Contents

26

Digital Electronics

INTRODUCTION

A continuously varying signal (voltage or current) is called an *analog signal*. For example, a sinusoidal voltage is an analog signal. In the previous chapter, we studied the behaviour of diodes and transistors primarily from the analog or continuous-signal point of view. In an analog electronic circuit, the output voltage changes continuously according to the input voltage variations. In other words, the output voltage can have an infinite number of values. A signal (voltage or current) which can have only two discrete values is called a *digital signal*. For example, a square wave is a digital signal. The semiconductors devices (*e.g.* diodes, transistors etc.) can be designed for two-state operation *viz.*, saturation and cut off. In that case, the output voltage can have only two states (*i.e.*, values), either *low or high. An electronic circuit that is designed for two-state operation is called a **digital circuit**.

* The exact value of voltage is unimportant if the voltage is distinguishable as low or high.

The branch of electronics which deals with digital circuits is called **digital electronics**. When most of us hear the term *digital*, we immediately think of "digital calculator" or "digital computer". This is attributed to the dramatic way the low-cost, powerful calculators and computers have become accessible to an average person. Now digital circuits are being used in many electronic products such as video games, microwave ovens and oscilloscopes. Digital techniques have also replaced a lot of the older "analog circuits" used in consumer products such as radios, TV sets and high-fidelity sound recording and playback equipment. In this chapter, we shall discuss the fundamental aspects of digital electronics.

26.1 Analog and Digital Signals

(*i*) Analog signal. A continuously varying signal (voltage or current) is called an *analog signal*. For example, an alternating voltage varying sinusoidally is an analog signal [See Fig. 26.1]. If such an analog signal is applied to the input of a transistor amplifier, the output voltage will also vary sinusoidally. This is the analog operation *i.e.*, the output voltage can have an infinite number of values. Due to many-valued output, the analog operation is less reliable.

(*ii*) Digital signal. A signal (voltage or current) that can have only two discrete values is called a *digital signal*. For example, a square wave is a digital signal [See Fig. 26.2]. It is because this signal has only two values viz, +5 V and 0 V and no other value. These values are labelled as *High* and *Low*. The High voltage is + 5 V and the Low voltage is 0 V. If proper digital signal is applied to the input of a transistor, the transistor can be driven between *cut off* and *saturation.* In other words, the transistor will have two-state operations *i.e.*, output is either low or high. Since digital operation has only two states (*i.e.*, *ON* or *OFF*), it is far more reliable than many-valued analog operation. It is because with two-states operation, all the signals are easily recognised as either low or high.

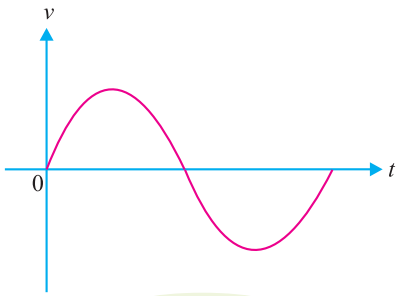


Fig. 26.1

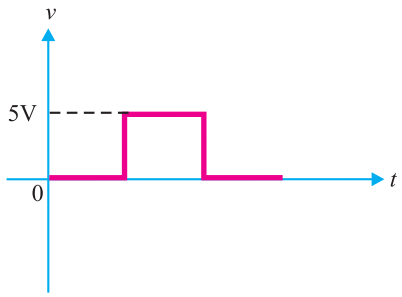


Fig. 26.2

26.2 Digital Circuit

An electronic circuit that handles only a digital signal is called a **digital circuit.**

The output voltage of a digital circuit is either low or high and no other value. In other words, digital operation is a two-state operation. These states are expressed as (*High* or *Low*) or (*ON* or *OFF*) or (1 or 0). Therefore, a digital circuit is one that expresses the values in digits 1's or 0's. Hence the name digital. The numbering concept that uses only the two digits 1 and 0 is the *binary numbering system.* Therefore, the first step would be to discuss this number system.

26.3 Binary Number System

A number system is a code that uses symbols to count the number of items. The most common and familiar number system is the decimal number system. The decimal number system uses the symbols 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. Thus, the decimal system uses 10 digits for counting the items. A binary system uses only two digits (0 and 1) for counting the items. The reader may wonder how to count the items in a binary system. Let us see how it is done.

Counting in Decimal and Binary systems. Figure 26.3 shows the counting of stones in decimal as well as binary system. As you will see, the counting in the binary number system is performed much the same way as in the decimal number system.

Stones	Decimal	Binary
No stone	0	0
●	1	1
●●	2	10
●●●	3	11
●●●●	4	100
●●●●●	5	101
●●●●●●	6	110
●●●●●●●	7	111
●●●●●●●●	8	1000
●●●●●●●●●	9	1001

Fig. 26.3

(*i*) Let us first see how items are counted in decimal system. In this system, the count starts as 0, 1, ..., 9. After 9, we are to write the next number. To do so, we use the second digit of the decimal system (*i.e.*, 1) followed by the first digit (*i.e.*, 0). So after 9, the next number is 10. The count again continues as 10, 11, 12 ....., 19. After 19, we use the third digit of the system (*i.e.*, 2) followed by the first digit (*i.e.*, 0) and the count continues as 20, 21, ... etc. In this way, we get the number upto 99. In order to represent a number next to 99, we use three decimal digits (100). That is to say second digit of the decimal system (*i.e.*, 1) followed by two first digits (*i.e*, two zeros).

(*ii*) Let us now turn to binary system. Note that 0 and 1 count in the binary system is the same as in the decimal counting. To represent 2 stones, we use the second binary digit (*i.e.*, 1) followed by the first (*i.e.*, 0). This gives binary number 10 (read as one-zero and not ten) as an equivalent of 2 in the decimal system. Likewise, 3 in the decimal system can be represented by the binary number 11 (read as one-one and not eleven). After this, the two binary digits are exhausted. We shall use three digits to represent the next binary number. Thus, to represent 4 (four), we use the second binary digit followed by two first binary digits. This gives the binary *100 (read as one-zero-zero) as equivalent to 4 in the decimal system. Here is a simple way to find binary equivalents. *Each time the two digits 1 and 0 in one position are exhausted (counted as high as they will go), a 1 is added at the left, all digits to the right are made 0, and the count continues.* The reader may apply this simple rule to find next binary numbers.

Notes :

(*i*) Each binary digit (0 or 1) is referred to as a *bit*. A string of four bits is called as a *nibble* and eight bits make a *byte*. Thus, 1001 is a nibble and 10010110 is a binary byte.

(*ii*) The binary number system is the most useful in digital circuits because there are only two digits (0 and 1).

26.4 Place Value

Consider the decimal number 642. This can be expressed as :

$$642 = 600 + 40 + 2$$

Note that in a multidigit decimal number (*i.e.*, 642 in the present case), each position has a value that is 10 times the value of the next position to its immediate right. In other words, every position can be expressed as :

* Note that the procedure is similar to that which was used to write 100 (hundred) in the decimal system.

$$642 = 6 \times 10^2 + 4 \times 10^1 + 2 \times 10^0$$

Thus, we find that values of various positions in a decimal number system are powers of 10 *i.e.*, equal to the number of digits used in the system. This number is called *base* or *radix* of the system. Thus, the decimal system has base of 10 (ten).

For the decimals, the digit to the extreme right is referred to as the *least significant digit* (*LSD*) because its positional value or weight is the lowest. For the decimal number 642, 2 is the *LSD*. The left-most digit in the decimal number is the *most significant digit* (*MSD*) because its positional value or weight is the highest. For the decimal number 642, 6 is the *MSD* with a value of 600.

Binary number system. In the binary number system, only two digits (0 and 1) are used. Therefore, the *base* of this system is 2. In a binary number, each position has a value that is 2 times the value of the next position to its immediate right. In other words, every position can be expressed by 2 raised to some power. We know that binary number 1001 is equal to the decimal number 9. This can be readily shown as under :

$$1001 = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 9$$

For binary numbers, the digit at the extreme right is referred to as *least significant bit* (*LSB*). In the binary number 1001, the 1 at the right is the *LSB*. The left-most digit is called the *most significant bit* (*MSB*). In the binary number 1001, the 1 at the left is the *MSB* with the value of 8 in decimal terms.

26.5 Decimal to Binary Conversion

There are many methods to perform this conversion. The method described here is called *double-dabble* because it requires successive divisions by 2. This method can be summarised as under :

Divide progressively the decimal number by 2 and write down the remainder after each division. Continue this process till you get a quotient of 0 and remainder of 1, the conversion is now complete. The remainders, taken in reverse order, form the binary number [See Fig. 26.4].

Note that 13 is first divided by 2, giving a quotient of 6 with a remainder of 1. This remainder becomes the 2^0 position in the binary number. The 6 is then divided by 2, giving a quotient of 3 with a remainder of 0. This remainder becomes the 2^1 position in the binary number.

Continuing this procedure, the equivalent binary number is 1101.

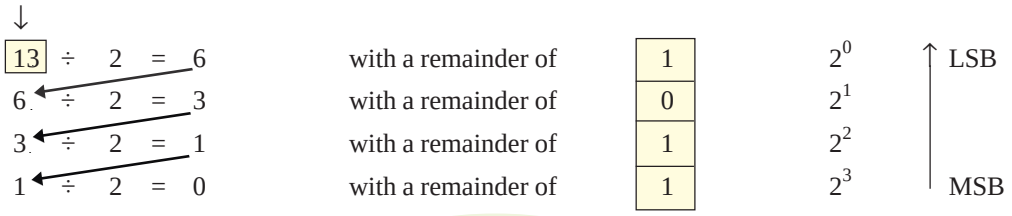


Fig. 26.4

Example 26.1. *Convert the decimal number 37 to its equivalent binary number.*

Solution. Using double-dabble method, we find that the equivalent binary number is 100101. It is a usual practice to mention the base of the number system. The decimal system has a base of 10 while binary system has a base of 2.

$$\therefore \quad (37)_{10} = \mathbf{(100101)_2}$$

2	37
2	18 – 1
2	9 – 0
2	4 – 1
2	2 – 0
2	1 – 0
	0 – 1

Note. This notation avoids the confusion that may arise because decimal number also involves the digits 0 and 1. Thus, $(101)_{10}$ denotes the decimal number hundred one while the binary number $(101)_2$ is equivalent to decimal number 5.

Example 26.2. *Convert the decimal number 23 to its equivalent binary number.*

Solution. Using double-dabble method, we find that the equivalent binary number is 10111.

$\therefore \quad (23)_{10} = \mathbf{(10111)_2}$

Note that binary number 10111 has five bits.

2	23
2	11 – 1
2	5 – 1
2	2 – 1
2	1 – 0
	0 – 1

26.6 Binary to Decimal Conversion

Binary numbers can be converted to equivalent decimal numbers quite easily. Suppose you are given the binary number 110011. Its conversion to equivalent decimal number involves the following two steps :

(i) Place the decimal value of each position of the binary number.

(ii) Add all the decimal values to get the decimal number.

1	1	0	0	1	1
2^5	2^4	2^3	2^2	2^1	2^0

Thus, $(110011)_2 = 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

$= 32 + 16 + 0 + 0 + 2 + 1 = 51$

$\therefore \quad (110011)_2 = (51)_{10}$

Note. In binary to decimal conversion, all positions containing 0 can be ignored. Only add the decimal values of the positions where 1 appears. Thus, in case of the above binary number,

$(110011)_2 = 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^1 + 1 \times 2^0$

$= 32 + 16 + 2 + 1 = 51$

Example 26.3. *Convert the binary number 110001 to its equivalent decimal number.*

Solution. The binary number along with its decimal values of various positions is shown.

1	1	0	0	0	1
2^5	2^4	2^3	2^2	2^1	2^0

$\therefore \quad (110001)_2 = 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^0$

$= 32 + 16 + 1 = 49$

or $(110001)_2 = \mathbf{(49)_{10}}$

26.7 Octal Number System

The octal number system has a radix of *eight* so that it uses eight digits : 0, 1, 2, 3, 4, 5, 6 and 7. The position weights in the system are powers of eight. The digit positions of first six powers of eight are:

$8^0 = 1 \; ; \; 8^1 = 8 \; ; \; 8^2 = 64$

$8^3 = 512 \; ; \; 8^4 = 4096 \; ; \; 8^5 = 32768$

The octal number system is frequently used in digital circuits due to two principal reasons. First, it can be easily converted to binary. Secondly, there are significantly fewer digits in any given octal number than in the corresponding binary number so that it is much easier to work with shorter octal numbers.

1. Decimal-to-Octal Conversion. To convert a decimal number to octal, we employ the same repeated-division method that we used in decimal-to-binary conversion. However, here the division factor is 8 instead of two. The following examples illustrate decimal-to-octal conversion.

(i) To convert decimal number 91 to octal number, the procedure is as under :

Division	*Remainder*	
$91 \div 8 = 11$	3	(LSB)
$11 \div 8 = 1$	3	
$1 \div 8 = 0$	1	(MSB)

$\therefore \quad (91)_{10} = (133)_8$

(ii) As another example, consider the conversion of decimal number 266 to octal number.

Division	*Remainder*	
$266 \div 8 = 33$	2	(LSB)
$33 \div 8 = 4$	1	

$4 \div 8 = 0 \qquad 4 \quad (MSB)$

$\therefore \quad (266)_{10} = (412)_8$

2. Octal-to-Decimal Conversion. An octal-to-decimal conversion can be done in the same manner as a binary-to-decimal conversion *i.e.* simply add up the position weights to obtain the decimal number. The following examples illustrate octal-to-decimal conversion.

(*i*) To convert octal number $(133)_8$ to decimal number, the procedure is as under :

Position weights $8^2 \quad 8^1 \quad 8^0$

Octal number $1 \quad 3 \quad 3$

$\therefore \quad (133)_8 = (8^2 \times 1) + (8^1 \times 3) + (8^0 \times 3)$

$= 64 + 24 + 3 = 91$

$\therefore \quad (133)_8 = (91)_{10}$

(*ii*) As another example, consider the conversion of octal number $(372)_8$ to decimal number.

Position weights $8^2 \quad 8^1 \quad 8^0$

Octal number $3 \quad 7 \quad 2$

$\therefore \quad (372)_8 = (8^2 \times 3) + (8^1 \times 7) + (8^0 \times 2)$

$= 192 + 56 + 2 = 250$

$\therefore \quad (372)_8 = (250)_{10}$

3. Octal-to-Binary Conversion. The advantage of octal number system is the ease with which an octal number can be converted to a binary number and vice-versa. It is because eight is the third power of two, providing a direct correlation between three-bit groups in a binary number and the octal digits *i.e.* each three-bit group of binary bits can be represented by one octal digit. Therefore, conversion from octal to binary is performed by converting *each* octal digit to its 3-bit binary equivalent. The eight possible digits are converted as shown in the adjoining table.

Octal and Binary Equivalents	
Octal Digit	Binary Bits
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

(*i*) The conversion of octal number $(472)_8$ to binary number is done as under :

4	7	2
↓	↓	↓
100	111	010

Therefore, octal 472 is equivalent to binary 100111010 *i.e.*

$(472)_8 = (100111010)_2$

(*ii*) As another example, consider the conversion of octal number $(5431)_8$ to binary number.

5	4	3	1
101	100	011	001

Therefore, octal 5431 is equivalent to binary 101100011001 *i.e.*

$(5431)_8 = (101100011001)_2$

4. Binary-to-Octal Conversion. The conversion of binary number to octal number is simply the reverse of the above process. The bits of the binary number are grouped into groups of *three* bits starting at the LSB. Then each group is converted to its octal equivalent. To illustrate this method, consider the conversion of binary number $(100111010)_2$ to octal number. The procedure is as under:

$$\underbrace{100}_{\downarrow\ 4}\quad\underbrace{111}_{\downarrow\ 7}\quad\underbrace{010}_{\downarrow\ 2}$$

$\therefore \quad (100111010)_2 = (472)_8$

Note that there are fewer digits in the octal number than in the corresponding binary number. Therefore, it is much easier to work with shorter octal numbers.

Sometimes the binary number will not have even groups of 3 bits. In that case, we can add one or two 0s to the left of the MSB of the binary number to fill the last group. This point is illustrated below for the binary number 11010110.

$$\underbrace{011}_{3}\quad\underbrace{010}_{2}\quad\underbrace{110}_{6}$$

Note that a 0 is placed to the left of the MSB to produce even groups of 3 bits.

Example 26.4. *Convert the following decimal numbers to octal equivalent.*

(*i*) *76* (*ii*) *255* (*iii*) *372*

Solution.

(*i*)

Division			*Remainder*	
$76 \div 8$	=	9	4	(LSB)
$9 \div 8$	=	1	1	
$1 \div 8$	=	0	1	(MSB)

$\therefore \quad (76)_{10} = \mathbf{(114)_8}$

(*ii*)

Division			*Remainder*	
$255 \div 8$	=	31	7	(LSB)
$31 \div 8$	=	3	7	
$3 \div 8$	=	0	3	(MSB)

$\therefore \quad (255)_{10} = \mathbf{(377)_8}$

(*iii*)

Division			*Remainder*	
$372 \div 8$	=	46	4	(LSB)
$46 \div 8$	=	5	6	
$5 \div 8$	=	0	5	(MSB)

$\therefore \quad (372)_{10} = \mathbf{(564)_8}$

Example 26.5. *Convert octal number $(24.6)_8$ to the equivalent decimal number.*

Solution.

$$(24.6)_8 = (2 \times 8^1) + (4 \times 8^0) + (6 \times 8^{-1})$$
$$= 16 + 4 + 0.75 = 20.75$$

$\therefore \quad (24.6)_8 = \mathbf{(20.75)_{10}}$

Example 26.6. *Convert $(177)_{10}$ to its 8-bit binary equivalent by first converting to octal.*

Solution. We shall first convert $(177)_{10}$ to its equivalent octal number as under :

Division			*Remainder*	
$177 \div 8$	=	22	1	(LSB)
$22 \div 8$	=	2	6	
$2 \div 8$	=	0	2	(MSB)

$\therefore \quad (177)_{10} = (261)_8$

We now convert the octal number $(261)_8$ to its equivalent binary number as under :

$$\begin{array}{ccc} 2 & 6 & 1 \\ \downarrow & \downarrow & \downarrow \\ 010 & 110 & 001 \end{array}$$

Therefore, the binary equivalent is 010110001. We remove the leading zero to express the result as 8 bits

$$\therefore \quad (177)_{10} = \mathbf{(10110001)_2}$$

26.8 Hexadecimal Number System

The hexadecimal system uses a radix of 16. Therefore, it has 16 possible digit symbols. The first ten digits in the hexadecimal system are represented by the numbers 0 through 9 (0, 1, 2, 3, 4, 5, 6, 7, 8 and 9) and the letters A though F are used to represent the numbers 10, 11, 12, 13, 14 and 15 respectively. The adjoining table shows the relationships among hexadecimal, decimal and binary. Note that each hexadecimal digit represents a group of four binary digits.

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

As is true for binary and decimal numbers, each digit in the hexadecimal system has a positional value or weight. For the right most digit of a hex (abbreviation for hexadecimal) number, the positional weight is 16^0 (= 1), the next digit to the left has a positional weight of 16^1 (= 16) and so on. The positional weight distribution of a hex number system is given below:

	16^3	16^2	16^1	16^0
etc. ←	4096	256	16	1

1. Decimal-to-Hex Conversion. To convert a decimal number to hex number, the technique is the same as used for decimal-to-binary conversion or decimal-to-octal conversion. Recall that we did decimal-to-binary conversion using repeated division by 2 and decimal-to-octal conversion using repeated division by 8. Likewise, decimal-to-hex conversion is done using repeated division by 16. Let us illustrate the decimal-to-hex conversion procedure. Suppose we are to convert the decimal number 423 to hex number.

Division	*Remainder*	
$423 \div 16 = 26$	7	(LSB)
$26 \div 16 = 1$	10	
$1 \div 16 = 0$	1	(MSB)

$$\therefore \quad (423)_{10} = (1\ A7)_{16}$$

Note that 10 is represented by the letter A.

2. Hex-to-Decimal Conversion. In order to convert a hex number to its decimal equivalent, simply add up the position weight of each digit in the hex number. The following example illustrates this conversion.

$$(356)_{16} = (3 \times 16^2) + (5 \times 16^1) + (6 \times 16^0)$$
$$= 768 + 80 + 6 = 854$$
$$\therefore \quad (356)_{16} = (854)_{10}$$

3. Hex-to-Binary Conversion. The conversion from hex to binary is performed by converting each hex digit to its 4-bit binary equivalent (See above table). The following example illustrates this point. Here, we shall convert hex number $(9\ F2)_{16}$ to its binary equivalent.

$$\begin{array}{ccc} 9 & F & 2 \\ \downarrow & \downarrow & \downarrow \\ 1001 & 1111 & 0010 \end{array}$$

$$\therefore \quad (9F2)_{16} = (100111110010)_2$$

4. Binary-to-Hex Conversion. The conversion from binary to hex is just the reverse of the above process. The binary number is grouped into groups of *four* bits and each group is converted to its equivalent hex digit. The following example illustrates this point. Here, we shall convert binary number $(1110100110)_2$ to its *equivalent hex number.

$$\begin{array}{ccc} \underbrace{0011} & \underbrace{1010} & \underbrace{0110} \\ 3 & A & 6 \end{array}$$

$$\therefore \quad (1110100110)_2 = (3A6)_{16}$$

Example 26.7. *Convert decimal number 541 to hexadecimal.*

Solution.

	Division	*Remainder*	
$541 \div 16$	$= 33$	13	(LSB)
$33 \div 16$	$= 2$	1	
$2 \div 16$	$= 0$	2	(MSB)

$$\therefore \quad (541)_{10} = \mathbf{(21D)_{16}}$$

Example 26.8. *Convert decimal number 378 to a 16-bit number by first converting to hexadecimal.*

Solution.

	Division	*Remainder*	
$378 \div 16$	$= 23$	10	(LSB)
$23 \div 16$	$= 1$	7	
$1 \div 16$	$= 0$	1	(MSB)

Thus $(378)_{10} = (17A)_{16}$. We can easily convert this hex number to binary 000101111010. Therefore, we can express $(378)_{10}$ as a 16-bit binary number by adding four leading 0s.

$$(378)_{10} = \mathbf{(0000000101111010)_2}$$

Example 26.9. *Convert $(B2F)_{16}$ to octal.*

Solution. It is easier to first convert hex to binary and then to octal.

$$\begin{aligned} (B2F)_{16} &= 1011 \quad 0010 \quad 1111 && \text{.... conversion to binary} \\ &= 101 \quad 100 \quad 101 \quad 111 && \text{.... 3-bit groupings} \\ &= 5 \quad 4 \quad 5 \quad 7 \end{aligned}$$

$$\therefore \quad (B2F)_{16} = \mathbf{(5457)_8}$$

26.9. Binary-Coded Decimal Code (BCD Code)

Circuits and machines can deal readily with binary numbers, but people are used to working with decimal numbers. Moreover, there are considerably fewer decimal digits required to represent a number than there are binary. It is much easier to remember just a few digits than it is to remember many. Thus whenever there is an interface between digital circuits and people, the interface data usually takes the decimal from. As a result, the digital circuits must utilise some binary code to conveniently represent the decimal numbers. The code used for this purpose is called BCD code. *In a BCD code, each decimal number is represented by a 4-bit binary number.* For example, to convert decimal number $(489)_{10}$ to BCD, the procedure is as under :

* Zeros are added, as needed, to complete 4-bit group.

4 8 9
0100 1000 1001

Note that the highest BCD value that a 4-bit binary number could represent is 9 which would be $(1001)_2$ in binary. Clearly, only the 4-bit binary numbers from 0000 through 1001 are used.

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

The adjoining table shows the BCD code. Each of the decimal digits (0 through 9) is represented by its binary equivalent. Since a decimal digit can be as large as 9, four bits are required to code each decimal digit (the binary code for 9 is 1001).

Note that each decimal digit is assigned a 4-bit binary number even though the binary equivalent may require fewer than four binary places. This way, circuits which use BCD always handle the string of binary bits in four - place groups. When using BCD code, remember that all zeros must be retained, unlike a binary number where leading zeros can be dropped. The BCD code is used when it is necessary to transfer decimal information into and out of a digital machine. Examples of digital machines include the digital clocks, calculators, digital voltmeters and frequency counters.

Example 26.10. *What decimal number is represented by the BCD string given below ?*

0100 0000 0010

Solution. Divide the BCD number into 4-bit groups and convert each to decimal.

$\underbrace{0100}_{4}$ $\underbrace{0000}_{0}$ $\underbrace{0010}_{2}$

Therefore, the equivalent decimal number is $(402)_{10}$.

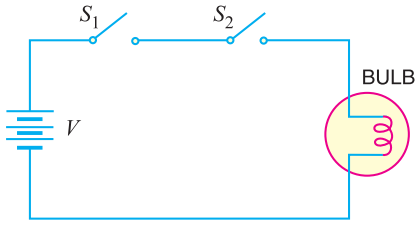
Note. To avoid confusion between BCD and true binary, a BCD string is often separated into groups of 4 binary bits or a subscript BCD is sometimes attached to the string as illustrated under :

0100 0000 0010 or 010000000010_{BCD}

26.10 Logic Gates

A digital circuit with one or more input signals but only one output signal is called a **logic gate**.

Since a logic gate is a switching circuit (*i.e.* a digital circuit), its output can have only one of the two possible states *viz.*, either a high voltage (1) or a low voltage (0) — it is either *ON* or *OFF*. Whether the output voltage of a logic gate is high (1) or low (0) will depend upon the conditions at its input. Fig. 26.5 shows the basic idea of a *logic gate using switches.



(*i*)

S_1	S_2	Bulb
open	open	OFF
open	closed	OFF
closed	open	OFF
closed	closed	ON

(*ii*)

Truth Table

S_1	S_2	Output
0	0	0
0	1	0
1	0	0
1	1	1

(*iii*)

Fig. 26.5

* In itself, the circuit is not actually a logic gate but the logic is similar. The actual gate circuits are made with diodes and transistors. In other words, switches S_1 and S_2 are replaced by diodes or transistors.

(*i*) When S_1 and S_2 are open, the bulb is *OFF*.

(*ii*) When S_1 is open and S_2 closed, the bulb is *OFF*.

(*iii*) When S_2 is open and S_1 closed, the bulb is *OFF*.

(*iv*) When both S_1 and S_2 are closed, the bulb is *ON*.

Note that output (*OFF* or *ON*) depends upon the conditions at the input.

The four possible combinations of switches S_1 and S_2 are shown in the table on the previous page. It is clear that when either of the switches (S_1 or S_2) or both are open, the bulb is *OFF*. In binary language, when either of the inputs or both the inputs are low (0), the output is low. When both switches are closed, the bulb is *ON*. In terms of binary language, when both the inputs are high (1), the output is high. It is usual practice to show the conditions at the input and output of a logic gate in the binary form as shown in the table on the previous page. Such a table is called **truth table**.

The term "logic" is usually used to refer to a decision-making process. A logic gate makes logical decisions regarding the existence of output depending upon the nature of the input. Hence, such circuits are called logic circuits.

26.11 Three Basic Logic Gates

A logic gate is a circuit that has one or more input signals but only one output signal. All logic gates can be analysed by constructing a truth table. A truth table lists all input possibilities and the corresponding output for each input. The three basic logic gates that make up all digital circuits are **(*i*)** OR gate **(*ii*)** AND gate and **(*iii*)** NOT gate. We shall first discuss these three basic logic gates and then the combination of these gates. The following points may be noted about logic *gates :

(*i*) A binary 0 represents 0 V and binary 1 represents + 5V**. It is common to refer to binary 0 as LOW input or output and binary 1 as HIGH input or output.

(*ii*) A logic gate has only one output signal. The output will depend upon the input signal/signals and the type of gate.

(*iii*) The operation of a logic gate may be described either by **truth table** or **Boolean algebra.**

26.12 OR Gate

An OR gate is a logic gate that has two or more inputs but only one output. However, the output *Y* of an OR gate is LOW when *all* inputs are LOW. The output *Y* of an OR gate is HIGH if any or all the inputs are HIGH.

It is called OR gate because the output is high if any or all the inputs are high. For the same reason, an OR gate is sometimes called "any or all gate". For example, consider a 2-input OR gate. The output *Y* will be high if either or both inputs are high.

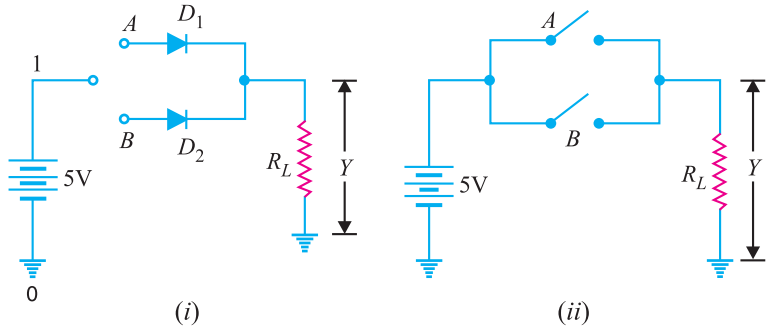
OR gate operation. Fig. 26.6 (*i*) shows one way to build a 2-input OR gate while Fig. †26.6 (*ii*) shows its simplified schematic diagram. The input voltages are labeled as *A* and *B* while the output voltage is *Y*. Note that negative terminal of the battery is grounded and corresponds to 0 state (LOW level). The positive terminal of the battery (+5 V) corresponds to 1 state (HIGH level). There are only four input-output possibilities.

* A *gate* can be regarded as a *barrier* which when closed prevents the passgae of information but if open allows the signal/signals to pass through freely.

** In digital systems, the binary information is represented by two voltage levels, generally +5 V and 0 V. So 5 V is used to represent binary 1 and 0 V is used to represent binary 0.

† As you can see in Fig. 26.6 (*ii*) that output is high when *either* or *both* of the input switches are closed but not when both are open.

(*i*) When both *A* and *B* are connected to ground, both diodes are non-conducting. Hence, the output voltage is ideally zero (low voltage). In terms of binary, when $A = 0$ and $B = 0$, then $Y = 0$ as shown in the truth table in Fig. 26.6 (*iii*).



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

(*iii*)

Fig. 26.6

(*ii*) When *A* is connected to ground and *B* connected to the positive terminal of the battery, diode D_2 is forward biased and diode D_1 is non-conducting. Therefore, diode D_2 conducts and the output voltage is ideally +5 V. In terms of binary, when $A = 0$ and $B = 1$, then $Y = 1$ [See Fig. 26.6 (*iii*)].

(*iii*) When *A* is connected to the positive terminal of the battery and *B* to the ground, diode D_1 is on and diode D_2 is off. Again the output voltage is +5 V. In binary terms, when $A = 1$ and $B = 0$, then $Y = 1$ [See Fig. 26.6 (*iii*)].

(*iv*) When both *A* and *B* are connected to the positive terminal of the battery, both diodes are on. Since the diodes are in parallel, the output voltage is +5 V. In binary terms, when $A = 1$ and $B = 1$, then $Y = 1$ [See Fig. 26.6 (*iii*)].

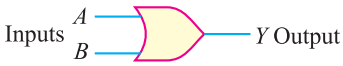


Fig. 26.7

It is clear from the truth table that *for OR gate, the output is high if any or all of the inputs are high. The only way to get a low output is by having all inputs low*. Fig. 26.7 shows the logic symbol of OR gate. Note that the symbol has curved line at the input.

Boolean expression. The algebra used to symbolically describe logic functions is called Boolean algebra. The "+" sign in Boolean algebra refers to the logical OR function. The Boolean expression for OR function is

$$A + B = Y$$
$$\uparrow$$
OR symbol

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

The adjoining table shows possibilities for the inputs. According to this table, when 0 is ORed with 0, the result equals 0. Also, any variable ORed with 1 equals 1. The OR function can be summed up as under :

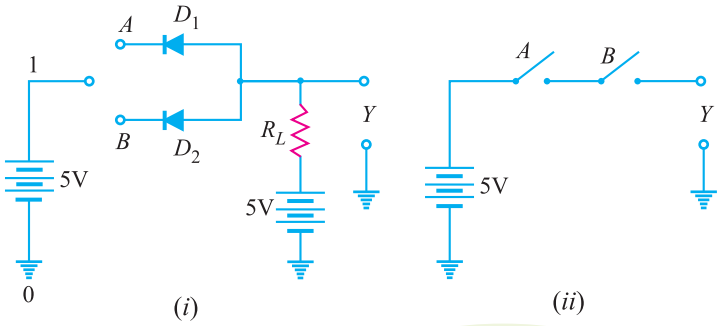
0 ORed with 0 equals 0
0 ORed with 1 equals 1
1 ORed with 1 equals 1

26.13 AND Gate

The AND gate is a logic gate that has two or more inputs but only one output. The output *Y* of AND gate is HIGH when all inputs are HIGH. However, the output *Y* of AND gate is LOW if any or all inputs are LOW.

It is called AND gate because output is HIGH only when all the inputs are HIGH. For this reason, the AND gate is sometimes called "all or nothing gate". For example, consider a 2-input AND gate. The output will be HIGH when both the inputs are HIGH.

AND gate operation. Fig. 26.8 (*i*) shows one way to build a 2-input AND gate while *Fig. 26.8 (*ii*) shows its simplified schematic diagram. There are only four input-output possibilities.



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

(*iii*)

Fig. 26.8

(*i*) When both *A* and *B* are connected to ground, both the diodes (D_1 and D_2) are forward biased and hence they conduct current. Consequently, the two diodes are grounded and output voltage is zero. In terms of binary, when $A = 0$ and $B = 0$, then $Y = 0$ as shown in truth table in Fig. 26.8 (*iii*).

(*ii*) When *A* is connected to the ground and *B* connected to the positive terminal of the battery, diode D_1 is forward biased while diode D_2 will not conduct. Therefore, diode D_1 conducts and is grounded. Again output voltage will be zero. In binary terms, when $A = 0$ and $B = 1$, then $Y = 0$. This fact is shown in the truth table.

(*iii*) When *B* is connected to the ground and *A* connected to the positive terminal of the battery, the roles of diodes are interchanged. Now diode D_2 will conduct while diode D_1 does not conduct. As a result, diode D_2 is grounded and again output voltage is zero. In binary terms, when $A = 1$ and $B = 0$, then $Y = 0$. This fact is indicated in the truth table.

(*iv*) When both *A* and *B* are connected to the positive terminal of the battery, both the diodes do not conduct. Now, the output voltage is +5 V because there is no current through R_L.

It is clear from the truth table that *for AND gate, the output is high if all the inputs are high. However, the output is low if any or all inputs are low.* Fig. 26.9 shows the logic symbol of AND gate. This is the symbol you should memorise and use from now on for AND gates.

Boolean expression. The Boolean expression for AND function is

$$A \,.\, B = Y$$
$$\uparrow$$
AND symbol

where the multiplication **dot stands for the AND operation. The adjoining table shows the possibilities for the inputs. Table tells us that 0 ANDed with any variable equals 0. Also, 1 ANDed with 1 equals one. The AND function can be summed up as under :

0 ANDed with 0 equals 0
0 ANDed with 1 equals 0
1 ANDed with 1 equals 1

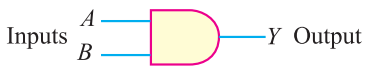


Fig. 26.9

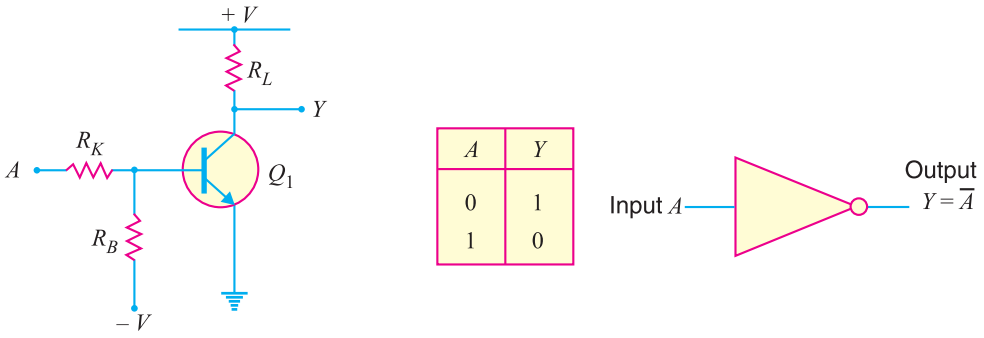
A . B	=	Y
0 . 0	=	0
0 . 1	=	0
1 . 0	=	0
1 . 1	=	1

* Note that two switches used to represent the OR function were connected in parallel. If the switches are connected in series [See Fig. 26.8 (*ii*)], AND function is obtained. The output is high if both the switches are closed. The output will be low if either switch is open.

** Note that the multiplication dot is often omitted, so expression may appear as $AB = Y$.

26.14 NOT Gate or Inverter

The NOT gate or inverter is the simplest of all logic gates. It has only one input and one output, where the output is opposite of the input. The NOT gate is often called inverter because it inverts the input.



(*i*) (*ii*) (*iii*)

Fig. 26.10

Figure 26.10 (*i*) shows a *typical inverter circuit. When A is connected to ground, the base of transistor Q_1 will become negative. This negative potential causes the transistor to cut off and collector current is zero and output is + V volts. In binary terms, when $A = 0$, $Y = 1$. If sufficiently large positive voltage is applied at A, the base of the transistor will become positive, causing the transistor to conduct heavily. Therefore, the output voltage is zero. In binary terms, when $A = 1$, $Y = 0$. Fig. 26.10 (*ii*) shows truth table for an inverter. It is clear from the truth table that whatever the input to the inverter, the output assumes opposite polarity. If the input is 0, the output will be 1 ; if the input is 1, the output will be 0.

Figure 26.10 (*iii*) shows the logic symbol for NOT gate or inverter. Note that small bubble on the inverter symbol represents inversion. The Boolean expression for NOT function is

$$Y = \overline{A}$$

Note that bar above the input A represents inversion.

If $A = 0$, then $Y = \overline{0}$ or $Y = 1$.

If $A = 1$, then $Y = \overline{1}$ or $Y = 0$.

26.15 Combination of Basic Logic Gates

The OR, AND and NOT gates are the three basic circuits that make up all digital circuits. We shall discuss a few combinations of these basic circuits.

(*i*) NAND gate. It is a combination of AND gate and NOT gate. In other words, output of AND gate is connected to the input of a NOT gate as shown in Fig. 26.11 (*i*). Clearly, the output of a NAND gate is opposite to the AND gate. This is illustrated in the truth table for the NAND gate. Note that truth table for NAND gate is developed by *inverting the outputs* of the AND gate.

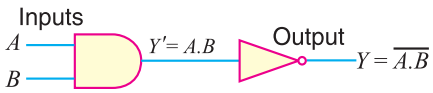
The Boolean expression for NAND function is

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

This Boolean expression can be read as Y = not A . B. To perform the Boolean algebra operation,

* Note that resistors R_K and R_B form a voltage divider between ground and the negative voltage.

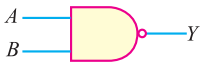
first the inputs must be ANDed and then the inversion is performed. *Note that output from a NAND gate is always 1 except when all of the inputs are 1.* Fig. 26.11 (*iii*) shows the logic symbols for a NAND gate. The little bubble (small circle) on the right end of the symbol means to invert the AND.



(*i*)

Inputs		Output	
A	*B*	AND (*Y′*)	NAND (*Y*)
0	0	0	1
1	0	0	1
0	1	0	1
1	1	1	0

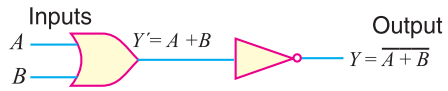
(*ii*)



(*iii*)

Fig. 26.11

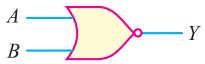
(*ii*) NOR gate. It is a combination of OR gate and NOT gate. In other words, output of OR gate is connected to the input of a NOT gate as shown in Fig. 26.12 (*i*). Note that output of OR gate is inverted to form NOR gate. This is illustrated in the truth table for NOR gate. It is clear that truth table for NOR gate is developed by *inverting the outputs* of the OR gate.



(*i*)

Inputs		Output	
A	*B*	OR (*Y′*)	NOR (*Y*)
0	0	0	1
1	0	1	0
0	1	1	0
1	1	1	0

(*ii*)



(*iii*)

Fig. 26.12

The Boolean expression for NOR function is

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

This Boolean expression can be read as *Y* = not *A* or *B*. To perform the Boolean algebra operation, first the inputs must be ORed and then the inversion is performed. *Note that output from a NOR gate is high (1) only when all the inputs are low (0). If any of the inputs is high (1), the output is low (0).* Fig. 26.12 (*iii*) shows the logic symbol for a NOR gate. The bubble (small circle) at the *Y* output indicates inversion.

26.16 NAND Gate as a *Universal Gate

The NAND gate is universal gate because its repeated use can produce other logic gates. The table below shows how NAND gates can be connected to produce inverter (*i.e.*, NOT gate), AND gate and OR gate.

* It may be noted that NOR gate is also a universal gate.

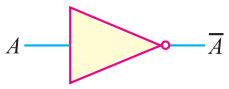
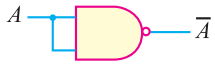
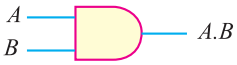
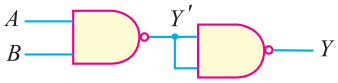
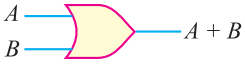
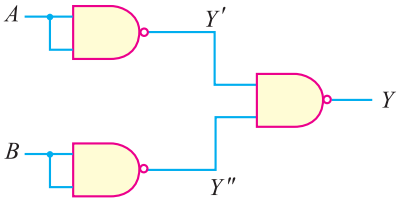
Logic Function	Symbol	Circuit using NAND gates only
Inverter		
AND		
OR		

Fig. 26.13

(*i*) NOT gate from NAND gate. When two inputs of NAND gate are joined together so that it has one input, the resulting circuit is NOT gate. The truth table also shows this fact.

A	*B* (= *A*)	*Y*
0	0	1
1	1	0

(*ii*) AND gate from NAND gates. For this purpose, we use two NAND gates in a manner as shown above. The output of first NAND gate is given to the second NAND gate acting as inverter (*i.e.*, inputs of NAND gate joined). The resulting circuit is the AND gate. The output *Y*′ of first NAND gate (AND gate followed by NOT gate) is inverted output of AND gate. The second NAND gate acting as inverter further inverts it so that the final output *Y* is that of AND gate. The truth table also shows this fact.

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

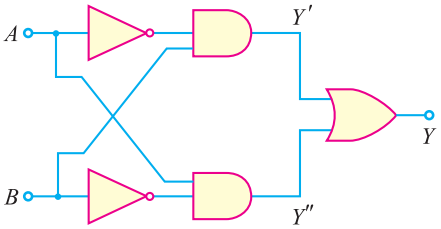
(*iii*) OR gate from NAND gates. For this purpose, we use three NAND gates in a manner as shown above. The first two NAND gates are operated as NOT gates and their outputs are fed to the third. The resulting circuit is OR gate. This fact is also indicated by the truth table.

A	*B*	$Y' = \overline{A}$	$Y'' = \overline{B}$	*Y*
0	0	1	1	0
1	0	0	1	1
0	1	1	0	1
1	1	0	0	1

26.17 Exclusive OR Gate

The name exclusive OR gate is usually shortened to XOR gate. The XOR gate can be obtained by using OR, AND and NOT gates as shown in Fig. 26.14 (*i*).

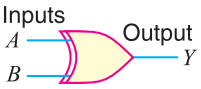
Fig. 26.14 (*ii*) shows the truth table for XOR gate. The table shows that the output is HIGH (1) if any but not all of the inputs are HIGH (1). This *exclusive* feature eliminates the similarity to the OR gate. The OR gate truth table is also given so that you can compare the OR gate truth table with XOR gate truth table. The logic symbol for XOR gate is shown in Fig. 26.14 (*iii*). Note that the symbol is similar to that of OR gate except for the additional curved line at the input side.



(*i*)

Inputs		Output	
A	*B*	OR	XOR
0	0	0	0
1	0	1	1
0	1	1	1
1	1	1	0

(*ii*)



(*iii*)

Fig. 26.14

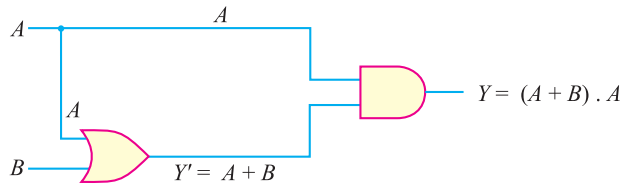
The logic operations in the circuit are as under :

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

Note that 0 ANDed with 1 is 0 and 1 ANDed with 1 is 1.

Example 26.11. *Obtain the truth table for the circuit shown in Fig. 26.15 (i).*

Solution. Figure 26.15 (*ii*) shows the truth table for the circuit. The truth table can be obtained very easily if the reader remembers the following simple Boolean operations :



(*i*)

A	*B*	*Y′* = *A* + *B*	*Y* = *Y′* . *A*
0	0	0	0
1	0	1	1
0	1	1	0
1	1	1	1

(*ii*)

Fig. 26.15

(*i*) *0 *ORed with 0 = 0 ; 1 ORed with 1 = 1 ; 1 ORed with 0 = 1*

(*ii*) *0 **ANDed with 0 = 0 ; 0 ANDed with 1 = 0 ; 1 ANDed with 1 = 1*

Thus, when *A* = 0 and *B* = 0, then *A* ORed with *B* = 0 *i.e.*, *Y′* = 0. When *Y′* (= 0) is ANDed with *A* (= 0), the result is 0. Again when *A* = 1 and *B* = 0, then *A* ORed with *B* is 1 *i.e.*, *Y′* = 1. Now *Y′* (= 1) ANDed with *A* (= 1), the result is 1.

Example 26.12. *Obtain the truth table for the circuit shown in Fig. 26.16.*

* Note that *A* + *B* means *A* ORed with *B*.

** Note that *A* . *B* means *A* ANDed with *B*.

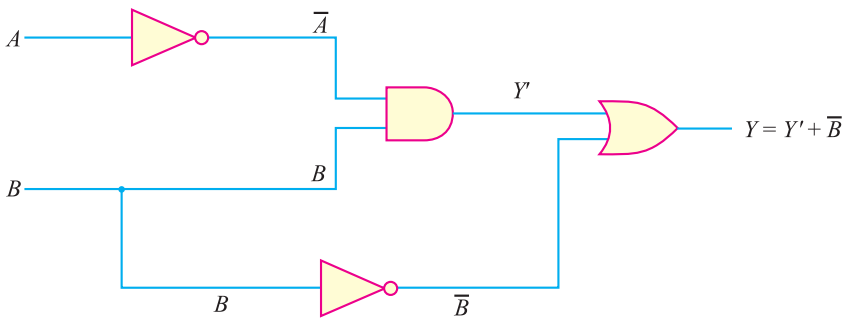


Fig. 26.16

Solution. The truth table for the circuit is shown below :

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

(i) When $A = 0$ and $B = 0$, then $\overline{A} = 1$. Now Y' is equal to $\overline{A}$ (= 1) ANDed with B (= 0). The result is 0. Then Y' (= 0) ORed with $\overline{B}$ (= 1) is 1 *i.e.*, $Y = 1$.

(ii) When $A = 1$ and $B = 0$, then $\overline{A} = 0$. Now Y' is equal to $\overline{A}$ (= 0) ANDed with B (= 0) and the result is 0 *i.e.*, $Y' = 0$. Then Y' (= 0) ORed with $\overline{B}$ (= 1) is 1 *i.e.*, $Y = 1$.

The reader can proceed in a similar way to find the other output values.

26.18 Encoders and Decoders

A digital circuit can process numbers in binary form. However, most of the information we handle is in decimal form. Therefore, a digital machine must perform the following functions :

(i) Convert the information from decimal to digital (binary) form.

(ii) Process the digital information.

(iii) Convert the digital output back to decimal form.

The circuit that converts decimal form to digital (binary) form is called **encoder** and the circuit that converts digital form to decimal form is called **decoder.** Fig. 26.17 shows encoding and decoding in a digital calculator. Here the input is the decimal number 5 punched in at the keyboard. The encoder changes the decimal number 5 to the digital form as the binary digit 0101. The central processing unit (CPU) contains digital logic circuits for necessary calculations. Here all operations are carried out in binary form. The output of

Encoders and Decoders

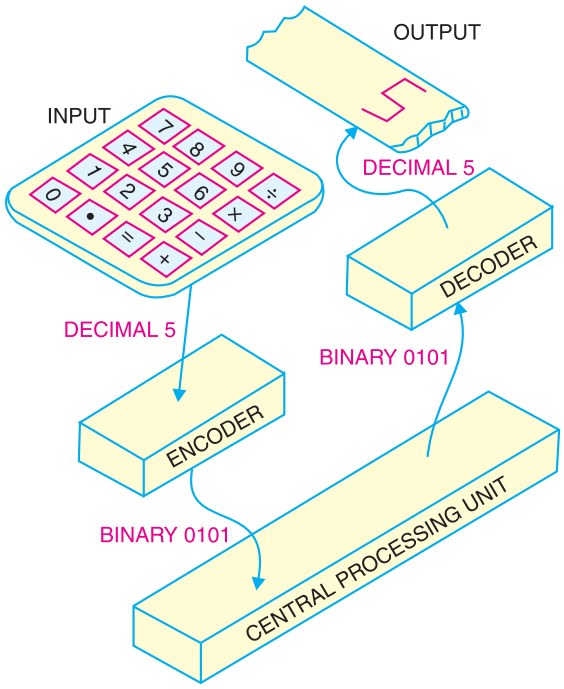


Fig. 26.17

CPU is fed to the decoder which changes the binary signal back to the decimal form. The output display is in the decimal form, showing the original number 5.

26.19 Advantages and Disadvantages of Digital Electronics

The world of electronics can be classified as either digital or analog circuits. An increasing majority of applications in electronics use digital techniques to perform operations that were once performed using analog methods. It is worthwhile to give advantages and disadvantages of digital electronics.

Advantages. The chief reasons for the shift to digital technology are :

(*i*) *Digital systems are generally easier to design.* It is because the circuits that are used are *switching circuits* where *exact* values of voltages or currents are not important, only the range (HIGH or LOW) in which they fall is important.

(*ii*) *Digital circuits provide greater accuracy and precision.* It is because digital circuits can handle as many digits of precision as you need simply by adding more switching circuits. In analog systems, precision is usually limited to three or four digits because the values of voltage and current are directly dependent on the circuit components.

(*iii*) *Digital circuits are less affected by noise.* Suprious fluctuations in voltage (noise) are not as critical in digital systems as in analog systems. It is because in a digital circuit, the exact value of a voltage is not important as long as the noise is not large enough to prevent us from distinguishing a HIGH from a LOW.

(*iv*) *More digital circuitry can be fabricated on IC chips.* Analog system uses such devices (high-value capacitors, inductors, transformers) that cannot be economically integrated. For this reason, analog systems cannot achieve the same degree of integration as digital circuits.

(*v*) Information storage is easy with digital circuits.

Disadvantages. (*i*) The real world is mainly analog. However, the digital circuits can handle only digital signals. This necessitates encoders and decoders which increase the cost of the equipment.

(*ii*) There are situations where using only analog techniques is simpler and more economical. For example, the process of signal amplification is most easily accomplished using analog circuitry.

However, advantages of digital techniques outweigh the disadvantages. For this reason, we are fast switching to digital techniques.

26.20 Boolean Algebra

Digital circuits perform the binary arithmetic operations with binary digits 1 and 0. These operations are called logic functions or logical operations. *The algebra used to symbolically describe logic functions is called* **Boolean algebra.** Boolean algebra is a set of rules and theorems by which logical operations can be expressed symbolically in equation form and be manipulated mathematically. As with the ordinary algebra, the *letters of alphabet (*e.g. A, B, C* etc.) can be used to represent the variables. Boolean algebra differs from ordinary algebra in that Boolean constants and variables can have only two values ; 0 and 1. There are four connecting symbols used in Boolean algebra *viz.*

(*i*) equals sign (=) **(*ii*)** plus sign (+)

(*iii*) multiply sign (·) **(*iv*)** bar (–)

(*i*) Equals sign (=). The equals sign in Boolean algebra refers to the standard mathematical equality. In other words, the logical value on one side of the sign is identical to the logical value on the other side of the sign. Suppose we are given two logical variables such that $A = B$. Then if $A = 1$, then $B = 1$ and if $A = 0$, then $B = 0$.

(*ii*) Plus sign (+). The plus sign in Boolean algebra refers to the logical *OR operation.* Thus, when the statement $A + B = 1$ appears in Boolean algebra, it means A ORed with B equals 1. Consequently, either $A = 1$ or $B = 1$ or both equal 1.

(*iii*) Multiply sign (·). The multiply sign in Boolean algebra refers to *AND operation.* Thus, when the statement $A \cdot B = 1$ appears in Boolean algebra, it means A ANDed with B equals 1. Consequently, $A = 1$ and $B = 1$. The function $A \cdot B$ is often written as AB, omitting the dot for convenience.

(*iv*) Bar sign (–). The bar sign in Boolean algebra refers to *NOT operation.* The NOT has the effect of inverting (complementing) the logical value. Thus, if $A = 1$, then $\overline{A} = 0$.

26.21 Boolean Theorems

We now discuss the basic Boolean theorems that are useful in manipulating and simplifying Boolean expressions. For convenience, we divide the theorems into two groups :

(*i*) Single variable theorems

(*ii*) Multivariable theorems

(*i*) Single variable theorems. These theorems refer to the condition when only one input to the logic gate is variable. Table 26.1 gives single variable Boolean theorems.

* For example, A might represent a certain digital circuit input or output and at any time, we must have either $A = 0$ or $A = 1$.

Table 26.1

Theorem 1 :	$A + 0 = A$
Theorem 2 :	$A \cdot 1 = A$
Theorem 3 :	$A + \overline{A} = 1$
Theorem 4 :	$A \cdot \overline{A} = 0$
Theorem 5 :	$A + A = A$
Theorem 6 :	$A \cdot A = A$
Theorem 7 :	$A + 1 = 1$
Theorem 8 :	$A \cdot 0 = 0$
Theorem 9 :	$\overline{\overline{A}} = A$

Theorem 1. $(A + 0 = A)$. This theorem can be verified by ORing a variable A with a 0 and is illustrated in Fig. 26.18. Here one input to OR gate is always 0 and the other input A can be a value 1 or 0. When A is at 1, the output is 1 which is equal to A. When A is at 0, the output is 0 which is also equal to A (= 0). Therefore, a variable ORed with 0 is equal to the value of the variable. This is easy to remember since 0 added to anything does not effect the value of the variable, either in regular addition or OR addition.

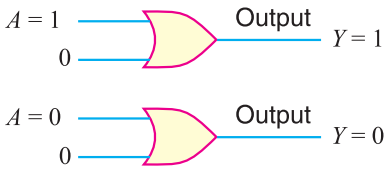


Fig. 26.18

Theorem 2. $(A \,.\, 1 = A)$. This theorem can be verified by ANDing a variable A with a 1 and is illustrated in Fig. 26.19. Here one input to AND gate is always 1 and the other can be a value 1 or 0. If A is 1, the output of the AND gate is 1 because both the inputs are now 1's. If A is 0, the output of the AND gate is a 0. Therefore, a variable ANDed with a 1 is equal to the value of the variable $(A \cdot 1 = A)$. This is easy to remember because AND operation is just like ordinary multiplication.

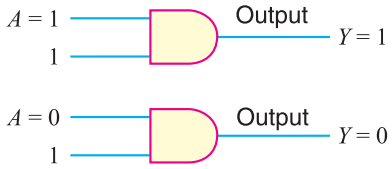


Fig. 26.19

Theorem 3. $\left(A + \overline{A} = 1\right)$. This theorem can be easily explained. If a variable A and its complement $(\overline{A})$ are ORed, the result is always 1. If A is a 0, then $0 + \overline{0} = 0 + 1 = 1$. If A is a 1, then $1 + \overline{1} = 1 + 0 = 1$. Fig. 26.20 illustrates this theorem.

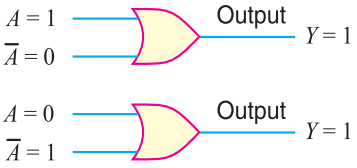


Fig. 26.20

Theorem 4. $\left(A \cdot \overline{A} = 0\right)$. This theorem states that if a variable A is ANDed with its complement, the result is zero. This is readily apparent because either A or $\overline{A}$ will always be 0. Therefore, when one of the inputs to an AND gate is 0, the output is always 0. Fig. 26.21 illustrates this theorem.

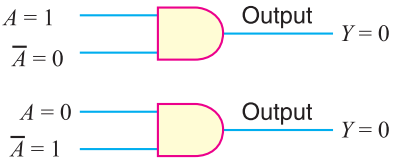


Fig. 26.21

Theorem 5. $\left(A + A = A\right)$. This theorem states that when a variable A is ORed with itself, the output is equal to the variable. Thus, if A is a 0, then 0 + 0 = 0 and

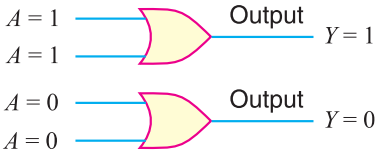


Fig. 26.22

if A is a 1, then 1 + 1 = 1. Fig. 26.22 illustrates this theorem.

Theorem 6. ($A \cdot A = A$). This theorem states that if a variable A is ANDed with itself, the result is equal to the variable. For example, if $A = 0$, then $0 \cdot 0 = 0$ and if $A = 1$, then $1 \cdot 1 = 1$. For either case, the output of an AND gate is equal to the value of the input variable A. Fig. 26.23 illustrates this theorem.

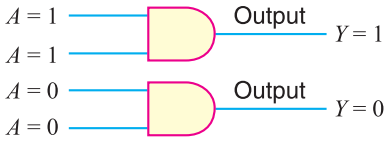


Fig. 26.23

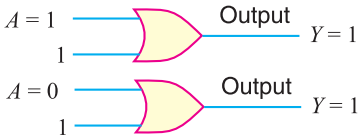


Fig. 26.24

Theorem 7. ($A + 1 = 1$). This theorem states that when a variable A is ORed with 1, the outuput is always equal to 1. Fig. 26.24 illustrates this theorem. One input to an OR gate is always 1 and the other input A can be either 1 or 0. Now 1 on an input to OR gate produces 1 on the output regardless of the value of the variable on the other input.

Theorem 8. ($A \cdot 0 = 0$). This theorem states that variable A ANDed with 0 always produces 0. Recall that any time one input to an AND gate is 0, the output is 0 regardless of the value of the variable A on the other input. This theorem is illustrated in Fig. 26.25.

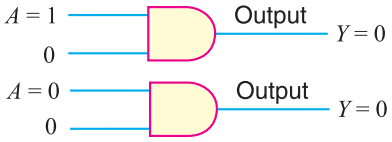


Fig. 26.25

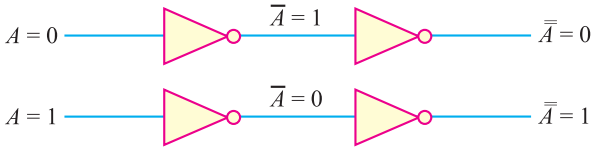


Fig. 26.26

Theorem 9. $\left(\bar{\bar{A}} = A\right)$. This theorem states that if a variable A is complemented twice, the result is the variable itself. Starting with A and inverting (complementing) it once gives $\bar{A}$. Inverting it once more gives A — the original value. This theorem is illustrated in Fig. 26.26.

Duality Principle. Before moving to multivariable theorems, this would be the right place to mention an important property of Boolean algebra called *duality principle*. It is stated below :

The duality principle states that a Boolean expression remains valid if operators OR and AND are interchanged and 1's and 0's in the expression are also interchanged.

In order to understand this principle, consider the Boolean Theorem 1 *viz.*

$$A + 0 = A$$

According to duality principle, this Boolean expression remains valid if OR function is replaced by AND function and 0 by 1. In that case, the Boolean expression becomes :

$$A \cdot 1 = A$$

Note that this is Boolean Theorem No. 2. Therefore, Boolean Theorem 2 is dual of Boolean Theorem 1 and *vice-versa*. Applying duality principle, Theorem 4 is dual of Theorem 3 and *vice-versa*, Theorem 6 is dual of Theorem 5 and *vice-versa*, Theorem 8 is dual of Theorem 7 and *vice-versa*. To apply duality principle to a Boolean expression, we simply interchange OR and AND operator and replace 1's by 0's and 0's by 1's.

(*ii*) Multivariable theorems. These theorems refer to the condition when more than one input to the logic gate are variable. Table 26.2 gives multivariable Boolean theorems.

Table 26.2

Theorem 10	:	$A + B = B + A$	*Commutative Law*
Theorem 11	:	$A \cdot B = B \cdot A$	
Theorem 12	:	$A + (B + C) = (A + B) + C$	*Associative Law*
Theorem 13	:	$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	
Theorem 14	:	$A \cdot (B + C) = A \cdot B + A \cdot C$	*Distributive Law*
Theorem 15	:	$(A + B) \cdot (C + D) = A \cdot C + B \cdot C + A \cdot D + B \cdot D$	
Theorem 16	:	$A + A \cdot B = A$	
Theorem 17	:	$\overline{(A+B)} = \overline{A} \cdot \overline{B}$	*De Morgan's Theorems*
Theorem 18	:	$\overline{(A \cdot B)} = \overline{A} + \overline{B}$	

The following points may be noted about these theorems :

(*a*) Theorems 10 and 11 obey **commutative law**. *This law states that the order in which the variables are ORed or ANDed makes no difference.*

Fig. 26.27

Fig. 26.28

Figure 26.27 illustrates the commutative law as applied to the OR gate while Fig. 26.28 illustrates the commutative law as applied to an AND gate.

(*b*) Theorems 12 and 13 obey **associative law.** *This law states that in the ORing or ANDing of several variables, the result is the same regardless of the grouping of the variables.*

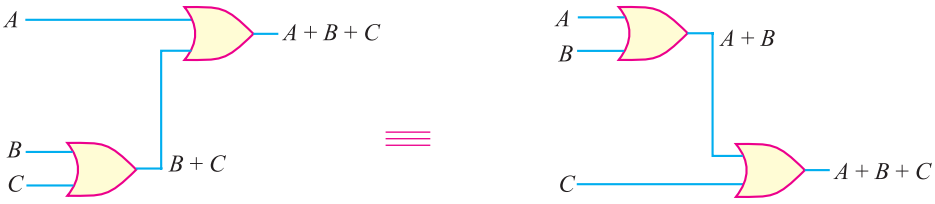


Fig. 26.29

Fig. 26.30

Figure 26.29 illustrates the associative law as applied to the OR gate, while Fig. 26.30 illustrates the associative law as applied to an AND gate.

(*c*) Theorems 14 and 15 obey **distributive law**. *This law states that a Boolean expression can be expanded by multiplying term-by-term just the same as in ordinary algebra.*

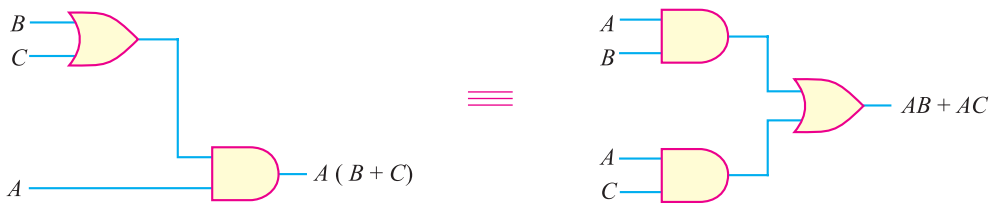


Fig. 26.31

Fig. 26.31 illustrates the distributive law in terms of gate implementation.

(*d*) We will prove Theorem 16 by factoring and using Theorems 2, 7, 10 and 14.

$$\begin{aligned} A + A \cdot B &= A \cdot 1 + A \cdot B && \text{...Theorem 2} \\ &= A \cdot (1 + B) && \text{...Theorem 14} \\ &= A \cdot (B + 1) && \text{...Theorem 10} \\ &= A \cdot 1 && \text{...Theorem 7} \\ &= A && \text{...Theorem 2} \end{aligned}$$

(*e*) Theorems 17 and 18 are the two most important theorems of Boolean algebra and were contributed by the great mathematician named *De Morgan*. Therefore, these theorems are called De Morgan's theorems.

26.22 De Morgan's Theorems

De Morgan's theorems are extremely useful in simplifying expressions in which a product or sum of variables is inverted. The two theorems are :

(*i*) $\overline{(A+B)} = \overline{A} \cdot \overline{B}$

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

(*i*) The first De Morgan's theorem may be stated as under :

When the OR sum of two variables is inverted, this is equal to inverting each variable individually and then ANDing these inverted variables i.e.,

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

In this expression, *A* and *B* are the two variables. The L.H.S. is the complement of the OR sum of the two variables. The R.H.S. is the AND product of individual inverted variables.

(*ii*) The second De Morgan's theorem may be stated as under :

When the AND product of two variables is inverted, this is equal to inverting each variable individually and then ORing them i.e.,

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

In this expression, *A* and *B* are the two variables. The L.H.S. is the complement of the AND product of the two variables. The R.H.S. is the OR sum of the individual inverted variables.

26.23 Operator Precedence

The operator precedence for evaluating Boolean expression is (*i*) parenthesis (*ii*) NOT (*iii*) AND and (*iv*) OR. In other words, the expression inside the parenthesis must be evaluated before all other operations. The next operation that holds precedence is the complement, then follows the AND and finally the OR. For example, consider the Boolean expression :

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

The sequence of operations will be :

(*i*) The expression inside the parenthesis (*i.e.* $C + D$) will be evaluated first.

(*ii*) Then $\overline{B}$ will be evaluated.

(*iii*) Then the results of the two (*i.e.* $\overline{B}$ and $C + D$) will be ANDed.

(*iv*) Finally, the result of the product will be ORed with A.

Example 26.13. *Using Boolean algebraic techniques, simplify the following expression* :

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

Solution. $$Y = A \cdot B \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot B \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot B \cdot C \cdot \overline{D} + A \cdot B \cdot C \cdot \overline{D} \quad ...(i)$$

Step 1 : Take out the common factors as below :

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

Step 2 : Apply Theorem 3 $(A + \overline{A} = 1)$:

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

Step 3 : Again factorise :

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

Step 4 : Apply Theorem 3 $(C + \overline{C} = 1)$:

$$Y = B\,\overline{D} \cdot 1 = B\overline{D}$$

This is the simplified form of exp. (*i*).

Example 26.14. *Using Boolean techniques, simplify the following expression* :

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

Solution. $$Y = AB + A\,(B + C) + B\,(B + C) \quad ...(i)$$

Step 1 : Apply Theorem 14 (distributive law) to second and third terms:

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

Step 2 : Apply Theorem 6 $(B \cdot B = B)$:

$$Y = AB + AB + AC + B + BC$$

Step 3 : Apply Theorem 5 $(AB + AB = AB)$:

$$Y = AB + AC + B + BC$$

Step 4 : Factor B out of last 2 terms :

$$Y = AB + AC + B\,(1 + C)$$

Step 5 : Apply commutative law and Theorem 7 $(1 + C = C + 1 = 1)$:

$$Y = AB + AC + B \cdot 1$$

Step 6 : Apply Theorem 2 $(B \cdot 1 = B)$:

$$Y = AB + AC + B$$

Step 7 : Factor B out of first and third terms :

$$Y = B\,(A + 1) + AC$$

Step 8 : Apply Theorem 7 $(A + 1 = 1)$:

$$Y = B \cdot 1 + AC$$

Step 9 : Apply Theorem 2 $(B \cdot 1 = B)$:

$$Y = B + AC$$

This is the simplified form of exp. (*i*).

Example 26.15. *Simplify the following Boolean expressions to a minimum number of literals :*

(*i*) $Y = A + \overline{A}B$ (*ii*) $Y = AB + \overline{A}C + BC$

Solution. (*i*)

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

$$= A + AB + \overline{A}B \quad [\because A = A + AB \text{ from Theorem 16}]$$

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

$$= A + B \quad [\because A + \overline{A} = 1 \text{ from Theorem 3}]$$

$$\therefore \quad Y = A + B$$

(*ii*)

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

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

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

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

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

$$\therefore \quad Y = AB + \overline{A}C$$

Example 26.16. *Determine output expression for the circuit shown below and simplify it using De Morgan's theorem.*

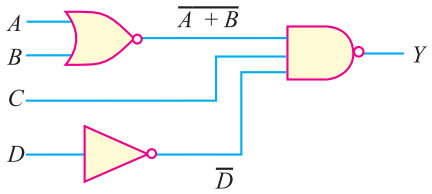


Fig. 26.32

Solution. The output expression for the circuit shown above is :

$$Y = \overline{[(\overline{A + B}) \cdot C \cdot \overline{D}]}$$

Using De Morgan's theorem :

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

$$\therefore \quad Y = A + B + \overline{C} + D$$

Example 26.17. *Find the complement of the expressions given below :*

(*i*) $Y = AB\overline{C} + A\overline{B}\overline{C}$

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

Solution. (*i*)

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

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

Applying De Morgan's theorem :

$$\overline{Y} = \overline{(AB\overline{C})} \cdot \overline{(A\overline{B}\overline{C})}$$

Again applying De Morgan's theorem to the each expression inside the brackets :

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

(*ii*)

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

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

Applying De Morgan's theorem :

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

Again applying De Morgan's theorem to the expression inside the bracket :

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

Applying De Morgan's theorem for the third time we get :

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

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

Example 26.18. *Simplify the following Boolean expressions* :

(*i*) $Y = (A + B + C) \cdot (A + B)$

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

(*iii*) $Y = 1 + A\,(B \cdot \overline{C} + BC + \overline{B}\,\overline{C}) + A\overline{B}C + AC$

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

Solution. (*i*)

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

Using $A \cdot A = A$, we get,

$$Y = A + AB + AB + B + AC + BC$$
$$= A + AB + B + AC + BC \qquad [\because AB + AB = AB]$$
$$= A + B + AC + BC \qquad [\because A + AB = A]$$
$$= A\,(1 + C) + B\,(1 + C)$$
$$= A \cdot 1 + B \cdot 1 \qquad [\because 1 + C = 1]$$

$\therefore$ $$Y = A + B$$

(*ii*)

$$Y = AB + ABC + A\overline{B}\overline{C}$$
$$= AB + AB\,(C + \overline{C})$$
$$= AB + AB \qquad [\because C + \overline{C} = 1]$$

$\therefore$ $$Y = AB$$

(*iii*)

$$Y = 1 + A\,(B \cdot \overline{C} + BC + \overline{B}\,\overline{C}) + A\overline{B}C + AC$$

Using $1 + A = 1$, we get,

$$Y = 1 + A\overline{B}C + AC \qquad [\because 1 + A\,(B\overline{C} + BC + \overline{B}\overline{C}) = 1]$$
$$= 1 + AC$$

$\therefore$ $$Y = 1$$

Thus, because of the first term Y reduces to 1. Therefore, any Boolean expression ORed with 1, results in 1.

(*iv*)

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

Applying De Morgan's theorem :

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

Again applying De Morgan's theorem :

$$Y = (\overline{A} \cdot B \cdot \overline{C}) \cdot (\overline{B} \cdot C) = 0 \qquad [\because B \cdot \overline{B} = 0,\ C \cdot \overline{C} = 0]$$

Example 26.19. *Simplify the following Boolean expression* :

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

Solution. $Y = A\overline{B}D + A\overline{B}\,\overline{D}$

Factoring out the common variables $A\overline{B}$ (using Theorem 14), we get ,

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

Using Theorem 3, $D + \overline{D} = 1$.

$$\therefore \quad Y = A\overline{B} \cdot 1$$

Using Theorem 2, we get,

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

Example 26.20. *Simplify the following Boolean expression* :

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

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

The expression can be expanded by multiplying out the terms [Theorem 15].

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

Using Theorem 4, $\overline{A} \cdot A = 0$. Also $B \,.\, B = B$ [Theorem 6].

$$\therefore \quad Y = 0 + \overline{A} \cdot B + B \cdot A + B$$

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

Factoring out the variable B [Theorem 14], we have,

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

Using Theorem 7, $A + 1 = 1$.

$$\therefore \quad Y = B(\overline{A} + 1)$$

Again using Theorem 7, $\overline{A} + 1 = 1$.

$$\therefore \quad Y = B \cdot 1$$

Finally, using Theorem 2, we have,

$$Y = B$$

26.24 Combinational Logic Circuits

We can combine two or more logic gates to form a logic circuit or digital circuit. When the resulting logic or digital circuit has no feedback and no memory, it is often called combinational logic circuit.

A logic circuit consisting of two or more logic gates that has no feedback and no memory is called a ***combinational logic circuit.***

A combinational logic circuit is constructed using OR, AND and NOT gates. Therefore, the basic building block for combinational circuits is the logic gate. Since a combinational logic circuit has no feedback and no memory, its output depends *only* on the current value of its inputs.

26.25 Boolean Expressions for Combinational Logic Circuits

A combinational logic circuit (or digital circuit) often consists of several different logic gates, interconnected in such a way as to perform a specific logic function. By using the laws, theorems and techniques of Boolean algebra, we can find the Boolean expression for any combinational logic circuit. Let us find Boolean expression for some logic circuits.

(*i*) Fig. 26.33 shows a logic circuit. It consists of two AND gates and one OR gate. Each of the three gates has two input variables. Because gate G_1 is an AND gate and its two inputs are A and B, its output is AB. The gate G_2 is also an AND gate and its two inputs are C and D so that its output is CD. The gate G_3 is an OR gate so that its output is the ORing of AB and CD, producing $AB + CD$. Therefore, the Boolean expression for the output of this logic circuit is

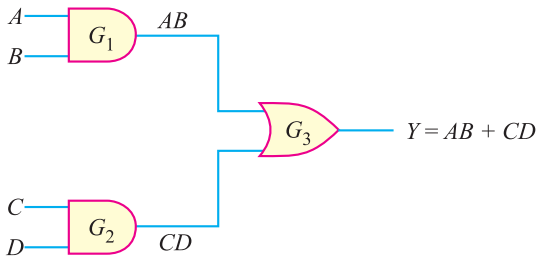


Fig. 26.33

$$Y = AB + CD$$

(*ii*) Fig. 26.34 shows a logic circuit consisting of two AND gates and one OR gate. There are four (A, B, C and D) input variables.

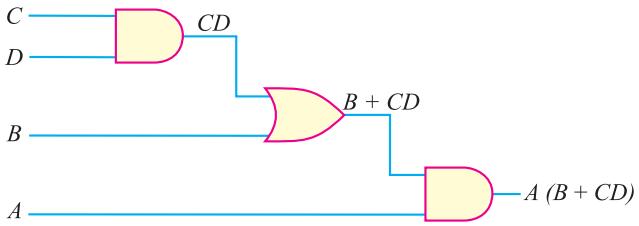


Fig. 26.34

The variable C is ANDed with D and its output is CD. Then CD is ORed with B, giving an output $(B + CD)$. This sum is then ANDed with A, resulting in the following Boolean expression :

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

(*iii*) Fig. 26.35 shows a logic circuit consisting of three AND gates and one OR gate. There are six input variables *viz*. A, B, C, D, E and F.

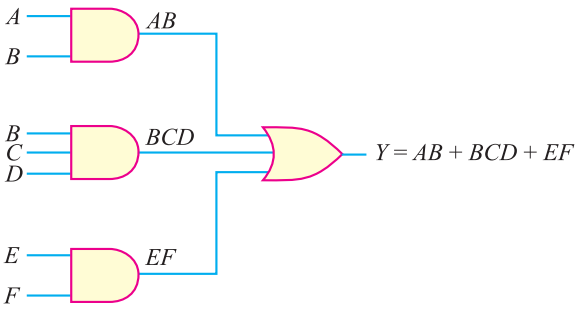


Fig. 26.35

The variables A and B are ANDed to produce AB, B, C and D are ANDed to produce BCD and E and F are ANDed to produce EF. The input to OR gate is AB, BCD and EF so the output of this gate is the ORing of the inputs *i.e.*

$$Y = AB + BCD + EF$$

26.26 AND and OR Operations in Boolean Expression

It is very important to interpret AND and OR operations in the Boolean expression. Sometimes, there may be confusion as to which operation (AND or OR) in a Boolean expression is to be performed

first. For example, the Boolean expression *$A \cdot B + C$ can be interpreted in two different ways *viz.*

(*i*) $A \cdot B$ is ORed with C *or* **(*ii*)** A is ANDed with the term $B + C$

In order to avoid this confusion, it will be understood that if Boolean expression contains both AND and OR operations, the AND operations are performed first unless there are brackets in the expression in which case the operation inside the brackets is to be performed first. This is the same rule that is used in ordinary algebra to determine the order of operations. Let us illustrate this important point with examples.

(*i*) Consider the Boolean expression: $Y = A \cdot B + C$. Since there are no brackets in this expression, AND operation is to be performed first. In other words, first A and B are ANDed to produce $A \cdot B$ and then $A \cdot B$ is ORed with C to give the final function: $Y = A \cdot B + C$. The logic circuit representing this Boolean expression is shown in Fig. 26.36.

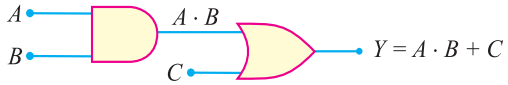


Fig. 26.36

(*ii*) Consider the Boolean expression: $Y = (A + B) \cdot C$. Since $A + B$ is in the bracket, the OR operation is to be performed first. In other words, A is ORed with B to give $(A + B)$ and then $(A + B)$ is ANDed with C to produce $Y = (A + B) \cdot C$. The logic circuit representing this Boolean expression is shown in Fig. 26.37.

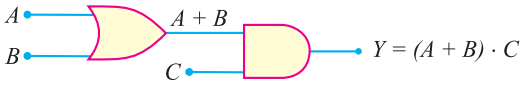


Fig. 26.37

Example 26.21. *Write the Boolean expression for the digital circuit shown in Fig. 26.38.*

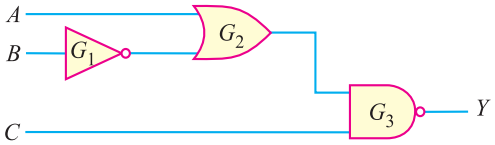


Fig. 26.38

Solution. The output of G_2 is $A + \bar{B}$. This output of G_2 is NANDed with C to yield $Y = \overline{(A + \bar{B})\, C}$. Note that bracket around $(A + \bar{B})$ is required otherwise the expression may be interpreted that A is NORed with $\bar{B}\, C$.

Example 26.22. *Write the Boolean expression for the logic circuit (digital circuit) shown in Fig. 26.39.*

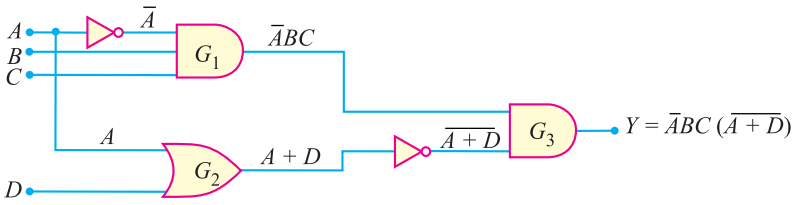


Fig. 26.39

Solution. There are two AND gates (viz. G_1 and G_3) and one OR gate (G_2). Also the circuit has two inverters. The input A is fed to G_1 through the inverter. Therefore, $\bar{A}$, B and C are ANDed by G_1 to produce an output of $\bar{A}BC$. The input A is ORed with D by G_2 to produce an output of $(A + D)$. The inputs $\bar{A}BC$ and $\overline{(A + D)}$ to G_3 are ANDed to produce the final output :

$$Y = \bar{A}BC\ \overline{(A + D)}$$

* Recall that it indicates A ANDed with B and can be expressed as $A{\cdot}B$ or AB.

Example 26.23. *Write the Boolean expression for the digital circuit shown in Fig. 26.40.*

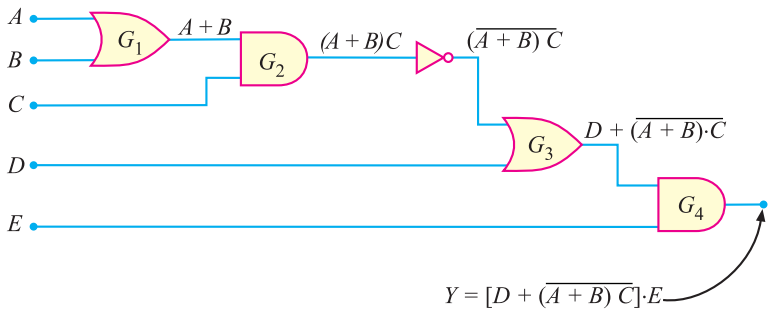


Fig. 26.40

Solution. The output of G_1 is $A + B$. Then $(A + B)$ and C are ANDed to produce an output of $(A + B)\,C$. It is easy to see $\overline{(A + B)\,C}$ and D are ORed by G_3 to produce an output of $D + \overline{(A + B) \cdot C}$. Finally, $D + \overline{(A + B) \cdot C}$ and E are ANDed by G_4 to produce the final output Y given by ;

$$Y = [D + \overline{(A + B)\,C}\,] \cdot E$$

Example 26.24. *Illustrate the commutative law of (i) addition and (ii) multiplication of two variables.*

Solution.

(*i*) The commutative law of addition states that the order in which the variables are ORed makes no difference. This law for the addition of two variables A and B can be written as :

$$A + B = B + A$$

Fig. 26.41

Fig. 26.41 illustrates the commutative law as applied to the OR gate.

(*ii*) The commutative law of multiplication states that the order in which the variables are ANDed makes no difference. This law for the multiplication of two variables A and B can be written as :

$$AB = BA$$

Fig. 26.42

Fig. 26.42 illustrates the commutative law as applied to the AND gate.

Example 26.25. *Illustrate associative law of (i) addition and (ii) multiplication as applied to Boolean algebra.*

Solution.

(*i*) The associative law of addition states that in the ORing of several variables, the result is the same regardless of the grouping of the variables. This law for the addition of the three variables A, B and C can be written as:

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

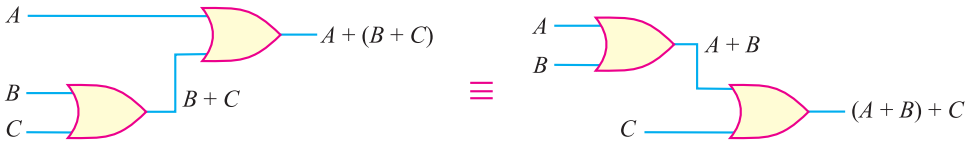


Fig. 26.43

Fig. 26.43. illustrates this law as applied to OR gates.

(ii) The associative law of multiplication states that it makes no difference in what order the variables are grouped when ANDing several variables. This law for the multiplication of the three variables *A*, *B* and *C* can be written as :

$$A\,(BC) = (AB)C$$

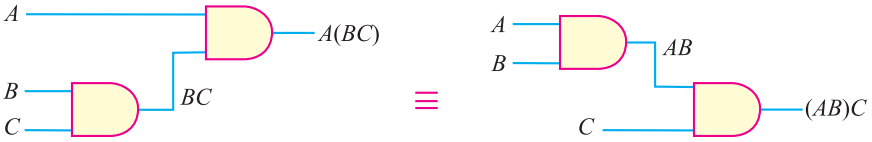


Fig. 26.44

Fig. 26.44 illustrates this law as applied to AND gates.

Example 26.26. *Illustrate distributive law as applied to Boolean algebra.*

Solution. This law states that ORing several variables and ANDing the result with a single variable is equivalent to ANDing the single variable with each of the several variables and ORing the products. The distributive law can be written for three variables (*A*, *B* and *C*) as under :

$$A\,(B + C) = AB + BC$$

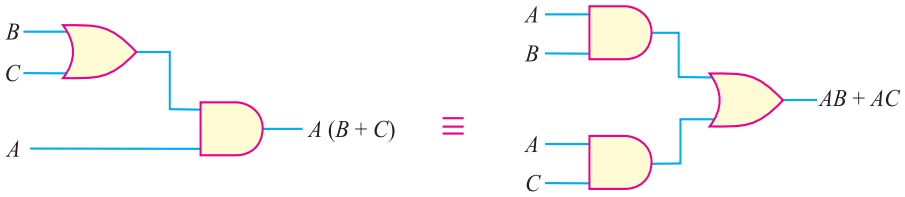


Fig. 26.45

Fig. 26.45 illustrates the distributive law in terms of gate implementation.

Note. The commutative law, the associative law and the distributive law of Boolean algebra are applicable regardless of the number of variables involved.

26.27 Truth Table from Logic Circuit

The output of a logic circuit for the given inputs can be determined directly from the circuit *without* using the Boolean expression. The procedure is to take a given set of inputs and work through each gate of the circuit to determine the final output. Let us illustrate the procedure with an example. Consider the logic circuit shown in Fig. 26.46. It consists of two AND gates and one OR gate. Each of the three gates has two input variables. Each of the input variables can be either a high (1) or a low (0). Since there are four input variables, there are 16 possible combinations of the input variables ($2^4 = 16$). We shall develop the truth table for the circuit without using the Boolean expression for the circuit.

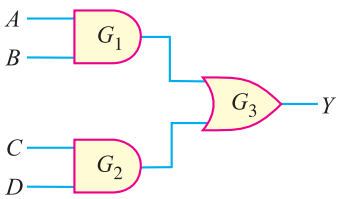


Fig. 26.46

First, suppose each input is low (*i.e.* $A = 0$, $B = 0$, $C = 0$ and $D = 0$). Under this condition, we shall examine the output of each gate in the circuit to arrive at the final output Y. If the inputs to gate G_1 are low (0), the output of G_1 is low (0). Also the output of G_2 is low (0) because both inputs are low (0). The two low inputs to G_3 also make its output low (0). Therefore, the output of the logic circuit is low when all the inputs are low as shown in the truth table. Similarly, we can find the output of the circuit for the remaining 15 input combinations.

Truth Table

Y = AB + CD

Inputs *A B C D*	*G_1 Output* (*AB*)	*G_2 Output* (*CD*)	*G_3 Output* *Y*
0 0 0 0	0	0	0
0 0 0 1	0	0	0
0 0 1 0	0	0	0
0 0 1 1	0	1	1
0 1 0 0	0	0	0
0 1 0 1	0	0	0
0 1 1 0	0	0	0
0 1 1 1	0	1	1
1 0 0 0	0	0	0
1 0 0 1	0	0	0
1 0 1 0	0	0	0
1 0 1 1	0	1	1
1 1 0 0	1	0	1
1 1 0 1	1	0	1
1 1 1 0	1	0	1
1 1 1 1	1	1	1

Note. We can also construct the truth table by developing the Boolean expression from the logic circuit. The logic circuit shown above (See Fig. 26.46) is redrawn as shown in Fig. 26.47. The Boolean expression for this logic circuit is

$$Y = AB + CD$$

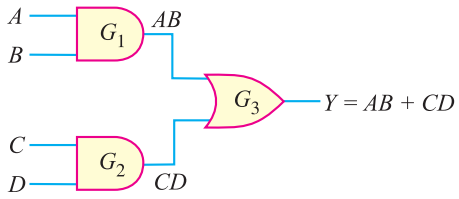


Fig. 26.47

Now simply put in the values for each combination of inputs and use the Boolean rules and laws to determine the final output Y. For example, when $A = 1$, $B = 1$, $C = 1$ and $D = 0$, the final output is

$$\begin{aligned} Y &= AB + CD \\ &= 1.1 + 1.0 \\ &= 1 + 0 \\ &= 1 \end{aligned}$$

Similary, we can find Y for the remaining 15 input combinations to get the truth table shown above.

26.28 Developing Logic Circuit from its Boolean Expression

We have seen how we can derive the Boolean expression for a given logic circuit. We now do the opposite *i.e.* create the logic circuit from its Boolean expression. Consider the following Boolean expression :

$$X = AB + CD$$

The function is composed of two terms, *AB* and *CD*, and it contains four variables. The first term is formed by ANDing *A* and *B* and the second term is formed by ANDing *C* and *D*. These two terms are then ORed to form the function *X*. The logic gates required to implement $X = AB + CD$ are as follows :

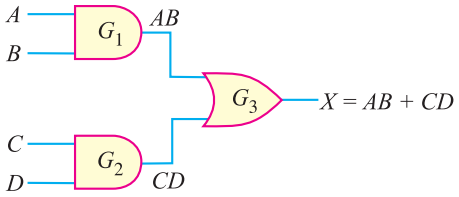


Fig. 26.48

(*i*) Two two-input AND gates to form *AB* and *CD*.

(*ii*) One two-input OR gate to form the final function *X* (= *AB* + *CD*).

The logic circuit that performs the function $X = AB + CD$ is shown in Fig. 26.48.

Example 26.27. *Drawn the logic circuit that implements the expression* $X = AB + CDE$.

Solution.

$$X = AB + CDE$$

The function is composed of two terms, *AB* and *CDE*, and it contains a total of five variables. The first term is formed by ANDing *A* and *B* and the second term is formed by ANDing *C*, *D* and *E*. These two terms are then ORed to form the function *X*. The logic gates required to implement $X = AB + CDE$ are as follows :

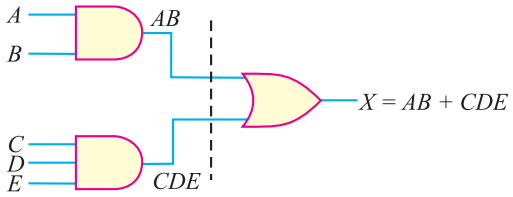


Fig. 26.49

(*i*) One two-input AND gate to form *AB*.

(*ii*) One three-input AND gate to form *CDE*.

(*iii*) One two-input OR gate to form the final function *X* (= *AB* + *CDE*).

The logic circuit that performs the function $X = AB + CDE$ is shown in Fig. 26.49.

Example 26.28. *Draw the logic circuit that implements the expression* $X = AB + \bar{B}C$.

Solution.

$$X = AB + \bar{B}C$$

The function is composed of two terms, *AB* and $\bar{B}C$, and it contains three variables. The first term is formed by ANDing *A* and *B* and the second term is formed by ANDing $\bar{B}$ and *C*. These two terms are then ORed to form the final function. The logic gates required to implement $X = AB + \bar{B}C$ are as follows :

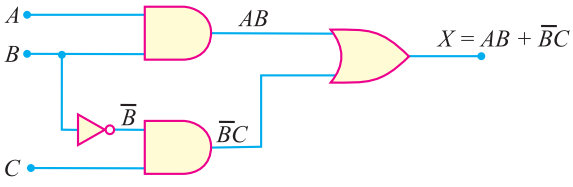


Fig. 26.50

(*i*) One inverter to form $\bar{B}$.

(*ii*) Two two-input AND gates to form *AB* and $\bar{B}C$.

(*iii*) One two-input OR gate to form the final function *X* (= *AB* + $\bar{B}C$).

The logic circuit that performs the function $X = AB + \bar{B}C$ is shown in Fig. 26.50.

Example 26.29. *Draw the logic circuit that implements the expression* $X = AB\,(C\bar{D} + EF)$.

Solution.

$$X = AB\,(C\bar{D} + EF)$$

A breakdown of this equation shows that the term AB and the term $C\bar{D} + EF$ are ANDed. The term AB is formed by ANDing the variables A and B. The term $C\bar{D} + EF$ is formed first by ANDing C and $\bar{D}$, ANDing E and F and then ORing these two terms. The logic gates required to implement $X = AB\,(C\bar{D} + EF)$ are as follows :

(*i*) One inverter to form $\bar{D}$.

(*ii*) Two two-input AND gates to form $C\bar{D}$ and EF.

(*iii*) One two-input OR gate to form $C\bar{D} + EF$.

(*iv*) One two-input AND gate to form AB.

(*v*) One two-input AND gate to form X [$=AB\,(C\bar{D} + EF)$].

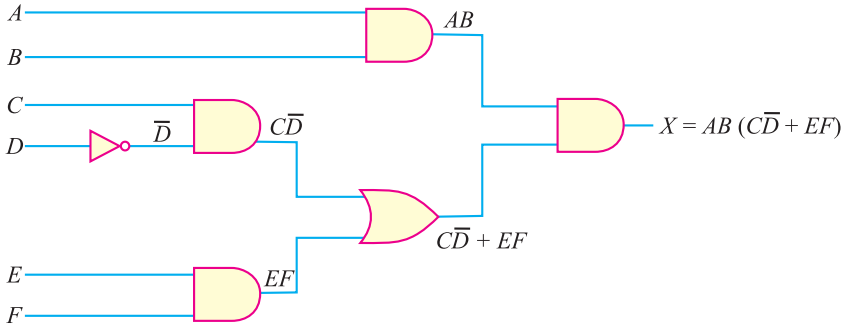


Fig. 26.51

The logic circuit that forms the function $X = AB\,(C\bar{D} + EF)$ is shown in Fig. 26.51.

26.29 Sum-of-Products Form

The sum-of-products form of a Boolean expression consists of two or more AND terms (*i.e.* products) that are ORed together. For example, $AB + CD$ is a sum-of-products expression. Here AND terms AB and CD are ORed (added). Other examples of this form are :

(*i*) $ABC + \bar{A}B\bar{C}$

(*ii*) $A\bar{B}C + DEF + A\overline{EF}$

(*iii*) $\bar{A} + BC\bar{D} + EFG$

The sum-of-products is a very useful form of a Boolean expression due to the straightforward manner is which it can be implemented in logic gates. For example, Fig. 26.52 shows the logic circuit that results in a sum-of-products form. It has simply two steps: ANDing and then ORing. Therefore,

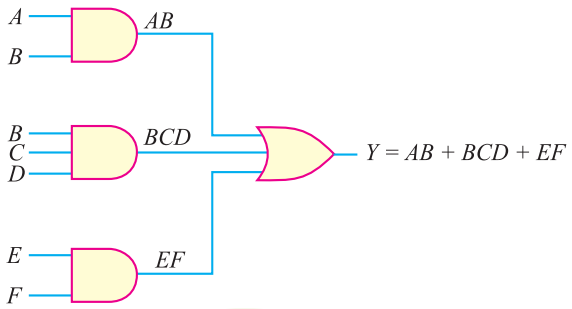


Fig. 26.52

this form is always only a *two-level* gate network *i.e.* the maximum number of gates through which a signal must pass in going form input to output is two (excluding inversions, but these can also be worked in).

26.30 Simplification of Boolean Expressions

The form of Boolean expression determines how many and which types of logic gates are needed as well as how they are connected together. The more complicated the Boolean expression, the more complex the logic circuit will be. It is, therefore, desirable to simplify an expression as much as possible to get the simplest logic circuit. Note that the new expression can be used to implement a logic circuit that is equivalent to the original logic circuit but contains fewer gates and connections. While simplifying a Boolean expression, the following two steps may be very helpful:

(*i*) Put the original expression into the sum-of-products form by the repeated use of rules, theorems and techniques of Boolean algebra.

(*ii*) Once it is in this form, the product terms are checked for common factors and factoring is performed wherever possible.

Illustration. Fig. 26.53 shows the logic circuit. The Boolean expression for this circuit is

$$X = AC\bar{D} + \bar{A}B\,(CD + BC)$$

We require *five AND gates*, *two OR gates* and *two inverters* to implement this expression. In all, nine gates are needed. We shall now use laws, rules and techniques of Boolean algebra to get the simplest expression for the given function.

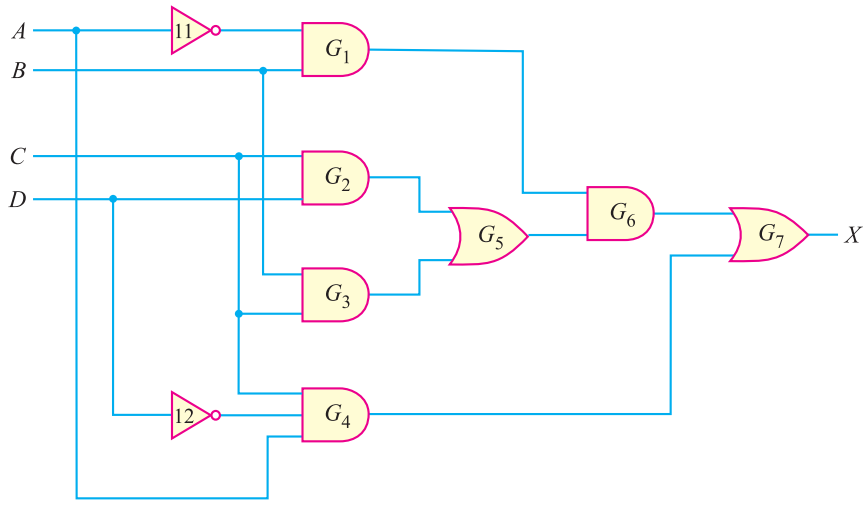


Fig. 26.53

Step 1. Apply distributive law to the second term by multiplying the term $CD + BC$ by $\bar{A}B$. The result is :

$$X = AC\bar{D} + \bar{A}BCD + \bar{A}BBC$$

Step 2. Applying the rule $BB = B$ to the third term, we have,

$$X = AC\bar{D} + \bar{A}BCD + \bar{A}BC$$

Step 3. Note that C is common to every term so that it can be factored out using distributive law.

$$\therefore \quad X = C\,(A\bar{D} + \bar{A}BD + \bar{A}B)$$

Step 4. We see that the term $\bar{A}B$ appears in the last two terms within the bracket and can be factored out of those two terms.

$$\therefore \quad X = C\,[A\bar{D} + \bar{A}B\,(D + 1)]$$

Since $D + 1 = 1$, $X = C\,(A\bar{D} + \bar{A}B)$

It appears that this equation cannot be simplified any further, but is can be written in a slightly different way by applying the distributive law (this results in the sum-of-products form):

$$X = AC\bar{D} + \bar{A}BC$$

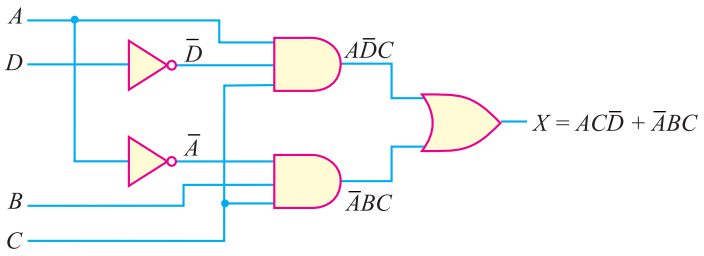


Fig. 26.54

Implementing this equation (*i.e.* $X = AC\bar{D} + \bar{A}BC$) into the logic circuit, it only requires *two three-input AND gates*, *two inverters* and *one two-input OR gate* as shown in Fig. 26.54. Note that this minimised circuit is equivalent to the original circuit of Fig. 26.53 but only requires five gates (instead of nine).

Example 26.30. *Simplify the expression :*

$$X = \bar{A}\bar{B}C + A\bar{B}C + AB\bar{C} + ABC$$

Solution.

$$X = \bar{A}\bar{B}C + A\bar{B}C + AB\bar{C} + ABC$$

Step 1. Note that the first two terms have $\bar{B}C$ as common factors while the last two terms have AB as common factors.

$$\therefore \quad X = \bar{B}C\,(A + \bar{A}) + AB\,(C + \bar{C})$$

Step 2. $A + \bar{A} = 1$ and $C + \bar{C} = 1$ so that :

$$X = \bar{B}C \cdot 1 + AB \cdot 1$$

Step 3. Since $\bar{B}C \cdot 1 = \bar{B}C$ and $AB \cdot 1 = AB$ so that :

$$\boldsymbol{X = AB + \bar{B}C}$$

Note that not only is the Boolean expression simplified, but so is the resultant logic circuit.

Example 26.31. *Simplify the expression :*

$$X = AB + A\,(B + C) + B\,(B + C)$$

Solution.

$$X = AB + A\,(B + C) + B\,(B + C)$$

Step 1. Applying distributive law to the second and third terms, we have,

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

Step 2. Now $BB = B$ and $AB + AB = AB$ so that :

$$X = AB + AC + B + BC$$

Step 3. $B + BC = B(1 + C) = B \cdot 1 = B$

$\therefore$ $X = AB + AC + B$

Step 4. Factoring B out, we have,

$$X = B(A + 1) + AC$$

Step 5. $A + 1 = 1$ so that $B(A + 1) = B \cdot 1 = B$.

$\therefore$ $\boldsymbol{X = B + AC}$

The original expression is simplified as far as it can go. Once you get acquainted with Boolean simplification techniques, you can combine many individual steps.

Example 26.32. *Simplify the circuit of Fig. 26.55 (i).*

(*i*) (*ii*)

Fig. 26.55

Solution. The Boolean expression for the circuit shown in Fig. 26.55 (*i*) is

$$X = (\bar{A} + B)(A + \bar{B})$$

Step 1. Multiplying out to get the sum-of-products form, we have,

$$X = \bar{A}A + \bar{A}\bar{B} + BA + B\bar{B}$$

Step 2. Now $\bar{A}A = 0$ and $B\bar{B} = 0$ so that :

$$X = \bar{A}\bar{B} + AB$$

This expression is implemented in Fig. 26.55 (*ii*). If you compare the circuit with the original circuit, you see that both circuits contain the same number of gates and connections. *Therefore, the simplification process has not produced a simpler circuit but it has produced an alternative circuit.*

26.31 Binary Addition

The addition of two binary numbers is performed in exactly the same manner as the addition of decimal numbers. In fact, binary addition is simpler, because we have only two digits (0 and 1). When adding binary numbers, the following rules apply :

$$0 + 0 = 0$$
$$1 + 0 = 1$$
$$1 + 1 = 10 = 0 + \text{carry of 1 into next position}$$
$$1 + 1 + 1 = 11 = 1 + \text{carry of 1 into next position}$$

It is not necessary to consider the addition of more than two binary numbers at a time. It is because the circuitry that actually performs the addition in a digital system can handle only two numbers at a time. When more than two numbers are to be added, the first two are added together and then their sum is added to the third number and so on.

Examples. Let us illustrate binary addition with two examples.

* This is the alternate symbol for the inverter.

(*i*) Let us add the following two binary numbers :

$$\begin{array}{r} 1\,0\,1 \\ +\ 0\,1\,0 \\ \hline 1\,1\,1 \end{array}$$

The addition is done column by column starting at the right as it is in the decimal number system. For the first column, 1 + 0 = 1 and for the second column, 0 + 1 = 1. For the third or last column, 1 + 0 = 1. This produces a sum of 111.

(*ii*) Let us now add the following two binary numbers :

$$\begin{array}{r} 1\,0\,1\,1\,1 \\ +\ 1\,0\,0\,1\,0 \\ \hline 1\,0\,1\,0\,0\,1 \end{array}$$

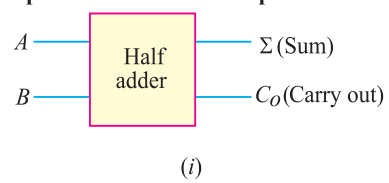
For the first column, 1 + 0 = 1. In the second column, we have, 1 + 1 = 10 = 0 with a carry of 1. For the third column, we have, 1 + 0 + carry of 1 = 1 + 1 = 10 = 0 with a carry of 1. For the fourth column, 1 + 0 + 0 = 1 + 0 = 1. For the fifth column, we have 1 + 1 = 10 = 0 with a carry of 1. This produces a sum of 101001.

Binary addition is the most important arithmetic operation in digital systems. It is because the operations of subtraction, multiplication and division as they are performed in most modern digital computers and calculators actually use only addition as their basic operation. To be an intelligent worker on digital equipment, you must master binary addition.

26.32 Electronic Adders

A logic circuit that performs the function of binary addition is called **electronic adder** *or* **adder.** The adder circuit consists of properly connected logic gates. There are usually two forms of the adder in common applications *viz.* (*i*) half adder (*ii*) full adder.

(*i*) Half Adder (HA). A logic circuit that can add two binary bits is called a half adder (*HA*). Fig. 26.56 (*i*) shows the block symbol for the half adder. The adder circuit would need two inputs and two outputs. The two inputs are for the two digits to be added, either 0 or 1. One output terminal is for the sum of the two inputs and the other output is for the carry, if necessary. Fig. 26.56 (*ii*) shows the addition table of the adder and can be thought of as a truth table. The numbers being added are on the input side of the table. The truth table has two output columns, one column for the sum and one column for the carry. The sum column is labeled with summation symbol Σ. The carry column is labeled with C_o. The C_o stands for carry output or *carry out*. Thus the half adder circuit has two inputs (A, B) and two outputs (Σ, C_o).



(*i*)

Truth Table

Inputs		Outputs	
B	A	Σ	C_O
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1
Binary digits to be added		Sum	Carry out
		XOR	AND

(*ii*)

Fig. 26.56

The half adder would behave according to the truth table shown in Fig. 26.56 (*ii*). Take a careful look at the truth table. The output columns (sum and carry) can be produced by using two gates as under :

(*a*) The sum column is the output of XOR gate. Remember that XOR gate has HIGH output when either input is HIGH but not when both inputs are the same.

(*b*) The carry column is the output of the AND gate. Both inputs must be HIGH for there to be a HIGH in the output.

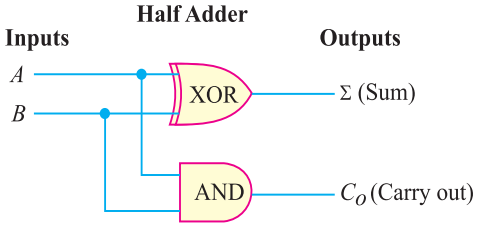
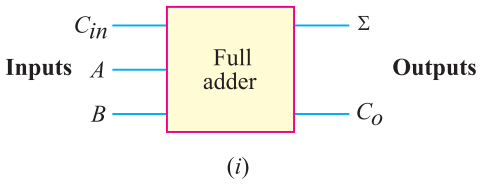


Fig. 26.57

Thus we can produce half adder using a two-input AND gate and a two-input XOR gate as shown in Fig. 26.57. The half adder circuit adds only the LSB column (1s column) in a binary addition problem. The reason is that it has no input for a carry-in.

***(ii)* Full Adder (FA).** A full adder adds two binary bits plus a carry input (C_{in}) to produce the sum (Σ) and carry (C_o) outputs. Fig. 26.58 (*i*) shows the block diagram of a full adder. It is formed by using two half adder circuits and an OR gate as shown in Fig. 26.58 (*iii*). Note the carry-in input which requires the extra half adder. The output of the OR gate forms the carry-out (C_o) output.

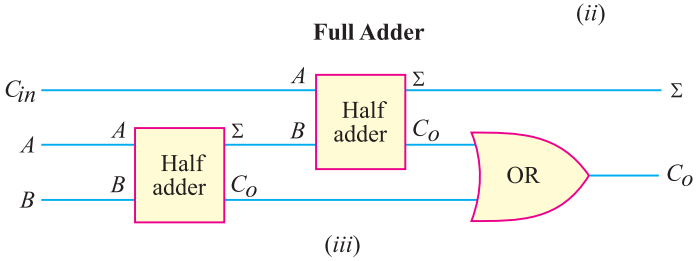


(*i*)

Truth Table

Inputs			Outputs	
C_{in}	B	A	Σ	C_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	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
Carry + B + A			Sum	Carry out

(*ii*)



(*iii*)

Fig. 26.58

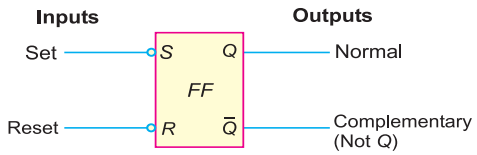
The full adder has three inputs : C_{in}, A and B. These three inputs must be added to get the Σ and C_o outputs. Fig. 26.58 (*ii*) shows the truth table for a full adder. It shows all the possible combinations of A, B and C_{in}. The full adder is used for binary addition in all places except the 1s place which means 2s, 4s, 8s etc.

26.33 Flip-Flops

A *flip-flop* is a bistable circuit made up of logic gates. A bistable circuit can exist in either of two stable states indefinitely and can be made to change its state by means of some external signal. The most important use of this property is that a flip-flop can "store" binary information. We have seen that a logic gate can make a logical decision based on the immediate conditions at the input terminals. However, the gates normally do not have a memory characteristic to retain the input data. On the other hand, flip-flops have the valuable feature of remembering. The reason is that a flip-flop circuit

is bistable. Because the flip-flop's output remains at a 0 or 1 depending on the last input signal, the flip-flop can be said to "remember". Another name for the flip-flop is *bistable multivibrator.* We shall discuss two important types of flip-flops *viz.* (*i*) *R–S* flip-flop and (*ii*) *J–K* flip-flop.

(*i*) R–S Flip-Flop. The **R–S* type is the basic flip-flop logic circuit. Fig. 26.59 (*i*) shows the logic symbol for an *R–S* flip flop. It has two inputs called *set* (*S*) and *reset* (*R*). The two outputs are labeled as *Q* and $\overline{Q}$. In flip-flops, the outputs are always opposite or complementary. In other words, if output *Q* = 1, then output $\overline{Q}$ = 0 and so on. Note the small bubbles for inversion at the *S* and *R* input terminals. The bubbles show that this *FF* has active LOW inputs. Logic 0 is required to activate the *R* or *S* input.



(*i*)

Truth Table

Mode of operation	Inputs		Outputs	
	S	*R*	*Q*	$\overline{Q}$
Prohibited	0	0	1	1
Set	0	1	1	0
Reset	1	0	0	1
Disabled	1	1	No change	

(*ii*)

Fig. 26.59

The exact behaviour of a flip-flop is defined by its truth table shown in Fig. 26.59 (*ii*). In the left column, the four modes of operation are :

(*a*) *Prohibited mode.* When the *S* and *R* inputs are both 0, both outputs go to a logical 1. This is called a prohibited state for the flip-flop. This mode is not used because it drives both outputs HIGH. A flip-flop must operate with complementary outputs.

(*b*) *Set mode.* The second line of truth table shows that when input *S* is 0 and *R* is 1, the *Q* output is set to logical 1. This is called the set mode. In this *FF*, the bubble at *S* means that a LOW here makes *Q* go HIGH.

(*c*) *Reset mode.* The third line of the truth table shows that when input *R* is 0 and *S* is 1, output *Q* is reset (cleared) to 0. This is called the reset mode. In this *FF*, the bubble at *R* means a LOW here resets *Q* to make it LOW.

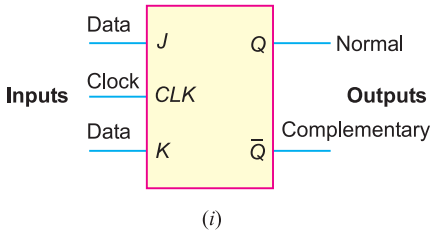
(*d*) *Disabled mode.* The fourth line of the truth table shows both inputs (*R* and *S*) at 1. This is the idle or at rest condition and leaves *Q* and $\overline{Q}$ in their previous complementary states. This is called disabled state because there is no change. The outputs stay as they were, with *Q* either set or reset. Because of the disabled mode, the flip-flop "remembers" the preceding state by remaining at that state until it is switched. That operation is possible because a flip-flop is a bistable circuit.

(*ii*) J–K Flip-Flop. The *J–K* flip-flop is probably the most widely used and is considered the universal flip-flop because it can be used in many ways. The logic symbol for the *J–K* flip-flop is illustrated in Fig. 26.60 (*i*). The inputs labeled *J* and *K* are the data inputs. The input labeled *CLK* is the clock input. Outputs *Q* and $\overline{Q}$ are the usual normal and complementary outputs.

Fig. 26.60(*ii*) shows the truth table for the *J–K* flip-flop. This table shows four useful modes of operation. When the *J* and *K* inputs are both 0, the flip-flop is in the *hold* (or *disabled*) *mode*. In the hold mode, the data inputs have no effect on the outputs. The outputs "hold" the last data present. Lines 2 and 3 of the truth table show the reset and set conditions for the *Q* output. Line 4 of the truth

* It is reset/set type flip-flop and hence the name *R–S* flip-flop.

table illustrates the useful *toggle* position of the *J–K* flip-flop. When both data inputs *J* and *K* are at 1, repeated *clock pulses cause the output to turn off-on-off-on-off-on and so on. This off-on action is like a toggle switch and is called *toggling*. Each clock pulse toggles the outputs to switch to their opposite states.



(*i*)

Truth Table

Mode of operation	Inputs			Outputs		
	CLK	*J*	*K*	*Q*	$\bar{Q}$	Effect on output Q
Hold	⊓	0	0	No change		No change – disable
Reset	⊓	0	1	0	1	Reset or cleared to 0
Set	⊓	1	0	1	0	Set to 1
Toggle	⊓	1	1	Toggle		Changes to opposite state

(*ii*)

Fig. 26.60

MULTIPLE-CHOICE QUESTIONS

1. The binary number 10101 is equivalent to decimal number .....
(*i*) 19 (*ii*) 12
(*iii*) 27 (*iv*) 21

2. The universal gate is .....
(*i*) NAND gate (*ii*) OR gate
(*iii*) NOT gate (*iv*) none of the above

3. The inverter is .....
(*i*) NOT gate (*ii*) OR gate
(*iii*) AND gate (*iv*) none of the above

4. The inputs of a NAND gate are connected together. The resulting circuit is .....
(*i*) OR gate (*ii*) AND gate
(*iii*) NOT gate (*iv*) none of the above

5. The NOR gate is OR gate followed by .....
(*i*) AND gate (*ii*) NAND gate
(*iii*) NOT gate (*iv*) none of the above

6. The NAND gate is AND gate followed by .....
(*i*) NOT gate (*ii*) OR gate
(*iii*) AND gate (*iv*) none of the above

7. Digital circuit can be made by the repeated use of .....
(*i*) OR gates (*ii*) NOT gates
(*iii*) NAND gates (*iv*) none of the above

8. The only function of NOT gate is to .....
(*i*) stop a signal
(*ii*) invert input signal
(*iii*) act as a universal gate
(*iv*) none of the above

9. When an input signal 1 is applied to a NOT gate, the output is .....
(*i*) 0
(*ii*) 1
(*iii*) either 0 or 1
(*iv*) none of the above

10. In Boolean algebra, the bar sign (–) indicates .....
(*i*) OR operation
(*ii*) AND operation
(*iii*) NOT operation
(*iv*) none of the above

* The clock pulses occur at some fixed interval, say after every 10 microsecond.

11. The given Boolean expression is

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

If $A = 1$ and $B = 1$, then $Y =$

(*i*) 1 (*ii*) 0

(*iii*) either 1 or 0 (*iv*) none of the above

12. In Boolean algebra, the plus sign (+) indicates .....

(*i*) AND operation

(*ii*) OR operation

(*iii*) NOT operation

(*iv*) none of the above

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

(*i*) $\overline{A} + \overline{B}$ (*ii*) $\overline{A} - \overline{B}$

(*iii*) $\overline{A} \cdot \overline{B}$ (*iv*) none of the above

14. $\overline{(A \cdot B)}$ = ...

(*i*) $\overline{A} + \overline{B}$ (*ii*) $\overline{A} \cdot \overline{B}$

(*iii*) $\overline{A} - \overline{B}$ (*iv*) none of the above

15. $A + A \cdot B$ = ...

(*i*) B (*ii*) A

(*iii*) $\overline{A} + B$ (*iv*) none of the above

Answers to Multiple-Choice Questions

1. (*iv*)	**2.** (*i*)	**3.** (*i*)	**4.** (*iii*)	**5.** (*iii*)
6. (*i*)	**7.** (*iii*)	**8.** (*ii*)	**9.** (*i*)	**10.** (*iii*)
11. (*ii*)	**12.** (*ii*)	**13.** (*iii*)	**14.** (*i*)	**15.** (*ii*)

Chapter Review Topics

1. Write a short note on analog and digital signals.

2. What is a digital circuit ?

3. What is binary number system ?

4. How will you make decimal to binary conversion ?

5. How will you make binary to decimal conversion ?

6. What is a logic gate ?

7. What are the three basic logic gates ?

8. Describe OR function with a 2-input OR gate.

9. Explain AND function with a 2-input AND gate.

10. What is a NAND gate ?

11. What is a NOR gate ?

12. How will you obtain NOT gate from NAND gate ?

13. What is indicated by plus (+), dot (.) and bar (—) in a Boolean expression ?

14. State De Morgan's theorems.

15. What are encoders and decoders ?

Problems

1. Convert decimal number 23 into equivalent binary number. **[$(10111)_2$]**

2. Simplify the expression $Y = A\,C\,D + \overline{A}\,B\,C\,D$. **[$Y = A\overline{C} + \overline{B}D$]**

3. Simplify the expression $Y = \overline{(\overline{A} + C) \cdot (B + \overline{D})}$ to one having only single variables inverted.

[$Y = A\overline{C} + \overline{B}D$]

4. Find the complement function of $Y = \overline{A}\,B\,\overline{C} + \overline{A}\,\overline{B}\,C$. **[$(A + \overline{B} + C)\,(A + B + \overline{C})$]**

5. Simplify the expression $Y = A \cdot B + A \cdot \overline{B}$. **[Y = A]**

6. Simplify the expression $Y = A \cdot B \cdot C + B \cdot C$. **[$Y = B \cdot C$]**

7. Simplify the following Boolean expression to a minimum number of literals :

$$Y = A(\overline{A} + B)$$ **[Y = AB]**

8. Simplify the following Boolean function to a minimum number of literals :

$$Y = \overline{A}\,\overline{B}\,C + \overline{A}\,B\,C + A\,\overline{B}$$ **[$Y = \overline{A}\,C + A\overline{B}$]**

9. Simplify the expression : $Y = (A + B)\,(\overline{A} + C)\,(B + C)$ **[$Y = (A + B)\,(\overline{A} + C)$]**

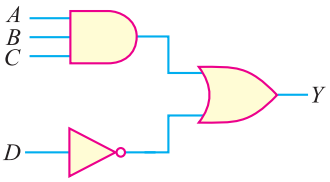
10. Find the complement of the function :

$$Y = A(\overline{B}\,\overline{C} + B\,C)\,H$$ **[$\overline{Y} = \overline{A} + (B + C)\,(\overline{B} + \overline{C})$]**

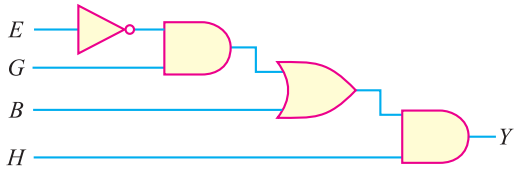
11. Draw the logic circuit for the following Boolean expressions :

(*i*) $Y = ABC + \overline{D}$ (*ii*) $Y = (\overline{E}G + B)\,H$

Ans.



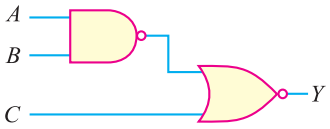
(*i*)



(*ii*)

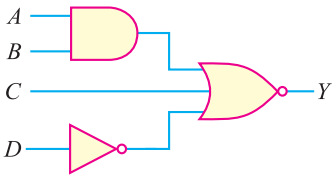
12. A logic circuit is given by the Boolean escperssion : $Y = \overline{\overline{AB} + C}$. Draw the logic circuit for this expression.

Ans.

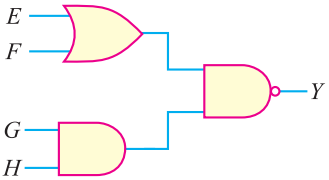


(*iii*)

13. Write the Boolean expressions for the logic circuits shown in Fig. 26.61.



(*i*)



(*ii*)

Fig. 26.61

[Ans. :- (*i*) $Y = \overline{AB + C + \overline{D}}$ (*ii*) $Y = \overline{(E + F)(GH)}$

14. Write the Boolean expression for the logic circuit shown in Fig. 26.62.

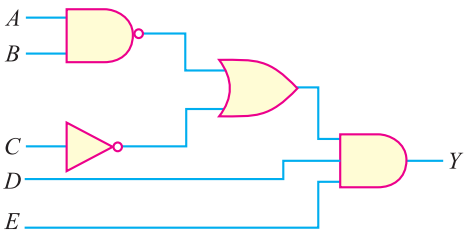


Fig. 26.62

[Ans. :- $Y = (\overline{AB} + \overline{C})\,DE$]

15. Write the BCD code for 829.

[100000101001]

16. What is the decimal number for 10000111000 BCD ?

[438]

Discussion Questions

1. Why is logic circuit name so ?
2. What is the importance of digital techniques ?
3. Why is analog system unreliable ?
4. What is the importance of NAND gate ?
5. What is Boolean algebra ?
6. What is the importance of De Morgan's theorems in Boolean Algebra ?
7. What is the meaning of + sign in Boolean expression ?
8. Give two differences between decimal and binary systems.
9. What are the disadvantages of digital circuits ?
10. What are the advantages of Boolean theorems ?
11. What is the meaning of sign . in Boolean expression ?
12. What is a universal gate ? Why is it so named?
13. Most of information we handle is in decimal form. Will a digital circuit process this information as such?
14. What role is played by encoder and decoder?

Top