101 + 111

Step 2: Since there are 3 variables, $2^3 = 8$ possible combinations are possible. The SOP expression contains 5 of these combinations, so the POS must contain other 3 combinations, which are 001, 100 and 110.

Step 3: POS expression is, $\left(A+B+\overline{C}\right)\left(\overline{A}+B+C\right)\left(\overline{A}+\overline{B}+C\right)$

Method 2: The conversion between canonical forms can be done by another method:

The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function. This is because the original function is expressed by those minterms that make the function equal to 1, whereas its complement as a 1 for those minterms that the function is a 0. As an example consider the function,

$$F(A, B, C)=\overline{A}\,\overline{B}\,\overline{C}+\overline{A}B\overline{C}+\overline{A}BC+A\overline{B}C+ABC$$

$$=\sum(0, 2, 3, 5, 7)$$

This has a complement that can be expressed as

$$\overline{F}=\sum(1, 4, 6)=m_1+m_4+m_6$$

The complement of F, $\overline{F}$ is obtained by De Morgan's theorem,

$$\begin{aligned}\overline{F}&=\left(\overline{m_1+m_4+m_6}\right)\\&=\overline{m}_1\cdot\overline{m}_4\cdot\overline{m}_6\\&=M_1\cdot M_4\cdot M_6\\&=\pi M(1, 4, 6)\\&=\pi(1, 4, 6)\end{aligned}$$

$$\overline{m}_j=M_j$$

Example 1.122: Convert the canonical SOP into canonical POS $F(A, B, C) = \sum(1, 3, 6, 7)$.

Solution:

$$F = \sum(1, 3, 6, 7)$$

$$\overline{F} = \sum(0, 2, 4, 5)$$

$$= m_0 + m_2 + m_4 + m_5$$

By DeMorgan's theorem,

$$F = \overline{m_0 + m_2 + m_4 + m_5}$$

$$= \overline{m}_0 \cdot \overline{m}_2 \cdot \overline{m}_4 \cdot \overline{m}_5$$

$$= M_0 \cdot M_2 \cdot M_4 \cdot M_5$$

$$= \pi(0, 2, 4, 5)$$

Using this complementary relationship, find logical function in terms of maxterms. For example, for a 4 variable if

$$F = \sum(0, 2, 4, 6, 8, 10, 12, 14)$$

then the complement is $F = \pi M(1, 3, 5, 7, 9, 11, 13, 15)$.

1.26 KARNAUGH MAPS

A Karnaugh map is a graphical representation of a truth table that can be used to reduce a logic circuit to its simpliest terms. The size of the Karnaugh map depends on the amount of inputs that are listed in the truth table. An example of a three variable Karnaugh Map is shown below:

	$\overline{A}\overline{B}$	$\overline{A}B$	AB	$A\overline{B}$
C	$\overline{A}\overline{B}C$	$\overline{A}BC$	ABC	$A\overline{B}C$
$\overline{C}$	$\overline{A}\overline{B}\overline{C}$	$\overline{A}B\overline{C}$	$AB\overline{C}$	$A\overline{B}\overline{C}$

The terms withing a Karnaugh Map are obtained by combining the row and column boolean expression that are shown at the top and left margins of the Karnaugh Map. Each combined term within the Karnaugh Map corresponds to a single line of inputs in a truth table. Terms that have a line over them corresponds to a low or zero input. Terms without any marking corresponds to a high or 1 input.

1.26.1 Constructing a Karnaugh Map

(1) Two variable maps:

A	*B*	*Y*
0	0	0
0	1	0
1	0	1
1	1	1

	$\overline{B}$	B
$\overline{A}$	0	0
A	1	1

	$\overline{B}$	B
$\overline{A}$	$\overline{A}\overline{B}$	$\overline{A}B$
A	$A\overline{B}$	AB

(2) Three variable maps:

A	*B*	*C*	*Y*
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

	$\overline{C}$	C
$\overline{A}\overline{B}$	0	0
$\overline{A}B$	1	0
AB	1	1
$A\overline{B}$	0	0

(3) Four variable maps:

A	*B*	*C*	*D*	*Y*
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	1	1	0	1

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

Pairs, Quads and Octets

Pairs

- Pair eliminates are variable and its complements.
- Pair of 1's horizontally and vertially adjacent.

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

$$Y = ABCD + ABC\overline{D}$$
$$= ABC\left(D + \overline{D}\right)$$
$$= ABC$$

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

$$Y = ABC\overline{D} + A\overline{B}C\overline{D}$$
$$= AC\overline{D}\left(B+\overline{B}\right)$$
$$= AC\overline{D}$$

***Quads*:** A quad is a group of four 1's that are horizontally or vertially adjacent

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

$$AB\overline{C}\,\overline{D} + AB\overline{C}D + ABCD + ABC\overline{D}$$
$$= ABC\left(D+\overline{D}\right) + AB\overline{C}\left(D+\overline{D}\right)$$
$$= ABC + AB\overline{C}$$
$$= AB\left(C+\overline{C}\right)$$
$$= AB$$

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

$$ABCD + ABC\overline{D} + A\overline{B}CD + A\overline{B}C\overline{D}$$
$$= ABC\left(D+\overline{D}\right) + A\overline{B}C\left(D+\overline{D}\right)$$
$$= AC\left(B+\overline{B}\right)$$
$$= AC$$

***Octet*:** An octet is a group of eight 1's.

An octet eliminates three variables and their complements.

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

$$Y = AB\overline{C}\,\overline{D} + AB\overline{C}D + ABCD + ABC\overline{D} +$$
$$A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}D + A\overline{B}CD + A\overline{B}C\overline{D}$$
$$= ABC\left(D+\overline{D}\right) + AB\overline{C}\left(D+\overline{D}\right) +$$
$$A\overline{B}\,\overline{C}\left(D+\overline{D}\right) + A\overline{B}C\left(D+\overline{D}\right)$$
$$= AB\left(C+\overline{C}\right) + A\overline{B}\left(C+\overline{C}\right)$$
$$= AB + A\overline{B} = A\left(B+\overline{B}\right) = A$$

1.26.2 Karnaugh Map Simplifications

Encircle octets first, the quads second and the pairs last.

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

$$Y = A\overline{C} + C\overline{D} + \overline{A}\overline{B}D$$

***Overlapping Groups*:** It is possible to use the same 1 more than once.

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

$$Y = A + B\overline{C}D$$

It is valid to encircle the 1's as shown below. But the isolated 1 results in a more complicated equation.

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

$$Y = A + \overline{A}B\overline{C}D$$

Rolling the map

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

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

Visualize the picking up the karnaugh map and rolling it so that the left side touches the right side. By doing so, the two pairs can be realised as Quad.

$\therefore$ The quad has the equation,

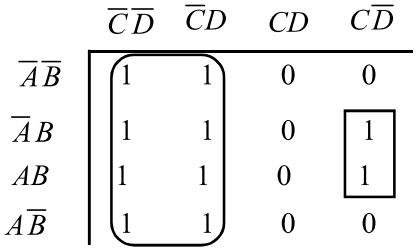
$$Y = B\overline{D}$$

Proof: To show whether the rolling is valid or not.

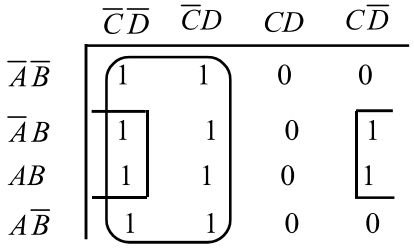
$$Y = B\overline{C}\,\overline{D} + BC\overline{D}$$

$$= B\overline{D}\left(C + \overline{C}\right) = B\overline{D}$$

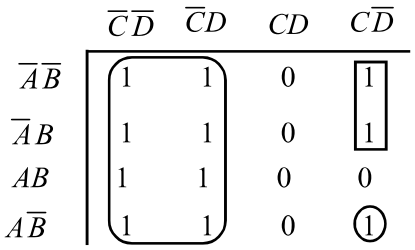
***Rolling and Overlapping*:** It is possible to overlap and roll the map to get large groups.



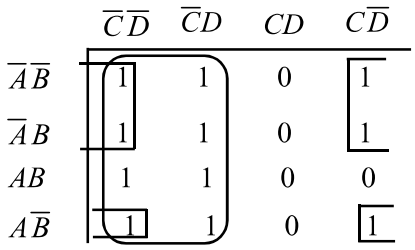
$$Y = \overline{C} + BC\overline{D}$$



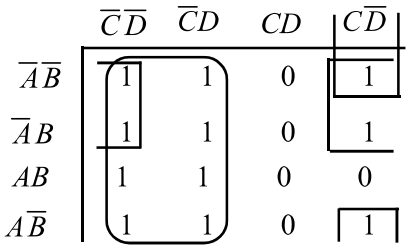
$$Y = C + B\overline{D}$$



$$Y = \overline{C} + A\overline{B}C\overline{D} + \overline{A}C\overline{D}$$



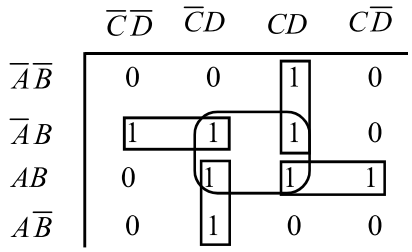
$$Y = \overline{C} + A\overline{D} + A\overline{B}\,\overline{D}$$



$$Y = \overline{C} + \overline{A}\overline{D} + \overline{B}C\overline{D}$$

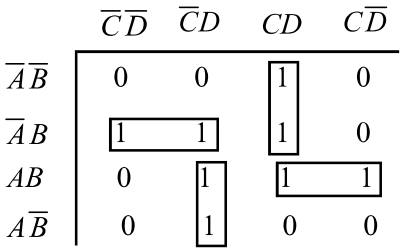
Eliminating Redundant Groups

Redundant group is a group whose 1's are already used by other groups. The redundant group is eliminated as shown in **Figure 1.26 (b).**



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

Fig. 1.26 (a)

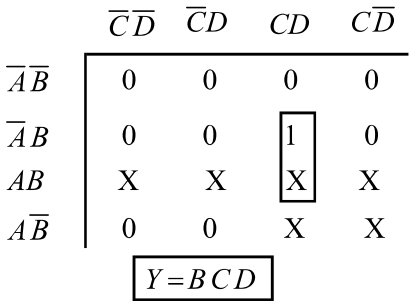


	$\overline{C}\,\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\,\overline{B}$	0	0	1	0
$\overline{A}B$	1	1	1	0
AB	0	1	1	1
$A\overline{B}$	0	1	0	0

Fig. 1.26 (b)

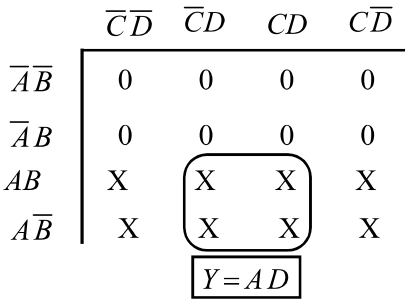
1.26.3 Don't Care Condition

In some digital system, certain output conditions never occur during normal operation. Therefore corresponding output never appears. Since the output never appears it is indicated by an '*X*' in the truth table.



	$\overline{C}\,\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\,\overline{B}$	0	0	0	0
$\overline{A}B$	0	0	1	0
AB	X	X	X	X
$A\overline{B}$	0	0	X	X

$Y = BCD$



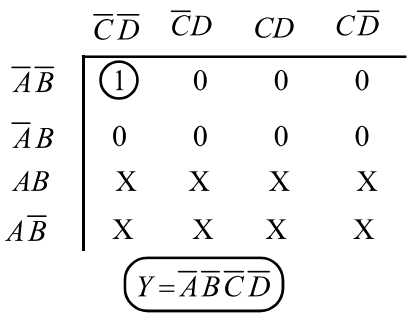
	$\overline{C}\,\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\,\overline{B}$	0	0	0	0
$\overline{A}B$	0	0	0	0
AB	X	X	X	X
$A\overline{B}$	X	X	X	X

$Y = AD$

Solution:

1. Given the truth table, draw a Karnaugh map with other 0's, 1's and don't cares.
2. Enclose the actual 1's on the Karnaugh map in the largest groups you can find by treating the don't cares as 1's.
3. After the actual 1's have been included in groups, disregard the remaining don't cares by visualizing them as 0's.

Example:



	$\overline{C}\,\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\,\overline{B}$	1	0	0	0
$\overline{A}B$	0	0	0	0
AB	X	X	X	X
$A\overline{B}$	X	X	X	X

$Y = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$

Here don't cares are of no help. The best way is, encircle the isolated 1, while treating don't cares as 0's.

1.26.4 Reducing Karnaugh Maps

The rules for reducing Karnaugh Maps are as follows:

- ❖ All of the 1's in the Karnaugh Map are called minterms.
- ❖ The 1's can be reduced in groups of 2, 4 and 8.
- ❖ Minterms that are next to each other horizontally or vertically, can be grouped together.
- ❖ Minterms that have been grouped in a Karnaugh Map, can be reduced to the boolean terms that are common with all the terms in the group.
- ❖ Minterms that cannot be grouped together, cannot be reduced.
- ❖ Use a minterm for grouping more than once.
- ❖ All 1's must be accounted for.

Example 1.123: Determine the Karnaugh Map and reduced boolean equation for the truth table shown in **Table 1.22.**

TABLE 1.22 : Truth Table

C	*B*	*A*	OUTPUT
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Without reduction the Boolean Equation for the above truth table is:

$$Y = \overline{A}\,\overline{B}\,\overline{C} + A\,\overline{B}\,\overline{C} + \overline{A}\,B\,\overline{C} + A\,B\,\overline{C} + \overline{A}\,\overline{B}\,C$$

Each minterm corresponds to an instance in the truth table when the output is high. The Karnaugh Map for the above truth table, with the allowed groupings are shown below:

	$\overline{A}\,\overline{B}$	$\overline{A}\,B$	AB	$A\,\overline{B}$
C	1	0	0	0
$\overline{C}$	1	1	1	1

The map shows two groupings that cover each minterm. Each of these groupings will reduce to one term.

The two terms that are grouped together $\overline{A}\,\overline{B}\,C$ reduces to $\overline{A}\,\overline{B}\,\overline{C}$. This is because $\overline{A}$ and $\overline{B}$ are common to both terms.

The four terms $\overline{A}\,\overline{B}\,\overline{C}$ and $\overline{A}\,B\,\overline{C}$ and $A\,B\,\overline{C}$ and $A\,\overline{B}\,\overline{C}$ that are grouped together reduces to $\overline{C}$. This is because $\overline{C}$ is the only input common to all four terms.

Therefore the boolean equation

$$Y=\overline{A}\,\overline{B}C+\overline{A}\,\overline{B}\,\overline{C}+\overline{A}\,B\,\overline{C}+A\,B\,\overline{C}+A\,\overline{B}\,\overline{C} \quad \text{reduces to}$$

$$Y=\overline{A}\,\overline{B}+\overline{C}$$

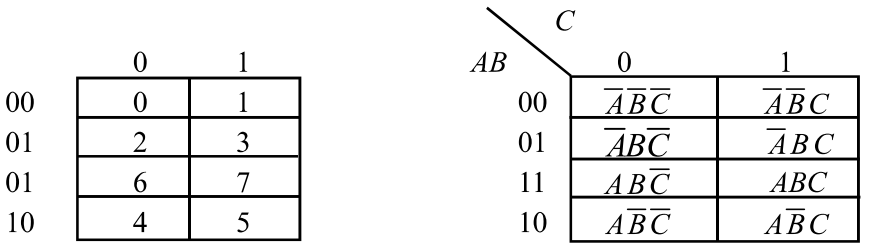
1.26.5 Simplification of Sum of Product Expression

The procedure to simplify the SOP expression using K-map as follows:

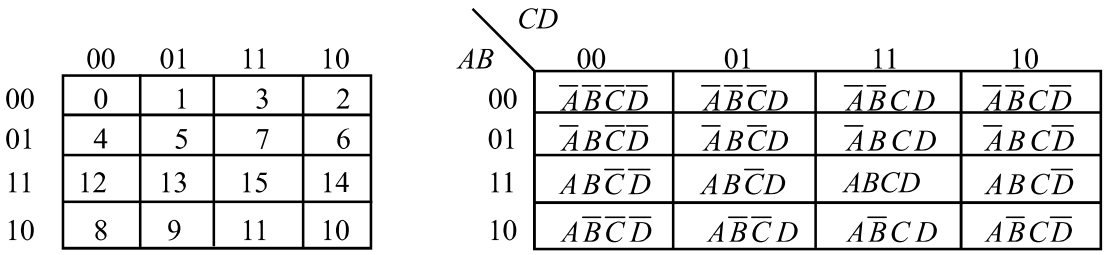
- ❖ Plot the K-map and enter the 1's in those cells corresponding to the combinations for which function value is 1.
- ❖ Check the K-map for adjacent 1's and encircle those 1's which are not adjacent to any other 1's.
- ❖ Check for those 1's which are adjacent to only one other 1 and encircle such pairs.
- ❖ A group must contain either 1,2,4,8 or 16 ones (1's), which are all powers of two.
- ❖ Combine any pairs necessary to include any 1's that have not yet been grouped.
- ❖ Form the simplified expression by summing product terms of all the groups.

The minterms for variable and standard product terms and represented by 2 varibale K map, 3 variable K-map and 4 variable K-map are shown in **Figurre 1.27.**

(a) Two Variable Map



(b) Three Variable Map



(c) 4 Variable Map

Fig.1.27: Representation of functions in the Map

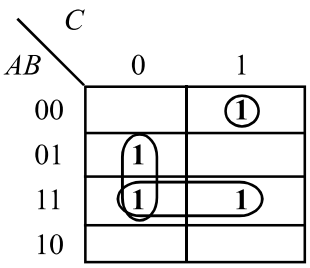
Example 1.124: Simplify the following SOP expression on a Karnaugh Map.

$$\overline{A}\,\overline{B}C+\overline{A}B\overline{C}+AB\overline{C}+ABC$$

Solution: The expression is evaluated as follows:

$$\overline{A}\,\overline{B}C+\overline{A}B\overline{C}+AB\overline{C}+ABC$$

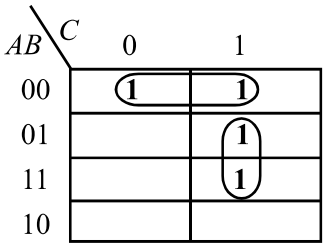
001 010 110 111



$$F=B\overline{C}+AB+\overline{A}\,\overline{B}C$$

Example 1.125: Simplify the SOP by using K-map

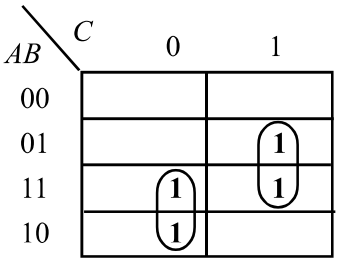
$$Y=\sum \text{m}\,(0,1,3,7)$$



$$Y=\overline{A}\,\overline{B}+BC$$

Example 1.126: Simplify the expression $Y=m_3+m_4+m_6+m_7$

Solution:



$$Y=BC+A\overline{C}$$

Example 1.127: Simplify the Boolean expression using K map $F=\overline{A}C+\overline{A}B+A\overline{B}C+BC$.

Solution: The given expression is not a standard SOP form. First convert this non-standard SOP to standard SOP and then simplify the expression using K-map.

$$\overline{A}C=\overline{A}C\left(B+\overline{B}\right)=\overline{A}BC+\overline{A}\overline{B}C$$

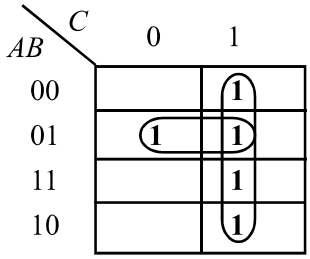
$$\overline{A}B=\overline{A}B\left(C+\overline{C}\right)=\overline{A}BC+\overline{A}B\overline{C}$$

$$BC=BC\left(A+\overline{A}\right)=ABC+\overline{A}BC$$

$$\therefore F=\overline{A}C+\overline{A}B+A\overline{B}C+BC$$

$$=\left(\overline{A}BC+\overline{A}\overline{B}C\right)+\left(\overline{A}BC+\overline{A}B\overline{C}\right)+A\overline{B}C+\left(ABC+\overline{A}BC\right)$$

$$=\overline{A}BC+\overline{A}\overline{B}C+\overline{A}B\overline{C}+A\overline{B}C+ABC$$



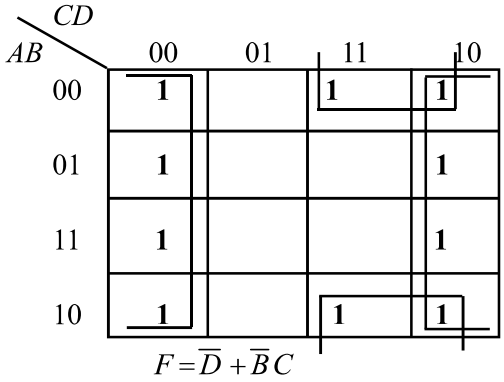
$F=C+\overline{A}B$

Example 1.128: Use a *K* map to minimize the following SOP expression:

$$F=A\overline{B}\overline{C}\overline{D}+\overline{A}\overline{B}\overline{C}\overline{D}+\overline{A}B\overline{C}\overline{D}+AB\overline{C}\overline{D}+$$

$$\overline{A}\overline{B}CD+A\overline{B}CD+ABC\overline{D}+A\overline{B}C\overline{D}+\overline{A}\overline{B}C\overline{D}+\overline{A}BC\overline{D}$$

Solution:

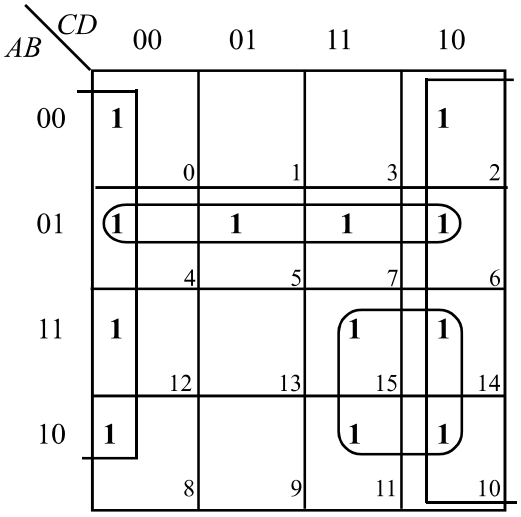


$F=\overline{D}+\overline{B}C$

Example: 1.129: Simplify the expression

$$F = \sum \text{m}(0,2,4,5,6,7,8,10,11,12,14,15)$$

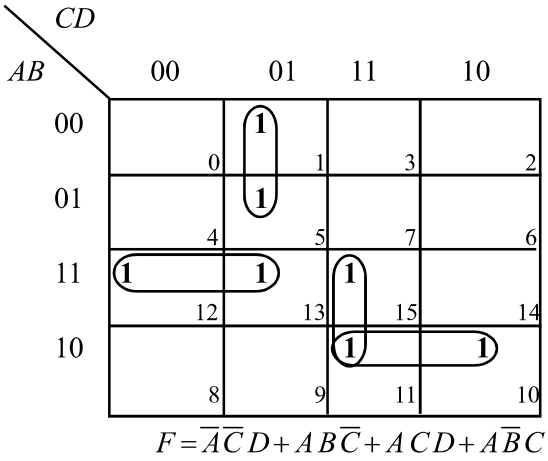
Solution:



$$F = \overline{D} + A\overline{B} + AC$$

Example 1.130: Simplify the expression using K-map

$$F = m_1 + m_5 + m_{10} + m_{11} + m_{12} + m_{13} + m_{15}$$



$$F = \overline{A}\,\overline{C}\,D + AB\overline{C} + ACD + A\overline{B}C$$

Example 1.131: Simplify the following SOP expression on a K-map

$$\overline{B}\,\overline{C} + A\overline{B} + AB\overline{C} + A\overline{B}C\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + A\overline{B}CD$$

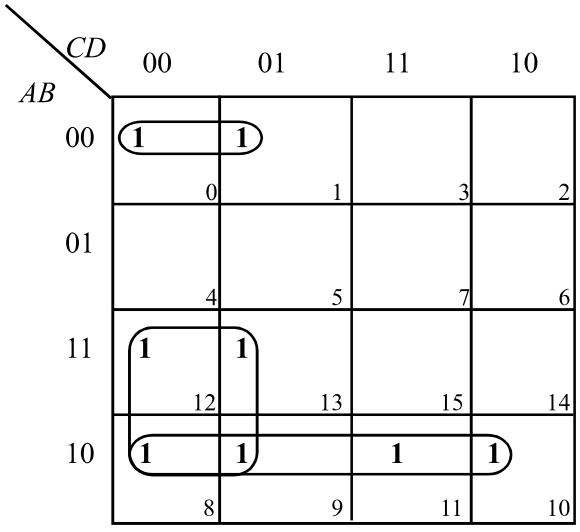
Solution: The SOP expression is obviously not in standard form because each product term does not have 4 variables. The first and second term are both missing 2 variables, the third term is missing one variable and the rest of the terms are standard. First expand the terms by including all combinations of the missing variables numerically as follows:

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

Repeated terms are cancelled under the rule $A + A = A$, therefore,

$$F = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D + A\overline{B}\overline{C}\overline{D} + A\overline{B}\overline{C}D + A\overline{B}CD + A\overline{B}C\overline{D} + AB\overline{C}\overline{D} + AB\overline{C}D$$

$$= \sum(0, 1, 8, 9\ 10, 11, 12, 13)$$

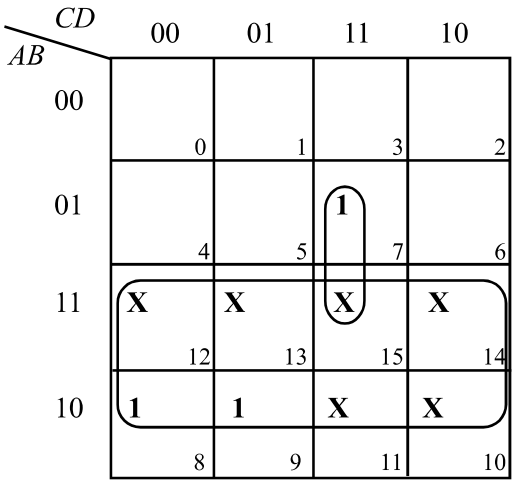


$$F = A\overline{B} + \overline{A}C + \overline{A}\overline{B}\overline{C}$$

Example 1.132: Simplify using K-map

$$F(A, B, C, D) = \sum m\,(7,8,9) + d(10,11,12,13,14,15)$$

Solution:



$F = A + BCD$

Note: Without don't cares, $F = A\overline{B}C + \overline{A}BCD$

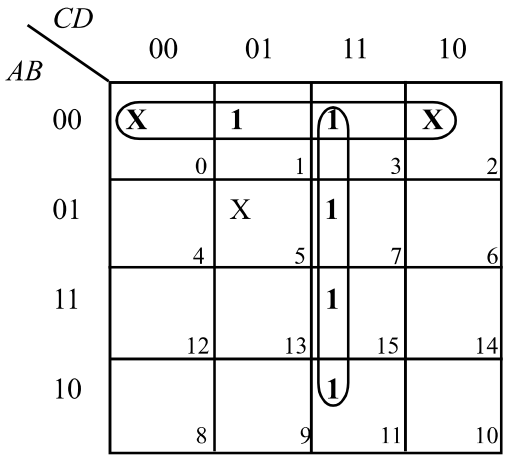
With don't cares, $F = A + BCD$

Therefore, it is clear that, the advantage of using don't care terms is to get the simplest expression.

Example 1.133: Simplify using K-map

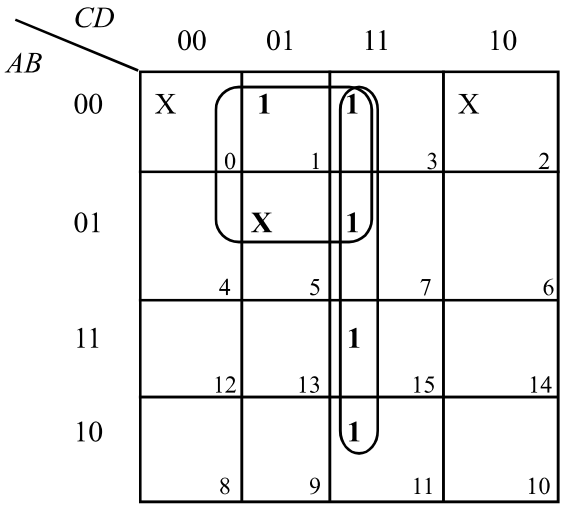
$$F(A, B, C, D) = \sum m\,(1,3,7,11,15) + d(0,2,5)$$

Solution:



$F = \overline{A}\overline{B} + CD$

(or)



$F = \overline{A}D + CD$

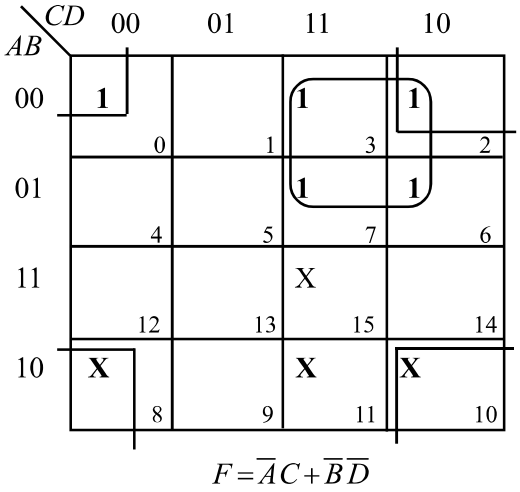
$F = \overline{A}\overline{B} + CD$ (or)

$A\overline{D} + CD$

Example 1.134: Using the K-Map method, simplify the following Boolean function

$$F = \sum m\,(0,2,3,6,7) + \sum d\,(8, 10, 11, 15)$$ **(April 2005)**

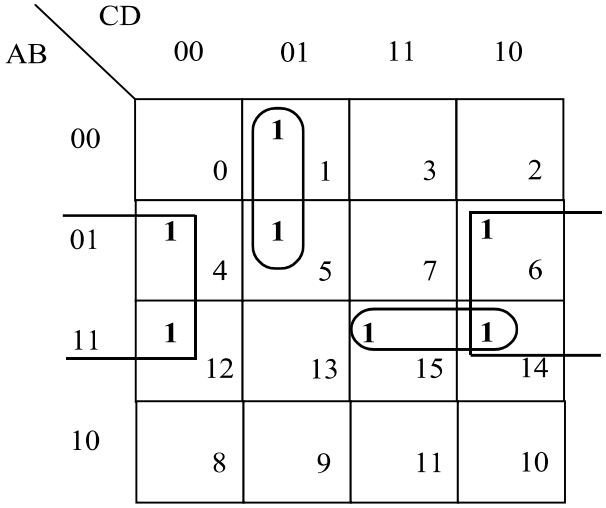
Solution:



$F = \overline{A}C + \overline{B}\overline{D}$

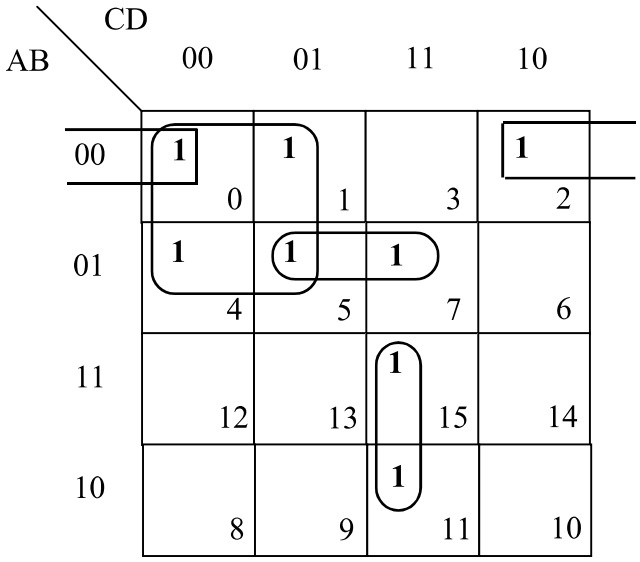
Example: Simplify the following Boolean function F using Karnaugh map method.

Example 1.135: $F(A,B,C,D) = \sum(1,4,5,6,12,14,15)$ **(Dec 2011)**



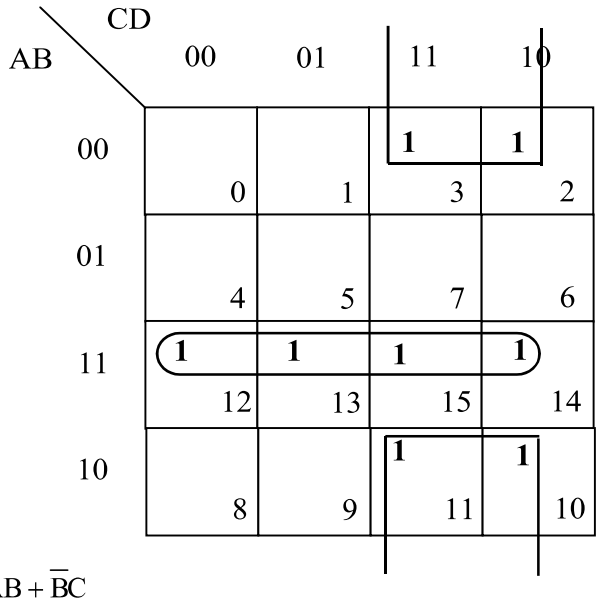
$F(A,B,C,D) = B\overline{D} + \overline{AC}D + ABC$

Example 1.136: $F(A,B,C,D) = \sum(0,1,2,4,5,7,11,15)$ **(Dec 2011)**



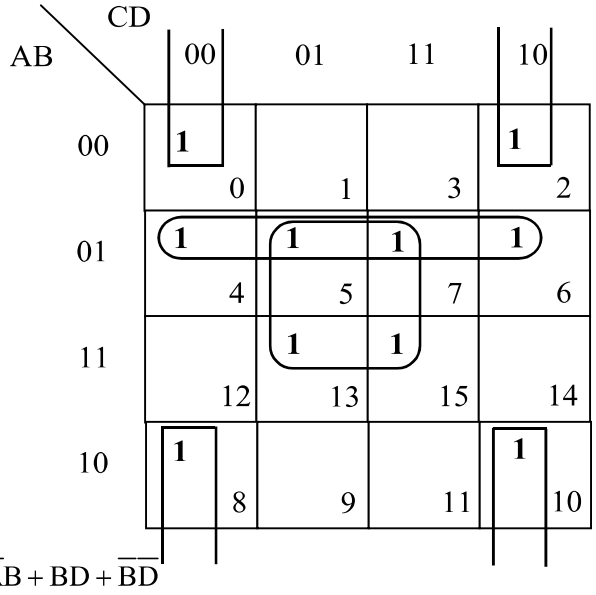
$F(A,B,C,D) = \overline{A}\,\overline{C} + \overline{A}B\overline{D} + ACD + \overline{A}\,\overline{B}\,\overline{D}$

Example 1.137: $F(A,B,C,D) = \sum(2,3,10,11,12,13,14,15)$ **(Dec 2011)**



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

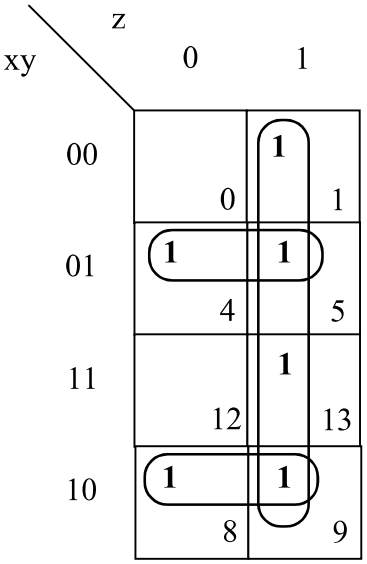
Example 1.138: $F(A,B,C,D) = \sum(0,2,4,5,6,7,8,10,13,15)$ **(Dec 2011)**



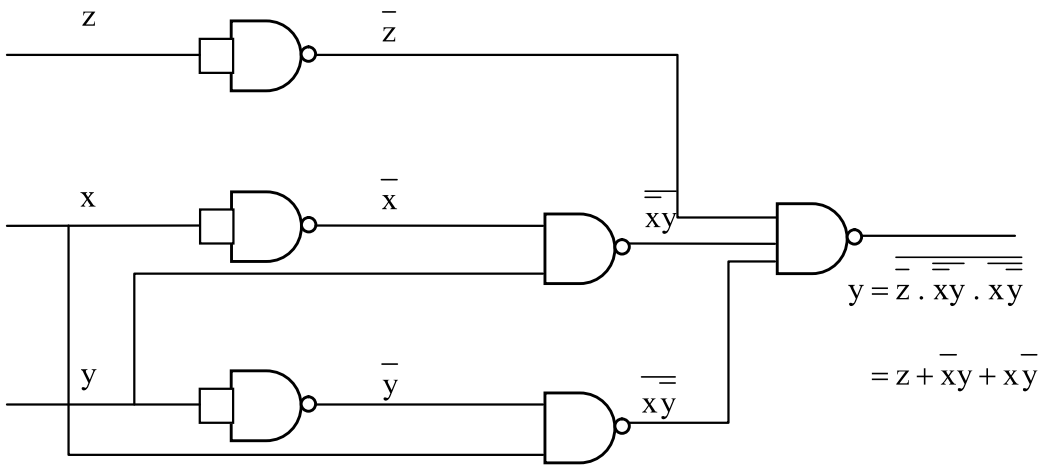
$F(A,B,C,D) = \overline{A}B + BD + \overline{B}\overline{D}$

Example 1.139: Implement the switching function. **(May 2012)**

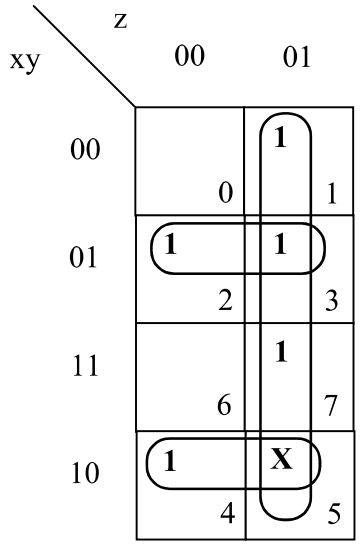
$F(x,y,z) = \sum m(1,2,3,4,5,7)$ with NAND gates.



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

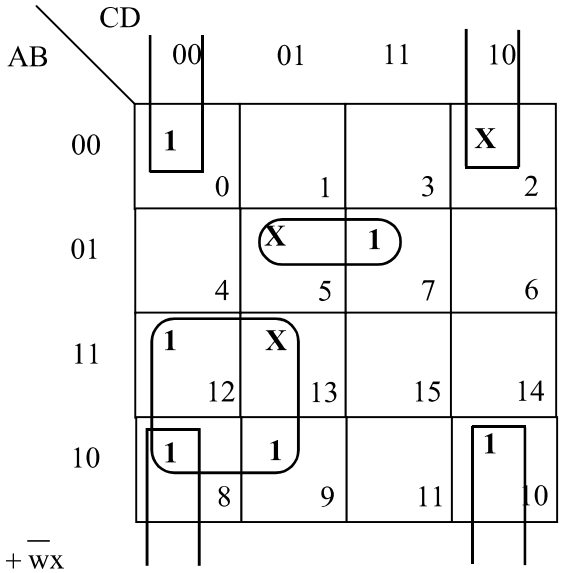


Example 1.140: $f(A,B,C) = \sum m(0,1,3,7) + \sum d(2,5)$ **(May 2013)**



$f = C + \overline{A}\,\overline{B}$

Example 1.141: F(w, x, y, z) = ∑m(0, 7, 8, 9, 10, 12) + ∑d(2, 5, 13) **(May 2013)**



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

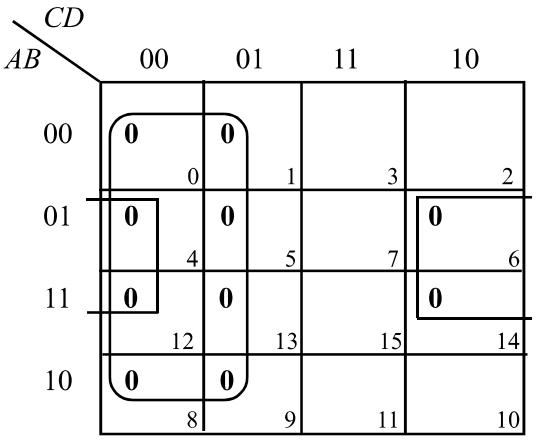
1.26.6 Simplification of Product of Sum expression

To simplify a POS expression, for each maxterm in the expression in, a '0' has to be entered in the corresponding cells and groups must be formed with '0' cells, instead of 1 cells to get he minterm (SOP) expression. The simplified term corresponding to each group can be obtained by the OR operation of the variables that are same for all cells of that group. Here, a variable corresponding to '0' has to be represented in an uncomplemented form.

Example 1.142: Simplify, the POS expression using K map method.

$$F = \pi\,(0, 1, 4, 5, 6, 8, 9, 12, 13, 14)$$

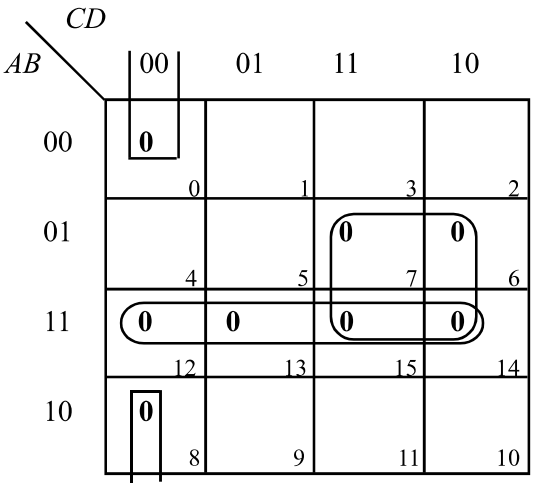
Solution:



$$F = C \cdot \left(\overline{B} + D\right)$$

Example 1.143: Simplify the POS expression

$$F = \pi\,(0,6,7,8,12,13, 14,15)$$

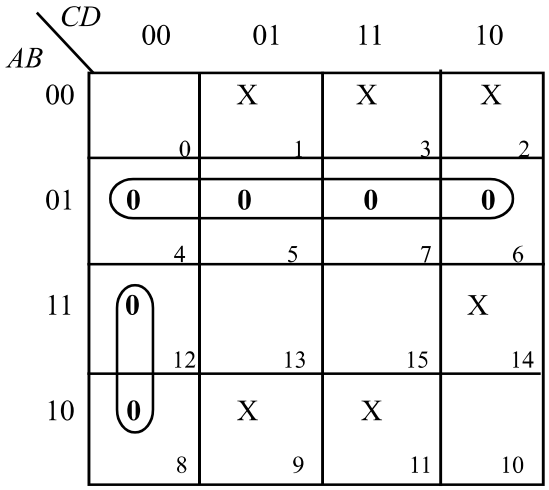


$$F = \left(\overline{A} + \overline{B}\right)\left(\overline{B} + \overline{C}\right)\left(B + C + D\right)$$

Example 1.144: Simplify the expression

$$F(A, B, C, D) = \pi M\ (4,5,6,7,8,12)\ .\ d(1,2,3,9,11,14)$$

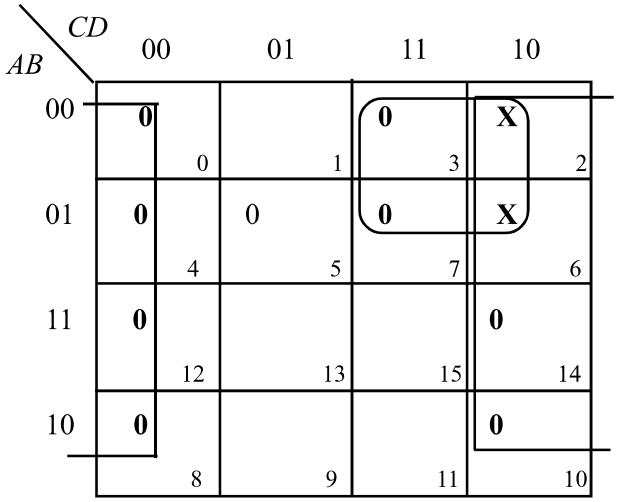
Solution:



$$F = \left(A + \overline{B}\right)\left(\overline{A} + C + D\right)$$

Example 1.145: Simplify the POS expression,

$$F = \pi\ \mathrm{M}\ (0,3,4,7,8,10,12,14).\mathrm{d}(2,6)$$



$$F = D\left(A + \overline{C}\right)$$

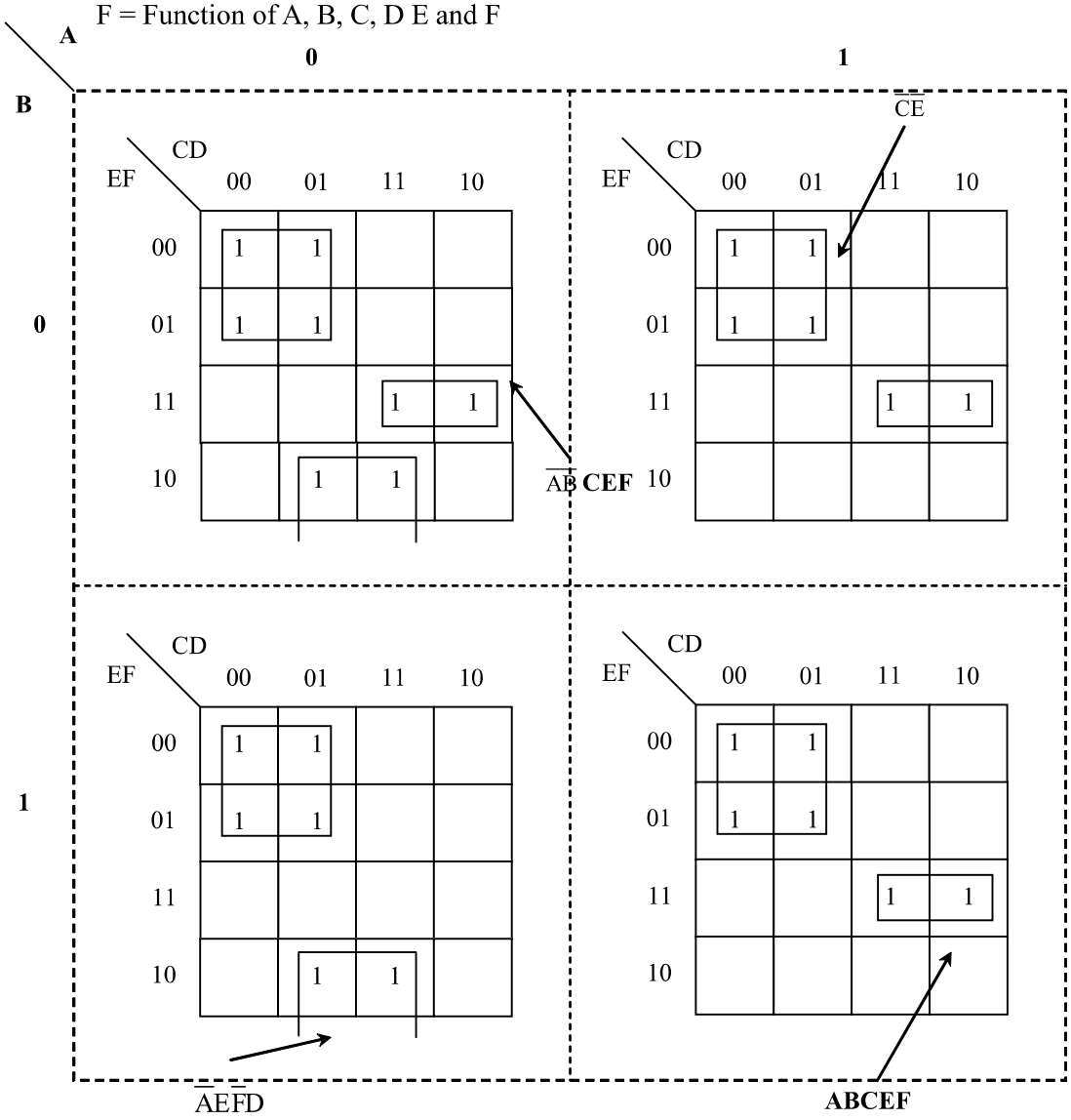
Example 1.146: Determine the minterm sum of product form of the switching function.

$F = \sum (0,1,4,5,6,11,14,15,16,17,20,22,30,32,33,36,37,48,49,52,53,59,63)$ **(Dec. 2010)**

Solution:

Fill up the K-map with the variable given

F = Function of A, B, C, D E and F



Four 1's in each box form a group of 16 bits and their reduced function is $\overline{C}\overline{E}$. Therefore

$F = \sum (0,1,4,5,6,11,14,15,16,17,20,22,30,32,33,36,37,48,49,52,53,59,63)$

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

Example 1.147: Minimize the following expression using Karnaugh map.

$Y = A'BC'D' + A'BC'D + ABC'D' + AB'C'D + A'B'CD'.$ **(May 2011)**

AB \ CD	00	01	11	10
00	0	0	0	1
01	1	1	0	0
11	1	0	0	0
10	0	1	0	0

$Y = \overline{A}B\overline{C} + B\overline{C}\overline{D} + \overline{A}\overline{B}C\overline{D} + A\overline{B}\overline{C}C$

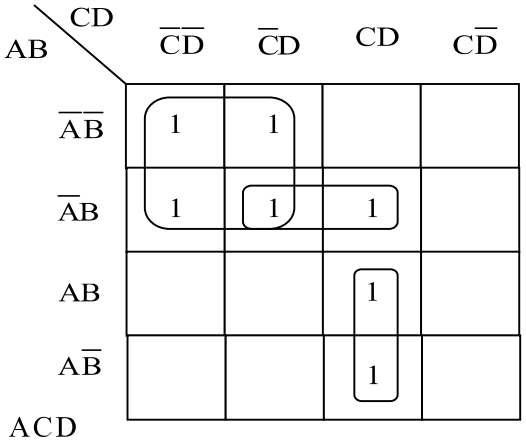
Example 1.148: Simplify the following Boolean function F using Karnaugh map method.

(i) $\mathbf{F(A,B,C,D) = \sum(1,4,5,6,12,14)}$ **(Dec 2010)**

AB \ CD	$\overline{C}\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\overline{B}$		1		
$\overline{A}B$	1	1		1
AB	1			1
$A\overline{B}$				

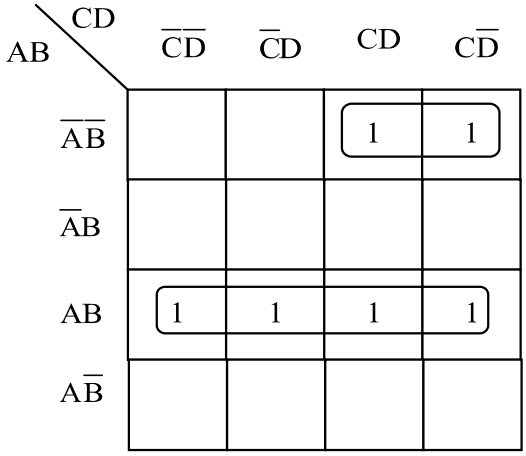
$F = B\overline{C}\,\overline{D} + \overline{A}C\overline{D} + BC\overline{D}$

(ii) $\mathbf{F(A,B,C,D) = \sum(0,1,4,5,7,11,15)}$



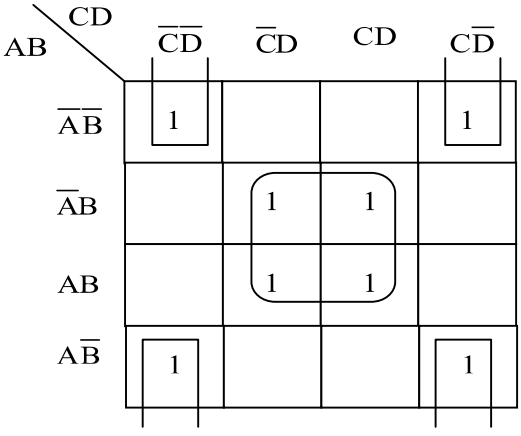
$F = \overline{A}\,\overline{C} + \overline{A}BD + ACD$

(iii) $\mathbf{F(A,B,C,D) = \sum(2,3,12,13,14,15)}$



$F = AB + \overline{A}\overline{B}C$

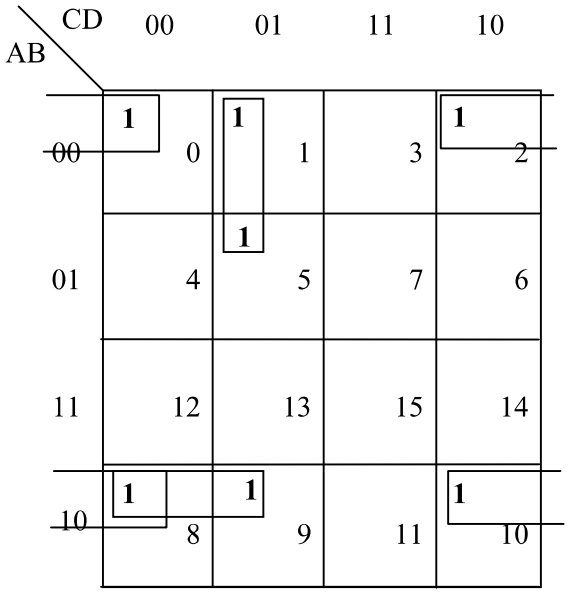
(iv) $\mathbf{F(A,B,C,D) = \sum(0,2,5,7,8,10,13,15)}$



$F = BD + \overline{B}\,\overline{D}$

Example 1.149: Simplify F(A, B, C, D) = ∑ (0, 1, 2, 5, 8, 9, 10) in sum of products and product of sums using K-map. **(Dec 2012)**

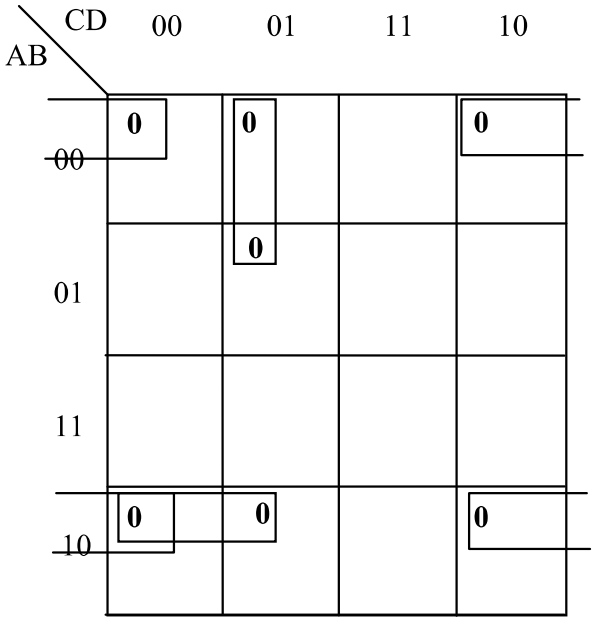
Solution:



SOP:

$$F = \overline{B}\overline{D} + \overline{A}\overline{C}D + A\overline{B}\overline{C}$$

POS:



POS: $F = (B + D)(A + C + \overline{D})(\overline{A} + B + C)$

1.26.7 FIVE VARIABLE MAP

The five-variable map consists of 2 four-variable maps with A, B, C, D and E. Variable A distinguishes between the two maps, as indicated on the top of the diagram as A = 0 and A = 1. Minterms 0–15 belong with A = 0 and minterms 16 –31 with A = 1.

A = 0

BC \ DE	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

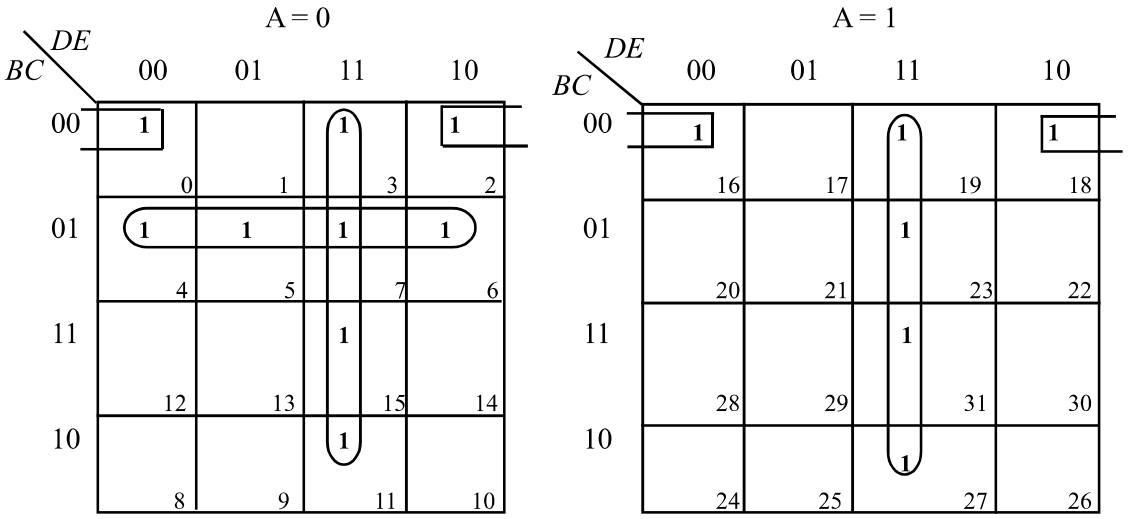
A = 1

BC \ DE	00	01	11	10
00	16	17	19	18
01	20	21	23	22
11	28	29	31	30
10	24	25	27	26

Example 1.150

Simplify the Boolean expression:

$F(A, B, C, D, E) = \Sigma(0, 2, 3, 4, 5, 6, 7, 11, 15, 16, 18, 19, 23, 27, 31)$



$$F = DE + \overline{A}\,\overline{B}\,C + \overline{B}\,\overline{C}\,\overline{E}$$

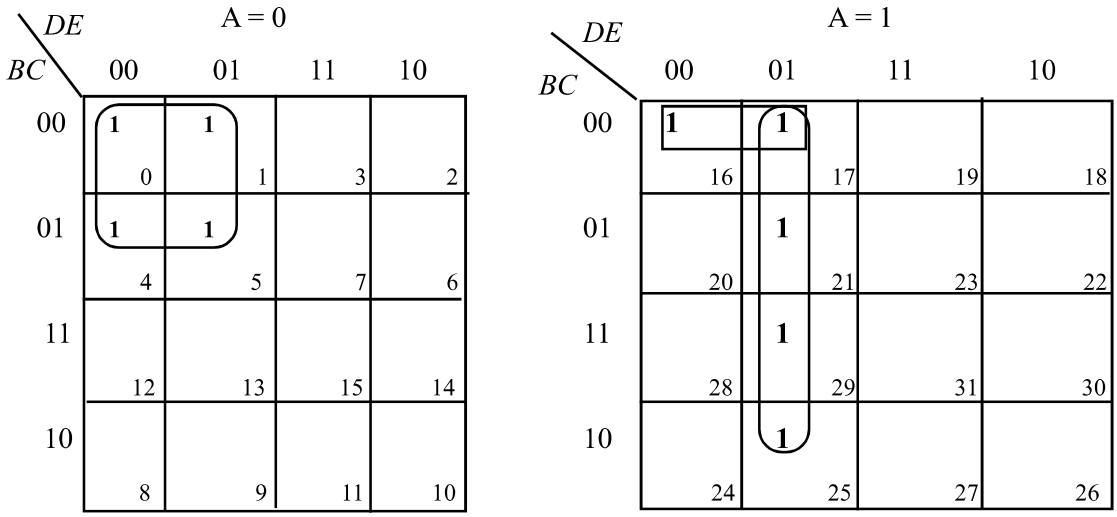
In DE and $\overline{B}\,\overline{C}\,\overline{E}$ terms, A is not included because the adjacent squares belong to both $A = 0$ and $A = 1$.

In $\overline{A}\,\overline{B}\,C$ term, it is necessary to include $\overline{A}$ because all the squares are associated with $A = 0$.

$$F = DE + \overline{A}\,\overline{B}\,C + \overline{B}\,\overline{C}\,\overline{E}$$

Example 1.151: Simplify the Boolean function

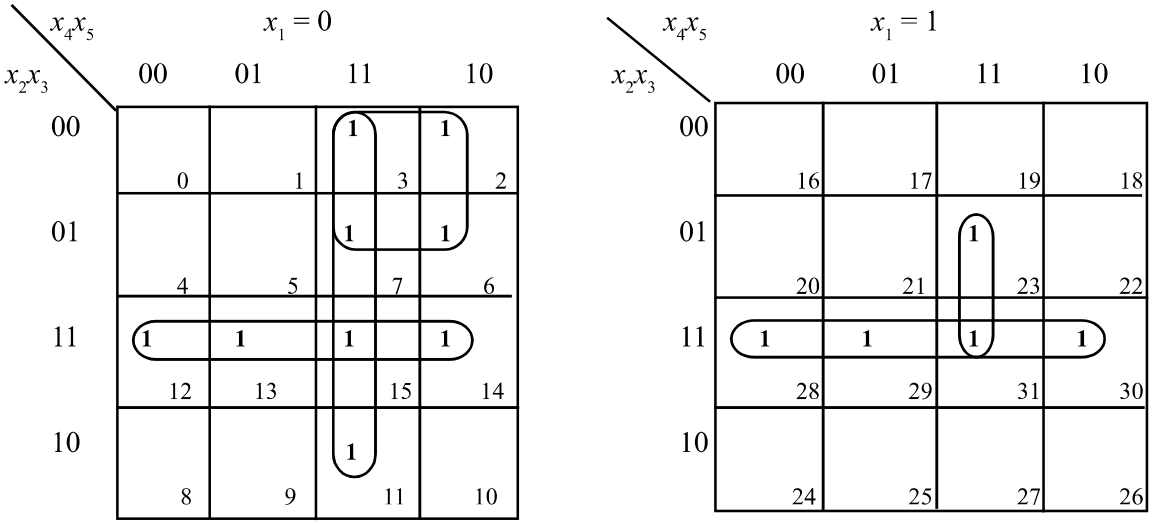
$F(A, B, C, D, E) = \Sigma(0, 1, 4, 5, 16, 17, 21, 25, 29)$



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

Example 1.152: Find the minimal sum of product form for the following switching function:

$f(x_1, x_2, x_3, x_4, x_5) = \Sigma m(2, 3, 6, 7, 11, 12, 13, 14, 15, 23, 28, 29, 30, 31)$ **(May 2006)**

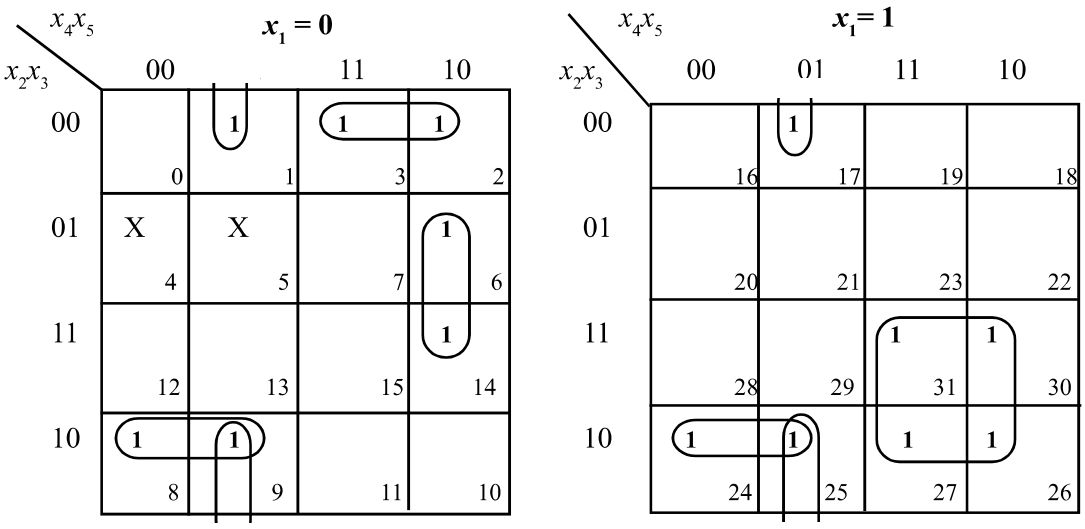


$$F = x_2x_3 + \overline{x}_1x_4x_5 + \overline{x}_1\overline{x}_2x_4 + x_1x_3x_4x_5$$

Example 1.153: Find the minimal sum of product expression for the following switching function:

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

(May 2006)



$$F = x_2\bar{x}_3\bar{x}_4 + \bar{x}_3\bar{x}_4x_5 + x_1x_2x_4 + \bar{x}_1\bar{x}_2\bar{x}_3x_4 + \bar{x}_1x_3\bar{x}_4\bar{x}_5$$

1.27 TABULATION METHOD (QUINE-McCLUSKEY METHOD)

The *K* map method of minimization of logic functions is convenient as long as the number of variables does not exceed 4 or 5. When the number of variable increases, K map become very difficult. To avoid this difficult, Quine-McCluskey or Tabulation method can be used. This method is developed by Quine and McCluskey. The procedure for simplification of Boolean function by Quine- McCluskey method is as follows:

- ❖ Each minterm should be expressed by its binary representation.
- ❖ Arrange the minterms based on the number of 1's
- ❖ Compare each binary number from one group to other and if they differ only one bit position, put dash (–) mark and copy the remaining term. Please tick (√) mark after each comparison.
- ❖ Apply the same process described in step 3 for the resultant column and these cycles have to be continued unitl no new list can be found (i.e., no further elimination of literals)
- ❖ List the unchecked (unticked) implicant and form prime implicant chart.

Prime implicant chart

- ❖ The prime implicants should be represented in rows and each minterm of the function in a column.

- The cross (X) mark should be placed in each row to show the comparison of minterms that make the prime implicants.
- Search for single **X** column and select prime implicants corresponding to that dot by putting the star (*) mark in front of it.
- Prime implicants that cover minterms with a single cross in their column are called essential prime implicants.
- Write the simplified expression using prime implicants.

Example 1.154: Simplify the Boolean function by using tabulation method

$$F(A,B,C,D) = \sum m\,(0,2,3,6,7,8,10,12,13)$$

Solution: The minterms are represented in the binary form as shown in **Table 1.23(a)**

Table 1.23(a) Binary representation of minterms

Minterm	*Binary equivalent*
0	0 0 0 0
2	0 0 1 0
3	0 0 1 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
10	1 0 1 0
12	1 1 0 0
13	1 1 0 1

The above binary representation are grouped into a number of sections in terms of the number of 1's as shown in **Table 1.23(b)**

Table 1.23(b) Group of minterms for different number of 1's

Number of 1's	*Minterm*	*A B C D*	
0	0	0 0 0 0	√
1	2	0 0 1 0	√
	8	1 0 0 0	√
2	3	0 0 1 1	√
	6	0 1 1 0	√
	10	1 0 1 0	√
	12	1 1 0 0	√
3	7	0 1 1 1	√
	13	1 1 0 1	√

Any two number in these groups which differ from each other by only variable can be chosen and combined, to get 2 cell combination as shown in **Table 1.23(c)**.

Table 1.23(c) 2-cell combination

Combination	*A*	*B*	*C*	*D*	
(0,2)	0	0	–	0	√
(0,8)	–	0	0	0	√
(2,3)	0	0	1	–	√
(2,6)	0	–	1	0	√
(2,10)	–	0	1	0	√
(8,10)	1	0	–	0	√
(8,12)	1	–	0	0	
(3,7)	0	–	1	1	√
(6,7)	0	1	1	–	√
(12,13)	1	1	0	–	

Table 1.23(d) 4 cell combination

Combination	*A*	*B*	*C*	*D*
(0,2,8,10)	–	0	–	0
(2,3,6,7)	0	–	1	–

From the 2-cell combinations, are variable and a dash (–) in the same position can be combined to form 4-cell combination as shown in **Table 1.23(d)**.

The cells (0,2) and (8,10) from the same 4 cell combination as the cells (0,8) and (2,10). The order in which the cells are placed in a combination does not have any effect. Thus the (0,2,8,10) combination may be given as (0,8,2,10)

i.e., (0,2,8,10) = (0,8,2,10)

(2,3,6,7) = (2,6,3,7)

Using Table 1.28(c) and (d) the prime implicants table can be as shwon in **Table 1.23(e)**.

Table 1.23(e) Prime Implicant Table

Prime Implicants	**Minterms**								
	0	**2**	**3**	**6**	**7**	**8**	**10**	**12**	**13**
(8, 12)						×		×	
(12,13)*								×	×
(0,2,8,10)*	×	×				×	×		
(2,3,6,7)*		×	×	×	×				
	√		√	√	√				√

The columns having only one cross (X) mark correspond to essential prime implicants. A tick mark put against every column which has only one cross mark. A star (*) mark is placed against every essential prime implicant. The sum of the prime implicants gives the function in its minimal SOP form. Therefore, $F = (1\ 1\ 0\ -) + (-\ 0\ -\ 0) + (0\ -\ 1\ -)$

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

Example 1.155: Find the minimal SOP for the given function using Quine-McCluskey method

$$F = \sum m\,(0,1,2,8,10,11,14,15)$$ **(April 2005)**

Solution: **Binary representation of minterms**

Minterms	*Binary equivalent*
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
8	1 0 0 0
10	1 0 1 0
11	1 0 1 1
14	1 1 1 0
15	1 1 1 1

Group of minterms for different number of 1's:

Number of 1's	*Minterm*	*A*	*B*	*C*	*D*	
0	0	0	0	0	0	√
1	1	0	0	0	1	√
	2	0	0	1	0	√
	8	1	0	0	0	√
2	10	1	0	1	0	√
3	11	1	0	1	1	√
	14	1	1	1	0	√
4	15	1	1	1	1	√

2 cell combination					
Combination	*A*	*B*	*C*	*D*	
(0,1)	0	0	0	–	
(0,2)	0	0	–	0	√
(0,8)	–	0	0	0	√
(2,10)	–	0	1	0	√
(8,10)	1	0	–	0	√
(10,11)	1	0	1	–	√
(10,14)	1	–	1	0	√
(11,15)	1	–	1	1	√
(14,15)	1	1	1	–	√

4 cell combination				
Combination	*A*	*B*	*C*	*D*
(0,2,8,10)	–	0	–	0
(10,11,14,15)	1	–	1	–

Here (0,2,8,10) = (0,8,2,10), (10,11,14,15) = (10,14,11,15)

Prime Implicants Table

Prime Implicants	Minterms							
	0	**1**	**2**	**8**	**10**	**11**	**14**	**15**
(0,1)*	×	×						
(0,2,8,10)*	×		×	×	×			
(10,11,14,15)*			×	×	×	×	×	×
		√	√	√		√	√	√

$F = [(0\ 0\ 0\ -) + (-\ 0\ -\ 0) + (1\ -\ 1\ -)]$

$F = \overline{A}\,\overline{B}\,\overline{C} + \overline{B}\,\overline{D} + AC$

Example 1.156: Simplify the following function using tabulation method

$$F(A,B,C,D) = \sum m\,(1,2,3,5,9,12,14,15) + \sum d\,(4,8,11)$$ **(Dec 2005)**

Solution: The don't care conditions are used to find the prime implicants; but it is not compulsory to include don't care term in the final expression.

Binary representation of minterms

Minterms	*Binary equivalent*
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
5	0 1 0 1
9	1 0 0 1
12	1 1 0 0
14	1 1 1 0
15	1 1 1 1
d4	0 1 0 0
d8	1 0 0 0
d11	1 0 1 1

Group of minterms for difference number of 1's:

Number of 1's	*Minterm*	*A*	*B*	*C*	*D*	
1	1	0	0	0	1	√
	2	0	0	1	0	√
	d4	0	1	0	0	√
	d8	1	0	0	0	√
2	3	0	0	1	1	√
	5	0	1	0	1	√
	9	1	0	0	1	√
	12	1	1	0	0	√
3	d11	1	0	1	1	√
	14	1	1	1	0	√
4	15	1	1	1	1	√

2-cell combinations					
Combination	*A*	*B*	*C*	*D*	
(1,3)	0	0	–	1	√
(1,5)	0	–	0	1	
(1,9)	–	0	0	1	√
(2,3)	0	0	1	–	
(4,5)	0	1	0	–	
(4,12)	–	1	0	0	
(8,9)	1	0	0	–	
(3,11)	–	0	1	1	√
(9,11)	1	0	–	1	√
(12,14)	1	1	–	0	
(11,15)	1	–	1	1	
(14,15)	1	1	1	–	

4-cell combinations				
Combination	*A*	*B*	*C*	*D*
(1,3,9,11)	–	0	–	1

Prime Implicant Table

Prime implicant	*A*	*B*	*C*	*D*
(1,5)	0	–	0	1
(2,3)	0	0	1	–
(4,5)	0	1	0	–
(4,12)	–	1	0	0
(8,9)	1	0	0	–
(8,12)	1	–	0	0
(12,14)	1	1	–	0
(11,15)	1	–	1	1
(14,15)	1	1	1	–
(1,3,9,11)	–	0	–	1

Select the minimum number of prime implicants which must cover all the minterms, except don't care minterms:

Prime Implicants	m_1	m_2	m_3	d_4	m_5	d_8	m_9	d_{11}	m_{12}	m_{14}	m_{15}
(1,5)*	×				×						
(2,3)*		×	×								
(4,5)				×	×						
(4,12)				×					×		
(8,9)						×	×				
(8,12)						×			×	×	
(12,14)*									×		×
(11,15)								×			×
(14,15)*										×	×
(1,3,9,11)*	×		×			×	×				
		√									

- Only m_2 column has single cross (X) mark and hence the prime implicant corresponding to it (2,3) is included in the final expression.
- m_1 column has 2 cross marks. We can include the prime implicants which has more minterms - (1,3,9,11).
- Columns d_4, d_8 and d_{11} are don't cares.
- m_5 is not included yet, therefore prime implicants (1,5) is included in the final expression.
- m_{12} is not included yet, therefore prime implicant (12,14) is included.
- m_{15} also can be included in the final expression by including prime implicant (14,15)

The final expression is,

$$F = (0 - 0\ 1) + (\ 0\ 0\ 1\ -) + (1\ 1 - 0\ ) + (1\ 1\ 1\ -) + (-\ 0 - 1)$$

$$= \overline{A}\,\overline{C}\,D + \overline{A}\,\overline{B}\,C + A\,B\,\overline{D} + A\,B\,C + \overline{B}\,D$$

Example 1.157: Simplify the given function

$$F = (A,B,C,D) = \sum m\,(2,3,7,9,11,13) + \sum d\,(1,\ 10,\ 15)$$

Solution: The don't cares are treated like required minterms when finding the prime implicants.

Minterm	***Binary Representation***
d1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
7	0 1 1 1
9	1 0 0 1
d10	1 0 1 0
11	1 0 1 1
13	1 1 0 1
d15	1 1 1 1

Group of minterms for different number of 1's:

Number of 1's	**Minterm**	***A B C D***
1	1	0 0 0 1 √
	2	0 0 1 0 √
2	3	0 0 1 1 √
	9	1 0 0 1 √
3	10	1 0 1 0 √
	7	0 1 1 1 √
	11	1 0 1 1 √
	13	1 1 0 1 √
4	15	1 1 1 1 √

2-cell combination					
Combination	***A***	***B***	***C***	***D***	
(1,3)	0	0	–	1	√
(1,9)	–	0	0	1	√
(2,3)	0	0	1	–	√
(2,10)	–	0	1	0	√
(3,7)	0	–	1	1	√
(3,11)	–	0	1	1	√
(9,11)	1	0	–	1	√
(9,13)	1	–	0	1	√
(10,11)	1	0	1	–	√
(7,15)	–	1	1	1	√
(11,15)	1	–	1	1	√
(13,15)	1	1	–	1	√

4-cell combination				
Combination	***A***	***B***	***C***	***D***
(1,3,9,11)	–	0	–	1
(2,3,10,11)	–	0	1	–
(3,7,11,15)	–	–	1	1
(9,11,13,15)	1	–	–	1

The don't care columns are omitted when forming prime implicants table

Prime implicants	**2**	**3**	**7**	**9**	**11**	**13**
(1,3,9,11)		×		×	×	
(2,3,10,11)*	×	×			×	
(3,7,11,15)*		×	×		×	
(9,11,13,15)*				×	×	×
	√		√			√

$$F = (-\ 0\ 1\ -) + (-\ -\ 1\ 1) + (1\ -\ -\ 1)$$
$$= \overline{B}\ C + CD + AD$$

Example 1.158: Simplify the following 5 vairiable expression using Mccluskey method.

F = ∑m(0, 1, 9, 15, 24, 29, 30) + d(8, 11, 31) **(Dec 2010)**

Solution:

	(1)	**(2)**	**(3)**
0	0000	0, 1 (1)	0, 1, 8, 9
1	0001	0, 8 (8)	
8	1000	1, 9(8)	
9	01001	8, 9(1)	
24	11000	8, 24(16)	
11	1011	9, 11(2)	
15	01111	11, 15(4)	
29	11101	15, 31(16)	
30	11110	30, 31(1)	
31	11111		

Prime implicant table

	0	1	9	15	24	29	30
0, 1, 8, 9	✓	✓	✓				
8, 24			✓		✓		
9, 11			✓				
11, 15				✓			
15, 31				✓			
29, 31							
30, 31							✓
	✓	✓			✓	✓	✓

Answer:

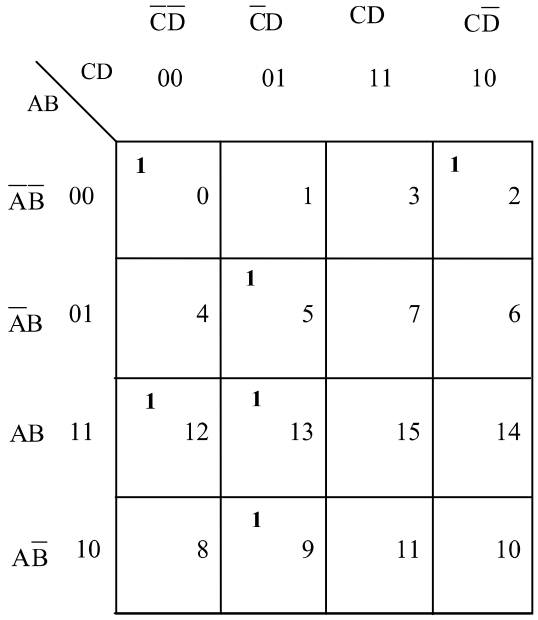
A'C'D' + BC'D'E' + A'BDE + ABCE + ABCD

Or

A'C'D' + BC'D'E' + BCDE + ABCE + ABCD

Example 1.159: Minimize the expression using Quine McCluskey (Tabulation) method

$Y' = A'B'C'D' + A'BC'D + ABC'D' + ABC'D + AB'C'D + A'B'CD'$ **(May 2012)**



Given, F(A, B, C, D) = {0, 2, 5, 9, 12, 13}

Min term	Binary equivalent			
0	0	0	0	0
2	0	0	1	0
5	0	1	0	1
9	1	0	0	1
12	1	1	0	0
13	1	1	0	1

Number of 1's	Minterms	A	B	C	D
0	0	0	0	0	0
1	2	0	0	1	0
2	5	0	1	0	1
	9	1	0	0	1
	12	1	1	0	0
3	13	1	1	0	1

2 – cell combination

Combination	A	B	C	D
(0, 2)	0	0	–	0
(5, 13)	–	1	0	1
(9, 13)	1	–	0	1
(12, 13)	1	1	0	–

Prime implicants	0	2	5	9	12	13
(0, 2)	×	×				
(5, 13)			×			×
(9, 13)				×		×
(12, 13)					×	×
	✓	✓	✓	✓	✓	

F = A'B'D' + BC'D + AC'D + ABC'

Example 1.160: Simplify the Boolean function using Quine McCluskey method:

$F(A, B, C, D, E, F) = \sum m(0, 5, 7, 8, 9, 12, 13, 23, 24, 25, 28, 29, 37, 40, 42, 44, 46, 55, 56, 57, 60, 61)$ **(May 2013)**

Solution:

Minterms	Binary equivalent					
0	0	0	0	0	0	0
5	0	0	0	1	0	1
7	0	0	0	1	1	1
8	0	0	1	0	0	0
9	0	0	1	0	0	1
12	0	0	1	1	0	0
13	0	0	1	1	0	1
23	0	1	0	1	1	1
24	0	1	1	0	0	0
25	0	1	1	0	0	1
28	0	1	1	1	0	0
29	0	1	1	1	0	1
37	1	0	0	1	0	1
40	1	0	1	0	0	0
42	1	0	1	0	1	0
44	1	0	1	1	0	0
46	1	0	1	1	1	0
55	1	1	0	1	1	1
56	1	1	1	0	0	0
57	1	1	1	0	0	1
60	1	1	1	1	0	0
61	1	1	1	1	0	1

Group of mintrems for different number of 1's

Number of 1's	Minterms	A	B	C	D	E	F
0	0	0	0	0	0	0	0
1	8	0	0	1	0	0	0
2	5	0	0	0	1	0	1
	9	0	0	1	0	0	1

	12	0	0	1	1	0	0
	24	0	1	1	0	0	0
	40	1	0	1	0	0	0
3	7	0	0	0	1	1	1
	13	0	0	1	1	0	1
	25	0	1	1	0	0	1
	28	0	1	1	1	0	0
	37	1	0	0	1	0	1
	42	1	0	1	0	1	0
	44	1	0	1	1	0	0
	56	1	1	1	0	0	0
4	23	0	1	0	1	1	1
	29	0	1	1	1	0	1
	46	1	0	1	1	1	0
	57	1	1	1	0	0	1
	60	1	1	1	1	0	0
5	55	1	1	0	1	1	1
	61	1	1	1	1	0	1

2-Cell Combination

Combination	**A**	**B**	**C**	**D**	**E**	**F**
(0, 8)	0	0	–	0	0	0
(8, 9)	0	0	1	0	0	–
(8, 12)	0	0	1	–	0	0
(8, 24)	0	–	1	0	0	0
(8, 40)	–	0	1	0	0	0
(5, 7)	0	0	0	1	–	1
(5, 13)	0	0	–	1	0	1
(5, 37)	–	0	0	1	0	1
(9, 13)	0	0	1	–	0	1
(9, 25)	0	–	1	0	0	1
(12, 13)	0	0	1	1	0	–
(12, 28)	0	–	1	1	0	0

(24, 25)	0	1	1	0	0	–
(24, 28)	0	1	1	–	0	0
(24, 56)	–	1	1	0	0	0
(40, 42)	1	0	1	0	–	0
(40, 44)	1	0	1	–	0	0
(40, 56)	1	–	1	0	0	0
(7, 23)	0	–	0	1	1	1
(13, 29)	0	–	1	1	0	1
(25, 29)	0	1	1	–	0	1
(25, 57)	–	1	1	0	0	1
(28, 29)	0	1	1	1	0	–
(28, 60)	–	1	1	1	0	0
(42, 46)	1	0	1	–	1	0
(44, 46)	1	0	1	1	–	0
(44, 60)	1	1	1	–	0	0
(56, 57)	1	1	1	0	0	–
(56, 60)	1	1	1	–	0	0
(23, 55)	–	1	0	1	1	1
(29, 61)	–	1	1	1	0	1
(57, 61)	1	1	1	–	0	1
(60, 61)	1	1	1	1	0	–

4 Cell Combination

Combination	**A**	**B**	**C**	**D**	**E**	**F**
(9, 13, 25, 29)	0	0	–	–	0	1
(12, 13, 28, 29)	0	–	1	1	0	–
(24, 25, 28, 29)	0	1	1	–	0	–
(24, 25, 56, 57)	–	1	1	0	0	–
(24, 28, 44, 60)	–	1	1	–	0	0
(24, 28, 56, 60)	–	1	1	–	0	0
(40, 42, 44, 46)	1	0	1	–	–	0
(25, 29, 57, 61)	–	1	1	–	0	1
(28, 29, 60, 61)	–	1	1	1	0	–

Prime Implicant Table

	0	5	7	8	9	12	13	23	24	25	28	29	37	40	42	44	46	55	56	57	60	61
(0, 8)	×			×																		
(8, 9)				×	×																	
(8, 12)				×		×			×													
(8, 24)				×					×													
(8, 40)				×										×								
(5, 7)		×	×																			
(5, 13)		×					×															
(5, 37)		×											×									
(40, 56)														×					×			
(7, 23)			×					×														
(23, 55)								×										×				
(9, 13, 25, 29)					×		×			×		×										
(12, 13, 28, 29)						×	×				×	×										
(24, 25, 28, 29)									×	×	×	×										
(24, 25, 56, 57)									×	×									×	×		
(24, 28, 44, 60)									×		×					×					×	
(24, 28, 56, 60)									×		×								×		×	
(40, 42, 44, 46)														×	×	×	×					
(25, 29, 57, 61)										×		×								×		×
(28, 29, 60, 61)											×	×									×	×
	✓												✓		✓		✓	✓				

F(A, B, C, D, E, F) = (00 –000) + (–00101) + (101– –0)

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

1.28 BASIC LOGIC GATES

There are three basic logic gates each of which performs a basic logic function, they are called NOT, AND and OR. All other logic functions can ultimately be derived from combinations of these three. For each of three basic logic gates a summary is given including the **logic symbol**, the corresponding **truth table** and the **Boolean expression**.

1.28.1 The NOT gate

The NOT gate is unique in that it only has one input. The logic symbol of NOT gate is shown in **Figure 1.28**.

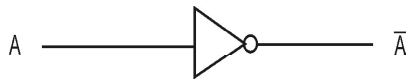


Fig. 1.28: Logic Symbol

The input to the NOT gate **A** is **inverted** i.e., the binary input state of 0 gives an output of 1 and the binary input state of 1 gives an output of 0.

$\bar{A}$ is known as "NOT A" or alternatively as the **complement** of A.

The truth table for the NOT gate appears as below:

TRUTH TABLE

A	$\bar{A}$
0	1
1	0

1.28.2 The AND gate

The AND gates has two or more inputs. The output from the AND gate is 1 if and only if all of the inputs are 1, otherwise the output from the gate is 0. The AND gate is drawn as shown in **Figure 1.29.**

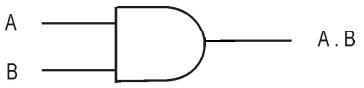


Fig. 1.29 : Logic Symbol

The output from the AND gate is written as **A . B**

The truth table for a two-input AND gate is given below:

TRUTH TABLE

A	*B*	*A* · *B*
0	0	0
0	1	0
1	0	0
1	1	1

1.28.3 The OR gate

The OR gate has two or more inputs. The output from the OR gate is 1 if any of the inputs is 1. The gate output is 0 if and only if all inputs are 0. The OR gate is drawn as shown in **Figure 1.30.**

Fig. 1.30 : Logic Symbol

The output from the OR gate is written as $A + B$.

The truth table for a two-input OR gate is given below:

TRUTH TABLE

A	*B*	*A* + *B*
0	0	0
0	1	1
1	0	1
1	1	1

1.29 OTHER LOGIC GATES

The three basic logic gates can be combined to provide more complex logical functions. Four important logical functions are described here, namely NAND, NOR, XOR and XNOR. In each case a summary is given including the **logic symbol** for that function, the corresponding **truth table** and the **Boolean expression**.

1.29.1 The NAND gate

The NAND gate has two or more inputs. The output from the NAND gate is 0 if and only if all of the inputs are 1 otherwise the output is 1. Therefore the output from the NAND gate is the NOT of A AND B (also known as the **complement** or **inversion** of $A \cdot B$). The NAND gate is shown in **Figure 1.31** where the small circle immediately to the right of the gate on the output line is known as an **invert bubble**.

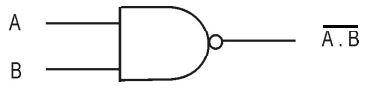


Fig. 1.31: Logic Symbol

The output from the NAND gate is written as $\overline{A \cdot B}$. The Boolean expression $\overline{A \cdot B}$ reads as "A NAND B". The truth table for a two-input NAND gate is given below:

TRUTH TABLE

A	*B*	$\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

1.29.2 The NOR gate

The NOR gate has two or more inputs. The output from the NOR gate is 1 if and only if all of the inputs are 0, otherwise the output is 0. This output behaviour is the NOT of A OR B. The NOR gate is drawn as shown in **Figure 1.32**.

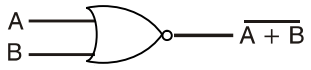


Fig. 1.32: NOR-Logic Symbol

The output from the NOR gate is written as **A+B** which reads "A NOR B".

The truth table for a two-input NOR gate is given below:

TRUTH TABLE

A	*B*	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

1.29.3 The Exclusive-OR (XOR) gate

The exclusive-OR or XOR gate has two or more inputs. For a two-input XOR the output is similar to that from the OR gate except it is 0 when the both inputs are 1. This cannot be extended to XOR gates comprising 3 or more inputs however.

In general, an XOR gate gives an output value of 1 when there are an odd number of 1's on the inputs to the gate. The truth table for a 3-input XOR gate below illustrates this point.

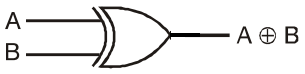


Fig. 1.33 : XOR-Logic Symbol

The XOR gate is drawn as shown in **Figure 1.33**.

The output from the XOR gate is written as $A \oplus B$ which reads **"A XOR B".**

The truth table for a two-input XOR gate looks like

TRUTH TABLE

A	*B*	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

$Y = A \oplus B = \overline{A}B + A\overline{B}$

For a 3-input XOR gate with inputs ***A, B*** and ***C*** the truth table is given by

A	*B*	*C*	$A \oplus B \oplus C$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

1.29.4 The Exclusive-NOR (XNOR) gate

The exclusive-NOR or XNOR gate has two or more inputs. The output is equivalent to inverting the output from the exclusive-OR gate described above. Therefore an equivalent circuit would comprise an XOR gate, the output of which feeds into the input of a NOT gate.

In general, an XNOR gate gives an output value of 1 when there are an even number of 1's on the inputs to the gate. The truth table for a 3-input XNOR gate below illustrates this point.

The XNOR gate is drawn using the same symbol as the XOR gate with an invert bubble on the output line as is illustrate in **Figure 1.34.**

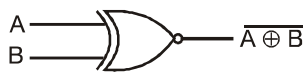


Fig. 1.34 : Logic Symbol

The output from the XNOR gate is written as $\overline{A \oplus B}$ which reads "A XNOR B".

$$Y = \overline{A \oplus B}$$

$$= \overline{\overline{A}B + A\overline{B}}$$

$$= \overline{\overline{A}B} + \overline{A\overline{B}}$$

$$= \left(A + \overline{B}\right)\left(\overline{A} + B\right)$$

$$Y = AB + \overline{A}\,\overline{B}$$

The truth table for a two-input XNOR gate looks like

TRUTH TABLE

A	B	$\overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

For a 3-input XNOR gate with inputs ***A, B*** and ***C*** the truth table is given by

A	B	C	$\overline{A \oplus B \oplus C}$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

1.30 GATE CONVERSIONS

Any logic gate can be replaced by sets of other interconnected logic gates. Therefore any digital design can be implemented with a small number of logic gate types. The gate conversion circuits are shown in **Figure 1.35.**

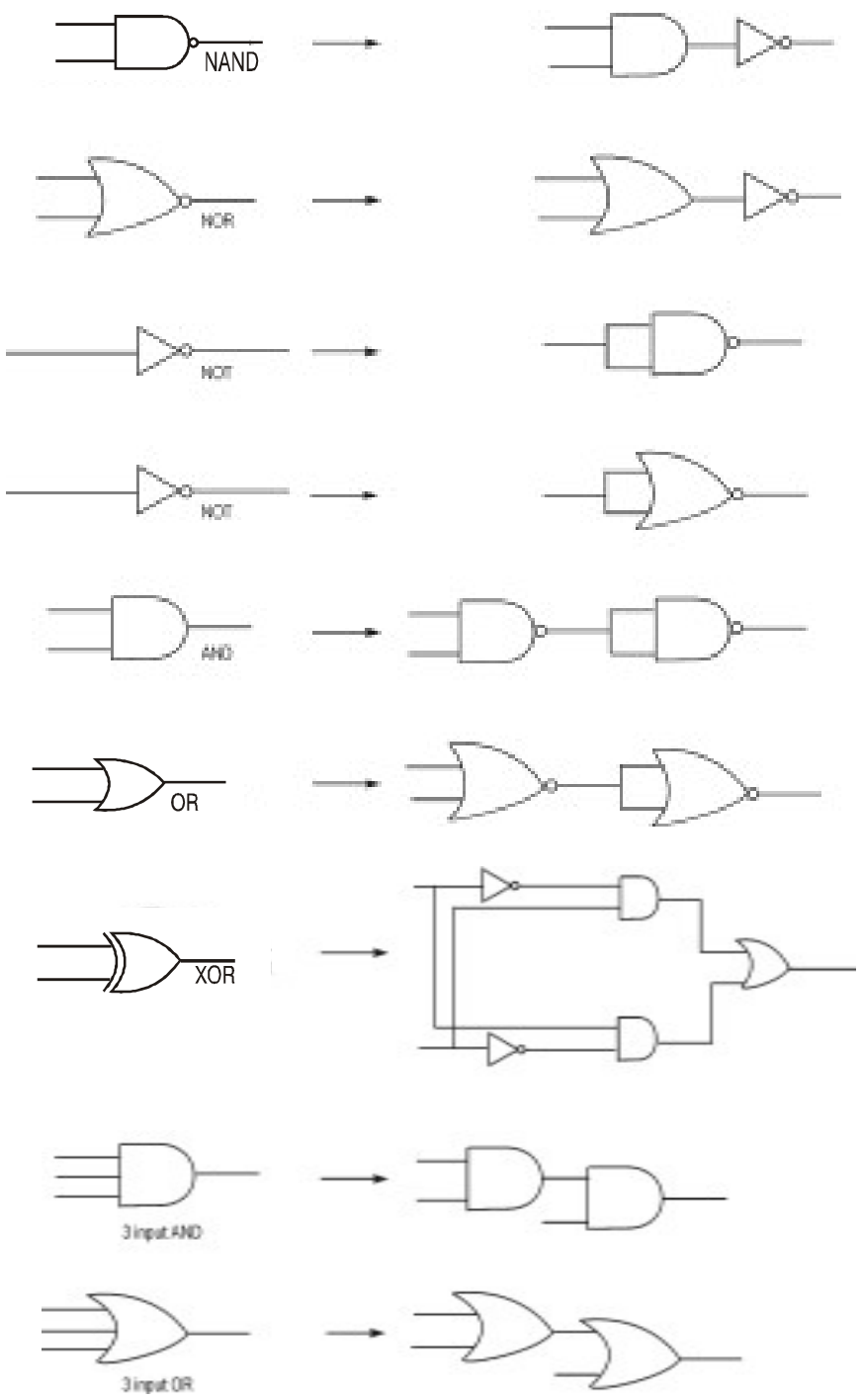


Fig. 1.35 : Gate Conversions

1.31 DIGITAL ICs

1.31.1 14 Pin DIP

The 14 Pin DIP (Dual-in-line package) was one of the first types of Integrated Circuits (IC's) developed. The term dual-in-line package comes from the two parallel sets of pins that are situated across each other on the IC. The typical layout of a 14 pin DIP is shown in **Figure 1.36.**

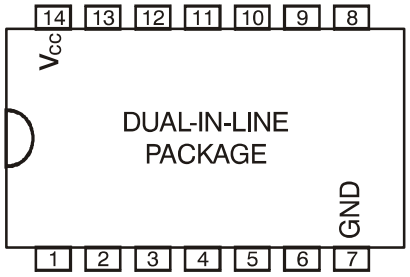


Fig. 1.36 : 14 Pin Dip

The pins are normally number counterclockwise with pin # 1 falling under the notch. For most IC's pin 7 is GND and pin 14 is Vcc. In order to utilize this IC, pin 14 must be connected to a supply voltage (usually 5V) and pin 7 must grounded. Then the various gates within the IC may be used for analysis.

1.31.2 74X04 Hex Inverter

The 74X04 is a hex inverter. The 04 is the number that determines the chip type. It is a hex inverter because it contains 6 inverters on a single IC. The X is in place of specific features that the IC may contain. The most commonly listed special feature is the 74LS04, where LS stands for Low Power Shottky Transistors. The pin-out for the 74X04 is shown in **Figure 1.37.**

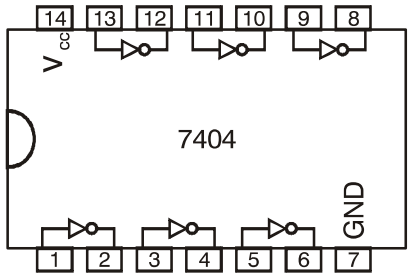


Fig. 1.37 : 7404 IC

1.31.3 74X08 Quad AND Gate

The 74X08 contains 4 AND gates. The 08 is the number that determines the chip type. It is called a quad AND gate because it contains 4 AND gates on a single IC. The pin-out for the 74X08 is shown in **Figure 1.38.**

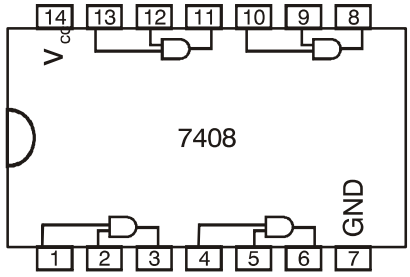


Fig. 1.38 : IC 7408

1.31.4 74X32 Quad OR Gate

The 74X32 contains 4 OR gates. The 32 is the number that determines the chip type. It is called a quad OR gate because it contains 4 OR gates on a single IC. The pinout for the 74X32 is shown in **Figure 1.39.**

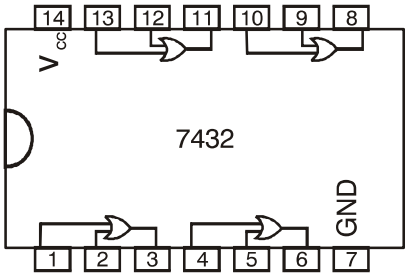


Fig. 1.39 : IC 7432

1.31.5 74X00 Quad NAND Gate

The 74X00 contains 4 NAND gates. The 00 is the number that determines the chip type. It is called a quad NAND gate because it contains 4 NAND gates on a single IC. The pin-out for the 74X00 is shown in **Figure 1.40.**

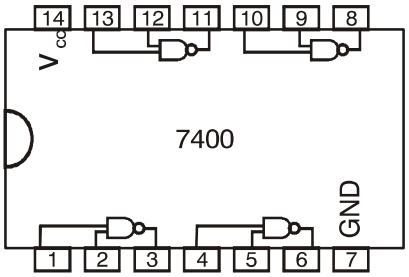


Fig. 1.40 : IC 7400

1.32 UNIVERSAL GATES

1.32.1 NAND GATE AS A UNIVERSAL GATE

The NAND gate is said to be a universal gate because any digital system can be implemented with it. The implementation of AND, OR and NOT operations with NAND gates is shown in **Figure 1.41** and EX-OR operations with NAND gates is shown in **Figure 1.42.**

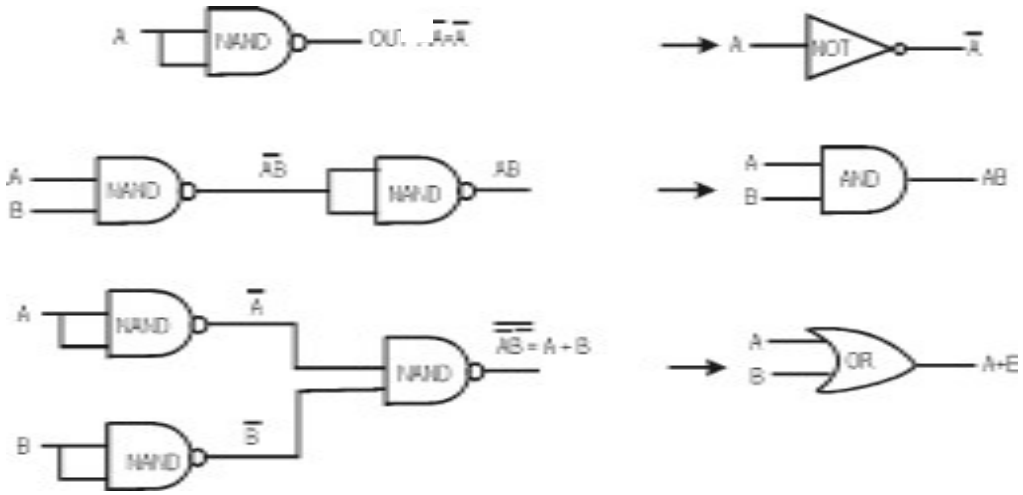


Fig. 1.41 : Implementation of basic gates using NAND

Exclusive OR with NAND

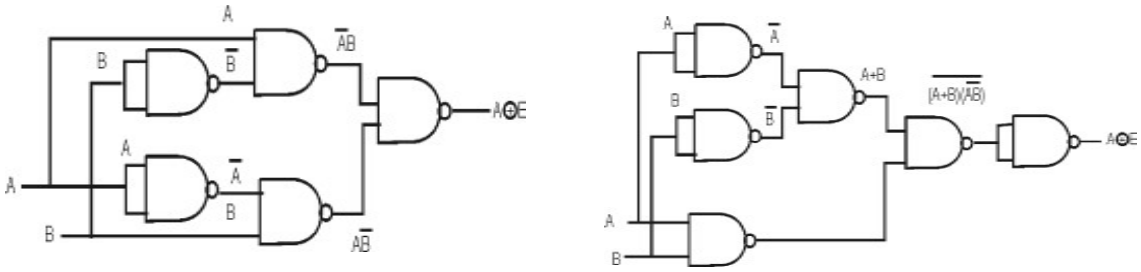


Fig. 1.42 : Implementation of EX-OR gate

1.32.2 NOR GATE AS A UNIVERSAL GATE

The NOR gate is called a universal gate because combinations of it can be used to accomplish all the basic functions. The implementation of basic gates using NOR gate is shown in **Figure 1.43.**

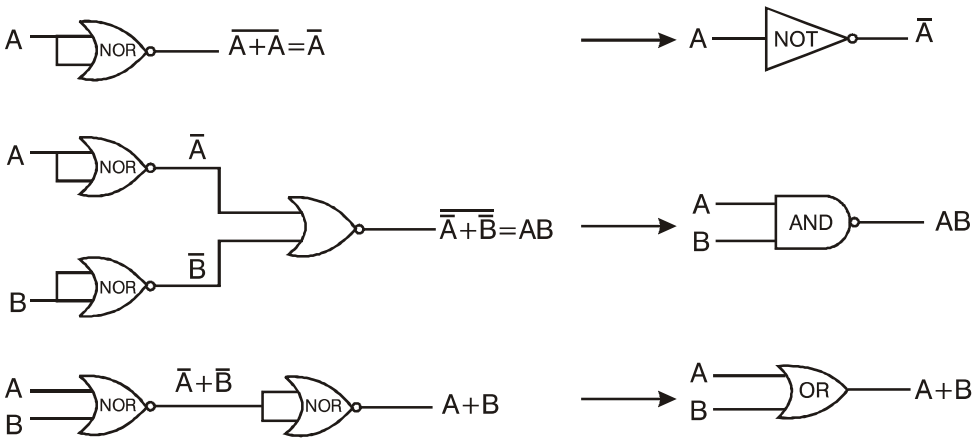


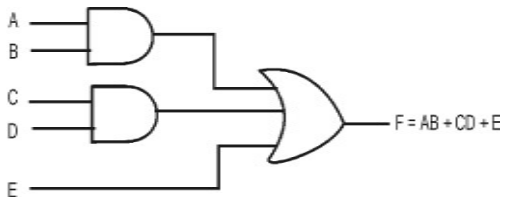
Fig. 1.43 : Implementation of Basic gates using NOR

1.33 IMPLEMENTATION OF LOGIC FUNCTIONS USING GATES

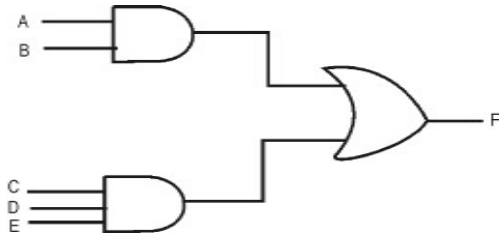
If the operation of the circuit is defined by a logic function, a logic circuit can be implemented directly from that function. For example, suppose we need a circuit that is defined by $X = ABC$. Then we immediately know that all that is needed is a 3 input AND gate. If we need a circuit that is defined by $X = A + \overline{B}$, we will use a two input OR gate with an inverter on one of the inputs $(\overline{B})$. The same reasoning used for these examples for these simple cases can be extended to more complex logic circuits.

Example 1.161: Draw the logic circuit using gate for the given function: $F = AB + CD + E$

Solution

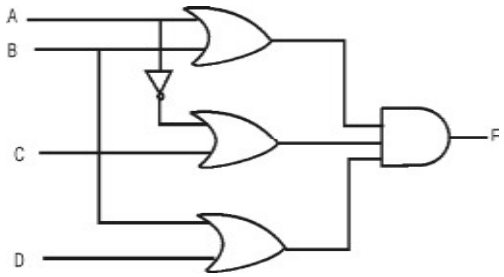


Example 1.162: Draw the logic circuit for $F = AB + CDE$



Example 1.163: Draw the logic circuit using basic gates for the function

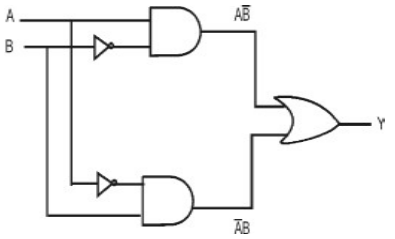
$$F = (A + B)(\overline{A} + C)(B + D)$$



Example 1.164: Draw the logic circuit using basic gates for EX-OR gate and EX-NOR gate.

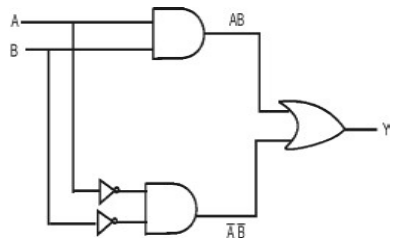
Solution: (i) The Boolean expression for EX-OR gate is

$$Y = A \oplus B = A\overline{B} + \overline{A}B$$



(ii) The Boolean expression for EX-NOR gate is,

$$Y = \overline{A \oplus B} = AB + \overline{A}\,\overline{B}$$



Example 1.165: Draw the logic circuit using basic gates for the function $F = AB\left(C\overline{D} + EF\right)$

Solution: $F = AB\left(C\overline{D} + EF\right) = ABC\overline{D} + ABEF$ (SOP form)

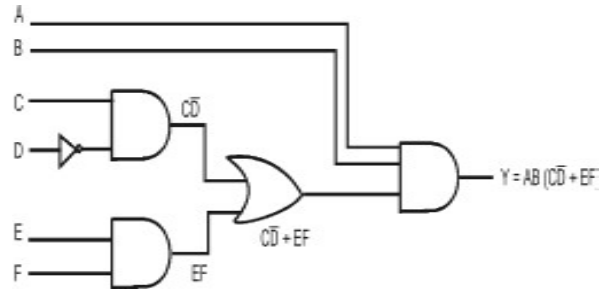


Fig. 1.44 : Logic circuit for $Y = AB\left(C\overline{D} + EF\right)$

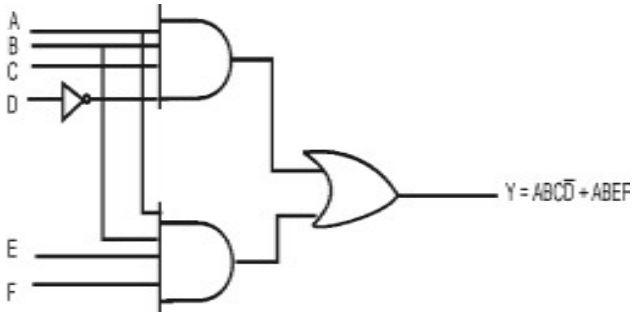
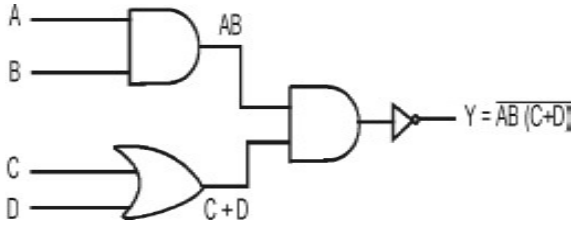


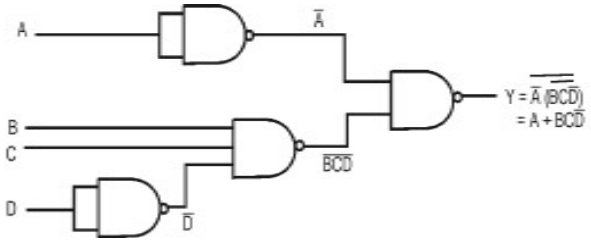
Fig. 1.45 : Logic circuit for $Y = ABC\overline{D} + ABEF$

Example 1.166: Draw the logic diagram for $Y = \overline{AB(C + D)}$

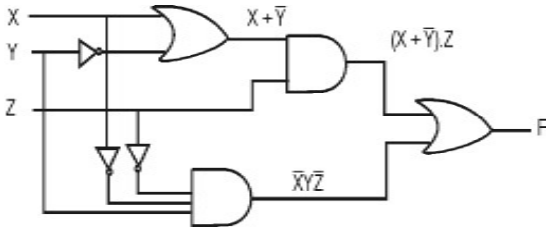


Example 1.167: Draw the logic circuit using NAND gates for

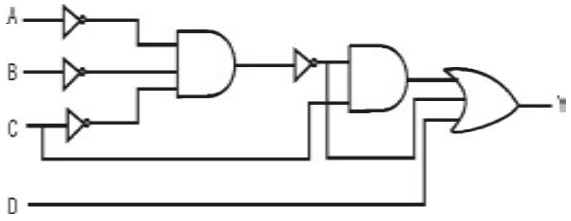
$$Y = A + BC\overline{D}$$



Example1.168: Draw the logic diagram for the function $F=\left(X+\overline{Y}\right)\cdot Z+\overline{X}\,Y\,\overline{Z}$.



Example 1.169: Reduce the given combinational logic circuit to a minimum form.



Solution: $Y=\left(\overline{\overline{A}\,\overline{B}\,\overline{C}}\right)C+\overline{\overline{A}\,\overline{B}\,\overline{C}}+D$

Applying DeMorgan's theorem an Boolean algebra,

$$Y=\left(\overline{\overline{A}+\overline{B}+\overline{C}}\right)C+\overline{\overline{A}+\overline{B}+\overline{C}}+D$$

$$=AC+BC+CC+A+B+C+D$$

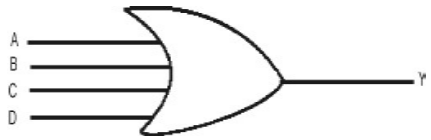
$$=AC+BC+C+A+B+C+D$$

$$=C\left(A+B+1\right)+A+B+D \qquad \left(C+C=C\right)$$

$$=C+A+B+D \qquad \left(A+B+1=1\right)$$

$$=A+B+C+D$$

The simplified circuit is,



Example 1.170:
Consider the combinational circuit shown in figure.

(i) Derive the Boolean expressions for T_1 through T_4. Evaluate the outputs F_1 and F_2 as a function of the four inputs.

$T_1 = \overline{B}C$

$T_2 = \overline{A}B$ $\qquad F_1 = T_3 + T_4$

$T_3 = T_1 + A$ $\qquad F_2 = T_2 + D$

$= \overline{B}C + A$

$T_4 = T_2 \oplus D$

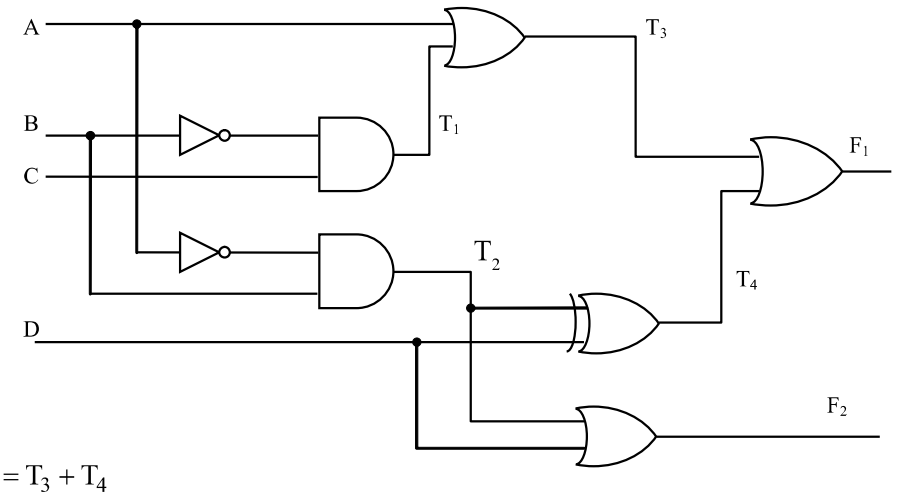
$= \overline{A}B\overline{D} + AD + \overline{B}D$

(ii) List the truth table with 16 binary combinations of the four input variables. Then list the binary values for T_1 through T_4 and outputs F_1 and F_2 in the table.

A	B	C	D	T_1	T_2	T_3	T_4	F_1	F_2
0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1	1	1
0	0	1	0	1	0	1	0	1	0
0	0	1	1	1	0	1	1	1	1
0	1	0	0	0	1	0	1	1	1
0	1	0	1	0	1	0	0	0	1
0	1	1	0	0	1	0	1	1	1
0	1	1	1	0	1	0	0	0	1
1	0	0	0	0	0	1	0	1	0
1	0	0	1	0	0	1	1	1	1

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

(iii) Plot the output Boolean function obtained in part (ii) on maps and show that simplified Boolean expressions are equivalent to the ones obtained in Part (i)



$$F_1 = T_3 + T_4$$

$$= T_1 + A + T_2 \oplus D$$

$$= \overline{B}C + A + \left[\left(\overline{A}B\right) \oplus D\right]$$

$$= A + \overline{B}C + \overline{A}B\overline{D} + D\left[\overline{\overline{A}B}\right]$$

$$= A + \overline{B}C + \overline{A}B\overline{D} + AD + \overline{B}D$$

$$= A + \overline{B}D + \overline{B}C + \overline{B}D$$

$$F_2 = T_2 + D$$

$$= \overline{A}B + D$$

$$= A + \overline{B}D + \overline{B}C + \overline{B}D$$

$$F_2 = T_2 + D$$

$$= \overline{A}B + D$$

1.34 NAND AND NOR IMPLEMENTATIONS

The NAND and NOR gates are the universal gates. NAND and NOR gates can be used to produce the AND, OR and NOT functions. Also, NAND and NOR gates are easier to fabricate with electronic components and are the basic gates used in all IC digital logic families.

When implementing any Boolean expression, it involves various logic gates and it needs various standard ICs. But all gates within the standard ICs are not utilized. For example to implement Boolean expression $\overline{AB}+CD$ we require two AND gates, one OR gate and one NOT gate. This requires 3 standard ICs. But gates utilized from ICs are very less. To improve utilization of ICs and reduce number of ICs required, NAND/NOR gates are used to implement Boolean expression. For this, conversion of AND/OR/NOT gates into NAND/NOR gates is needed.

The graphic symbols for NAND gate are shown in **Figure 1.46** and the graphic symbols for NOR gate are shown in **Figure 1.47.** A one-input NAND or NOR gate behaves like an inverter as shown in **Figure 1.48.**

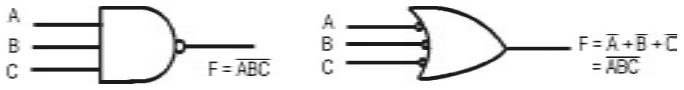


Fig. 1.46 : Graphic symbol for NAND gate

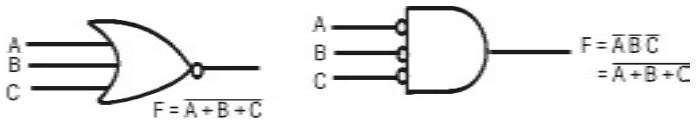


Fig. 1.47 : Graphic symbol for NOR gate

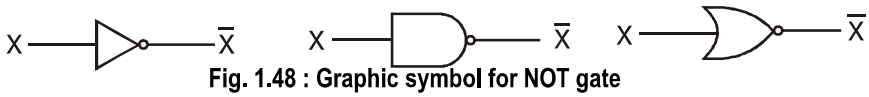


Fig. 1.48 : Graphic symbol for NOT gate

The rules for obtaining to NAND/NOR Logic:

1. Simplify the function and express it in sum of products (AND/OR logic).
2. For NAND implementation, add bubbles on the output of each AND gate and add bubbles on the input side to all OR gates.

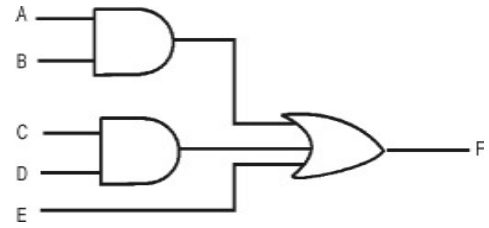
3. For NOR implementation, add bubbles on the output of each OR gate and add bubbles on the input side of all AND gates.
4. Add or subtract an inverter on each line that received a bubble in step 2 or 3.
5. Replace bubbled OR by NAND and bubbled AND by NOR.
6. Eliminate double inversions.

1.34.1 NAND gate implementation

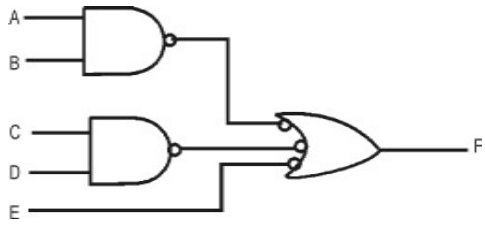
Example 1.171: Implement the Boolean function with NAND gates. $F = AB + CD + E$

Solution:

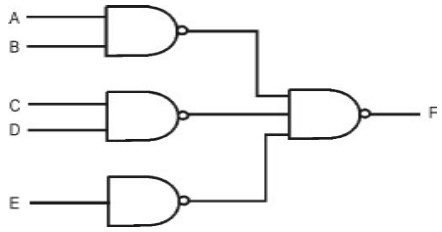
Step 1: Draw AND-OR circuit.



Step 2: Add bubbles on output of each AND gate and input of OR gate.



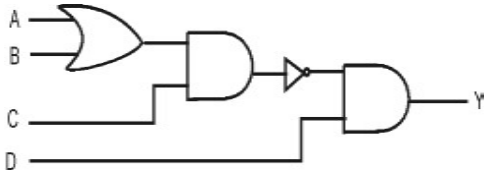
Step 3: Replace other gates by NAND gates.



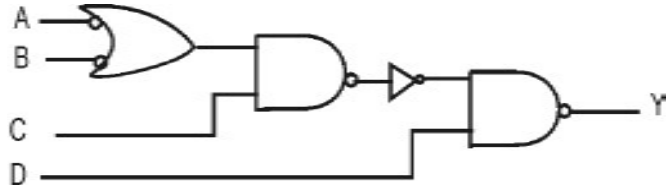
Example 1.172: Implement the Boolean expression with NAND gates $Y = \left(\overline{(A+B)C}\right)D$

Solution: Step 1: Draw original logic diagram for

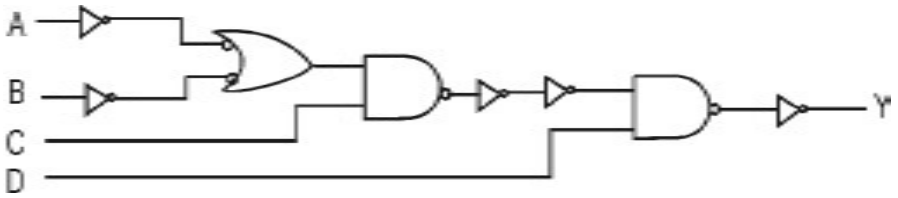
$$Y = \left(\overline{(A+B)C}\right)D$$



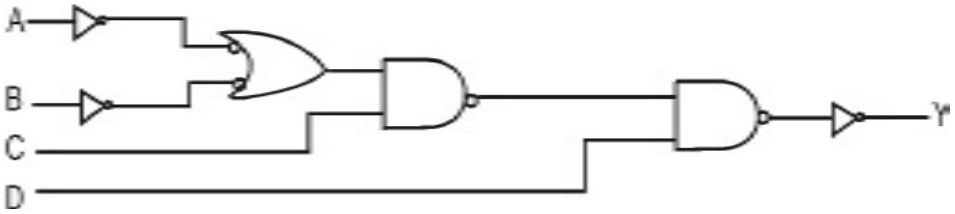
Step 2: Add bubbles on the output of the AND gates and input of the OR gate.



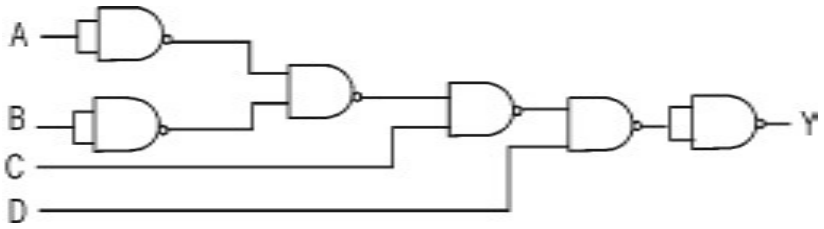
Step 3: Add inverters on each line that received a bubble.



Step 4: Eliminate double inversions.

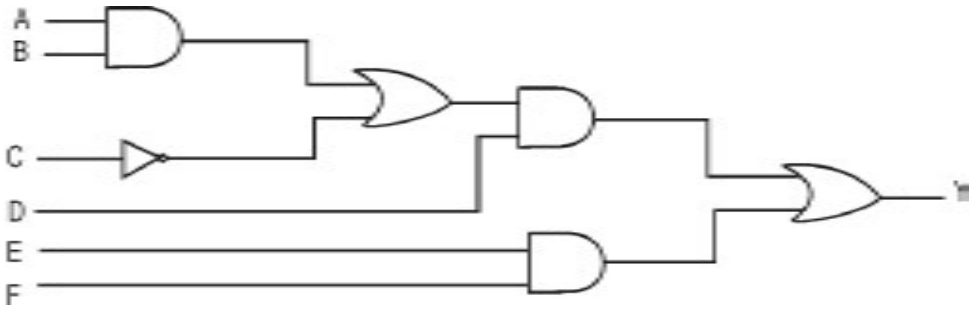


Step 5: Replace the other gates by only NAND gates.

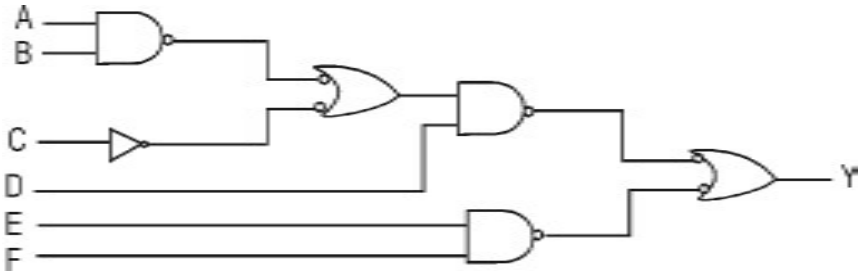


Example 1.173: Implement NAND gates for $Y=\left(AB+\overline{C}\right)D+EF$. ***(April 2004)***

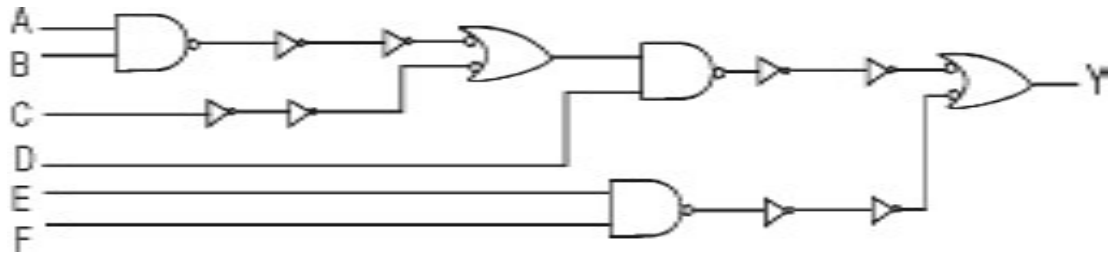
Solution: Step 1: Draw the original logic diagram for the expression $Y=\left(AB+\overline{C}\right)D+EF$



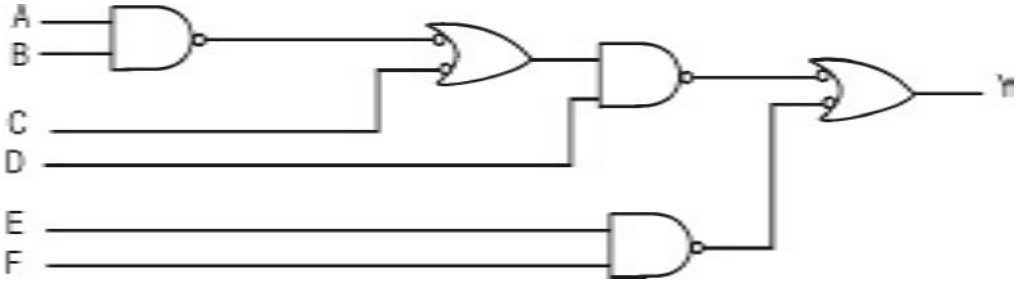
Step 2: Add bubbles on the output of the AND gates and input of the OR gates.



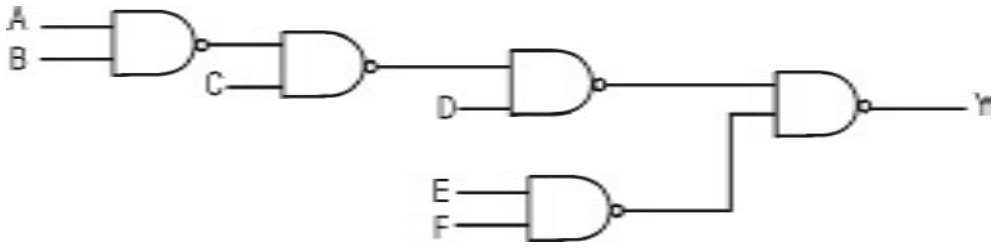
Step 3: Add inverters on each line that received a bubble.



Step 4: Eliminate double inversions.



Step 5: Draw the circuit with only NAND gates.



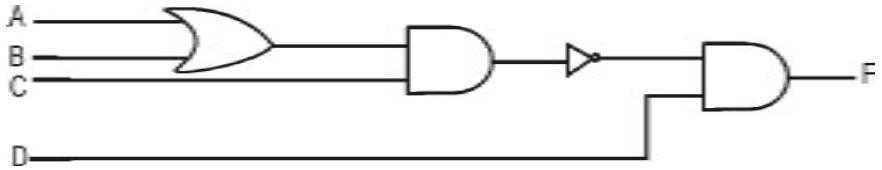
1.34.2 NOR gate implementation

Example 1.174: Implement the Boolean expression with NOR gates.

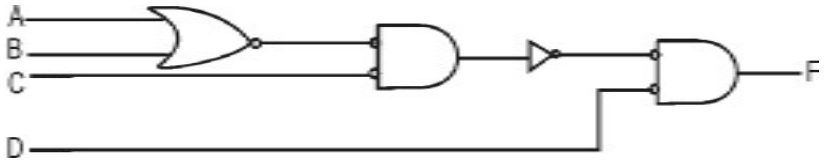
$$F = \overline{(A + B)C}.D$$

Solution:

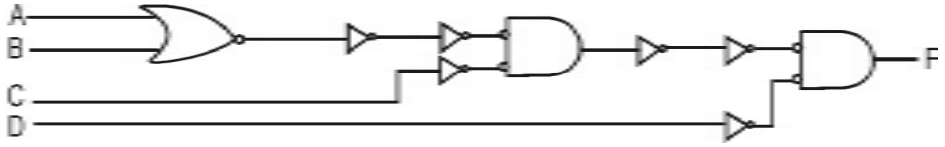
Step 1: Draw the original logic diagram for the given Boolean expression,



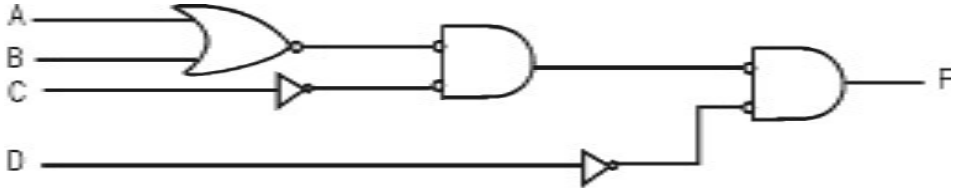
Step 2: Add bubbles on output of each OR gate and add bubbles on input of each AND gate.



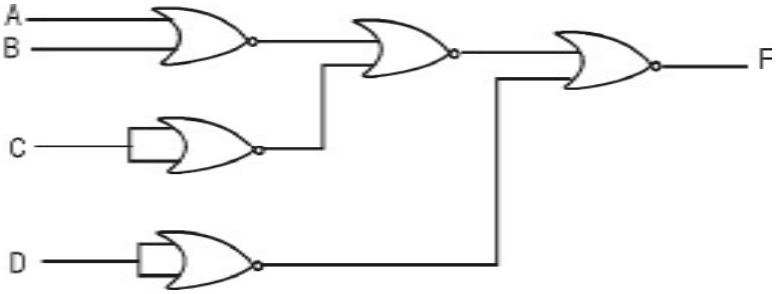
Step 3: Add inverters on each line that recieved bubbles.



Step 4: Eliminate double inversions.



Step 5: Draw the NOR diagram with one graphic symbol.

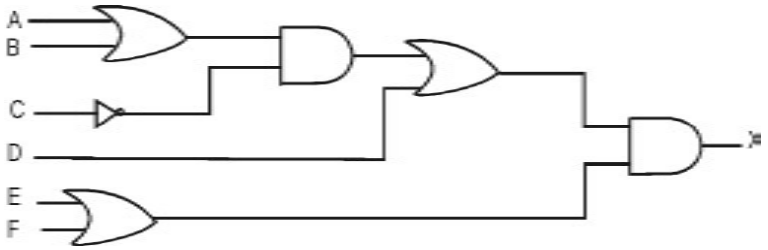


Example 1.175: Draw the multi level NOR circuit for the Boolean expression**:**

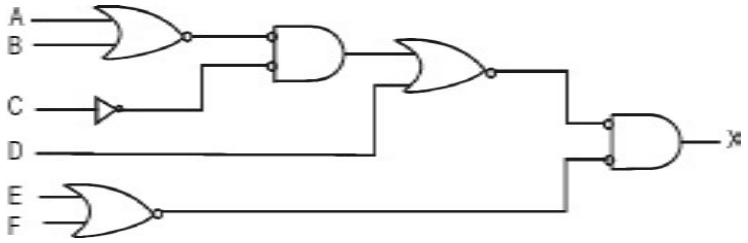
$$X=\left[\left(A+B\right)\overline{C}+D\right]\left(E+F\right)$$

Solution:

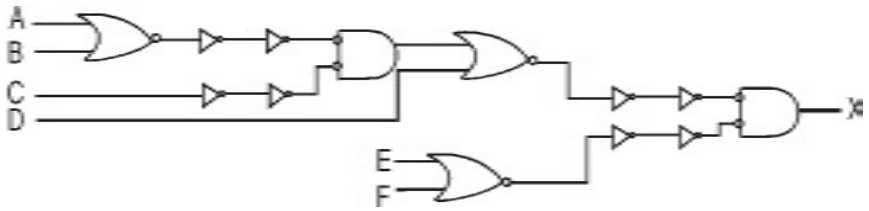
Step 1: Draw the original circuit diagram for $X=\left[\left(A+B\right)\overline{C}+D\right]\left(E+F\right)$



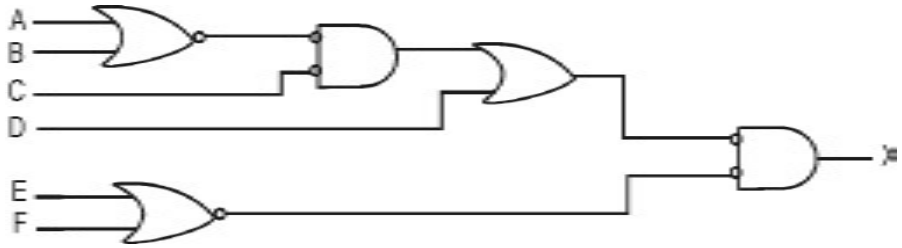
Step 2: Add bubbles on the output of OR gates and add bubbles on the input of AND gates.



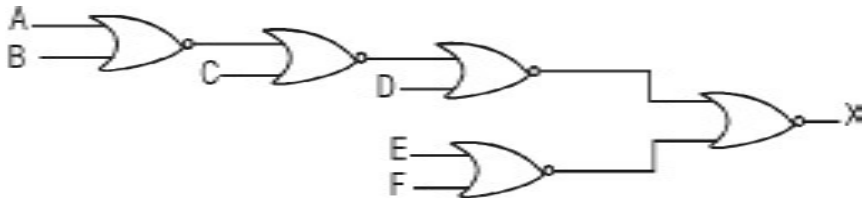
Step 3: Add inverters on each line that received bubbles.



Step 4: Eliminate Double Versions.



Step 5: Draw the NAND diagram using one graphic symbol.



1.35 MULTI LEVEL GATE IMPLEMENTATION

The maximum number of gates cascaded in series between a network input and the output is referred to as the number of levels of gates.

A function written in SOP form or in POS form corresponds directly to a 2-level gate network. AND-OR network means a 2-level network composed of a level of AND gates followed by an OR gate at the output. OR-AND network mean a 2-level network composed of a level of OR gates followed by an AND gate at the output. OR-AND-OR network means a 3-level network. The number of levels in an AND-OR network can usually be increased by factoring SOP expression from which it was derived. Similarly, the number of levels in an OR-AND network can usually be increased by multiplying out some of the terms in POS expression from which it was derived.

Logic designers are concerned with the number of levels in a network for several reasons. Sometime factoring or multiplying to increase the number of levels of gates. This will reduce the required number of gates and gate inputs and thus reduce the cost of the network. In other cases, increasing the number of levels will increase the cost. Now we will study some examples for 2 level, 3 level and 4 level networks.

1.35.1 Two Level AND-OR Network

The two level AND-OR network for the Boolean expression, $F = \overline{A}\overline{C}D + B\overline{C}D + BC\overline{D} + AC\overline{D}$ is shown in **Figure 1.49.**

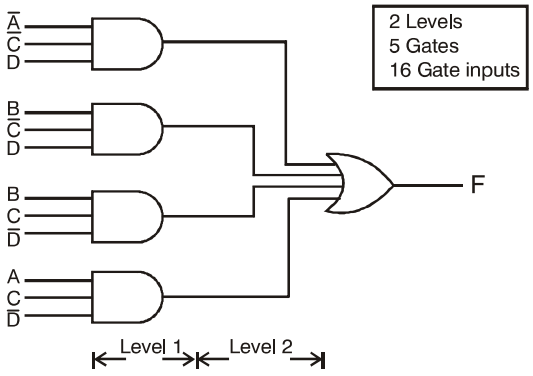


Fig. 1.49 : Two Level AND-OR