CHAPTER-1

Fundamental Concepts

1.1 ANALOG SIGNALS

We are very familiar with analog signals. The reading of a moving coil or moving iron voltmeter and ammeter, dynamometer wattmeter etc., are all analog quantities. The traceon a CRO screen is also analog. Analog methods for communication system have long been in use. Frequency division multiplexing is the means of analog communication. An electronicamplifier is an analog circuit. The low level analog signal (audio, video, etc.) is amplified to provide strength to the signal. Analog circuit systems (position control, process control) have been in use for the past many decades. Analog Computers use voltages, resistances and potentiometric rotations to represent the numbers and perform arithmetic operations. Analog differentiation, integration, etc., is also done. Operational amplifier is a very versatile analog electronic circuit used to perform a variety of operations (addition, subtraction, multiplication, division, exponentiation, differentiation, integration etc.). Analog integrated circuits arewidely used in electronic industry.

1.2 DIGITAL SIGNALS

The term digital is derived from digits. Any device or system which works on digits is a digital device or system. A digital voltmeter indicates the value of voltage in the form of digits, e.g., 230.25. Reading an analog instrument introduces human error and also requires more time. A digital reading is more accurate, eliminates human error and can be read quickly.

Communication systems have also gone digital. The initial signal waveform isalways analog. To use digital transmission, the signal waveform is sampled and the digital representation transmitted. The process of converting analog signal to digital form is also

known as digitizing. For multiple channels of transmission, Time Division Multiplexing is used.

Digital control systems are fast replacing analog control systems. In digital control systems the error is in the form of digital pulses.

Digital computers have revolutionalized the concept of computers. Their capability ranges from simple calculations to complex calculations using numerical techniques. Many computing tasks which required hours and days take only a few minutes on digital computers.

Digital signal processing is concerned with the representation of continuous time (analog) signals in digital form. It is based on Claude Shannon's* sampling theorem which states that "A band limited continuous time signal can be reconstructed in its entirety from a sequence of samples taken at intervals of less than $\frac{1}{2f_N}$ where f_N is the highest frequency present in the signal." It is essential that the analog signal is band limited which limits how much it can change between samples. The sampling rate has to high to be ensure accuracy.

Since the initial signal is always analog and the final required signal is also mostly analog, a digital system requires three essential aspects (1) conversion of analog signal to digital form (2) transmission of digital signal (3) reconstruction of analog signal from the received digital signal as shown in Fig. 1.1

A continuous time function x(t) is converted into a digital signal x(n) by an analog to digital (A/D) converter. The output of discrete time system is y(n) and is converted to continuous time function by digital to analog (D/A) converter. The discrete time system, in digital communications, is a digital communication channel. To achieve high fidelity, the sampling rate may have to be very high say 50000 samples per second. Each sample may be encoded by (say) 18 bits. The frequency f_s (in Fig. 1.1) must be more than twice f_N the highest frequency in the analog signal. Very large scale integration (VLSI) digital circuits have capability to sample at very fast rate so that high fidelity is achieved.

---

* Sampling is done to convert analog signal to digital signal.

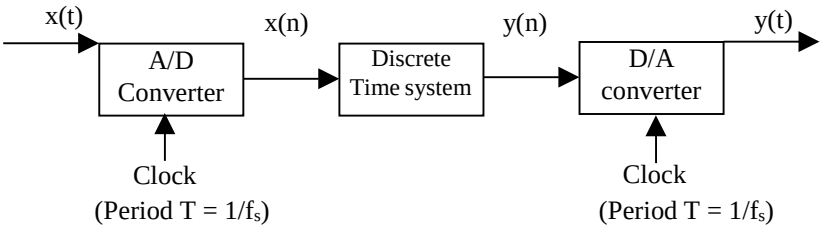


Fig. 1.1 Digital system

A DSP (digital signal processing) chip is the core of digital system used in cellular phones, modems, disk drives, digital automotive systems etc. It was invented only about 15 years ago but its applications have grown tremendously.

Digital methods have the following advantages over analog methods :

1. Digital devices work only in two states (say on and off). Thus their operation is very simple and reliable.
2. Digital display is very accurate and can be read at a fast speed. Human error is eliminated.
3. Electronic components exhibit change in behaviour due to ageing, change of ambient temperature etc. Therefore, the behaviour of analog circuits tends to be somewhat unpredictable. However, digital circuits are free from these defects.
4. Digital ICs are very cheap and compact in size.
5. Variety of digital ICs are available.
6. Power requirement of digital circuits is very low.
7. Digital systems have the characteristic advantage of memory. Thus information can be stored over a period of time. The space required for this stage is very small. One compact disc* can store information contained in many books.
8. Digital systems have high fidelity and provide noise free operations.
9. By integrating system peripheral functions on a DSP chip, the reliability can be enhanced and cost reduced.
10. When volumes are high, they can be manufactured at low cost.
11. The same digital system can be used with a variety of software for a number of tasks.
12. Standardisation & Repeatability.

1.3 BASIC DIGITAL CIRCUITS

In a digital system there are only a few basic operations performed, irrespective of the complexities of the system. These operations may be required to be performed a number of times in a large digital system like digital computer or a digital control system, etc. The basic operations are AND, OR, NOT, and FLIP-FLOP. The AND, OR, and NOT operations are discussed here and the FLIP-FLOP, which is a basic memory element used to store binary information (one bit is stored in one FLIP-FLOP).

1.3.1 The And Operation

A circuit which performs an AND operation is shown in Fig. 1.2. It has N inputs (N ≥ 2) and one output. Digital signals are applied at the input terminals marked A, B, …, N, the other terminal being ground, which is not shown in the diagram. The output is obtained at the output terminal marked Y (the other terminal being ground) and it is also a digital signal. The AND operation is defined as : the output is 1 if and only if all the inputs are 1. Mathematically, it is written as

$$\begin{aligned} Y &= A \text{ AND } B \text{ AND } C \ \ldots \text{ AND } N \\ &= A \cdot B \cdot C \cdot \ldots \cdot N \\ &= ABC \ \ldots N \end{aligned} \qquad \ldots(1.1)$$

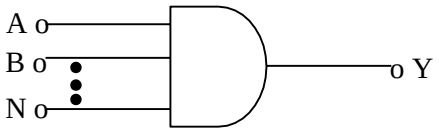


Fig. 1.2 The standard symbol for an AND gate

where A, B, C, … N are the input variables and Y is the output variable. The variables are binary, i.e. each variable can assume only one of the two possible values, 0 or 1. The *binary variables* are also referred to as *logical variables*.

Equation (1.1) is known as the *Boolean equation* or the *logical equation* of the AND *gate*. The term gate is used because of the similarity between the operation of a digital circuit and a gate. For example, for an AND operation the gate opens (Y = 1) only when all the inputs are present, i.e. at logic 1 level.

Truth Table Since a logical variable can assume only two possible values (0 and 1), therefore, any logical operation can also be defined in the form of a table containing all

possible input combinations (2^N combinations for N inputs) and their corresponding outputs. This is known as a *truth table* and it contains one row for each one of the input combinations.

For an AND gate with two inputs A, B and the output Y, the truth table is given in Table 1.1. Its logical equation is Y = AB and is read as "Y equals A AND B".

Since, there are only two inputs, A and B, therefore, the possible number of input combinations is four. It may be observed from the truth table that the input–output relationship for a digital circuit is completely specified by this table in contrast to the input–output relationship for an analog circuit. The pattern in which the inputs

Table 1.1 Truth table of a 2-input AND gate

Inputs		Output
A	B	$\overline{Y}$
0	0	0
0	1	0
1	0	0
1	1	1

are entered in the truth table may also be observed carefully, which is in the ascending order of binary numbers formed by the input variables. (See Chapter 2).

1.3.2 The OR Operation

Figure 1.3 shows an OR gate with N inputs (N ≥ 2) and one output. The OR operation is defined as: the output of an OR gate is 1 if and only if one or more inputs are 1. Its logical equation is given by

$$Y = A \text{ OR } B \text{ OR } C \ldots \text{ OR } N$$
$$= A + B + C + \ldots + N \qquad \ldots(1.2)$$

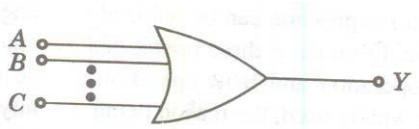


Fig. 1.3 The standard symbol for an OR gate

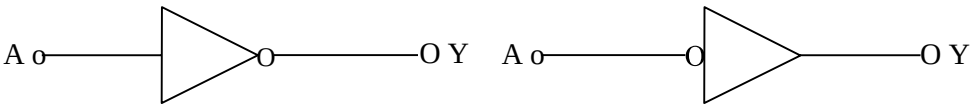
The truth table of a 2-input OR gate is given in Table 1.2. Its logic equation is Y = A + B and is read as "Y equals A or B".

Table 1.2 Truth table of a 2-input OR gate

Inputs		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

1.3.3 The NOT Operation

Figure 1.4. shows a NOT gate, which is also known as an *inverter*. It has one input (A) and one output (Y). Its logic equation is written as



(a) (b)

Fig. 1.4 The standard symbols for a NOT gate Book-2 Page 5

$$Y = \text{NOT } A$$
$$= \overline{A} \qquad \ldots(1.3)$$

and is read as "Y equals NOT A" or "Y equals complement of A". The truth table of a NOT gate is given in Table 1.3.

Table 1.3 Truth table of a NOT gate

Input A	Output Y
0	1
1	0

The NOT operation is also referred to as an inversion or complementation. The presence of a small circle, known as the *bubble*, always denotes inversion in digital circuits.

1.4 NAND AND NOR OPERATIONS

Any Boolean (or logic) expression can be realized by using the AND, OR and NOT gates discussed above. From these three operations, two more operations have been derived: the NAND operation and NOR operation. These operations have become very popular and are widely used, the reason being the only one type of gates, either NAND or NOR are sufficient for the realization of any logical expression. Because of this reason, NAND and NOR gates are known as *universal gates*.

1.4.1 The NAND Operation

The NOT-AND operation is known as the NAND operation. Figure 1.5a shows and N input ($N \geq 2$) AND gate followed by a NOT gate. The operation of this circuit can be described in the following way:

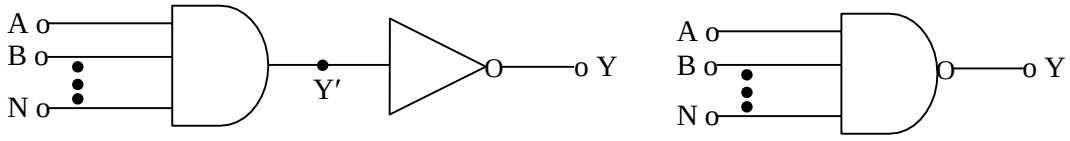
The output of the AND gate (Y′) can be written using Eq. (1.)

$$Y' = AB \ldots N \qquad \ldots(1.4)$$

Now, the output of the NOT gate (Y) can be written using Eq. (1.3)

$$Y = \overline{Y'} = (\overline{AB...N}) \qquad \ldots(1.5)$$

The logical operation represented by Eq. (1.5) is known as the NAND operation. The standard symbol of the NAND gate is shown in Fig. 1.5b. Here, a bubble on the output side of the NAND gate represents NOT operation, inversion or complementation.



(a) (b)

Fig. 1.5 (a) NAND operation as NOT-AND operation,

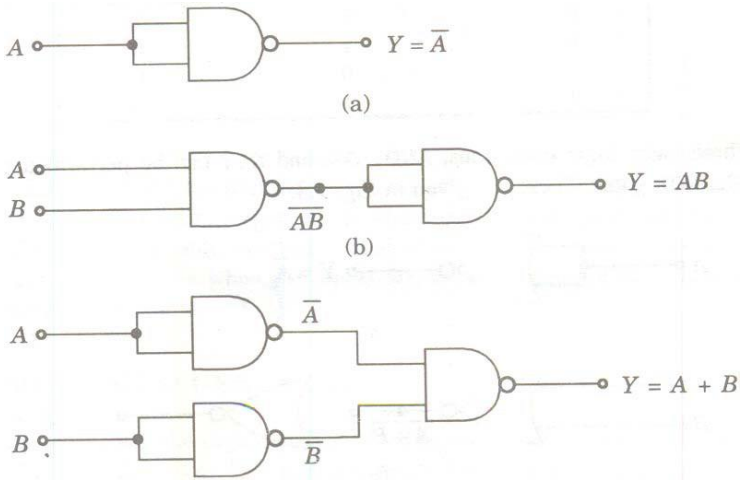
(b) Standard symbol for the NAND gate. Book-2 Page 6

The truth table of a 2-input NAND gate is given in Table 1.4. Its logic equation is $Y = \overline{A \cdot B}$ and, is read as "Y equals NOT (A AND B)".

Table 1.4 Truth table of a 2-input NAND gate

Inputs		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

The three basic logic operations, AND, OR and NOT can be performed by using only NAND gates. These are given in Fig. 1.6.



(c) (c)

Fig. 1.6 Realization of basic logic operations using NAND gates (a) NOT (b) AND (c) OR.

1.4.2 The NOR Operation

The NOT-OR operation is known as the NOR operation. Figure 1.7a shows an N input ($N \geq 2$) OR gate followed by a NOT gate. The operation of this circuit can be described in the following way:

The output of the OR gate Y′ can be written using Eq. (1.2) as

$$Y'. = A + B + \ldots + N \qquad \ldots(1.6)$$

and the output of the NOT gate (Y) can be written using Eq. (1.3)

$$Y = \overline{Y'} = \overline{A + B + ... + N} \qquad \ldots(1.7)$$

The logic operation represented by Equ. (1.7) is known as the NOR operation.

The standard symbol of the NOR gate is shown in Fig. 1.7b. Similar to the NAND gate, a bubble on the output side of the NOR gate represents the NOT operation.

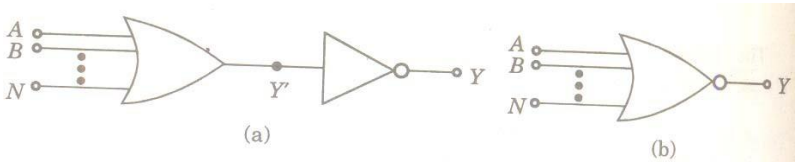


Fig. 1.7 (a) NOR operation as NOT-OR operation

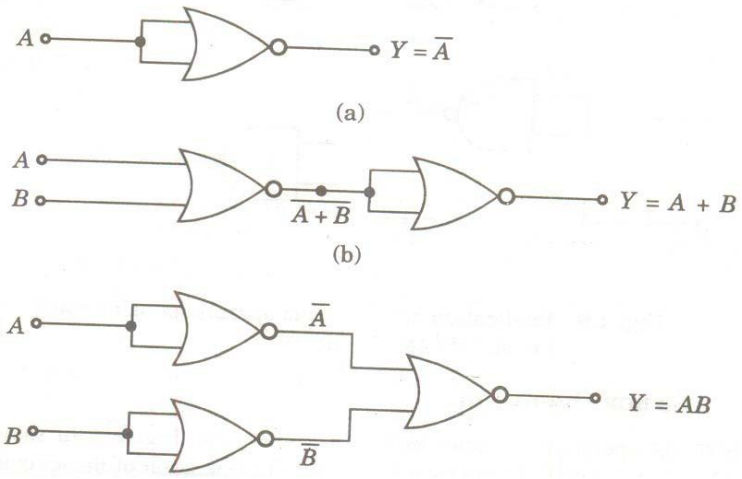
(b) Standard symbol for the NOR gate

Table 1.5 gives the truth table of a 2-input NOR gate. Its logic equation is $Y = \overline{A + B}$ and is read as "Y equals NOT (A OR B)"

Table 1.5 Truth table of a 2-input NOR gate

Inputs		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

The three basic logic operations, AND, OR, and NOT can be performed by using only the NOR gates. These are given in Fig. 1.8.



(c)

Fig. 1.8 Realization of basic logic operations using NOR gates (a) NOT (b) OR (c) AND

1.5 EXCLUSIVE–OR OPERATION

The EXCLUSIVE–OR (EX–OR) operation is widely used in digital circuits. It is not a basic operation and can be performed using the basic gates–AND, OR and NOT or universal gates NAND or NOR. Because of its importance, the standard symbol shown in Fig. 1.9 is used for this operation.

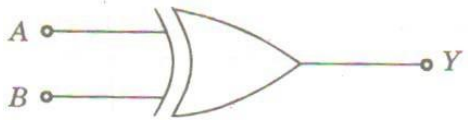


Fig. 1.9 Standard symbol for EX-OR gate.

The truth table of an EX–OR gate is given in Table 1.6 and its logic equation is written as

$$Y = A \text{ EX} - \text{OR } B = A \oplus B \qquad \ldots(1.8)$$

Table 1.6 Truth table of a 2-input EX–OR gate

Inputs		Output
A	B	Y
0	0	0
0	1	1

1	0	1
1	1	0

If we compare the truth table of an EX–OR gate with that of an OR gate given in Table 1.2, we find that the first three rows are same in both. Only the fourth row is different. This circuit finds application where two digital signals are to be compared. From the truth table we observe that when both the inputs are same (0 or 1) the output is 0, whereas when theinputs are not same (one of them is 0 and the other one is 1) the output is 1.

1.6 BOOLEAN ALGEBRA RELATIONS*

1.6.1 Commutative Law

$$A + B = B + A \quad \ldots(1.9)$$

$$A \cdot B = B \cdot A \quad \ldots(1.10)$$

Equations (1.9) and (1.10) mean that inputs can be interchanged in OR gate and AND gate.

Fig. 1.10 illustrates commutative law. In Fig. 1.10 (a) the two inputs to OR gate have been interchanged. The output is the same.

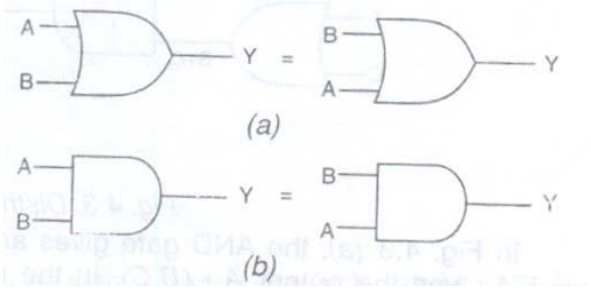


Fig. 1.10 Commutative law in Boolean algebra (a) ORing (b) ANDing

In Fig. 1.10 (b) the two inputs to AND gate have been interchanged. The output is the same.

1.6.2 Associative Law

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

$$A.\ (B.\ C) = (A.\ B)\ C$$

Equations with ( ) and ( ) are the Associative laws for ORing and ANDing.

---

* All Boolean relations are called laws or theorems.

Fig. 1.11 illustrates the associative law. In Fig. 1.11 (a) the nputs to OR gates have been grouped in two different ways but the output is the same, i.e., Y = A + B + C.

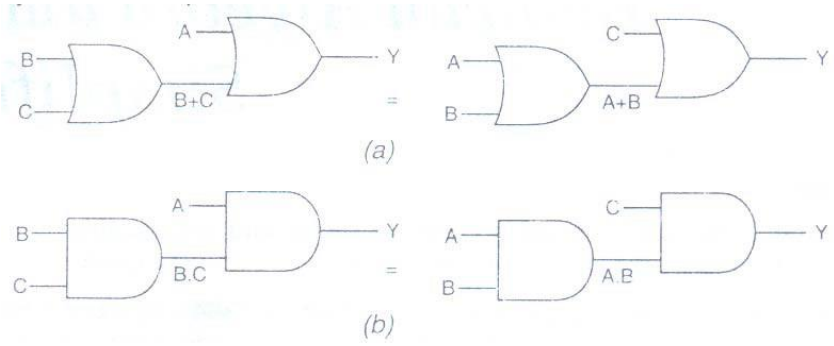


Fig. 1.11 Associative law in Boolean algebra (a) ORing (b) ANDing

In Fig. 1.11 (b) the inputs to AND gates have been grouped in two different ways without affecting the output. In each case the output is Y = A.B.C.

1.6.3 Distributive Law

$$A + (B.\ C) = (A + B).\ (A + C) \qquad \ldots(1.12)$$

$$A.\ (B + C) = A.\ B + A.\ C \qquad \ldots(1.13)$$

Fig. 1.12 illustrates the distributive law.

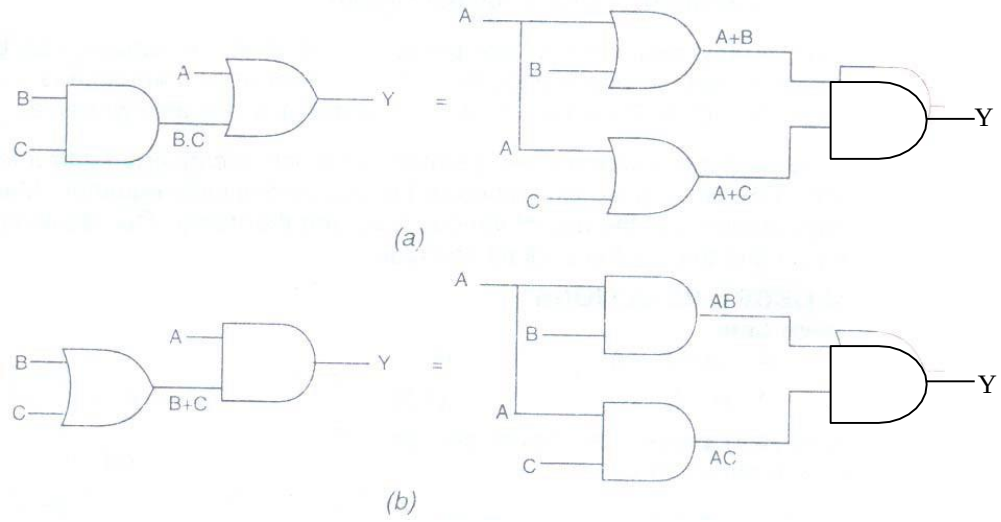


Fig. 1.12 Distributive law in Boolean algebra

In Fig. 1.12 (a), the AND gate gives an output B.C. This signal when fed to OR gate along with input A gives the output A + (B. C). In the circuit on RHS in Fig. 1.12 (a) the two OR gates given the output A + B and A + C respectively. The AND gate gives the output (A + B). (A + C).

In Fig. 1.12 (b) the OR gate gives the output (B + C). This is fed as input to AND gate along with A. On the RHS in Fig. 1.12 (b) the two AND gates give the outputs A. B and A. C respectively. The OR gate gives the output A.B + A. C.

Truth table for Equation (1.5) is given in Table 1.7. The correctness of Equations (1.9) to (1.12) can be seen by writing the truth table.

Table 1.7. Truth table for distributive law

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

1.6.4 OR Laws

If we study OR gate, the following laws become self evident

$$A + A = A \quad \ldots(1.14)$$

$$A + 1 = 1 \quad \ldots(1.15)$$

$$A + 0 = A \quad \ldots(1.16)$$

$$A + \overline{A} = 1. \quad \ldots(1.17)$$

Fig. 1.13(a) illustrates Eqn. from (1.14). If A = 0, output is 0 and if A = 1, output is 1. Thus any variable OR ed with itself equals the variable.

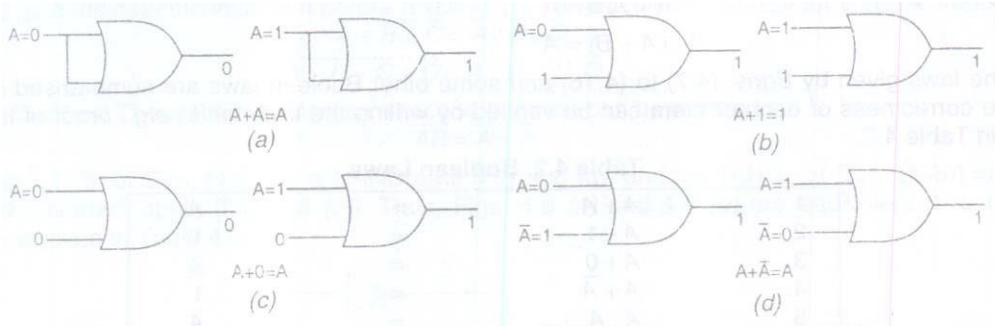


Fig 1.13 OR laws Book-1 Page 67

Fig. 1.13 (b) shows Eqn. (1.15). If one of the inputs to an OR gate is 1, the output is 1 irrespective of whether the other variable is 0 or 1.

Fig. 1.13 (c) shows Eqn. (1.16) where a variable A si ORed with 0. Any variable ORed with 0 equals the variable. If A = 0, output = 0 and A = 1, output = 1

Fig 1.3 (d) shows Eqn. (1.17). A variable ORed with its complement always equals 1.

1.6.5 AND Laws

The four AND laws in Boolean Algebra are

$$A.A = A \quad \ldots(1.18)$$

$$A.\ 1 = A \quad \ldots(1.19)$$

$$A.\ 0 = 0 \quad \ldots(1.20)$$

$$A.\ \overline{A} = 0 \quad \ldots(1.21)$$

Fig. 1.14 (a) illustrates Eqn. (1.18). Both inputs to AND gate are A which can be 0 or 1. In each case the output is equal to A.

Fig. 1.14 (b) illustrates Eqn. (1.19). If A = 0 and the other input is 1, the output is 0. If A = 1 and the other input is 1, the output is 1. Thus in both cases the output is equal to A.

If A = 0 and the other input is 0, the output = 0. If A = 1 and the other input is 0, the output = 0. Thus irrespective of the value of A, the output is 0 thus illustrating Eqn. (1.20) and shown in Fig. 1.14 (c).

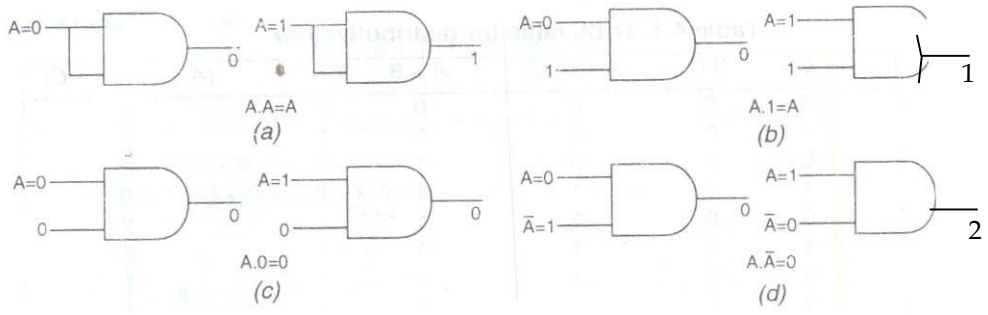


Fig. 1.14 AND laws

In Fig. 1.14 (d) A is ANDed with its complement $\overline{A}$. If A = 0, $\overline{A}$ = 1 and output is 0. If A = 1. $\overline{A}$ = 0 and output is again 0. Thus output is 0 irrespective of value of A

1.6.6 Double Inversion

The double inversion rule is

$$\overline{\overline{A}} = A$$

i.e., double complement of a variable equals the variable. This is illustrated in Fig. 1.15.

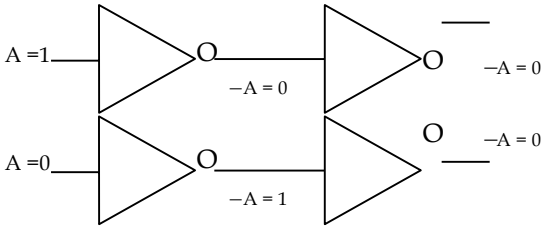


Fig. 1.15 Double inversion

1.6.7 Redundancy Law

$$A + A.B = A \quad \ldots(1.22)$$

$$A.(A + B) = A \quad \ldots(1.23)$$

The laws given by Eqns. (1.22) to (1.23) and some other Boolean laws are summarized in Table 1.8. The correctness of each of them can be verified by writing the truth table, e.g., proof of law 11 is shown in Table 1.9.

Table 1.8 Boolean Laws

1.	A + A	=	A
2.	A + 1	=	1
3.	A + 0	=	A
4.	A + A	=	1
5.	A. A	=	A
6.	A. 1	=	A
7.	A. 0	=	0

8.	$A.\ \overline{A}$	=	0
9.	$\overline{\overline{A}}$	=	A
10.	A + A. B	=	A
11.	A(A + B)	=	A
12.	$A(\overline{A} + B)$	=	A . B
13.	$A + \overline{A}.\ B$	=	A + B
14.	$\overline{A} + A.\ B$	=	$\overline{A}$+ B
15.	$\overline{A} + A.\ \overline{B}$	=	$\overline{A} + \overline{B}$

It is seen from Table 1.9 that law 7 can be obtained from law 2 by replacing + by $*$ and 1 by 0. Similarly law 6 can be obtained from law 3 by replacing + by. and 0 by 1. Such laws are known as dual laws.

Table 1.9 Proof of Boolean law 11 of Table 1.8

A	B	A + B	A(A + B)
0	0	0	0
1	0	1	1
0	1	1	0

1.7 DE MORGAN'S THEOREMS

(a) **First Theorem** : DE Morgan's first theorem is

$$\overline{A + B} = \overline{A}, \overline{B} \qquad \ldots(1.24)$$

The L.H.S. of Eqn. (1.24) is a NOR gate [Fig 1.16 (a)]. In the R.H.S. of Eqn. (4.17) the inputs are first inverted and then fed to the AND gate [Fig. 1.16 (b)]. Thus Figs. 1.16 (a) and 1.16 (b) are equivalent. Proof of this theorem is given in Table 1.10.

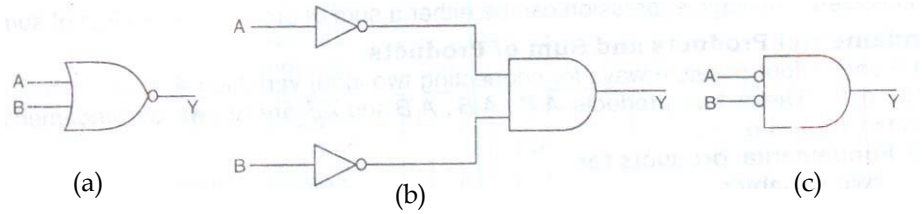


Fig 1.16. (a and b) De Morgan's first theorem (c) symbol for bubbled AND gate

Table 1.10. Proof of De Morgan's first theorem

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

Fig. 1.16 (b) is also known as bubbled AND gate and is shown in Fig. 1.16 (c). The bubbles before he AND gate indicate the inversion before ANDing. De Morgan's first theorem for 3 and 4 inputs is

$$\overline{A+B+C} = \overline{A}.\,\overline{B}.\,\overline{C} \qquad \ldots(1.25)$$

$$\overline{A+B+C+D} = \overline{A}.\,\overline{B}.\,\overline{C}.\,\overline{D} \qquad \ldots(1.26)$$

(b) **Second Theorem :** De Morgan's second theorem can be written as

$$\overline{AB} = \overline{A} + \overline{B} \qquad \ldots(1.27)$$

The L.H.S. of Eqn. (1.27) is a NAND gate [Fig. 1.17 (a)] and the R.H.S. of Eqn. (1.27) is an OR gate with inverted inputs [Fig. 1.17 (b)]. Thus, Figs. 1.17 (a) and 1.17 (b) are equivalent. Proof of this theorem is given in Table 4.5.

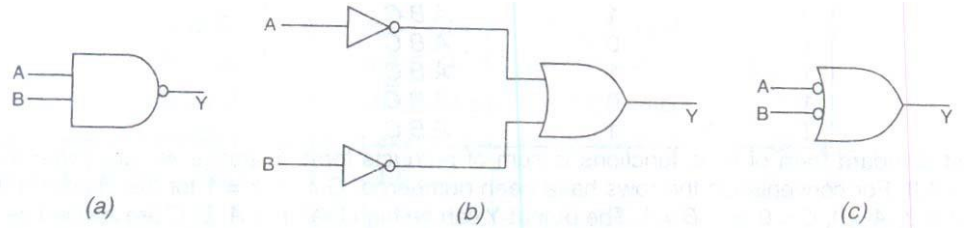
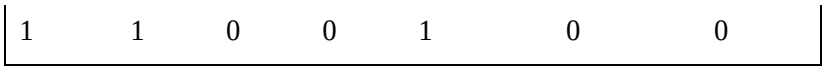


Fig. 1.17. De Morgan's second theorem (a) NAND gate (b) OR gate with inverted inputs (c) symbol for bubbled OR gate

Table 1.11. Proof of De Morgan's second theorem

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


1 1 0 0 1 0 0

De Morgan's second theorem for 3 and 4 inputs can be written as

$$\overline{ABC} = \overline{A} + \overline{B} + \overline{C} \quad \ldots(1.28)$$

$$\overline{ABCD} = \overline{A} + \overline{B} + \overline{C} + \overline{D} \quad \ldots(1.29)$$

In Fig. 1.17 (b) the inputs are first inverted and then fed to the OR gate. This gate can also be called a bubbled OR gate and is represented by Fig. 1.17 (c). The bubbles indicate the inversion which takes place before ORing.

1.8 SUMMARY

In this chapter, the basic concepts of the digital systems have been discussed. The basic features and advantages of these systems have been given briefly. The level of the treatment has been kept low to avoid any confusion. Table 1.12 summarizes the operation of all the gates introduced in this chapter. For convenience, two input gates have been taken andthe different symbols used for various operations are also given. A brief exposure to Boolean algebra has also been given.

Table 1.11 Summary of logic gates

1.12

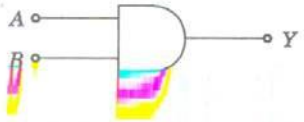
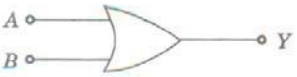
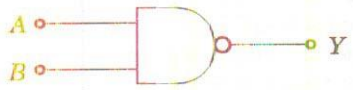
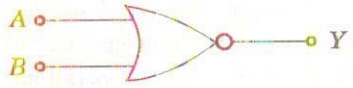
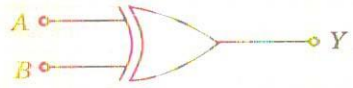
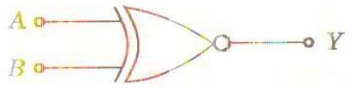
Gate	Logic diagram	Function	Truth table
AND		$Y = A \text{ AND } B$ $= A \cdot B$ $= A \cap B$ $= A \wedge B$ $= AB$	Inputs A B / Output Y 0 0 0 0 1 0 1 0 0
OR		$Y = A \text{ OR } B$ $= A + B$ $= A \cup B$ $= A \vee B$	Inputs A B / Output Y 0 0 0 0 1 1 1 0 1 1 1 1
NOT (inverter)		$Y = \text{NOT } A$ $= \overline{A}$	Input A / Output Y 0 1 1 0

Table 1.11 (*Contd.*)

Gate	Logic diagram	Function	Truth table		
NAND		$Y = A$ NOT AND B $= A$ NAND B $= \overline{A \cdot B}$ $= \overline{A \cap B}$ $= \overline{A \wedge B}$ $= A \uparrow B$ $= \overline{AB}$	Inputs A	B	Output Y
			0	0	1
			0	1	1
			1	0	1
			1	1	0
NOR		$Y = A$ NOT OR B $= A$ NOR B $= \overline{A + B}$ $= \overline{A \cup B}$ $= \overline{A \vee B}$ $= A \downarrow B$	Inputs A	B	Output Y
			0	0	1
			0	1	0
			1	0	0
			1	1	0
EX-OR		$Y = A$ EX-OR B $= A \oplus B$ $= A\overline{B} + \overline{A}B$	Inputs A	B	Output Y
			0	0	0
			0	1	1
			1	0	1
			1	1	0
EX-NOR		$Y = \overline{A \text{ EX-OR } B}$ $= A$ EX-NOR B $= A \odot B$ $= \overline{A\overline{B} + \overline{A}B}$ $= \overline{A}\,\overline{B} + AB$	Inputs A	B	Output Y
			0	0	1
			0	1	0
			1	0	0
			1	1	1

1. Analog signals depict continuous variation of the magnitude over a certain time whereas digital signals depict discrete values at various instants.
2. Analog instruments indicate the magnitude through the position of pointer on the scale. A digital instrument displays the actual magnitude in the form of digits.

3. Digital communication system, Digital control systems and Digital computers are widely used.
4. The representation of analog signal in digital form is by the use of digital signal processor (DSP).
5. We have to feed a program and data to a digital computer so that the computer may process the data and produce the desired output.
6. Digital computers work on binary numbers, i.e., 1 and 0.
7. Digital signals can be represented in positive or negative logic. In positive logic, the more positive level is level 1 and the other is level 0. In negative logic the more negative level is level 1 and the other is level 0. Positive logic is used more commonly.
8. An ideal pulse changes from low to high and high to low levels in zero time. An actual pulse has finite rise and full times.

PROBLEMS

1.1 Which of the following systems are analog and which are digital? Why?

(a) Pressure gauge
(b) An electronic counter used to count persons entering an exhibition
(c) Clinical thermometer
(d) Electronic calculator
(e) Transistor radio receiver
(f) Ordinary electric switch.

1.2 In the circuits of Fig. 1.18, the switches may be On (1) or OFF (0) and will cause the bulb to be ON (1) or OFF (0).

(a) Determine all possible conditions of the switches for the bulb to be ON (1)/ OFF (0) in each of the circuits.
(b) Represent the information obtained in part (a) in the form of truth table.

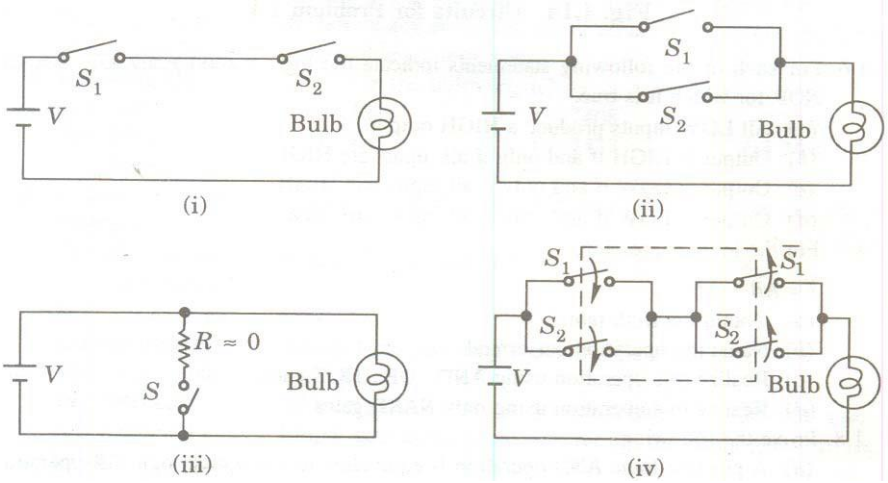


Fig. 1.18. Circuits for Problem 1.2

(c) Name the operation performed by each circuit (refer to Table 1.11).

1.3 The voltage waveforms shown in Fig. 1.19 are applied at the inputs of 2-input AND,

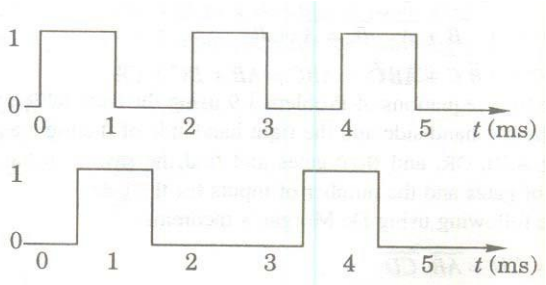


Fig. 1.19 Waveforms for Problem 1.3

OR, NAND, NOR, and EX-OR gates. Determine the output waveform in each case.

1.4 Find the relationship between the inputs and output for each of the gates shown in Fig. 1.19. Name the operation performed in each case.

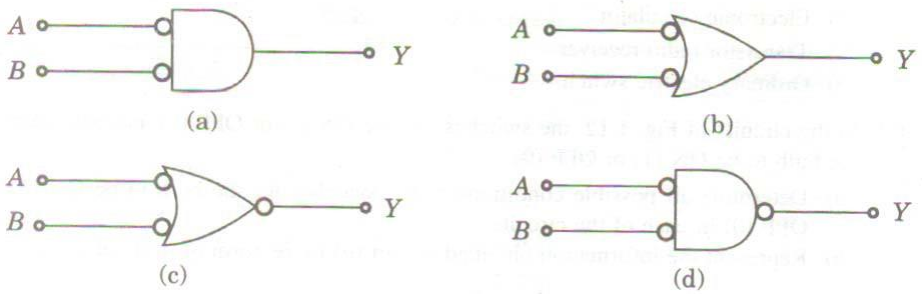


Fig. 1.19 Circuits for Problem 1.4

1.5 For each of the following statements indicate the logic gate(s), AND, OR, NAND, NOR for which it is true.

(a) All Low inputs produce a HIGH output.

(b) Output is HIGH if and only if all inputs are HIGH.

(c) Output is LOW if and only if all inputs are HIGH.

(d) Output is LOW if and only if all inputs are LOW.

1.6 For the logic expression,

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

(a) Obtain the truth table.

(b) Name the operation performed.

(c) Realize this operation using AND, OR, NOT gates.

(d) Realize this operation using only NAND gates.

1.8 Prove the following :

(a) A positive logic AND operation is equivalent to a negative logic OR operation and vice-versa.

(b) A positive logic NAND operation is equivalent to a negative logic NOR operation and vice-versa.

1.9 Prove the following using the Boolean algebraic theorems:

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

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

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

1.10 Prove the logic equations of Problem 1.9 using the truth table approach.

1.11 Realize the left hand side and the right hand side of the logic equations of Problem 1.9 using AND, OR, and NOT gates and find the saving in hardware in each case (number of gates and the number of inputs for the gates).

1.12 (a) Realize the logic equation (a) of Problem 1.12 using

(i) AND and OR gates.

(ii) Only NAND gates.

(b) Realize the logic equation (b) or Problem 1.12 using

(i) OR and AND gates

(ii) only NOR gates.

1.13 Verify that the following operations are commutative and associative

(a) AND (b) OR (c) EX–OR

1.14 Verify that the following operations are commutative but not associative.

(a) NAND (b) NOR

1.15 Realize the logic expression

$Y = A \oplus B \oplus C \oplus D$

using EX–OR gates.

1.16 For a gate with N inputs, how many combinations of inputs are possible? State the general rule to obtain the possible combinations.

1.17 Determine the IC chips required for the implementation of each of the circuits of Problem 1.13.

1.18 Make truth table for a 3-input

(a) AND gate (b) OR gate (c) NAND gate (d) NOR gate

1.19 Is it possible to use a 3-input gate as a 2-input gate for the following gates? If yes, how?

(a) AND (b) OR (c) NAND (d) NOR

1.20 Is it possible to INHIBIT (or DISABLE) AND, OR, NAND, NOR gates? If yes, how?

1.21 One of the inputs of a gate is used to control the operation of the gate and is labeled as ENBLE. Is it active-high or active-low if the gate is

(a) AND? (b) OR? (c) NAND? (d) NOR?

1.22 Is the INHIBIT input active-high or active-low in Prob. 1.24.

1.23 Realize a 3-input gate using 2-input gates for the following gates:

(a) AND (b) OR (c) NAND (d) NOR

1.24 Prove the following :

(a) $A \oplus B = \overline{A} \oplus \overline{B}$

(b) $\overline{A \oplus B} = A \oplus \overline{B} = \overline{A} \oplus B$

(c) $B \oplus (B \oplus A \cdot C) = A \cdot C$

CHAPTER-2

NUMBER SYSTEM & CODES

2.1 INTRODUCTION

A digital computer is also known as data processor. It processes data while solving a mathematical problem or doing translation from one language to another etc. Before a computer can process data, the data has to be converted into a form acceptable to a computer.

We use decimal number system in our work. This system has digits 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. Computers cannot use these numbers. Instead, a computer works on binary digits.A binary number system has only two digits 0 and 1. This is because of the reason that computers use integrated circuits with thousands of transistors. Due to variation of parameters the behaviour of transistor can be very erratic and quiescent point may shift from one position to another. Nevertheless the cut off and saturation points are fixed. When a transistor is cut off, a large change in values is needed to change the state to saturation Similaris the situation when it is in saturation. Thus a transistor is a very reliable two state device. One state represents digit 0 and the other state represents digit 1. All input voltages are recognized as either 0 or 1.

2.2 DECIMAL NUMBER SYSTEM

We are all familiar with the decimal number system. It uses ten digits (0, 1, 2, 3, 4, 5, 6, 7, 9) and thus its base is 10. The decimal number system of counting was evolved because we have 8 fingers and 2 thumbs on our two hands so that we can count 10. By using the different digits in different positions we can express any number. For numbers bigger than 9 we use two or more digits. The position of each digit in the number indicates the magnitude that this number represents. In the number 27 the digit 7 represents $7 \times 10°$ or 7 and the digit2 represents 2×10^1 or 20. The sum of 7 and 20 makes 27. Similarly the number 263 can be expressed as

Decimal $263 = (2 \times 10^2) + (6 \times 10^1) + (3 \times 10^0) = 200 + 60 + 3 = 263$.

Since the base in decimal number system is 10, the number 263 can be written as 263_{10}. The suffix 10 emphasizes the fact that the base is 10.

2.3 BINARY NUMBER SYSTEM

The binary number system has only two digits 0 and 1. Thus a binary number is a string of zeros and ones. Since it has only two digits, the base is 2.

The abbreviation of binary digit is bit. The binary number 1100 has 4 bits, 101011 has 6 bits and 11001010 has 8 bits. Each bit may represent either 0 or 1. A string of 8 bits is known as a byte. A byte is the basic unit of data in computers. In most computers, the data* is processed in strings of 8 bits or some multiples (i.e., 16, 24, 32 etc.). The computer memory also stores data in strings of 8 bits or multiples of 8 bits. Table 2.1 shows the 16 combinations of a 4 bit binary word.

Table 2.1. Binary, decimal, hexadecimal and octal equivalence

Binary				**Decimal**	**Hexadecimal**	**Octal**
0	0	0	0	0	0	0
0	0	0	1	1	1	1
0	0	1	0	2	2	2
0	0	1	1	3	3	3
0	1	0	0	4	4	4
0	1	0	1	5	5	5
0	1	1	0	6	6	6
0	1	1	1	7	7	7
1	0	0	0	8	8	10
1	0	0	1	9	9	11
1	0	1	0	10	A	12
1	0	1	1	11	B	13
1	1	0	0	12	C	14
1	1	0	1	13	D	15
1	1	1	0	14	E	16
1	1	1	1	15	F	17

It is interesting to note that our earlier system of counting and weighing was basically a binary system. Two Annas make one Two Annas. Two two annas make one Chawani

* Data means the names, numbers etc. needed to solve a problem.

(quarter of a rupee) and so on. Similar two Chhattaks (one sixteenth of seer) make one aad pao (quarter seer). Two aad pao make one pao (quarter seer) and so on.

As in the decimal system the binary number system is positionally weighted. The digital on the extreme right hand side has a weight of 2^0, the next one has a weight of 2^1 andso on. Since the base is 2, the binary number is written as (say) 1011_2. The suffix 2 emphasizes the fact that base is 2. If the number of binary digits is n, the highest decimal number which can be counted is 2^n–1. Thus with 4 binary digits we can count upto (2^4 – 1) or15 decimal number. If n = 8 we can count upto (2^8 – 1) = 255 decimal number.

2.3.1 Binary to Decimal Conversion

The procedure to convert a binary number to decimal is called dibble–dabble method. We start with the left hand bit. Multiply this value by 2 and add the next bit. Again multiply by 2 and add the next bit. Stop when the bit on extreme right hand side is reached.

An other fast and easy method to convert binary number to decimal number is as under:

1. Write the binary number.
2. Write the weights 2^0, 2^1, 2^2, 2^3 etc., under the binary digits starting with the bit on right hand side.
3. Cross out weights under zeros.
4. Add the remaining weights.

Example 2.1. Convert 100101_2 to decimal.

Solution : Left hand bit 1

Multiply by 2 and add next bit $2 \times 1 + 0 =$ 2

Multiply by 2 and add next bit $2 \times 2 + 0 =$ 4

Multiply by 2 and add next bit $2 \times 4 + 1 =$ 9

Multiply by 2 and add next bit $2 \times 9 + 0 = 18$

Multiply by 2 and add next bit $2 \times 18 + 1 = 37$

Therefore, $100101_2 = 37_{10}$

Example 2.2 Convert 1101_2 into equivalent decimal number.

Solution :	1	1	0	1	Binary number
	8	4	2	1	Write weights

8	4	2	1	Cross out weights under zeros
8 +	4 +	0 +	1 = 13	Add weights

Therefore, $1101_2 = 13_{10}$

Example 2.3. Convert 110011011001_2 into equivalent decimal number.

Solution :

1	1	0	0	1	1	0	1	1	0	0	1	Binary number
2048	1024	512	256	128	64	32	16	8	4	2	1	Write weights
2048	1024	512	256	128	64	32	16	8	4	2	1	Cross out weights under zeros

2048 + 1024 + 128 + 64 + 16 + 8 + 1 = 3289 Add weights

Thus $110011011001_2 = 3289_{10}$

2.3.2 Decimal to Binary Conversion

A systematic way to convert a decimal number into equivalent binary number is known as double dabble. This method involves successive division by 2 and recording the remainder (the remainder will be always 0 or 1). The division is stopped when we get a quotient of 0 with a remainder of 1. The remainders when read upwards give the equivalent binary number.

Example 2.4 Convert decimal number 10 into its equivalent binary number.

2	10	
2	5	remainder 0
2	2	remainder 1
2	1	remainder 0
	0	remainder 1

The binary number is 1010.

Example 2.5. Convert decimal number 25 into its binary equivalent.

Solution :

2	25	
2	12	remainder 1
2	6	remainder 0
2	3	remainder 0
		239

2	1	remainder 1
	0	remainder 1

The binary number is 11001.

2.3.3 BINARY ARITHMETIC

The rules for addition of binary numbers are as under :

0 + 0 = 0

0 + 1 = 1 + 0 = 1

1 + 1 = 10 i.e., 1 + 1 equals 0 with a carry of 1 to next higher column

1 + 1 + 1 = 11 i.e., 1 + 1 + 1 equals 1 with a carry of 1 to next higher column.

2.3.3.1. Binary Subtraction

The rules for subtraction of binary numbers are as under :

0 – 0 = 0

1 – 0 = 1

1 – 1 = 0

10 – 1 = 1

In both the operations of addition and subtraction, we start with the least significant bit (L.S.B.), i.e., the bit on the extreme right hand side and proceed to the left (as is done in decimal addition and subtraction).

Example 2.6. (a) Convert decimal numbers 15 and 31 into binary numbers. (b) Add the binary numbers and convert the result into decimal equivalent.

Solution : (a)

2	15	
2	7	remainder 1
2	3	remainder 1
2	1	remainder 1
	0	remainder 1

Binary number is 1111

2	31	
2	15	remainder 1
2	7	remainder 1
2	3	remainder 1
2	1	remainder
	0	remainder 1

Binary number is 11111

(b) Binary addition

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

The sum of binary numbers 1111 and 11111 is the binary number 101110

1	0	1	1	1	0	Binary number
32	16	8	4	2	1	Write weights
32	16	8	4	2	1	Cross cut weights under zero
		32 + 8 + 4 + 2 = 46				Add weights

Example 2.7. Subtract 10001 from 11001.

Solution :

	1	1	0	0	1		25
(–1)	1	0	0	0	1	(–)	17
Results		1	0	0	0		8

Example 2.8. Subtract 0111 from 1010.

Solution :

	1	0	1	0		10
(–1)	0	1	1	1	(–1)	7
	0	0	0	1		3

The least significant digit.* in the first number is 0. So we borrow 1 from the next digit and subtract 1 to give 1. Now in the second column we have 0, so we again borrow 1 from the next higher column and subtract 1 to give 1. In the third column, we borrow 1 from the next higher column and 1–1 gives 0. In the fourth column, 0 (after lending) – 0 gives 0.

2.3.3.2 Binary Multiplication

The four basic rules for binary multiplication are :

$0 \times 0 = 0$

$0 \times 1 = 0$

$1 \times 0 = 0$

$1 \times 1 = 1$

* The digit on the extreme RHS is the LSB and the digit on extreme LHS is the MSB (most significant digit)

The method of binary multiplication is similar to that in decimal multiplication. The method involves forming partial products, shifting successive partial products left one place and then adding all the partial products.

2.3.3.4 Binary Division

The division in binary system follows the same long division procedure as in decimal system.

Example 2.9. (a) Divide 110110_2 by 101.

(b) Convert 110110_2 and 101_2 into equivalent decimal number obtain division, convert results into binary and compare the results with those in part (a).

Solution : (a)

```
          1 0   1 0     1010 quotient
101 ) 1 1 0 1 1 0
      1 0 1
      -----------
        1 1 1
        1 0 1
        ---------
          1 0 0       remainder
```

(b)							
	1	1	0	1	1	0	Binary number
	32	16	8	4	2	1	Write weights
	32	16	6	4	2	1	Cross out weights under zero
	32 + 16 + 4 + 2 = 54						Add weights
				1	0	1	Binary number
				4	2	1	Write weights
				4	2	1	Cross out weights under zero
				4 + 1 = 5			Add weights

```
5 | 54
  |______
   10 – 4
```

quotient remainder

2	10	
2	5	remainder 0
2	2	remainder 1
2	1	remainder 0
	0	remainder 1

2	4	
2	2	remainder 0
2	1	remainder 0
	0	remainder 1

The quotient is 1010_2 and the remainder is 100_2. These are the same as in part (a)

2.4 SIGNED BINARY NUMBERS

To represent negative numbers in the binary system, digit 0 is used for the + sign and 1 for the –ve sign. The most significant bit is the sign bit followed by the magnitude bits. Numbers expressed in this manner are known as signed binary numbers.* The numbers may be written in 4 bits, 8 bits, 16 bits, etc. In every case, the leading bit represents the sign and the remaining bits represent the magnitude.

Example 2.10 Express in 16–bit signed binary system : (a) + 8, (b) –8, (c) 165, (d) –165.

Solution : (a)

2	8	
2	4	remainder 0
2	2	remainder 0
2	1	remainder 0
	0	remainder 1

The binary number is 1000.

For the 16 bit system, we use 16 bits, 0 (which stands for +) in the leading position, 1000 in the last 4 bits and 0 in the remaining 11 positions. So the signed 16 bit binary number is

+ 8 = 0000 0000 0000 1000

(b) In the leading bit we will have 1 (to represent the '–' sign). The rest of the representation is the same s in part (a).

* Signed binary numbers are also known as sign–magnitude numbers.

– 8 = 1000 0000 0000 1000

2	165	
2	82	remainder 1
2	41	remainder 0
2	20	remainder 1
2	10	remainder 0
2	5	remainder 0
2	2	remainder 1
2	1	remainder 0
	0	remainder 1

So the number is 10100101.

Using 0 in the leading bit (for + sign) the 16 bit signed binary number is

+ 165 = 0000 0000 1010 0101

(d) In the leading bit position we will have 1 (for the '–' sign).

Therefore, – 165 = 1000 0000 1010 0101

2.5 1'S COMPLEMENT

The 1's complement of a binary number is obtained by complementing each bit (i.e., 0 for 1 and 1 for 0).

Thus each bit in the original word is inverted to give the 1's complement. For example, for the number

1 1 0 0 1 0 0 1

1's complement is 0 0 1 1 0 1 1 0

2.6 2'S COMPLEMENT

The signed binary numbers required too much electronic circuitry for addition and subtraction. Therefore, positive decimal numbers are expressed in sign–magnitude form but negative decimal numbers are expressed in 2's complements.

2's complement is defined as the new word obtained by adding 1 to 1's complement* e.g.,

Let A = 0 1 0 1 i.e., 5

* 1′s complement of A is denoted by $\bar{A}$ and 2′s complement of A is denoted by A′.

1's complement $\quad \overline{A} = 1\ 0\ 1\ 0$

$\quad\quad\quad\quad\quad\quad + \quad\quad\quad 1$

2's complement $\quad A' = 1\ 0\ 1\ 1$ i.e., –5

Taking the 2's complement is the same as changing the sign of the given binary number. If we take the 2's complement twice we get the original number, e.g.,

$\quad\quad\quad\quad\quad\quad A' = 1\ 0\ 1\ 1$

1's complement $\quad \overline{A} = 0\ 1\ 0\ 0$

$\quad\quad\quad\quad\quad\quad + \quad\quad\quad 1$

2's complement $\quad A'' = 0\ 1\ 0\ 1 = A$

Thus $A'' = A$. In view of this every number and its 2's complement form a complementary pair. In a typical computer positive numbers are expressed in sign magnitude form but negative numbers are expressed as 2's complements. The positive numbers have a leading sign bit of 0 and negative numbers have a leading sign bit of 1.

2.6.1 2'S COMPLEMENT ADDITION, SUBTRACTION

The use of 2's complement representation has simplified the computer hardware* for arithmetic operations. When A and B are to be added, the B bits are not inverted so that we get

$$S = A + B \qquad \ldots(2.1)$$

When B is to be subtracted from A, the computer hardware forms the 2's complement of B and then adds it to A. Thus

$$S = A + B' = A + (-B) = A - B$$

Eqns. (2.1 and 2.2) represent algebraic addition and subtraction. A and B may represent either positive or negative numbers. Moreover, the final carry has no significance and is not used.

2.7 BINARY FRACTIONS

So far we have discussed only whole numbers. However, to represent fractions is also important. The decimal number 2568 is represented as

$$2568 = 2000 + 500 + 60 + 8 = 2 \times 10^3 + 5 \times 10^2 + 6 \times 10^1 + 8 \times 10^0$$

---

* Hardware means electronic, mechanical and magnetic devices in a computer. The computer program is known as software.

Similarly, 25.68 can be represented as

$$25.68 = 20 + 5 + 0.6 + 0.08 = 2 \times 10^1 + 5 \times 10^0 + 6 \times 10^{-1} + 8 \times 10^{-2}$$

2.7.1 Conversion of Binary to Decimal

In the binary system, the weights of the binary bits after the binary point, can be written as

$$0.1011 = 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4}$$

$$= 1 \times \frac{1}{2} + 0 \times \frac{1}{4} + 1 \times \frac{1}{8} + 1 \times \frac{1}{16}$$

$$= 0.5 + 0 + 0.125 + 0.0625 = 0.6875 \text{ (decimal)}$$

Example 2.11 Express the number 0.6875 into binary equivalent

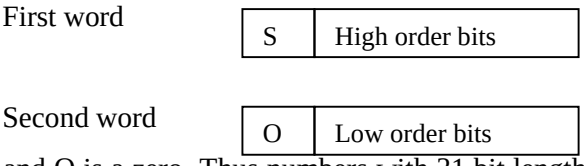
Solution :

Fraction	Fraction × 2	Remainder new fraction	Integer
0.6875	1.375	0.375	1 (MSB)
0.375	0.75	0.75	0
0.75	1.5	0.5	1
0.5	1	0	1(LSB)

The binary equivalent is 0.1011.

2.8 DOUBLE PRECISION NUMBERS

Most of the computers used in today's world are 16 bit or more. In these computers the numbers from +32, 767 to – 32,768 can be stored in each register. To store numbers greater than these numbers, double precision system is used. In this method two storage locations are used to represent each number. The format is



S is the sign bit and O is a zero. Thus numbers with 31 bit length can be represented in 16 bit registers. For still bigger numbers triple precision can be used. In triple precision 3 word lengths (each 16 bit) is used to represent each number.

2.9 FLOATING POINT NUMBERS

Most of the time we use very small and very large numbers, e.g., 1.02×10^{-12} and $6.5 \times 10^{+17}$. In binary representation, the numbers are expressed by using a mantissa and an exponent.

The mantissa has a 10 bit length and exponent has 6 bit length. Fig. 2.1 shows one such representation.

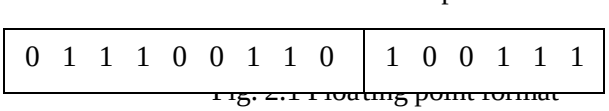


Fig. 2.1 Floating point format

The left most bit of mantissa is sign bit. The binary point is to the right of this sign bit.

The 6 bit exponent has a base of 2. The exponent can represent numbers 0 to 63. To express negative exponents the number 32_{10} (i.e., 100000_2) has been added to the exponent. It is known as excess –32 notation and is a common floating point format. Examples of exponent in excess – 32 format are

Table 2.2

Actual exponent	**Binary representation in excess–32 format**
– 32	000000
–1	011111
0	100000
+7	100111
+15	101111
+31	111111

The number represented in Table 2.2 is

Mantissa + 0.111001101

Exponent 100111

Subtracting 100000 from exponent, we get 000111. The number is

$$0.111001101_2 \times 2^7 = 1110011.01_2 = 115.25_{10}$$

Example 2.12. What does the floating point number 01101000000010101 represent.

Solution : Mantissa is +0. 110100000

Exponent is 010101

Subtracting 100000 from exponent, we get 110110101.

The given number is $+ 0.110100000 \times 2^{-11} = + 0.0000000000110100000_2$

$= + 0.000396728_{10}$

The advantage of floating point representation is that very large and very smallnumbers can be easily expressed. Since the above representation uses 10 bit long mantissa, the accuracy in above representation is 9 bit since 1 bit is used for sign). Fixed point 16 bit numbers are accurate to 15 bits. Thus breaking the 16 bit lengths into mantissa and exponent (to use floating point representation) reduces the accuracy to some extent. To ensure maximum accuracy the computers normalize the result of any floating point operation. In this process the most significant bit is placed next to the sign bit.

Example 2.13. Add the binary numbers 1 0 1 1 0 1. 0 1 0 1 and 1 0 0 0 1. 1 0 1.

Solution :

```
    1 0 1 1 0 1  .  0 1 0 1
+     1 0 0 0 1  .  1 0 1
-------------------------------
    1 1 1 1 1 0  .  1 1 1 1
```

Example 2.14. Convert the binary number 1 1 0 0 1. 0 0 1 0 1 1 to decimal.

Solution : The decimal equivalent is obtained as under

1	1	0	0	1	.	0	0	1	0	1	1	
2^4	2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	weights

Decimal equivalent = $1 \times 2^4\ 1 \times 2^3 + 1 \times 2^0 + 1 \times 2^{-3} + 1 \times 2^{-5} + 1 \times 2^{-6}$

$= 16 + 8 + 1 + 0.125 + 0.03125 + 0.015625 = 25.171875.$

2.10 OCTAL NUMEBR SYSTEM

The number system with base (or radix) eight is known as the octal number system. In this system, eight symbols, 0, 1, 2, 3, 4, 5, 6 and 7 are used to represent numbers. Similar to decimal and binary number systems, it is also a positional system and has, in general, two parts: integer and fractional, set apart by a radix (octal) point (·). For example, $(6327.4051)_8$ is an octal number. Using the weights it can be written as

$$(6327.4051)_8 = 6 \times 8^3 + 3 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 + .4 \times 8^{-1}$$

$$+ 0 \times 8^{-2} + 5 \times 8^{-3} + 1 \times 8^{-4}$$

$$= 3072 + 192 + 16 + 7 + \frac{4}{8} + 0 + \frac{5}{512} + \frac{1}{4096}$$

$$= (3287.5100098)_{10}$$

Thus, $(6327.4051)_8 = (3287.5100098)_{10}$

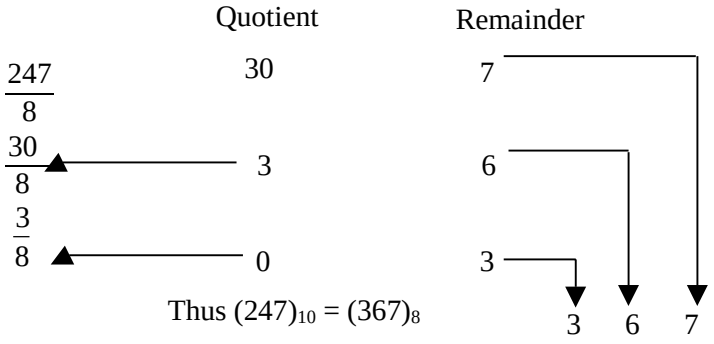
Using the above procedure, an octal number can be converted into an equivalent decimal number or a base -8 number can be converted into an equivalent base-10 number.

The conversion from decimal to octal (base-10 to base-8) is similar to the conversion procedure for base-10 to base-2 conversion. The only difference is that number 8 is used in place of 2 for division in the case of integers and for multiplication in the case of fractional numbers.

Example 2.15.

(a) Convert $(247)_{10}$ into octal

Solution (a)



Thus $(247)_{10} = (367)_8$

2.11 HEXADECIMAL NUMBER SYSTEM

Hexadecimal number system is very popular in computer uses. The base for hexadecimal number system is 16 which requires 16 distinct symbols to represent the numbers. These are numerals 0 through 9 and alphabets A through F. Since numeric digits and alphabets both are used to represent the digits in the hexadecimal number system,therefore, this is an alphanumeric number system. Table 2.3 gives hexadecimal numbers with their binary equivalents for decimal numbers 0 through 15. From the table, it is observed that there are 16 combinations of 4-bit binary numbers and sets of 4-bit binary numbers can be entered in the computer in the form of hexadecimal (hex.) digits. These numbers are required

to be converted into binary representation, using hexadecimal-to-binary converter circuits before these can be processed by the digital circuits.

Table 2.3 Binary and decimal equivalents of hexadecimal numbers

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

Example 2.16. Obtain decimal equivalent of hexadecimal number $(3A.2F)_{16}$

Solution : $(3A.\ 2F)_{16} = 3 \times 16^1 + 10 \times 16^0 + 2 \times 16^{-1} + 15 \times 16^{-2}$

$$= 48 + 10 + \frac{2}{16} + \frac{15}{16^2}$$

$$= (58.1836)_{10}$$

2.11.1 Decimal-to-Hexadecimal Conversion

The conversion from decimal to hexadecimal, the procedure used in binary as well as octal systems is applicable, using 16 as the dividing (for integer part) and multiplying (for fractional part) factor.

2.11.2 Hexadecimal-to-Binary Conversion

Hexadecimal numbers can be converted into equivalent binary numbers by replacing each hex digit by its equivalent 4–bit binary number.

Example 2.17. Convert $(2F9A)_{16}$ to equivalent binary number.

Solution : Using Table 2.7, find the binary equivalent of each hex digit.

$$(2F9A)_{16} = (0010\ 1111\ 1001\ 1010)_2$$
$$= (0010111110011010)_2$$

2.11.3 Binary-to-Hexadecimal Conversion

Binary number can be converted into the equivalent hexadecimal numbers by making groups of four bits starting from LSB and moving towards MSB for integer part and then replacing each group of four bits by its hexadecimal representation.

For the fractional part, and above procedure is repeated starting from the bit next to the binary point and moving towards the right.

Example 2.18 Convert the binary numbers of Example 2.24 to hexadecimal numbers.

Solution :

(a) $110\ 0111\ 0001.\ 0001\ 0111\ 1001 = (671.179)_{16}$

(b) $10\ 1101\ 1110.1100\ 1010\ 011 = (2DE.\ CA6)_{16}$

(c) $1\ 1111\ 0001.1001\ 101 = (1F1.99A)_{16}$

From the above examples, we observe that in forming the 4-bit groupings 0's may be required to complete the first (most significant digit) group in the integer part and the last (least significant digit) group in the fractional part.

2.11.4 Conversion from Hex-to-Octal and Vice-Versa

Hexadecimal numbers can be converted to equivalent octal numbers and octal numbers can be converted to equivalent hex numbers by converting the hex/octal number to equivalent binary and then to octal/hex, respectively.

Example 2.19 Convert the following hex numbers to octal numbers.

(a) A72E (b) 0.BF85

Solution :

(a) $(A72E)_{16} = (1010\ 0111\ 0010\ 1110)_2$

$= 001\ 010\ 011\ 100\ 101\ 110$

$= (1123456)_8$

$= 101\ 111\ 111\ 000\ 010\ 100$

$= (0.577024)_8$

Example 2.20 Convert $(247.36)_8$ to equivalent hex number.

Solution :

$(247.36)_8 = (010\ 100111.011\ 110)_2$

$= (0\ 1010\ 0111.0111\ 1000)_2$

$= (A\ 7.78)_{16}$

2.11.5 Hexadecimal Arithmetic

The rules for arithmetic operations with hexadecimal numbers are similar to the rules for decimal, octal and binary systems. The information can be handled only in binary form in a digital circuit and it is easier to enter the information using hexadecimal number system. Since arithmetic operations are performed by the digital circuits on binary numbers, therefore hexadecimal numbers are to be first converted into binary numbers. Arithmetic operations will become clear from the following examples.

Example 2.21 Add $(7F)_{16}$ and $(BA)_{16}$

Solution :

7F = 01111111

(+)BA = 10111010

$(139)_{16} = 1\ 00111001$

Example 2.22

Subtract (a) $(5C)_{16}$ from $(3F)_{16}$

(b) $(7A)_{16}$ from $(C0)_{16}$

Solution

(a) 3F = 00111111

– 5C = (+) 10100100 Two's complement of $(5C)_{16}$

1D = 11100011 Two's complement of result

Two's complement of 11100011 = 0001 1101 = $(1D)_{16}$

(b) C0 = 11000000

$-7A = (+)\ 10000110$ Two's complement of $(7A)_{16}$

$\overline{46} = \overline{101000110}$

Discard carry ↑

Multiplication and division can also be performed using the binary representation of hexadecimal numbers and then making use of multiplication and division rules of binary numbers.

2.12 CODES

Computers and other digital circuits process data in the binary format. Various binary codes are used to represent data which may be numeric, alphabets or special characters. Although, in every code used the information is represented in binary form, the interpretation of this binary information is possible only if the code in which this information is available is known. For example, the binary number 1000001 represents 65 (decimal) in straight binary, 41 (decimal) in BCD and alphabet A in ASCII code. A user must be very careful about the code being used while interpreting information available in the binary format. Codes are also used for error detection and error correction in digital systems.

2.12.1 BINARY CODED DECIMAL (BCD)

Computers work with binary numbers. We work with decimal numbers. A code is needed to represent decimal numbers and binary numbers.

A weighted binary code is one in which each number carries a certain weight. A string of 4 bits is known as nibble. Binary coded decimal (BCD) means that each decimal digit is represented by a nibble (binary code of 4 digits). Main BCD codes have been proposed, e.g., 8421, 2421, 5211, X53. Out of these 8421 code is the most predominant BCD code. The designation 8421indicates the weights of the 4 bits (8, 4, 2 and 1 respectively starting from the left most bit). When one refers to a BCD code, it always means 8421 code. Though 16 numbers (2^4) can be represented by 4 bits, only 10 of these are used. Table 2.4 shows the BCD code. The remaining 6 combinations, i.e., 1010, 1011, 1100, 1101, 1110 and 1111 are invalid in 8421 BCD code. To express any number in BCD code, each decimal number is replaced by the appropriate four bit code of Table 2.4. BCD code is used in pocket calculators, electronic counters, digital voltmeters, digital clocks etc. Early versions of computers also used BCD code. However, the BCD code was discarded for computers

because this code is slow and more complicated than binary. Table 2.5 shows some decimal numbers and their representation in octal, hexadecimal, binary and BCD systems.

Table 2.4. 8421 BCD code

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

Table 2.5. Number systems

Decimal	Octal	Hexadecimal	Binary	8421 BCD
0	0	0	0000 0000	0000 0000 0000
1	1	1	0000 0001	0000 0000 0001
2	2	2	0000 0010	0000 0000 0010
3	3	3	0000 0011	0000 0000 0011
4	4	4	0000 0100	0000 0000 0100
5	5	5	0000 0101	0000 0000 0101
6	6	6	0000 0110	0000 0000 0110
7	7	7	0000 0111	0000 0000 0111
8	10	8	0000 1000	0000 0000 1000
9	11	9	0000 1001	0000 0000 1001
10	12	A	0000 1010	0000 0001 0000
11	13	B	0000 1011	0000 0001 0001
12	14	C	0000 1100	0000 0001 0010

13	15	D	0000 1101	0000 0001 0110
14	16	E	0000 1110	0000 0001 0100
15	17	F	0000 1111	0000 0001 0101
16	20	10	0001 0000	0000 0001 0110
32	40	20	0010 0000	0000 0011 0010
64	100	40	0100 0000	0000 0110 0000
128	200	80	1000 0000	0001 0010 1000
200	310	C8	1100 1000	0010 0000 0000
255	377	FF	1111 1111	0010 0101 0101

2.12.1.1 BCD ADDITION

Addition is the most important arithmetic operation. Subtraction, multiplication and division can be done by using addition. The rules for BCD addition are :

1. Add the two numbers using binary addition (section 2.5). If the four bit sum is equal or less than 9(i.e., equal to or less than 1001) it is a valid BCD number.
2. If the four bit sum is more than 9 or a carry is generated from the group of 4 bits, the result is invalid. In such a case add 6(i.e., 0110) to the four bit sum to skip the 6 invalid states. If a carry is generated when adding 6, add the carry to the next four bit group.

Example 2.22. Represent the following decimal numbers in BCD and add (a) 5 and 4 (b) 7 and 6 (c) 15 and 17 (d) 131 and 162 (e) and 53.

Solution : (a) 5 0101 (b) 7 0111

\+ 4 0100 + 6 0110

9_{10} 1001 13_{10} 1101 invalid

0110

10011

(c) 15 0001 0101

\+ 17 0001 0111

\+ 32 0010 1100 Left group is valid, right group is invalid. Add 6

to the right
group and
carry to the
left group

$$\begin{array}{r} 0110 \\ \hline 0011\ 0010 \end{array}$$

(d) $\begin{array}{rr} 131 & 0001\ 0011\ \ 0001 \\ +\ \ 162 & \underline{0001\ 0110\ 0010} \\ \hline 293_{10} & 0010\ 1001\ 0011 \end{array}$

(e) $\begin{array}{rr} 67 & 0110\ 0111 \\ +\ 53 & 0101\ 0011 \\ \hline 120_{10} & \overline{10111010} \end{array}$

$$\begin{array}{r} 10111010 \\ \hline 0001\ 0010\ 0000 \end{array}$$

Both groups are invalid. Add 6 to each and add carry to next group

2.12.2 GRAY CODE

It is an unweighted code. The bit positions do not have any specific weights assigned to them. However, the most important characteristic of this code is that only a signal bit change occurs when going from one code number to next. (In binary systems all the 4 bits change when we go from 0111 to 1000. i.e., 7_{10} to 8_{10}). The single bit change property is important in some applications, e.g., shaft position encoders. In these applications the chances of error increase if more than one bit change occurs. Table 2.6 shows the 4 bit gray code.

It is seen in Table 2.6 that in gray code change is by 1 bit only at one times. Like binary Gray code can have any number of bits.

Table 2.6 Gray Code

Decimal	Binary	Gray code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101

2.12.2.1. Binary to Gray Conversion

The rules for changing binary number into equivalent Gray code are :

1. The left most bit (most significant bit) in Gray code is the same as the left most bit in binary

   1 0 1 1 Binary

   ↓

   1 Gray

2. Add the left most bit to the adjacent bit

   1 + 0 1 1

   1 1

3. Add the next adjacent pair

   1 0 + 1 1

   1 1 1 0

4. Add the next adjacent pair and discard carry

   1 0 1 + 1

   1 1 1 0

5. Continue the above process till completion.

2.12.2.2 Gray to Binary Conversion

the method to convert from Gray code to binary is an under:

1. Left most bit in binary is the same as the left most bit in Gray code.

   1 1 0 1 1 Gray

   ↓

   1 Binary

2. Add the binary MSB to the Gray digit in the adjacent position. Discard carry

   1 1 0 1 1 Gray

   ↓↗

   1 0 Binary

3. Add the binary digit generated in step 2 to the next Gray digit. Discard carry

   1 1 0 1 1 Gray

   ↓ ↗

1 0 0 Binary

4. Continue the above process till all the digits are covered. Discard carry in each case

1 1 0 1 1 Gray

↓ ↗

1 0 0 1 0 Binary

2.12.3 EXCESS 3 CODE

Excess 3 is a digital code obtained by adding 3 to each decimal digit and then converting the result to four bit binary. It is an unweighted code, i.e., no weights can be assigned to any of the four digit positions.

Table 2.7. Excess 3 code

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

Out of the possible 16 code combinations ($2^4 = 16$), only 10 are used in excess 3 code. The remaining 6, i.e., 0000, 0001, 0010, 1101, 1110 and 1111 are invalid in this code.

Example 2.24. Convert the following decimal numbers to excess 3 code (a) 14 (b) 32 (c) 46 (d) 430.

Solution : In each case add 3 to each digit in decimal number and then convert into binary.

(a) 1 4 (b) 3 2

+ 3 + 3 + 3 + 3

4 7 6 5

(c)

	↓	↓	
	0100	0111	
	4	6	
	+ 3	+ 3	
	7	9	
	↓	↓	
	0111	1001	

(d)

	↓	↓		
	0110	1010		
	4	3	0	
	+ 3	+ 3	+ 3	
	7	6	3	
	↓	↓	↓	
	0111	0110	0011	

2.12.4. EBCDIC Code

EBCDIC (Extended Binary Coded Decimal Interchange Code) is used in most of large computers for communication. It is eight bit code and uses BCD (binary coded decimal). This code also includes capital alphabets, lower case alphabets, numbers 0 – 9 and other symbols. Table 2.8 shows this code. e.g., letter B is written as :

B = 11000010

and letter d is written as : d = 10000010

Table 2.8 EBCDIC Table

Bit positions 0, 1→ Bit positions 2, 3 →	00	01	10	11
Bit positions 3, 5, 6, 7	00 01 10 11	00 01 10 11	00 01 10	11 0 01 10 11
0000	NUL DS	SP & –		0
0001	SOS	/	a j	A J 1
0010	FS		b k s	B K S 2
0011	TM		c l t	C L T 3
0100	PF RES BYP PN		d m u	D M U 4
0101	HT NL LF RS		e n v	E N V 5
0110	LC BS EOB UC		f o w	F O W 6
0111	DL IL PRE EOT		g p x	G P X 7
1000			h q y	H Q Y 8
1001			i r z	I R Z 9
1010	CC SM	¢ ! :		
1011		$, #		
1100		< * % @		
1101		( ) – ,		
1110		+ ; > =		
1111	CU1 CU2 CU3 1	| ? ″		

2.12.5 ERROR DETECTION CODES

Every digit of a digital system must be correct. An error in any digit can cause a problem because the computer may recognize it as something else. The correct ASCII code for A is 1000001. An error in one bit (i.e., 1000011) would mean C. Many methods have been devised to detect such errors.

2.12.5.1 Parity

Parity refers to the number of 1s in the binary word. When the number of 1s in the binary word is odd, it is said to have odd parity. When the number of words is even, it is said to have every parity, e.g.,

1100110	even parity
1000011	odd parity

One method for error detection is to use 7 bits for data and 8^{th} (most significant) bit for parity. The parity bit can be 1 or 0. To make odd parity, the parity bit is set to 1 or 0. If the word has odd number of 1s, the parity bit is set to 0. If the word has even number of 1s, the parity bit to set to 1 so as to make the total number of 1 odd. e.g.,

Table 2.9

Parity	Data	Total number of 1s
0	1100111	5
0	1101011	5
1	1000010	3
1	0000011	3

At the receiving point the parity is checked to see that it is odd. if it is even, an error has been committed and the data is required to be transmitted again.

In some computer systems even parity is also used, i.e., parity bit is set so as to make the total number of 1s even.

2.12.5.2 Check Sums

The above discussed parity check cannot detect two errors in the same word. If 01000011 or 01010111 is transmitted instead of 01100111, the errors will not be detected. One method to detect such cases is the check sums. As each word is transmitted, it is added to the previous word and the sum is retained at the sending end. E.g.,

Word A	0 0 0 1 0 0 1 1
Word B	1 0 0 1 0 1 0 0
SUM	1 0 1 0 0 1 1 1

Each successive word is added to the sum of the previous words. At the end of transmission, the sum (known as check sum) is also sent and is checked at the receiving point. Check sum method is commonly used in tele-processing.

2.12.5.3 Parity Data Codes

Parity can be added within each character. Two of these methods are known as 2 out of 5, and biquinary and are shown in Table 2.9.

Table 2.9 Parity data codes

Decimal	2 out of 5 code	Biquinary 5043210
0	00011	0100001
1	00101	0100010
2	00110	0100100
3	01001	0101000
4	01010	0110000
5	01100	1000001
6	10001	1000010
7	10010	1000100
8	10100	1001000
9	11000	1010000

The 2 out of code uses five bits to represent the 10 decimal digits. Each code word has two 1s. This facilitates decoding and easier error detection. If the number of 1s received is other than two, an error is indicated. It is used in communication systems.

The biquinary code has a 2 bit group and a 5 bit group. Each of these groups has a single 1. Its weight are 5043210. It is used in counters. The two bit group having weights 50 indicates whether the number is less than or equal to or greater than 5. The five bit group indicates the count below or above 5.

2.12.5.4 Error Correction Code

A method developed by RW Hammings and known as Hamming code is very commonly used for error correction. It contains parity bits located in proper positions.

To find the required number of parity bits, the following equation is used

$$2^p \geq m + p + 1 \qquad \ldots(2.3)$$

where m = number of information bits

p = number of parity bits

If m = 4, p must have a minimum value of 3 for Eqn. (2.3) to be satisfied. If m = 11,p must have a minimum value of 4 to satisfy Eqn. (2.3). the parity bits are located at each 2^n bit, e.g., for a bit data, the parity bits are located at positions 2^0, 2^1, 2^2, i.e., 1, 2, 4^{th} bit position starting with least significant bit (right most bit). Thus the format is

D_7 D_6 D_5 P_4 D_3 P_2 P_1

where P_1, P_2, P_4 indicate parity bits and the remaining are data (information bits).

For 11 bit data, the format is :

D_{15} D_{14} D_{13} D_{12} D_{11} D_{10} D_9 D_8 D_7 D_6 D_5 P_4 D_3 P_2 P_1

The assignment of parity bits in the four bit data is as under:

The bit P_1 is set so that it establishes even parity over bits 1, 3, 5 and 7 (i.e., data bits D_3, D_5, D_7 and itself P_1). P_2 is set for even parity over bits, 2, 3, 6 and 7 (i.e., D_3, D_6, D_7 and itself P_2). P_4 is set for even parity over bits 4, 5, 6, 7 (i.e., D_5, D_6, D_7 and itself P_4).

At the receiver end, each group is checked for even parity. If an error is indicated, it is located by forming a p bit binary number formed by the p parity bits. When the number of parity bits is 3 then binary number is 3 bit. The method is as discussed in examples 2.44 and 2.45.

Example 2.25 Data required to be transmitted is 1101. Formulate even parity Hamming code.

Solution : Since m = 4, p must be 3 to satisfy Eqn. (2.3). Parity bit positions are 2^0, 2^1, 2^2, i.e., bits 1, 2, and 4.

D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	1	0	0	1	1	0

P must be 0 so that there is even parity over bits 1, 3, 5, 7. P_2 must be 1 to create even parity over bits 2, 3, 6, 7 and P_4 must be 0 so that there is even parity over bits 4, 5, 6, 7. The values of P_1, P_2 and P_4 are indicated above.

Example 2.26 A seven bit Hamming code as received is 1111101. Check if it is correct. If not find the correct code if even parity is used.

Solution :

D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	1	1	1	1	0	1

Bits 4, 5, 6, 7 have even number of 1s. Hence no error

Bits 2, 3, 6, 7 have odd number of 1s. Hence error

Bits 1, 3, 5, 7 have even number of 1s. Hence no error

Evidently the error is in bit 2 position. The correct code is 1111111.

Example 2.27 The seven bit Hamming code as received is 0010001. Assuming that even parity has been used, check it is correct. If not find the correct code.

Solution :

D_7	D_6	D_5	P_4	D_3	P_2	P_1
0	0	1	0	0	0	1

Bits 4, 5, 6, 7 hae odd number of 1s. Hence error

Bits 2, 3, 6, 7 have even number of 1s. Hence no error

Bits 1, 3, 5, 7 have even number of 1s. Hence no error.

Evidently the error in bit 4. The correct code is 0011001.

Example 2.28 Solve the equation for x

$$x_{16} = 1111\ 1111\ 1111\ 1111_2$$

Solution : Replacing each 4 bit binary by hexadecimal using Table 2.1. We get

1111	1111	1111	1111
↓	↓	↓	↓
x_{16} = F	F	F	F_{16}

Hence $x_{16} = FFFF_{16}$

Example 2.29 How many memory locations can 14 address bits access.

Solution : Number of memory locations = 2^{14} = 16384.

Example 2.30 Solve the following ; (a) $110001_2 = x_{10}$ (b) $23.6_{10} = x_2$

(c) $65.535_{10} = x_{16}$ (d) $F8\ E6.39_{16} = x_{10}$.

Solution : (a)

1	1	0	0	0	1	Binary number
32	16	8	4	2	1	Write weights
32	16	8	4	2	1	Cross to weights under zero
32 + 16 + 1 = 49						Add weights

x = 49.

(b) Take the integer part

2	23
2	11–1
2	5–1
2	2–1
2	1–0
	0–1

Hence $23_{10} = 10111_2$.

Taking the fractional part

Fraction	**Fraction × 2**	**Remainder new fraction**	**Integer**
0.6	1.2	0.2	1
0.2	0.4	0.4	0
0.4	0.8	0.8	0
0.8	1.6	0.6	1
0.6	1.2	0.2	1
0.2	0.4	0.4	0
0.4	0.8	0.8	0

Hence $0.6_{10} = 1.00\ 1100_2$.

Hence $23.6_{10} = 10111.1001100_2$

(c) Taking the integer part

16	65
16	4–1
	0–4

Hence $65_{10} = 41_{16}$

Taking the fractional part

Fraction	Fraction × 16	Remainder new fraction	Integer
0.535	8.56	0.56	8
0.56	8.96	0.96	8
0.96	15.36	0.36	15 = F
0.36	5.76	0.76	5
0.76	12.16	0.16	12 = C
0.16	2.56	0.56	2
0.56	8.96	0.96	8

Hence $0.35 = 0.88F5\ C28_{16}$

Hence $65.535_{10} = 41.88F5\ C28_{16}$.

(d) Taking the integer part

F	8	E	6	Hexadecimal number
16^3	16^2	16^1	16^0	Write weight
$15 \times 16^3 + 8 \times 16^2 + 14 \times 16 + 6 \times 1$				Add weights

or $F8\ E6_{16} = 63718$

Taking the fraction part

0	3	9	Hexadecimal Number
	16^{-1}	16^{-2}	Write weights
$3 \times 16^{-1} + 9 \times 16^{-2}$			Add weights
$= 0.222656$			

Hence $F8\ E6.39_{16} = 63718.222656_{10}$

Example 2.31 In a new number system, X and Y are successive digits such that $(XY)_r = (25)_{10}$ and $(YX)_r = (31)_{10}$. Find X, Y, r.

Solution : Since the base is r

$$Xr + Yr^0 = 25$$

or $Xr + Y = 25$ …(i)

and $Yr + Xr^0 = 31$

or $Yr + X = 31$ …(11)

Also $Y = X + 1$ …(iii)

From Eqns. (i), (ii) and (iii)

$X = 3$, $Y = 4$ and $r = 7$.

Example 2.32 Solve the following (a) $(48.625)_{10} = (?)_2$ (b) $(6CD.A)_{16} = (?)_{10}$ (c) $(BCA3.AD)_{16} = (?)_2$ (d) $(446.25)_{10} = (?)_{16}$ (e) $(1010111.011)_2 = (?)_8$.

Solution :

Fraction	Fraction ×2	Remainder	Integer
		New Fraction	
0.625	1.25	.25	1
0.25	0.5	0.5	0
0.5	1.0	0	1

$(48.625)_{10} = (110000.101)_2$

2	48
2	24–0
2	12–0
2	6–0
2	3–0
2	1–0
	0–1

$48_{10} = 110000_2$

(b) 6 C D A

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

$= 6 \times 16^2 + 12 \times 16 + 13.\ 10 \times 16^{-1} = 1741.625_{10}$

(c) B C A 3 A D

↓ ↓ ↓ ↓ ↓ ↓

1011 1100 1010 0011 1010 1101

$BCA3.AD_{16} = 1011\ 1100\ 1010\ 0011.\ 1010\ 1101_2$

(d)

Fraction	Fraction ×16	New fraction	Integer
0.25	4	0	4

$(446.25)_{10} = (1BE.4)_{16}$

16	446
16	27–14 = E
16	1–11 = B
	0–1

$446_{10} = 1BE_{16}$

(e) 1 010 111 . 011

↓ ↓ ↓ . ↓

1 2 7 . 3

$(1010111.011)_2 = (127.3)_8$

Example 2.33 Multiply 10.101_2 by 0.101_2.

Solution

$$
\begin{array}{rrrrrrr}
 & & 1 & 0 & 1 & 0 & 1 \\
 & & 0 & . & 1 & 0 & 1 \\
\hline
 & & 1 & 0 & 1 & 0 & 1 \\
 & 0 & 0 & 0 & 0 & 0 & \\
1 & 0 & 1 & 0 & 1 & & \\
\hline
1 & 1 & 0 & 0 & 0 & 0 & 1
\end{array}
$$

SUMMARY

1. Digital computers are basically data processors and use binary numbers system. This system uses digits 0 and 1. Thus the base is 2.
2. Each binary digit is known as bit. A string of 4 bits is known as nibble and a string of 8 bits is known as byte. Computers Process data in strings of 8 bits or some multiplies, i.e., 16, 24 32 etc.
3. To convert a decimal number into binary, we divide the number successively by 2. The remainders when read upwards give the binary number.
4. To convert a binary number into decimal we multiply each binary digit by its weight. The weights are 2^0, 2^1, 2^2, 2^3… starting with the least significant bit (i.e., bit in the right most position).
5. The most significant bit denotes the sign in sign magnitude number form. 0 indicates positive, 1 indicates negative. The remaining bits are magnitude bits.
6. 1's complement is obtained by complementing each bit (i.e., 0 for 1 and 1 for 0). If the number is A, its 1's complement is denoted by $\overline{A}$.
7. 2's complement is obtained by adding 1 to 1's complement. It is denoted by A′.
8. The use of 2's complement representation simplifies the computer hardware.
9. In binary fractions, the weights of bits after binary point are 2^{-1}, 2^{-2}, 2^{-3} etc.
10. The rules for binary addition are 0 + 0 = 0, 0 + 1 = 1 + 0 = 1, 1 + 1 = 10 and 1 + 1 + 1 = 11.
11. The rules for binary subtraction are 0 – 0 = 0, 1 – 0 = 1, 1 – 1 = 0 and 10 – 1 = 1.

12. The rules for binary multiplication are $0 \times 0 = 0$, $0 \times 1 = 0$, $1 \times 0 = 0$, $1 \times 1 = 1$
13. In floating point representation, the numbers have a 10 bit long mantissa and 6 bit long exponent. The left most bit of mantissa is sign bit. The binary point is immediately to the right of this sign bit. To express negative exponents, the number 32_{10} (i.e., 100000_2) is added to the exponent.
14. In hexadecimal system the base is 16. The digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.
15. In octal system the base is 8. the digits are 0, 1, 2, 3, 4, 5, 6, 7.
16. In Binary Coded Decimal (BCD), each decimal digit is represented by 4 bits. 8421 BCD is the most prominent BCD code.
17. In Gray code representation only a single bit change occurs when going from one code to the next.
18. Excess – 3 code is obtained by adding 3 to every decimal digit and then converting the result into 4 bit binary.
19. Alphanumeric codes include binary codes for numbers, letters and symbols. ASCII and EBCDIC are the most common alphanumeric codes.
20. Error detection code enables detection of errors in the received data. Parity, check sums and parity data codes are used for this purpose.
21. Hamming code is used for error correction.

PROBLEMS

Determine the decimal numbers represented by the following binary numbers :

(a) 111001 (b) 101001 (c) 11111110

(d) 1100100 (e) 1101.0011 (f) 1010.1010

(g) 0.11100

Determine the binary numbers represented by the following decimal numbers ;

(a) 37 (b) 255 (c) 15 (d) 26.25

(e) 11.75 (f) 0.1

Add the following groups of binary numbers :

(a) 1011 + 1101

(b) 1010.11 01 + 101.01

Perform the following subtractions using 2's complement method.

(a) 0100 – 01001

(b) 01100 – 0001

(c) 0011.1001 – 0001.1110

Convert the following numbers from decimal to octal and then to binary. Compare the binary numbers obtained with the binary numbers obtained directly from the decimal numbers.

(a) 375 (b) 249 (c) 27.125

Convert the following binary numbers ot octal and then to decimal. Compare the decimal numbers obtained with the decimal numbers obtained directly from the binary numbers.

(a) 11011100.101010

(b) 01010011.010101

(c) 10110011

Encode the following decimal numbers in BCD code :

(a) 46 (b) 327.89 (c) 20.305

Encode the decimal numbers in Problem 2.9 to Excess-3 code.

Encode the decimal number 46 to Gray code.

Write your full name in

(a) ASCII code (b) EBCDIC code

Attach an even parity bit as MSB for

(a) ASCII code. (b) EBCDIC code.

What is the base in binary number system? What digits are used in this system?

What is meant by 1's complement, 2's complement?

What is octal system? What is the base? What digits are sued?

What is binary number? Discuss a procedure to convert (a) binary number to decimal, (b) decimal number into binary.

Explain (a) signed binary numbers, (b) 1's complement, (c) 2's complement.

What is a hexadecimal number? How can the following conversions be done. (a) Hexadecimal into binary. (b) Decimal to hexadecimal. (c) Hexadecimal to decimal. (d) Binary to hexadecimal?

2.18 What is ASCII code Discus it briefly.

CHAPTER-3

COMBINATIONAL LOGIC DESIGN

3.1 INTRODUCTION

Logic operations and Boolean algebra have already been discussed in Chapter 1. Boolean algebraic theorems are used for the manipulations of logical expressions. It has also been demonstrated that a logical expression can be realized using the logic gates. The number of gates and the number of input terminals for the gates required for the realization of a logical expression, in general, get reduced considerably if the expression can be simplified. Therefore, the simplification of logical expression is very important as it saves the hardware required to design a specific system. A large number of functions are available in IC form and therefore, we should be able to make optimum use of these ICs in the design of digital systems. That is, our aim should be to minimize the number of IC packages.

Basically, digital circuits are divided into two broad categories:

1. Combinational circuits, and

2. Sequential circuits,

In *combinational circuits*, the outputs at any instant of time depend upon the inputs present at that instant of time. This means there is no memory in these circuits. There are other types of circuits in which the outputs at any instant of time depend upon the present inputs as well as past inputs/outputs. This means that there are elements used to store past information. These elements are known as *memory*. Such circuits are known as *sequential circuits.* A sequential logic system may have combinational logic sub-systems. The design of combinational circuits will be discussed here. Sequential circuit design will be discussed later.

The design requirements of combinational circuits may be specified in one of the following ways:

1. A set of statements

2. Boolean expression, and

3. truth table.

The aim is to design a circuit using the gates already discussed or some other circuits which are in fact derived from the basic gates. As is usual in any engineering design, the number of components used should be minimum to ensure low cost, saving in space, power requirements, etc. There can be two different approaches to the design of combinational circuits. One of these is the traditional method, wherein the given Boolean expression or the truth table is simplified by using standard methods and the simplified expression is realized using the gates. The other method normally does not require any simplification of the logical expression or truth table, instead the complex logic functions available in medium scale integrated circuits (MSI) or large scale integrated circuits (LSI) can be directly used. Combinational circuit design using the traditional design methods has been discussed below.

The following methods can be used to simplify the Boolean function:

1. Algebraic method,

2. Karnaugh map technique,

3. Variable entered aping (VEM) technique, and

4. Quino-McCluskey method.

The algebraic method and the Karnaugh map (K-map) technique have been given here. The K-map is the simplest and most commonly used method. It can be used up to six variables. The variable entered mapping (VEM) has been discussed in Fletcher and the Quine-McCluskey method has been discussed in Hill and Peterson and the interested reader can refer to these books (see Bibliography).

The Quine-McClukey method is a formal method which in no way depends upon human intuition. It is, therefore, suitable for computer mechanization and is seldom used by logic designers manually. VEM technique is beyond the scope of this book although it is a

very powerful technique of logic simplification. Design of combinational logic circuits using MSI and LSI chips is postponed till these functions are discussed.

3.2 STANDARD FORMS OF LOGIC FUNCTIONS

Design of logic circuits starts with preparation of word equation or truth table for the desired output (0 or 1) for the given input conditions. A logic circuit to implement the above equation/truth table is to be synthesized. The logic expression can be either a sum of products or product of sums.[1]

3.2.1 Fundamental Products and Sum of Products

Fig. 3.1 shows four possible ways for connecting two input variables A, B and their complements $\overline{A}$, $\overline{B}$ to AND gate. These four products $\overline{A}\,\overline{B}$, $\overline{A}B$, $A\overline{B}$ and AB are known as fundamental products and are listed in Table 3.1.

Table 3.1 Fundamental products for two variables

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

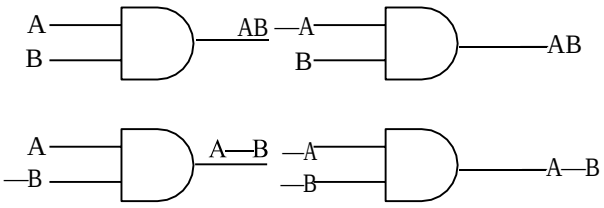


Fig. 3.1 Fundamental Products

Each fundamental product gives high output for the given input conditions of Table 4.6.

Table 3.2 lists the fundamental products for 3 variables A, B, C. As in the case of two variables all the fundamental products give high output for the given input, e.g., A = 1, B = 1 and C = 1gives high output when the three inputs to AND gate are 1, 1, 1. Similarly the other listings in Table 3.2 can be explained.

[1] SOP and POS are complementary forms.

Table 3.2 Fundamental products for three variables

A	B	C	Fundamental products
0	0	0	$\overline{A}\,\overline{B}\,\overline{C}$
0	0	1	$\overline{A}\,\overline{B}C$
0	1	0	$\overline{A}B\overline{C}$
0	1	1	$\overline{A}BC$
1	0	0	$A\overline{B}\,\overline{C}$
1	0	1	$A\overline{B}C$
1	1	0	$AB\overline{C}$
1	1	1	A B C

The first standard form of logic functions is sum of products form. Suppose we are given the truth table of Table 3.3. For convenience the rows have been numbered. Output Y = 1 for the condition of rows 3, 5, 7, 8. In row 3, A = 0, c = 0 and B = 1. The output Y can be high (i.e., 1) if A, B, C are ANDed as under

$$Y = \overline{A}\ B\ \overline{C} \qquad \ldots(3.1)$$

Similarly, we can interpret the remaining rows for high output.

Table 3.3

Row	A	B	C	Y
1	0	0	0	0
2	0	0	1	0
3	0	1	0	1
4	0	1	1	0
5	1	0	0	1
6	1	0	1	0
7	1	1	0	1
8	1	1	1	1

Rows 5, 7, 8 led to terms $A\overline{B}\,\overline{C}$, $A\ B\ \overline{C}$ and ABC. If any of these is 1, output Y will be 1. Thus the expressions for rows 3, 5, 7, 8 can be ORed. The required logic equation is

$$Y = \overline{A}\ B\ \overline{C} + A\ \overline{B}\ \overline{C} + A\ B\ \overline{C} + A\ B\ C \qquad \ldots(3.2)$$

Eqn. (3.2) is expanded form of us of products. Each term is known as minterm. It is seen that each minterm contains all the variables A, B, C. Digital circuit which implements Boolean eqn. (3.2) is shown in Fig. 3.2. The procedure for writing a Boolean expression as shown in Eqn. (3.2) is called expansion. The systematic procedure for writing the sum of products form is :

1. Write down all the terms.
2. Put blanks (or Xs) at places where letters must be provided to convert the term to a minterm.
3. Use combinations of Xs in each term to generate minterm. Where X is 0, write a letter with a bar, where X is 1 write a letter without bar.
4. Drop Out redundant term.

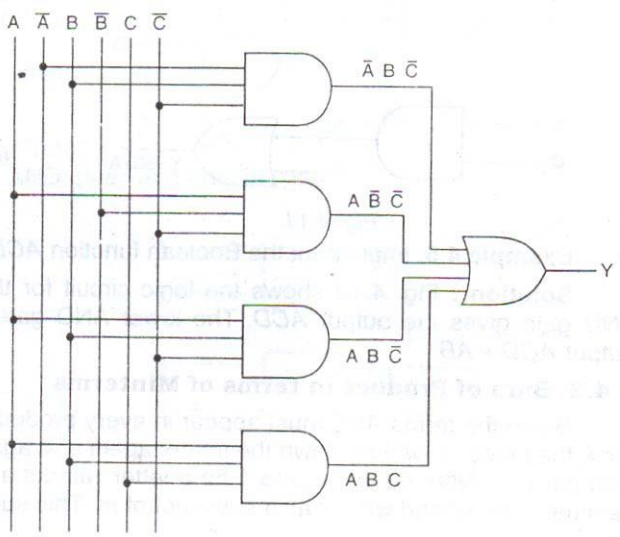


Fig. 3.2 Logical circuit for Eqn. (3.2)

Example 3.1 Write minterms of BC + A.

Solution : BC + A — Write terms

XBC + AXX — Add Xs where letters are missing

$\overline{A}$ BC, ABC — Vary Xs in XBC

A $\overline{B}\,\overline{C}$, A $\overline{B}$ C, A B $\overline{C}$, ABC — Vary Xs in AXX

Thus $BC + A = \overline{A}\,BC + ABC + A\,\overline{B}\,\overline{C} + A\,\overline{B}C + A\,B\,\overline{C} + ABC$

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

Example 3.2 Simplify the result of example 4.1 to show that it equals BC + A.

Solution : $\overline{A}\,B\,C + ABC + A\,\overline{B}\,\overline{C} + A\,\overline{B}C + A\,B\,\overline{C}$

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

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

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

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

Example 3.3 Write the minterms of ACD + AB.

Solution :-

ACD + AB	Write terms
AXCD + ABXX	Add Xs where letters are missing
ABCD, A $\overline{B}$CD	Vary Xs in AXCD
ABCD, ABC $\overline{D}$, AB$\overline{C}$D, AB$\overline{C}$ $\overline{D}$	Vary Xs in ABXX

Therefore, $ACD + AB = ABCD + A\,\overline{B}CD + ABCD + ABC\overline{D} + ABC\overline{D} + ABC\,\overline{D}\,\overline{D}$

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

Example 3.4 Implement the Boolean function BC + A

Solution : Fig. 3.3 shows the implementation. The AND gate gives the output BC. The term BC when ORed with A gives BC + A

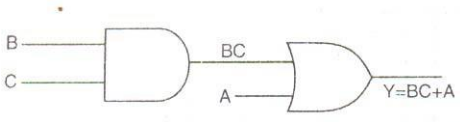


Fig. 3.3

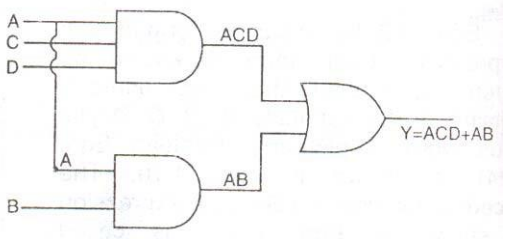


Fig. 3.4

Example 3.5 Implement the Boolean function ACD + AB.

Solution : Fig. 3.4 shows the logic circuit for the Boolean expression ACD + AB. The upper AND gate gives the output ACD. The lower AND gate gives AB. These are ORed to give the final output ACD + AB.

3.2.2 Sum of Product in terms of Minterms

Since the terms ABC must appear in every product, a short hand notation has been developed to save the labour in writing down the letters again and again. TO use this notation

substitute 0 for a letter with bar (i.e., NOT ed letter) and 1 for a letter without a br. Express the resultant binary number by a decimal number and write it as a subscript of m. This subscript is used in all truth tables and maps.

Example 3.6 Write the minterms designation for (a) ABCD (b) $A\overline{B}CD$ (c) $A\ \overline{B}\ \overline{C}\ \overline{D}$

Solution : (a)

A	B	C	D
1	1	1	1 = 15

Hence $ABCD = m_{15}$

(b)

A	$\overline{B}$	C	D
1	0	1	1 = 11

Hence $A\ \overline{B}\ CD = m_{11}$

(c)

A	$\overline{B}$	$\overline{C}$	$\overline{D}$
1	0	0	0 = 8

Hence $A\ \overline{B}\ \overline{C}\ \overline{D} = m_8$

Example 3.7 Write the complete expression for the minterm designation $Y = \Sigma\, m\,(1, 3, 5, 7)$

Solution : The expression $Y = \Sigma\, m(1, 3, 5, 7)$ should be read as Y is the sum of minterms 1, 3, 5, 7. Since the highest value of m is 7, there are 3 variables in the system. Let the variables by A, B, C.

Decimal number 7 corresponds to binary numbers 111 or ABC

Decimal number 1 = binary number 001 or $\overline{A}\ \overline{B}\ C$

Decimal number 3 = binary number 011 or $\overline{A}BC$

Decimal number 5 = binary number 101 or $A\overline{B}C$

Hence $Y = \Sigma\, m(1, 3, 5, 7) = \overline{A}\ \overline{B}C + \overline{A}\ BC + A\ \overline{B}\ C + ABC$

[In each of the above cases 0 and 1 are replaced by barred and unbarred letters A, B, C]

3.2.3 Product of Sums

The product of sums, as the name suggests, is an expression involving the product of two or more terms where each term contains sum of a number of variables. The product of sum form is a dual of sum of product form. It may also contain a single variable term. The expression A(A + B) (C + D) is an example of product of sum form. Fig. 3.5 shows a logic circuit corresponding to expression A(A + B) (C + D).

Example 3.8 Indicate the form of the expression

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

Draw a logic circuit to implement this expression.

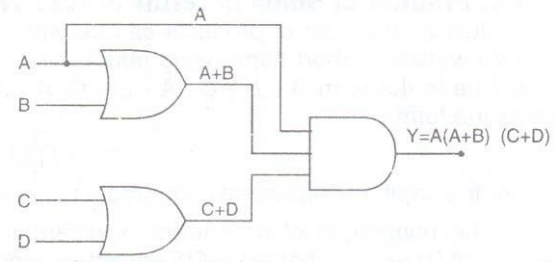


Fig. 3.5

Solution : It is a sum of products form. The logic circuit is shown in Fig. 3.6. It has three AND gates and one OR gate.[2]

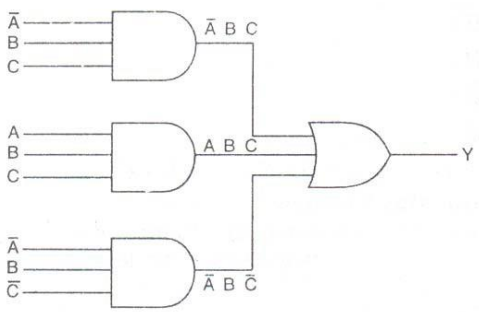


Fig. 3.6

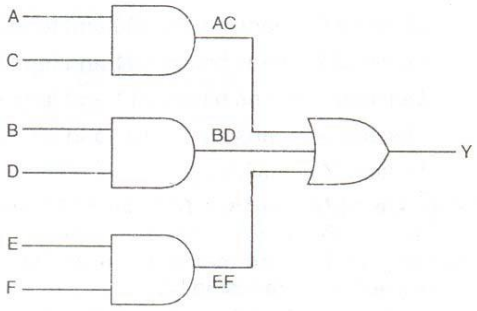


Fig. 3.7

[2] The sum of product form implementation will have one OR gate and as many AND gates as the number of product terms.

Example 3.9 Identify the form of the expression Y = AC + BC + EF

Draw logic circuit to implement the function.

Solution : It is a sum of product form. Fig. 3.7 shows the logic circuit. It has three AND gates and one OR gate.

Example 3.10 Identify the form of the expression

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

Draw logic circuit.

Solution : It is product of sum form. Fig. 3.8 shows the logic circuit. It has three OR gates and one AND gate.

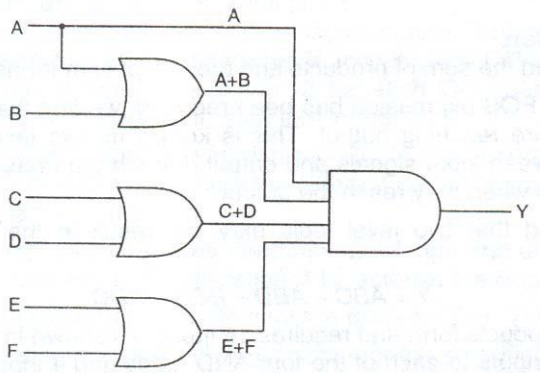


Fig. 3.8

3.2.4 Product of Sums in terms of Max Terms

Just as the sum of products can be written in short hand using minterms, the product of sums can be written in short hand using max terms. If a system has variables A, B, C then the max terms would be in the form $A + B + C$, $A + \overline{B} + C$, $A + B + C$ etc. A Boolean expression written in max terms takes the form

$$Y = \prod M(0, 1, 3, 4) \qquad \ldots(3.3)$$

where the capital $\prod$ represents the product and M stands for max terms.

The numbering of max terms is different from numbering of min terms. The unbarred letters represent 0s and the barred (NOT ed) letters represent 1s in forming the max terms designation.

Example 3.11 Write the full form of expression

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

Solution : Let the variables be A, B, C.

Decimal 0 means binary 000 and term is A + B + C

Decimal 1 means binary 001 and term is $A + B + \overline{C}$

Decimal 3 means binary 001 and term is $A + \overline{B} + \overline{C}$

Decimal 4 means binary 100 and term is $\overline{A} + B + C$

Hence, $Y = \prod M(0, 1,\ 3,\ 4) = (A + B + C)\ (A + B + \overline{C})(A + \overline{B} + \overline{C})(A + B + C)$

3.2.5 Complementary Nature of Minterms and Max Terms

The SOP and POS forms of Boolean expressions are complementary forms. Therefore, the min terms designation and max term designation are also complementary. Let the min term designation for a three variable expression be

$$Y = \sum m(0,\ 1,\ 4,\ 6)$$

The equivalent max term form is

$$Y = \prod M(2, 3, 5, 7)$$

For a three variable expression the total number of terms are $2^3 = 8$. Out of these the terms corresponding to decimal numbers 0, 1, 4, 6 are min terms and those corresponding to decimal numbers 2, 3, 5, 7 are max terms. Thus if min terms are known, the max terms can be written directly by visual inspection only.

Let the min term form for a 4 variable expression be

$$Y = \sum m(1,\ 4, 5, 7, 9, 11, 14)$$

Then the max term form is

$$Y = \prod M(0, 2, 3, 6, 8, 10, 12, 13, 15).$$

3.3 SIMPLIFICATION OF LOGICAL FUNCTIONS USING K-MAP

Simplification of logical functions with K-map is based on the principle of combining terms in adjacent cells. Two cells are said to be adjacent if they differ in only one variable. For example, in the two-variable K-maps, the top two cells are adjacent and the bottom two cells are adjacent. Also, the left two cells and the right two cells are adjacent. It can be verified that in adjacent cells one of the literals in same, whereas the other literal appears in uncomplemented form in one and in the complemented form in the other cell.

Similarly, we observe adjacent cells in the 3-variable and 4-variable K-maps. Table 3.4 gives the adjacent cells of each cell in 2-, 3-, and 4-variable K-maps. From this it becomes clear that if the Gray code in used for the identification of cells in K-map, physically adjacent (horizontal and vertical but not diagonal) cells differ in only one variable. Also, the left-most cells are adjacent to their corresponding right-most cells and similarly the top cells are adjacent to their corresponding bottom cells. The simplification of logical function is achieved by grouping adjacent 1's or 0's in groups of 2', where i = 1, 2,… n and n is the number of variables.

3.3.1 Grouping Two Adjacent Ones

If there are two adjacent ones on the map, these can be grouped together and the resulting term will have one less literal than the original two terms. It can be verified for each of the groupings of two ones as given in Table 5.5.

Table 3.4 Adjacent cells in K-maps

Cell with decimal number	Decimal numbers of adjacent cell		
	2-variable	3-variable	4-variable
0	1, 2	1, 2, 4	1, 2, 4, 8
1	0, 3	0, 3, 5	0, 3, 5, 9
2	0, 3	0, 3, 6	0, 3, 6, 10
3	1, 2	1, 2, 7	1, 2, 7, 11
4		0, 5, 6	0, 5, 6, 12
5		1, 4, 7	1, 4, 7, 13
6		2, 4, 7	2, 4, 7, 14
7		3, 5, 6	3, 5, 6, 15
8			0, 9, 10, 12
9			1, 8, 11, 13
10			2, 8, 11, 14
11			3, 9, 10, 15
12			4, 8, 13, 14
13			5, 9, 12, 15
14			6, 10, 12, 15
15			7, 11, 13, 14

Example 3.12 Simplify the K-map of Fig. 3.9

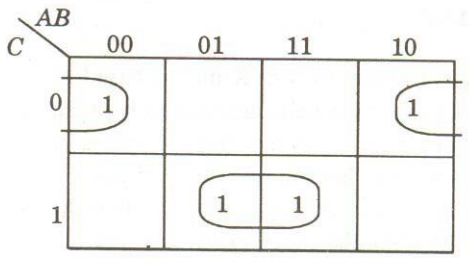


Fig. 3.9 K-map of Ex. 3.12

Solution : The standard SOP form of equation can be written by inspection as

$$Y = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}BC + ABC + A\overline{B}\overline{C} \quad \ldots(3.4)$$

If we combine the ones in adjacent cells (0, 4) and (3,7), Eq. (3.4) can be written as

$$Y = (\overline{A} + A)\overline{B}\,\overline{C} + (\overline{A} + A)BC \quad \ldots(3.4)$$

$$= \overline{B}\,\overline{C} + BC \quad \ldots(3.5)$$

Equation (3.5) can be directly obtained from the K-map by using the following procedure:

1. Identify adjacent ones, then see the values of the variables associated with these cells. Only one variable will be different and gets eliminated. Other variables will appear in ANDed form in the term, it will be in the uncomplemented form if it is 1 and in the complemented form if it is 0.
2. Determine the term corresponding to each group of adjacent ones. These terms are ORed to get the simplified equation in SOP form.

3.3.2 Grouping Four Adjacent Ones

Four cells form a group of four adjacent ones if two of the literals associated with the minterms/maxterms are not same and the other literals are same. Table 5.6 gives all possible groups of four adjacent ones for each cell in a 3-variable map. In case of 2-variable map, there is only one possibility corresponding to entry 1 in all

the four cells, and the simplified expression will be Y=1. That is, Y always equals 1 (independent of the variables).

On the basis of groupings of 4 adjacent ones given in Table 3.5, we can find the groupings in K-maps of four or more variables. In the case of a four-variable K- map, there are six possible groupings of 4-variables involving any cell. It is left to the reader to verify this fact.

Table 3.5. Groups of four adjacent ones in a 3-variable K-map

Cell with decimal number	Decimal numbers of cells forming groups of adjacent fours		
0	(0, 2, 6, 4),	(0, 1, 2, 3),	(0, 1, 4, 5),
1	(1, 0, 2, 3),	(1, 3, 7, 5),	(1, 0, 4, 5),
2	(2, 0, 6, 4),	(2, 3, 1, 0),	(2, 3, 6, 7),
3	(3, 1, 7, 5),	(3, 2, 1, 0),	(3, 2, 6, 7),
4	(4, 6, 2, 0),	(4, 5, 6, 7),	(4, 5, 0, 1),
5	(5, 1, 3, 7),	(5, 4, 6, 7),	(5, 4, 0, 1),
6	(6, 0, 2, 4),	(6, 7, 4, 5),	(6, 7, 2, 3),
7	(7, 1, 3, 4),	(7, 6, 4, 5),	(7, 6, 2, 3),

Example 3.13 Simplify the K-map of Fig. 3.10.

Solution The standard SOP form of equation can be written by inspection as

$$Y = m_0 + m_1 + m_3 + m_7 + m_8 + m_9 + m_{11} + m_{15}$$

$$= (m_0 + m_1 + m_8 + m_9) + (m_3 + m_7 + m_{15} + m_{11}) \quad \ldots(3.6)$$

In the K-map of Fig. 3.10, there are two groups of four adjacent ones. One corresponding to cells 0, 1, 8 and 9, and the other one corresponding to 3, 7, 15 and

11.

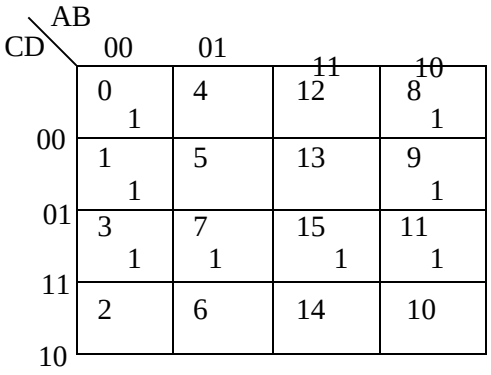


Fig. 3.10 K-map of Ex. 3.10

In Eq. (3.6), the minterms corresponding to each group are combined. The first term can be written as

$$m_0 + m_1 + m_8 + m_9 = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}\,D + A\,\overline{B}\,\overline{C}\,\overline{D} + A\,\overline{B}\,\overline{C}\,D$$

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

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

$$= \overline{B}\,\overline{C}[\overline{A}.1 + A.1]$$

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

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

In the first term or Eq, (3.6) we observe the following:

1. In this group of four minterms, two of the variables appear as $\overline{B}$ and $\overline{C}$ in all the four terms.

2. The variable A appears as A in two and as $\overline{A}$ in the other two minterms.

3. The variable D appears as D in two and as $\overline{D}$ in the other two minterms.

4. The combination of these four minterms results in one term with two literals which are present in all the four terms. Similarly, the second term of Eq. (3.6) is simplified to CD. Therefore, the K-map is simplified to

$$Y = \overline{B}\,\overline{C} + CD \qquad \ldots(3.7)$$

3.3.3 Grouping Eight Adjacent Ones

Eight cells form a group of eight adjacent ones if three of the literals associated with the minterms/maxterms are not same and the other literals are same. In case of 3-variable K-map, there is only one possibility of eight ones appearing in the K-map and this corresponds to output equal to 1, irrespective of the values of the input variables. Table 3.6 gives all possible groups of eight adjacent ones in a 4-variable K-map. From an understanding of this, we can easily find out such combinations for 5- and 6-variable K-maps. When eight adjacent ones are combined, the resulting equation will have only one term with the number of literals three less than the number of literals in the original minterms. Similar to the groupings of adjacent two and four ones, the literals which are common in all the eight minterms will be present and the literals which are not same get eliminated in the resulting term.

Table 3.6 Group of eight adjacent ones in 4-variable K-map

Decimal numbers of cells forming groups of adjacent eights in a 4-variable L-map
0, 4, 12, 8, 1, 5, 13, 9,
0, 4, 12, 8, 2, 6, 14, 10
0, 1, 3, 2, 4, 5, 7, 6
0, 1, 3, 2, 8, 9, 11, 10
1, 5, 13, 9, 3, 7, 15, 11
4, 5, 7, 6, 12, 13, 15, 14
12, 13, 15, 14, 8, 9, 11, 10
3, 7, 15, 11, 2, 6, 14, 10

The reader is advised to verify the simplification of eight adjacent ones into a single

term with three variables eliminated. For example, let us take the first group of eight adjacent

ones in Table 3.6. For all these eight cells, the variable C appears as $\overline{C}$ in the minterms and the other three variables are not same. Therefore, the grouping of these eight cells results in a term $\overline{C}$. Figure 3.11 shows the simplified expression for each of the groupings of eight ones for a 4-variable K-map.

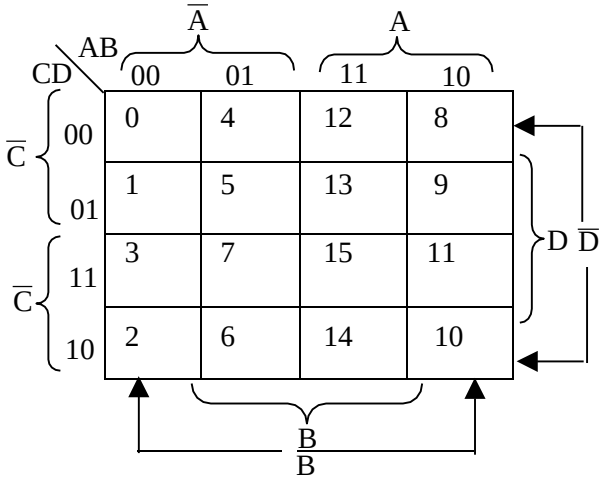


Fig. 3.11 Four-variable K-map illustrating the groupings of eight adjacent ones

3.3.4 Grouping 2, 4, and 8 Adjacent Zeros

In the above discussion, we have considered groups of 2, 4, and 8 adjacent ones. Instead of making the groups of ones, we can also make groups of zeros. The procedure is similar to the one used above and is as follows:

a. Group of two adjacent zeros result in a term with one literal less than the number of variables. The literal which is not same in the two maxterms gets eliminated.

b. Group of four adjacent zeros result in a term with two literals less than the number of variables. The two literals which are not same in all the four maxterms get eliminated.

c. Group of eight adjacent zeros result in a term with three literals less than the number of variables. The three literals which are not same in all the eight maxterms get eliminated.

We have considered groups of 2, 4, and 8 adjacent ones and zeros. The same logic can be extended to 16, 32, and 64 adjacent ones and zeros which occur in K-maps with more than 4 variables.

3.5 MINIMIZATION OF LOGICAL FUNCTIONS SPECIFIED IN MINTERMS/MAXTERMS OR TRUTH TABLE

3.5.1 Minimization of SOP Form

We have seen the advantages of simplifying a logical expression. If the expression is simplified to a stage beyond which it can not be further simplified, it will require minimum number of gates with minimum number of inputs to the gates. Such an expression is referred to as the *minimized expression.*

For minimizing a given expression in SOP form or for a given truth table, we have to prepare the K-map first and then look for combinations of ones on the K-map. We have to combine the ones in such a way that the resulting expression is minimum. To achieve this, the following algorithm can be used which will definitely lead to minimized expression:

a. Identify the ones which can not be combined with any other ones and encircle them. These are *essential prime implicants*.

b. Identify the ones that can be combined in groups of two in only one way. Encircle such groups of ones.

c. Identify the ones that can be combined with three other ones, to make a group of four adjacent ones, in only one way. Encircle such groups of ones.

d. Identify the ones that can be combined with seven other ones, to make a group of eight adjacent ones, in only one way. Encircle such groups of ones.

e. After identifying the essential groups of 2, 4, and 8 ones, if there still remains some ones which have not been encircled then these are to be combined

with each other or with other already encircled ones. Of course, however, we should combine the left-over ones in largest possible groups and in as few groupings as possible. In this, the groupings may not be unique and we should make the groupings in an optimum manner. You can verify that any one can be included any number of times without affecting the expression.

The above algorithm will be used to minimize the logical functions in the examples given.

Example 3.14 Minimize the four-variable logic function using K-map.

$$f(A, B, C, D) = \sum m(0, 1, 2, 3, 5, 7, 8, 9, 11, 14) \quad ..(3.7.1)$$

Solution : The K-map of Eq. (3.7.1) is shown in Fig. 3.12. The equation is minimized in the following steps:

1. Encircle 1in cell 14 which can not be combined with any other 1. The term corresponding to this is $ABC\overline{D}$.
2. There are at least two possible ways for every 1 forming groups of two adjacent ones. Therefore, we ignore it for the time being and go to the next step.
3. There is only one possible group of four adjacent ones involving each of the cells 8, 11, 5 or 7 and 2, and these are (8, 9, 0, 1),(11, 9, 1, 3), (5, 7, 3, 1) and (2, 3, 1, 0), respectively. Encircle these groups. The terms corresponding to these grops are $\overline{B}\,\overline{C}$, $\overline{B}D$, $\overline{A}\,D$, and $\overline{A}\,\overline{B}$, respectively.

Since all the ones have been encircles, therefore, the minimized equation is

$$f(A, B, C, D) = ABC\overline{D} + \overline{B}\,\overline{C} + \overline{B}D + \overline{A}D + \overline{A}\,\overline{B} \quad \ldots(3.8)$$

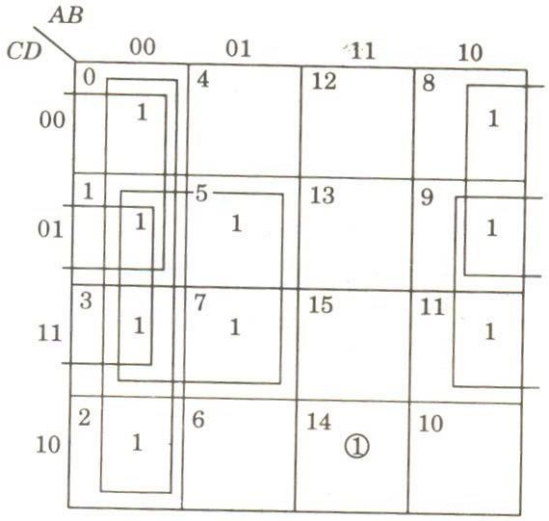


Fig. 3.12 K-map for Eq. (3.8)

Example 3.15 Determine the minimized expression in SOP form for the truth table given in Table 3.7.

Solution : The K-map for the truth table to Table 3.7 is shown in Fig. 3.13.

Using the minimization steps, we obtain the minimized expression

$$Y = \overline{B} + A\overline{C} + \overline{A}\,CD \qquad \ldots(3.9)$$

Table 3.7

Inputs				Output Y
A	B	C	D	
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0

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

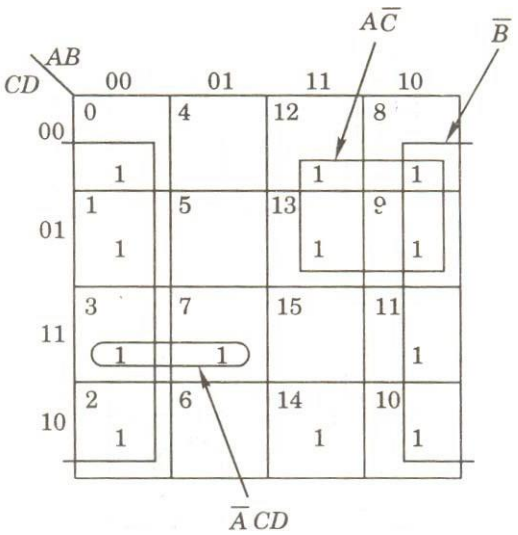


Fig. 3.13 K-map of Table 3.7.

3.5.2 Minimization of POS Form

For minimizing a given expression in POS form or for a given truth table we write zeros in the cells corresponding to maxterms for 0 outputs. The K-map is simplified by following the same procedure as used for SOP form with ones replaced by zeros. In this, groups of zeros are formed rather than groups of ones. We shall minimize the above two examples in POS form.

Example 3.16 Minimize the logic function of Eq. (3.10) in POS form.

Solution : Equation (3.10) can be expressed in standard POS form as

$$f(A, B, C, D) = \prod M\,(4, 6, 10, 12, 13, 15) \qquad \ldots(3.10)$$

The K-map corresponding to Eq. (3.10) is shown in Fig. 3.14.

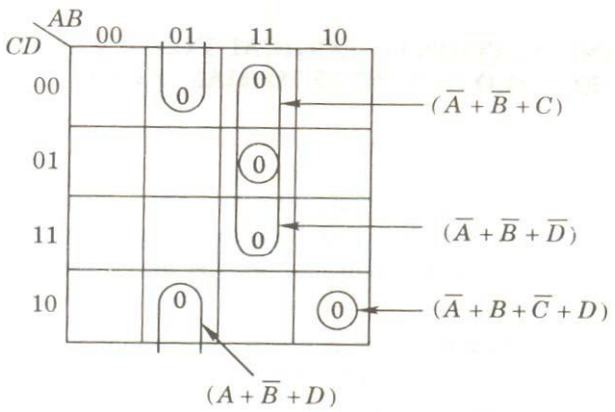


Fig. 3.14 K-map of Eq. (3.10)

Using steps similar to those outlined for SOP form, we obtain the minimized expression,

$$f = (\overline{A}+B+\overline{C}+D).((\overline{A}+\overline{B}+C).(\overline{A}+\overline{B}+\overline{D}).(A+\overline{B}+D) \qquad \ldots(3.11)$$

Example 3.17 Minimize the truth table give in Table 3.7 using maxterms.

Solution : The K-map is given in Fig. 3.15.

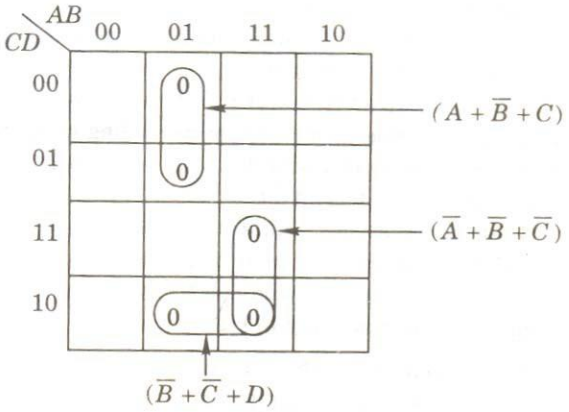


Fig. 3.15 K-map of Table 3.7.

The simplified expression is

$$Y = (A + \overline{B} + C)(\overline{A} + \overline{B} + \overline{C})(\overline{B} + \overline{C} + D) \quad \ldots(3.12)$$

Comparison of Eqs (3.11) and (3.12) confirms our generalizations made in Ex. 3.16 regarding the hardware requirements in the two methods.

3.6 MINIMIZATION OF LOGICAL FUNCTIONS NOT SPECIFIED IN MINTERMS/MAXTERMS

If the function is specified in one of the two standard forms, its K-map can be prepared and the function can be minimized. Now we consider the cases where the functions are not specified in standard forms. In such cases, the equations can be converted into standard forms,the K-map obtained and minimized. Alternately, we can directly prepare K-map using the following algorithm:

1. Enter ones for minterms and zeros for maxterms.
2. Enter a pair of ones/zeros for each of the terms with one variable less than the total number of variables.

3. Enter four adjacent ones/zeros for terms with two variables less than the total number of variables.
4. Repeat for other terms in the similar way.

One the K-map is prepared the minimization procedure is same as discussed earlier,

The following examples will help in understanding the above procedure:

Example 3.18 Minimize the four variable logic function

$$f(A, B, C, D) = AB\overline{C}D + \overline{A}BCD + \overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}\overline{D} + A\overline{C} + A\overline{B}C + \overline{B} \quad \ldots(3.13)$$

Solution : The method for obtaining K-map is

1. Enter 1 in the cell with A = 1, B = 1, C = 0, D =1

   corresponding to the minterm $AB\overline{C}D$.
2. Enter 1 in the cell with A = 0, B = 1, C = 1, D = 1

   corresponding to the minterm $\overline{A}BCD$.
3. Enter 1's in the two cells with A = 0, B = 0, C = 0

   corresponding to the term $\overline{A}\,\overline{B}\,\overline{C}$
4. Enter 1's in the two cells with A = 0, B = 0, D = 0

   (one of these is already entered) corresponding to the term $\overline{A}\,\overline{B}\,\overline{D}$.
5. Enter 1's in the two cells with A = 1, B = 0, C = 1

   corresponding to the term $A\overline{B}C$.
6. Enter 1's in the four cells with A = 1, C = 0

   (one of them is already entered) corresponding to the term $A\,\overline{C}$.
7. Enter 1's in the eight cells with B = 0 (all of them except one have already been entered) corresponding to the term $\overline{B}$.

The K-map is given in Fig. 3.16.

CD \ AB	00	01	11	10
00	1		1	1
01	1		1	1
11	1	1		1
10	1			1

Fig. 3.16 K-map of Eq. (3.13)

Example 3.19 Minimize the four variable logic function

$$f(A, B, C, D) = (A + B + \overline{C} + \overline{D}) \cdot (\overline{A} + C + \overline{D}) \cdot (\overline{A} + B + \overline{C} + \overline{D}) \cdot (\overline{B} + C) \cdot (\overline{B} + \overline{C}) \cdot (A + \overline{B}) \cdot (\overline{B} + \overline{D}) \quad \ldots(3.14)$$

Solution : The K-map cells in which 0's are to be entered corresponding to each term are given in Table 3.8. Even if a cell is involved in more than one terms, a 0 is to be entered only once.

Table 3.8. K-map cells with 0 entries

Term	Cell(s) with 0's
$A + B + \overline{C} + \overline{D}$	A = 0, B= 0, C = 1, D = 1
$\overline{A} + C + \overline{D}$	A = 1, C = 0, D = 1
$\overline{A} + B + \overline{C} + \overline{D}$	A = 1, B = 0, C = 1, D = 1
$\overline{B} + \overline{C}$	B = 1, C = 0
$\overline{B} + \overline{C}$	B = 1, C = 1
$A + \overline{B}$	A = 0, B = 1
$\overline{B} + \overline{D}$	B = 1, D = 1

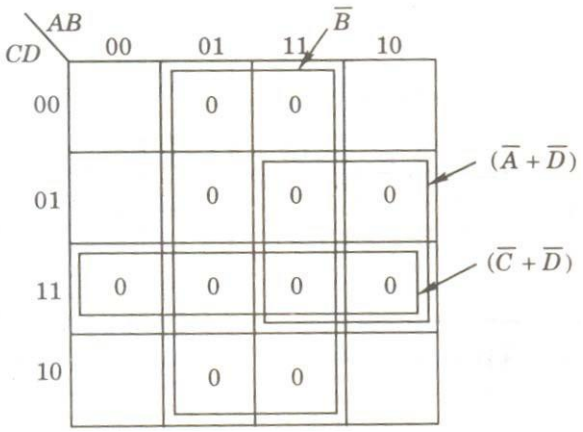


Fig. 3.17 K-map of Eq. (3.14)

The K-map is given in Fig. 5.17. The minimized expression is

$$f(A, B, C, D) = \overline{B} \cdot (\overline{A} + \overline{D})(\overline{C} + \overline{D}) \quad \ldots(3.15)$$

3.7 DON'T-CARE CONDITIONS

We enter 1's and 0's in the map corresponding to input variables that make the function equal to 1to 0, respectively. The maps are simplified using either 1's or 0's. Therefore, we make the entries in the map for either 1's or 0's. The cells which do not contain1 are assumed to contain 0 and vice-versa. This is not always true since there are cases in which certain combinations of input variables do not occur. Also, for some functions the outputs corresponding to certain combinations of input variables do not matter. In suchsituations the designer has a flexibility and it is left to him whether to assume a 0 or 1 as output for each of these combinations. This condition is known as *don't-care* condition and can be represented on the K-map as a × mark in the corresponding cell. The × mark in a cell may be assumed to be a 1 or a 0 depending upon which one leads to a simpler expression. Thefunction can be specified in one of the following ways:

1. In terms of minterms and don't-care conditions, For example,

$$f(A, B, C, D) = \sum m\,(1, 3, 7, 11, 15) + d(0, 2, 5) \qquad \ldots(3.16)$$

Its K-map and the minimized expression are given in Fig. 5.18a.

2. In terms of maxterms and don't-care conditions. For example,

$$f(A, B, C, D) = \prod m\,(4, 5, 6, 7, 8, 12) \,.\, d(1, 2, 3, 9, 11, 14) \qquad \ldots(3.17)$$

Its K-map and the minimized expression are given in Fig. 3.18b.

3. In terms of truth table. For example, consider the truth table of Table 3.9.

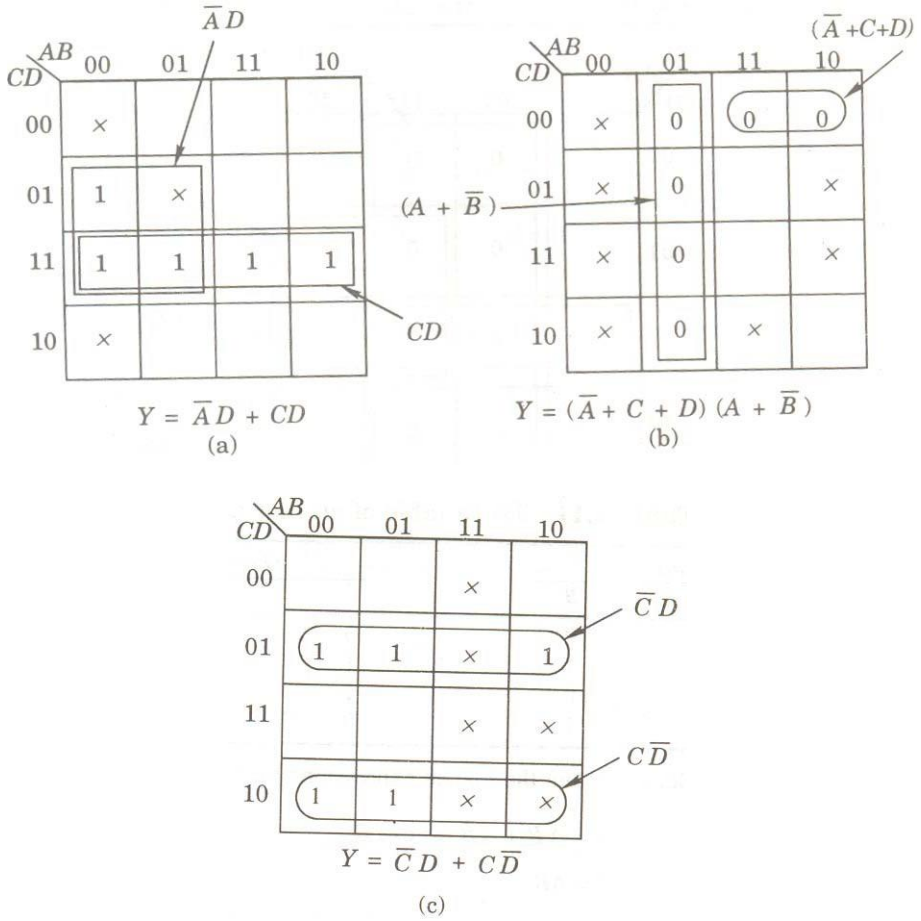


Fig. 3.18 K-maps with don't-care conditions.

Table **3.9**

Inputs				Output
A	B	C	D	Y
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0

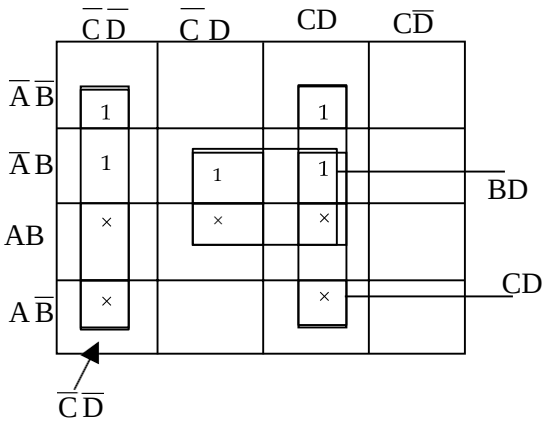
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	×
1	0	1	1	×
1	1	0	0	×
1	1	0	1	×
1	1	1	0	×
1	1	1	1	×

Its K-map and the minimized expression in SOP form are given in Fig. 3.18c.

Example 3.20 Given $f(A,B,C,D) = \sum m(0,3,4,5,7) + d(8, 9, 10, 11, 12, 13, 14, 15)$ …(3.17)

The 'd terms in Eqn. (4.33) refer to Don't care conditions.

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0



1	0	0	0	0
1	0	0	1	1
1	0	1	0	×
1	0	1	1	×
1	1	0	0	×
1	1	0	1	×
1	1	1	0	×
1	1	1	1	×

(a) (b)

Fig. 3.19a

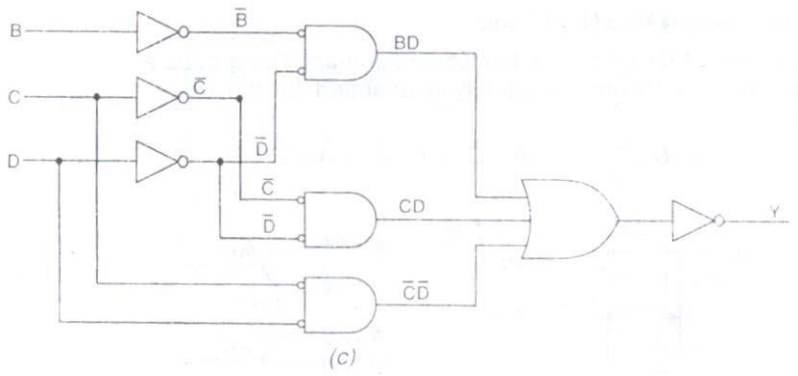


Fig. 3.19b

Fig. 3.19 (a) shows the Karnaugh map. The don't cares have been as Xs. When forming groups these don't cares can be treated as 1 or 0 whichever gives us the largest group. In Fig. 3.19 (a) the groups have been formed by treating Xs as 1s. (The reader should try to form groups by treating don't cares as 0s. This would yield smaller groups and hence require more costly hardware). The simplified Boolean expression is

$$Y = CD + BD + \overline{C}\,\overline{D} \qquad \ldots(3.18)$$

Fig. 3.19 (b) the NAND circuit corresponding to Eqn. (3.18).

Example 3.21 (a) Draw a logic circuit for the Boolean equation Y = AB + AC + BD + CD.

(b) Simplify the expression and draw logic circuit for the simplified expression.

Solution : (a) The logic circuit is shown in Fig. 3.20 (a). It requires 4 AND gates and one OR gate.

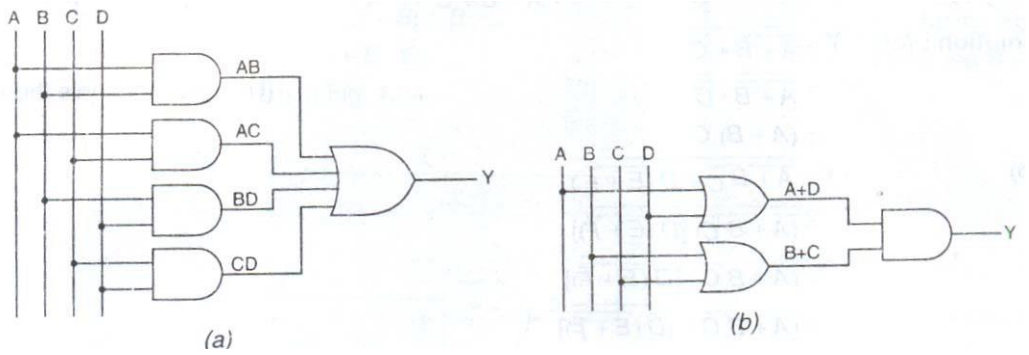


Fig. 320

(b) $Y = AB + AC + BD + CD$

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

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

Fig. 3.20 (b) shows the logic circuit. It requires 2 OR gates and 1 AND gate.

Example 3.22 (a) Draw a circuit to realize the function

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

(b) Simplify the function and draw a circuit to realize the simplified function. Draw a truth table showing the original and simplified function.

Solution : (a) The gate circuit for $Y = \overline{\overline{A.B} + \overline{A.C}}$ is showing in Fig. 3.22.

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

$= Y = \overline{\overline{A.B}.\overline{A.C}}$ (using De Morgan's theorem)

$= A.\ B\ .\ A\ .\ C = A.\ B.\ C$ (using the associative law)

The circuit for the simplified function is shown in Fig. 3.23. The truth table is dawn in Table 3.10.

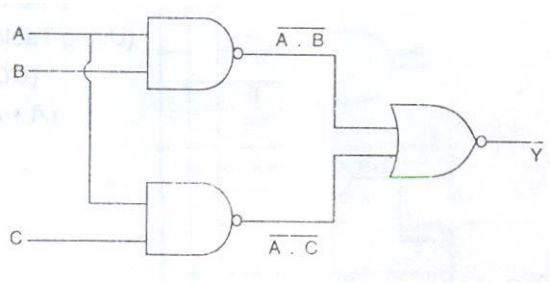


Fig. 3.22

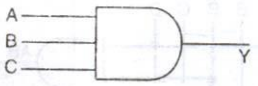


Fig. 3.23

Table 3.10. Truth table for Example 3.21

A	B	C	A.B	A.C	$\overline{A.B}$	$\overline{A.C}$	$\overline{\overline{A.B}+\overline{A.C}}$	$\overline{\overline{AB}+\overline{A.C}}$	A.B.C
0	0	0	0	0	1	1	1	0	0
0	0	1	0	0	1	1	1	0	0
0	1	0	0	0	1	1	1	0	0
0	1	1	0	0	1	1	1	0	0
1	0	0	0	0	1	1	1	0	0
1	0	1	0	1	1	0	1	0	0
1	1	0	1	0	0	1	1	0	0
1	1	1	1	1	0	0	0	1	1

Example 3.23 Simplify the expression

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

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

$$= \overline{\overline{A} + \overline{B} + \overline{A} + AB} \quad \text{(using De Morgan's theorem Eqn. 4.2)}$$

$= \overline{\overline{A} + \overline{B} + AB}$ (using Table 4.2)

$= \overline{\overline{A} + \overline{B} + A}$ (using Table 4.2)

$= \overline{A + \overline{A} + \overline{B}}$ (using Eqn. 4.1)

$= \overline{1 + \overline{B}}$ (using Table 4.2)

$= \overline{1}$ (using Table 4.2)

$= 0$ (Taking complement)

Example 3.24 Reduce to its minimum sum of products form. Then implement it in logic circuit.

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

Solution : $X = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$

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

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

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

$+ \overline{A}\,CD$

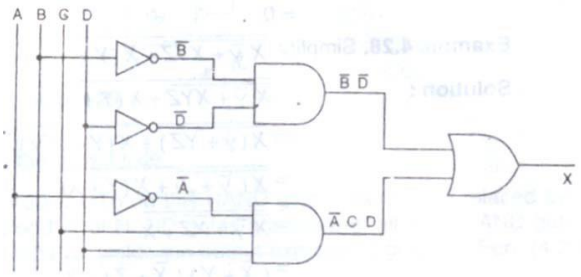


Fig. 3.24

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

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

$+ \overline{A}\,CD$

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

$+ \overline{A}\,CD$

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

or $X = \overline{B}\,\overline{D} + A\,CD$

Fig. 3.24 shows the logic circuit.

Example 3.25 Simplify and implement

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

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

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

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

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

$= AC + A\overline{B}\,C + BC + B\overline{B}C + \overline{A}\,B + \overline{A}\,\overline{C} + \overline{A}\,B + ABC$

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

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

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

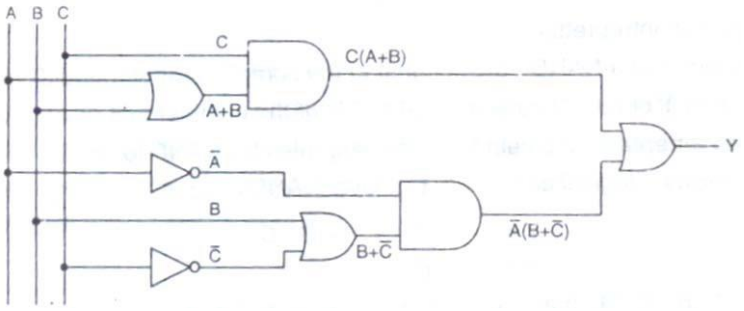


Fig 3.25 shows the logic circuit.

3.8 LOGIC CIRCUIT REALIZATION USING NAND AND NOR GATES ONLY

When a logic circuit is to be realized, the starting point is the Boolean expression or the truth table. The first step is the simplification of the Boolean expression using rules of Boolean Algebra and Demorgan's theorem. The resulting expression can be realized in a number of ways.

We have discussed the sum of products and product of sums forms of Boolean expressions. Both these expression can be realized using AND and OR gates.

As discussed in Chapter-I NAND and NOR gates are universal gates and any given Boolean expression can be realized using either NAND gates only or NOR gates only. Logic hardware is available in NAND, NOR< AND, OR, NOT forms and a logic designer has maximum flexibility. Many designers prefer to use either all NAND or all NOR gates in the circuit.

3.8.1 NAND Logic

Consider the expression

$$Y = \overline{AB + \overline{A} + \overline{B}} \qquad \ldots(3.19)$$

Eqn. (3.19) can be interpreted in two ways :

1. the output Y of a NAND gate is equal to the complement of the AND of variables A and B.
2. The output Y of NAND gate is equal to OR of the complement of input variables A and B.

The above two interpretations help in developing rules for NAND logic

Fig. 3.26 (a) shows a logical circuit with two level NAND logic.

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

or
$$Y = AB + \overline{C} \qquad \ldots(3.20)$$

It is seen in Eqn. (3.26) that input variables A and B appear as ANDed in the final output. Moreover, the output AB is ORed with complement of variable C. The term $\overline{AB}$ is input to gate G_1 alongwith input C. It is evident that :

1. All odd numbered gate levels (1, 3, 5 etc.) act as OR gates with single input variables complimented.

2. All even numbered gate levels (2, 4, 6 …) act as AND gates (because its

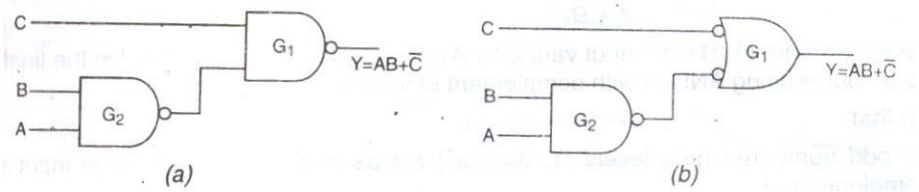


Fig. 3.26

inputs A and B are ANDed in the final output)

The numbering odd and even are from the output side.

Fig. 3.26 (a) can be redrawn as Fig. 3.26. (b). To do this NAND gate 1 has been replaced by a negative OR symbol. When more than one odd level NAND gates are present, all such NAND gates are replaced by negative OR symbols. Fig. 3.26 (b) yields the output expression given by Eqn. (3.26) directly.

3.8.2 NOR logic

Consider the Boolean expression

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

Above eqn. can be interpreted in two ways :

1. The output of a NOR gate is equal to the complement of the OR of input variables.
2. The output of a NOR gate is equal to the AND of the complements of inputs.

The above two interpretations help in developing rules for NOR logic.

Fig. 3.27 (a) shows a two level NOR logic circuit.

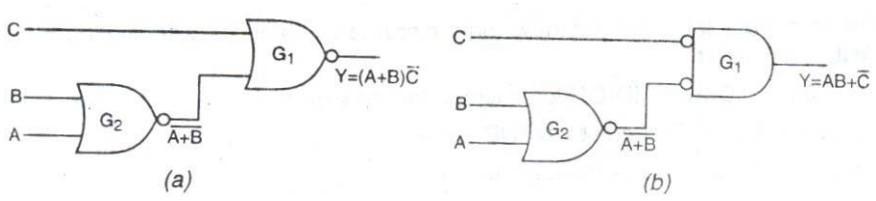


Fig. 3.27

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

or $$Y = (A + B)\ \overline{C} \qquad \ldots(3.21)$$

It is seen that in Eqn. (3.21) the input variables A and B appear as being ORed in the final output. Moreover, this output is being ANDed with complement of variable C.

It is seen that :

1. All odd numbered gate levels (1, 3, 5, …) act as AND gates with single input variable complemented.
2. All even numbered gate levels (2, 4, 6, …) act as OR gates because the inputs A and B are being ORed in the final output.

Fig. 3.27 (a) can be redrawn as in Fig. 3.27 (b). To do this NOR gate 1 has been replaced by a negative AND symbol. When more than one odd level gates are present, all such NOR gates are replaced by negative AND symbols. Fig. 3.27 gives the output given by Eqn. (3.21) directly.

3.9 DESIGN EXAMPLES

3.9.1 Arithmetic Circuits

1. Half-adder. A logic circuit for the addition of two one-bit numbers is referred to as an half-adder. The addition process is illustrated in Section 2.5 and is reproduced in truth table form in Table 3.11. Here, A and B are the two inputs and S (SUM) and C(CARRY) are the two outputs.

Table 3.11 Truth table of an half-adder

Inputs		Outputs	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

From the truth table, we obtain the logical expressions for S and C outputs as

$$S = \overline{A}B + A\overline{B} = A \oplus B \quad \ldots(3.22)$$

$$C = AB \quad \ldots(3.23)$$

The realization of an half-adder using gates is shown in Fig. 3.28.

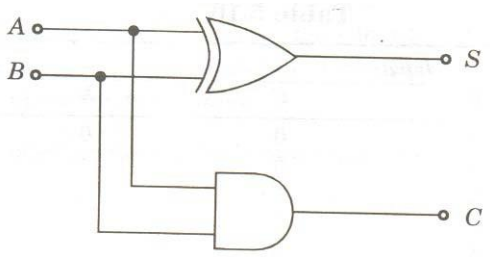


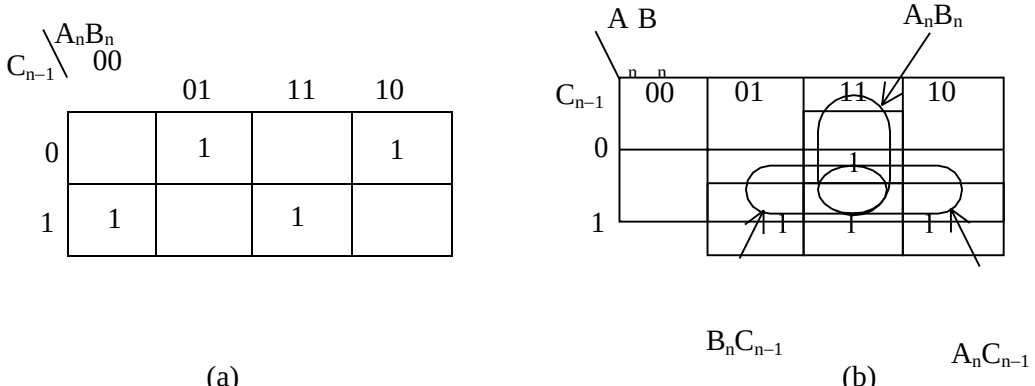
Fig. **3.28** Realization of an half-adder.

2. Full-adder. An half-adder has only two inputs and there is no provision to add a carry coming from the lower order bits when multibit addition is performed. For this purpose, a third input terminal is added and this circuit is used to add A_n, B_n, and C_{n-1}, where A_n and B_n are the nth order bits of the numbers A and B respectively and C_{n-1} is the carry generated from the addition of (n–1)th order bits. This circuit is referred to as *full-adder* and its truth table is given in Table 3.12.

Table 3.12 Truth table of a full-adder

Inputs			Outputs	
A_n	B_n	C_{n-1}	S_n	C_n
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

The K-maps for the outputs S_n and C_n are given in Fig. 3.29.



(a) (b)

Fig. 3.29 K-maps for (a) S_n (b) C_n

$$S_n = \overline{A}_n B_n \overline{C}_{n-1} \overline{A}_n \overline{B}_n C_{n-1} + A_n \overline{B}_n C_{n-1} + A_n B_n C_{n-1} \quad \ldots(3.24)$$

$$C_n = A_n B_n + B_n C_{n-1} + A_n C_{n-1} \quad \ldots(3.25)$$

The NAND-NAND realizations are given in Fig. 3.30

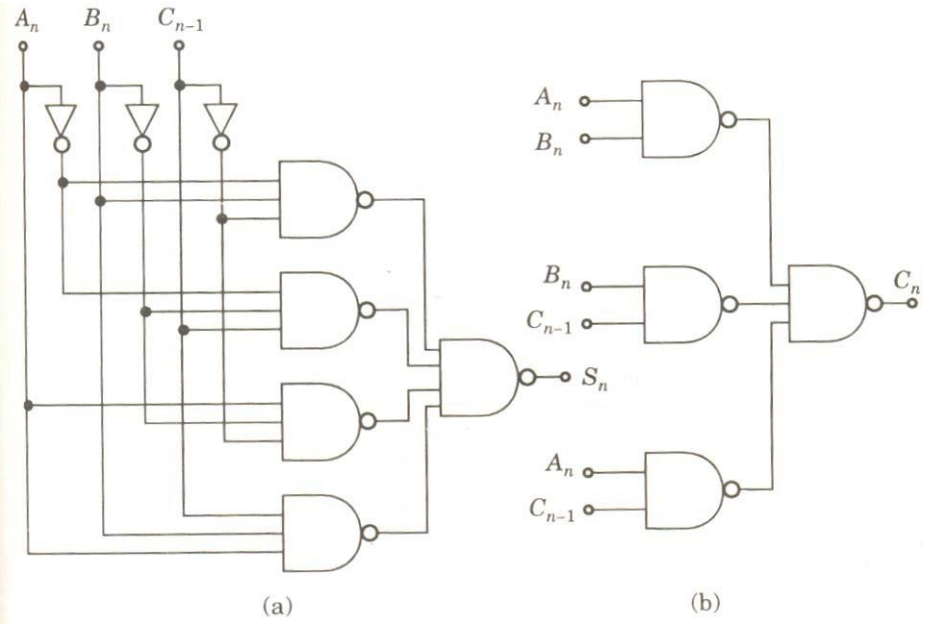


Fig. 3.30 NAND-NAND realization of (a) S_n (b) C_n.

3. Half-sub-tractor. A logic circuit for the subtraction of B (subtrahend) from A (minuend) where A and B are 1-bit numbers is referred to as a *half-sub-tractor*. The subtraction process is reproduced in truth table form in Table 3.13. Here, A and B are the two inputs and D(difference) and C (borrow) are the two outputs.

Table 3.13 Truth table of a half-sub-tractor

Inputs		Outputs	
A	B	D	C
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

From the truth table, and logical expressions for D and C are obtained as

$$D = \overline{A}B + A\overline{B} = A \oplus B \qquad \ldots(3.26)$$

$$C = \overline{A}\,B \qquad \ldots(3.27)$$

The realization of a half-subtractor using gates is shown in Fig. 3.21

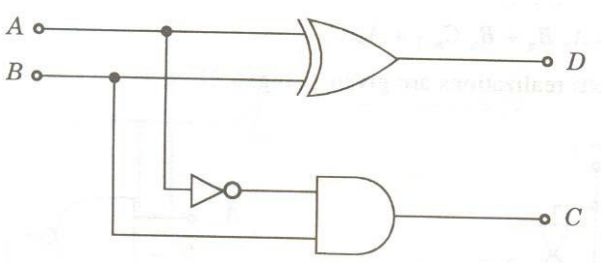


Fig. 3.31 Realization of a half-subtractor

4. Full-subtractor. Just like a full-adder, we require a *full-subtractor* circuit for performing multibit subtraction wherein a borrow from the previous bit position may also be there. A full-subtractor will have three inputs, A_n (minuend), B_n (sub-trahend) and C_{n-1} (borrow from the previous stage) and two outputs, D_n (difference) and C_n (borrow). Its truth table is given in Table 5.14. The K-map for the output D_n is exactly same as the K-map for S_n of the adder circuit and therefore, its realization is same as given in Fig. 3.30(a).

Table 3.14 Truth table of a full-subtractor

Inputs			Outputs	
A_n	B_n	C_{n-1}	S_n	C_n
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

The K-map for C_n is given in Fig. 3.32(a) and its realization is given in Fig. 3.32(b). The simplified expression for C_n is

$$C_n = \overline{A}_n B_n + \overline{A}_n C_{n-1} + B_n C_{n-1} \quad \ldots(3.28)$$

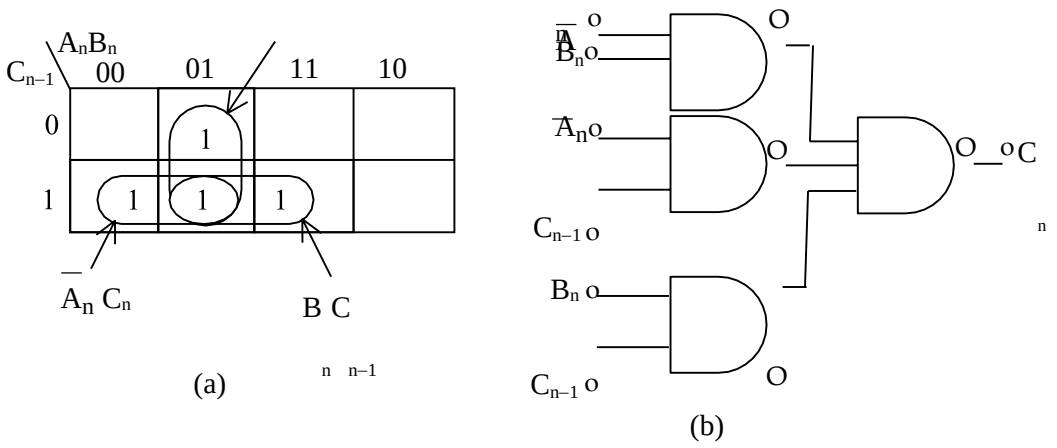
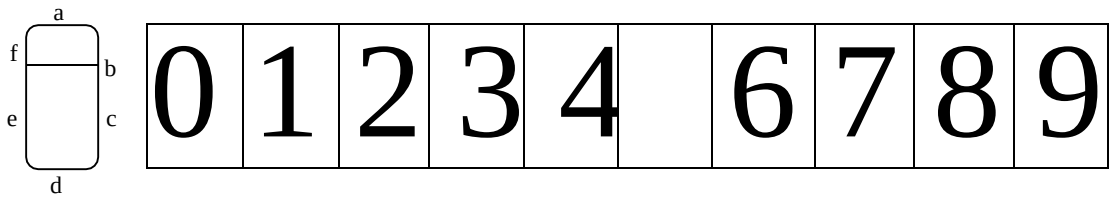


Fig. 3.32 (a) Truth table for C_n (b) Realization of C_n

3.9.2 BCD-to-7-Segment Decoder

A digital display that consists of seven LED segments is commonly used to display decimal numerals in digital systems. Most familiar examples are electronic calculators and watches where one 7-segment display device is used for displaying one numeral 0 through 9. For using this display device, the data has to be converted from some binary code to the code required for the display. Usually, the binary code used is natural BCD. Figure 3.33(a) shows the display device, Fig. 3.33(b) shows the segments which must be illuminated for each of the numerals and Fig. 3.33(c) gives the display system.



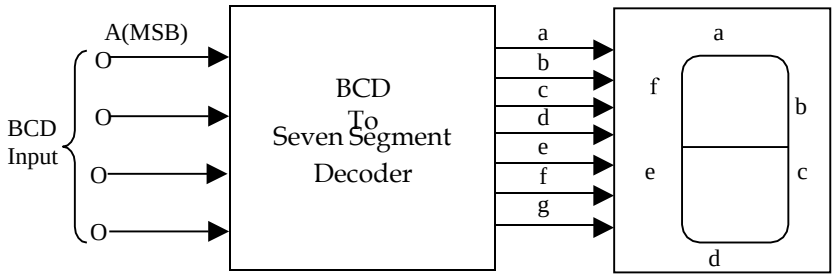


Fig. 3.33 (a) 7-segment display (b) display of numerals (c) display system

Table 3.15 gives the truth table of BCD-to-7-segment decoder. Here ABCD is the natural BCD code for numerals 0 through 9. The K-maps for each of the outputs a through g are given in Fig. 3.34. The entries in the K-map corresponding to six binary combinations not used in the truth table are x – don't-care. The K-maps are simplified and the minimum expressions are given by:

$$a = \overline{B}\,\overline{D} + BD + CD + A \quad \ldots(3.29)$$

$$b = \overline{B} + \overline{C}\,\overline{D} + CD \quad \ldots(3.30)$$

$$c = B + \overline{C} + D = \overline{\overline{B}\,C\,\overline{D}} \quad \ldots(3.31)$$

$$d = \overline{B}\,\overline{D} + C\overline{D} + \overline{B}C + B\overline{C}\,D \quad \ldots(3.32)$$

$$e = \overline{B}\,\overline{D} + C\overline{D} \quad \ldots(3.33)$$

$$f = A + \overline{C}\,D + B\overline{C} + BD \quad \ldots(3.34)$$

$$g = A + B\overline{C} + BC + C\,D \quad \ldots(3.35)$$

Table 3.15 Truth table of BCD-to-7 segment decoder

Decimal digit displayed	Input				Outputs						
	$\overline{A}$	$\overline{B}$	$\overline{C}$	$\overline{D}$	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1

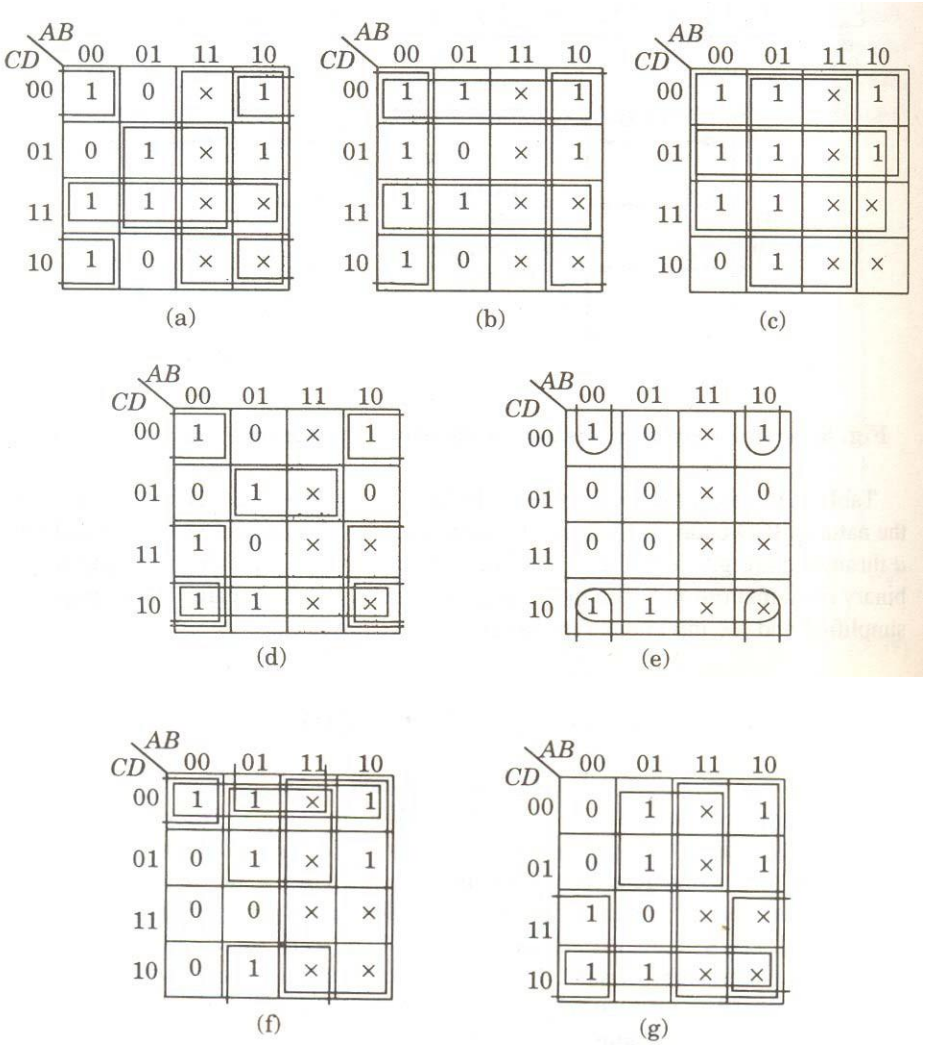


Fig. 3.34 K-maps of Table 3.15

The NAND gate realizations are shown in Fig. 3.35.

We see from the realizations of Fig. 3.35 that a term with single literal must be inverted and then applied to the second level NAND gate.

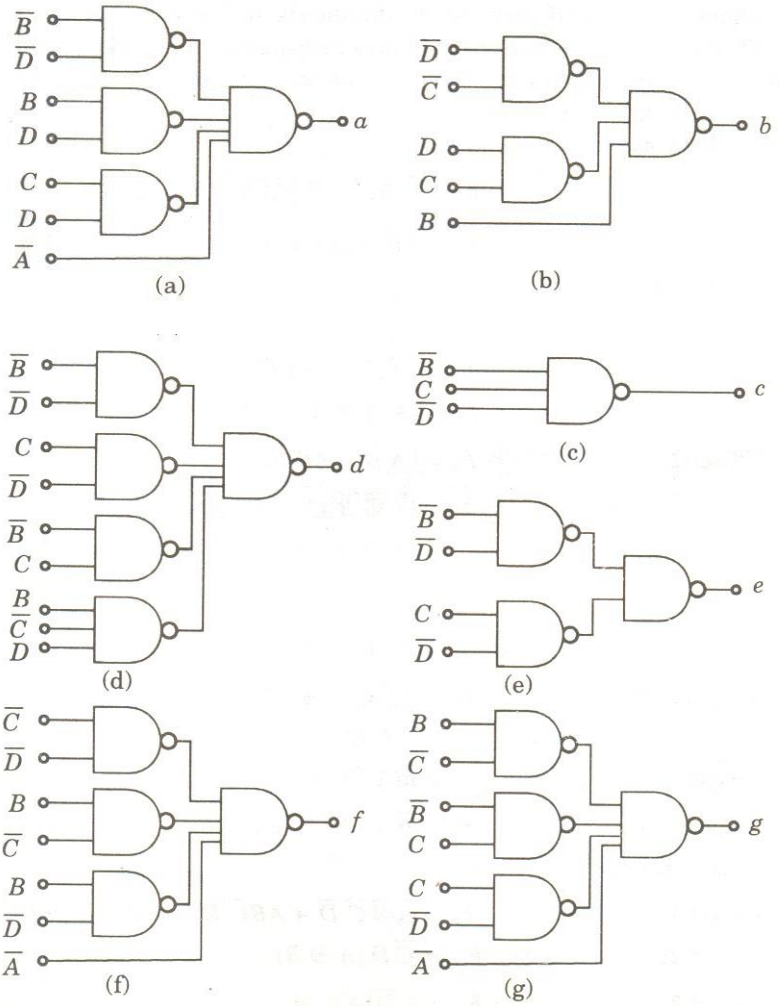


Fig. 3.35 NAND gate realizations of Eqns. (3.29)-(3.25).

3.10 BCD ADDER

We have discussed the BCD* number representation in chapter 2. In BCD system each decimal digit is represented by its equivalent 4 bit binary number, e.g., 63 is written as 01100011 where 0110 represents

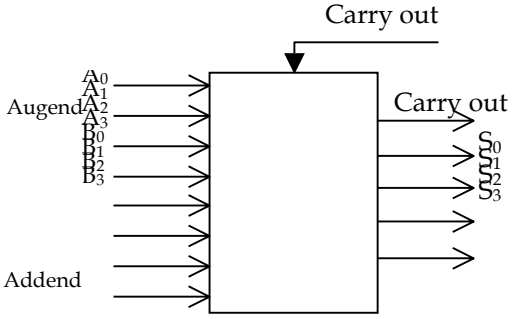


Fig. 3.36 Block diagram of BCD adder

* BCD means Binary Coded Decimal.

6 and 0011 represents 3. Since the highest decimal number is 9 (i.e., 1001) the binary combinations 1010, 1011, 1100, 1101, 1110 and 1111 are not valid in BCD system. In BCD addition, as and when the sum exceeds 9, we add 6 to the sum.

A block diagram of BCD adder has been shown in Fig. 3.36. It has 9 bits of inputs.4 bits of augend, 4 bits of addend and one bit for carry in from the lower stage. The output has 5 bits, i.e., 4 bits of SUM and one bit for carry out.

Fig. 3.37 shows the logic diagram of BCD adder made up from full and half adders, OR gates and AND gate. It has provision for taking care of invalid combinations in BCD. A_3 A_2 A_1 A_0 are the 4 bits of Augend while B_3 B_2 B_1 B_0 are the 4 bits of addend. S_3 S_2 S_1 S_0 are the output bits. The circuitry which determines when a carry is to be transmitted to the next most significant digits to be added consists of both the full binary adder to which sum (S) outputs from the adders for weight 8, 4, 2 inputs are connected and the OR gate to which the carry (C) from the eight position bit is connected. It is evident that a carry should be generated when 8 AND 2 or 8 AND 4 AND 2 sum outputs from the adder represent 1s or when the carry outputs from the eight position adder contains a 1.

When the 4 bit result is not valid in BCD, the method to obtain correct result is to add decimal 6 (i.e., 0110) to the invalid result. This addition means adding 1s in the weight 4 and weight 2 output lines. The two half adders and the full adders in the lower part of Fig. 3.37 perform this function. The examples of this correction are

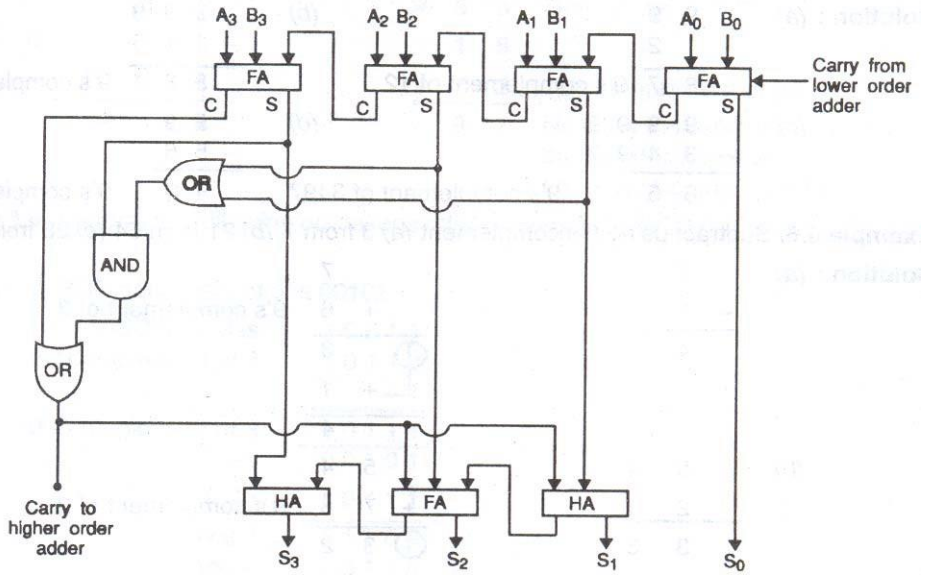


Fig. 3.37 BCD adder

6 + 7 = 13 or 0110 + 0111 = 1101

+ 0110

1 0011 = 3

↓

Carry

8 + 6 = 14 or 1000 + 0110 = 1110

+ 0110

1 0011 = 3

↓

Carry

3.11 LOGIC CIRCUIT FOR FORMING 9'S COMPLEMENT OF BCD NUMBER

Fig. 3.38 shows a logic circuit for forming 9's complement of a BCD number. The inputs are A_3 A_2 A_1 A_0. The weights for these inputs are 8, 4, 2 and 1 respectively.

If the input is the BCD of any decimal number, the output will be 9's complement of that number, e.g., if input is 0011 (i.e., decimal 3) the output will be 0110 (decimal 6). The circuit consists of one NOT gate, one XOR gate and NOR gate. The validity of circuit can be checked by Karnaugh's graphs. The truth tables for the NOR, XOR and NOT gates in Fig. 3.38 are shown in Table 3.16–3.18. By combining these we can form the truth table of the complete circuit. This truth table shown in Table 3.19.

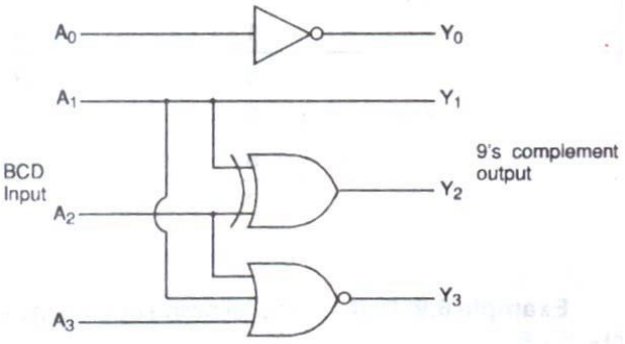


Fig. 3.38 Logic circuit for forming 9's complement of BCD number

Table 3.16 Truth table for 3 input NOR gate in Fig. 3.38

A_3	A_2	A_1	$Y_3 = \overline{A_3 + A_2 + A_1}$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0

Table 3.17 Truth table for 2 input XOR gate in Fig. 3.38

A_2	A_1	$Y_2 = A_2 \oplus A_1$
0	0	0
0	1	1
1	0	1
1	1	0

Table 3.18 Truth table for NOT gate in Fig. 3.38

A_0	Y_0
0	1
1	0

Table 3.19 Truth table for logic circuit of Fig. 3.38

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	1	0	0	1
0	0	0	1	1	0	0	0
0	0	1	0	0	1	1	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	0	0
0	1	1	0	0	0	1	1
0	1	1	1	0	0	1	0
1	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0

3.12 BCD ADDER/SUBTRACTOR

BCD addition and subtraction is very easy is 10's complement and is used in calculators. The logic circuits of Fig. 3.37 and Fig. 3.38 are connected in tandem.

A block diagram of BCD adder/subtractor is shown in Fig. 3.39 (a). Let $B_3\ B_2\ B_1\ B_0$ be the first BCD number and let $A_3\ A_2\ A_1\ A_0$ be the number to be added or subtracted from the first number. The number $B_3\ B_2\ B_1\ B_0$ is fed directly to the BCD adder/subtractor while

A_3 A_2 A_1 A_0 passes through a logic circuit for forming 10's complement of BCD numbers (i.e., logic of Fig. 3.38).

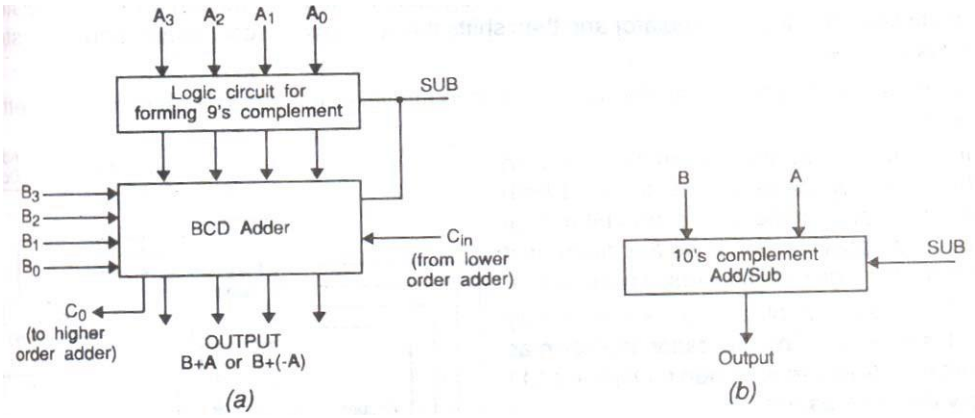


Fig. 3.39 BCD adder/subtractor (a) block diagram (b) symbol

When SUB is low, the A bits pass through the logic circuit as it is and BCD adder gives the sum.* $S = A + B$

When SUB is high, A bits are converted to 9's complement by the logic circuit. Moreover, a High SUB adds 0001 to the output of the logic circuit. This is equivalent to forming 10's complement of A bits. Thus we get the result as $-A + B$ or $B + (-A)$, i.e., number A is subtracted from number B. A symbol for BCD adder/subtractor is shown in Fig. 3.39 (b).

3.13 BINARY MULTIPLIER

We have discussed the rules for binary multiplication in chapter 2. These are $0 \times 0 = 0$, $0 \times 1 = 0$, $1 \times 0 = 0$ and $1 \times 1 = 1$. To understand the working of a binary multiplier, let us multiply binary number 111 by 101. The method proceeds as under

			1	1	1	Multiplicand
			1	0	1	Multiplier
			1	1	1	First partial product
		0	0	0		Second partial product
		0	1	1	1	Sum of first and second partial products
	1	1	1			Third partial product
1	0	0	0	1	1	Final result

From the above multiplication we can write the following rules for binary multiplication. The logic circuit for binary multiplier must follow these rules.

1. The partial product is 000, if the multiplier bit is 0 and is equal to multiplicand if multiplier bit is 1.
2. The product register* needs twice as many bits as the multiplicand register.
3. The first partial product is shifted one place to the right (relative to second partial product) when adding.

From the above rules and procedure it is evident that logic circuit for binary multiplication dos the following.

1. It stores the multiplicand and multiplier in separate registers say Y and B respectively.
2. It recognizes a bit as 0 or 1.
3. It stores partial products in accumulator (i.e., B).
4. If the right most bit in multiplier register is 0, it shifts the combined accumulator and B register right one place. If the right most bit in the B register is 1, it adds

* Register is a digital device for temporary storage of binary number.

the contents of Y register (i.e., multiplicand) to the accumulator and then shifts the contents of accumulator and B register right by one place.

5. After each step it examines the right most bit in multiplier register (i.e., B register) whether the bit is 0 or 1.

6. In addition to above it examines the sign bits of multiplicand and multiplier. If both arethe same, it stores the sign bit of final result as positive. If the two are different, it stores the sign bit of final result as negative.

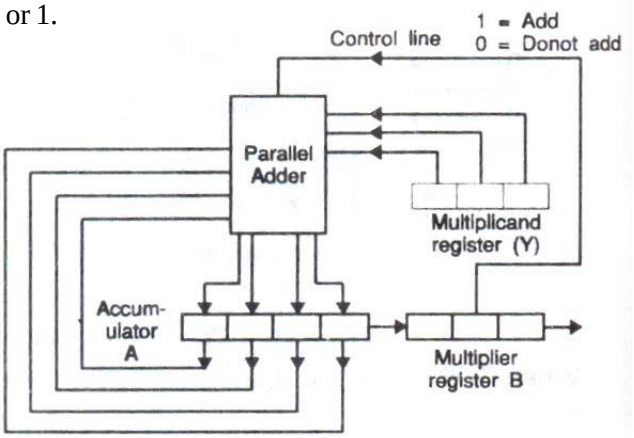


Fig. 3.40

Fig. 3.40 shows a block diagram of binary multiplier. This process of multiplication is known as add and shift multiplier circuit. When multiplying 111 by 101 the various steps are :

1. Binary 111 is loaded in register Y and 101 in register B. Accumulator contents in the beginning are 0000.
2. Binary 1 (LSB of register B) is applied to control line. Contents of Y register are added to accumulator.
3. Registers A and B are shifted to the right by one bit.
4. Binary 0 is applied to the control line.
5. Registers A and B are shifted to the right by one bit.
6. Binary 1 (MSB of register B) is applied to the control line. Contents of Y (starting from MSB) and partly in register B(ending with LSB).

3.14 BINARY DIVISION

Though the division process appearing to be almost of the same complexity as multiplication, actually it is more complex. This is due to the reason that at every step it has to be checked whether the result of a particular division process has left a positive or negative remainder. If the remainder is negative, 0 is put in the quotient register.

The basic division process is by subtraction and shifting to the left. The dividend (the number to be divided) is stored in the accumulator register (say A). The divisor is stored in register Y, the quotient in register B and the remainder in register A (accumulator). The steps in the division process are :

1. Contents of Y register are subtracted from the accumulator (A). After subtraction the remainder may be positive or negative is zero.
2. If the remainder is negative, it is evident that the division will not go. Therefore,a 0 is placed in the LSB of B register. The contents of register A are restored by adding the remainder to the divisor. The registers A and B are then shifted left by one bit.
3. If the remainder is positive or zero, this part of division process has succeeded. The registers A and B are shifted one bit left and 1 is put in the LSB of register B.
4. The above process is repeated M + 1 times where M is the number of shifts. A counter is used to count the shifts till the first 1 bits of registers Y and A are aligned.

3.15 MAGNITUDE COMPARATOR

The function of a magnitude comparator is to compare the magnitudes of two numbers and indicate which one is bigger of the two. The simplest form of a magnitude comparator indicates whether the two numbers are equal or not.

An exclusive OR is a comparator. If the two input bits are equal its output is 0, if not the output is 1. Thus a 1 bit comparator is just an exclusive OR gate as shown in Fig. 3.41.

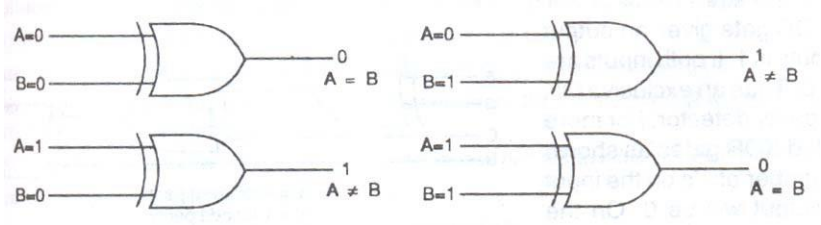


Fig. 3.41 One bit magnitude comparator

When he numbers to be compared are of two bits we need the circuit of Fig. 3.42. LSBs of the two numbers are A_0 and B_0 and are fed to XOR gate 1. The MSB of the numbersare A_1 and B_1 and are fed to XOR gate 2. If the two numbers are equal, outputs of both XOR gates are 0 each and the outputs of NOT gates are 1 each. Then the output of AND gate is 1. Thus a High output of AND gate indicates A = B. If A ≠ B, one of the XOR gates gives High output and the final output of And gate is low.

The above process can be expanded for numbers with more bits. Fig. 3.43 shows a magnitude comparator for 4 bit numbers. It is seen that the combination of XOR and NOT gate in Fig. 3.42 is an exclusive NOR gate. Therefore, the circuit of Fig. 3.43(a) shows

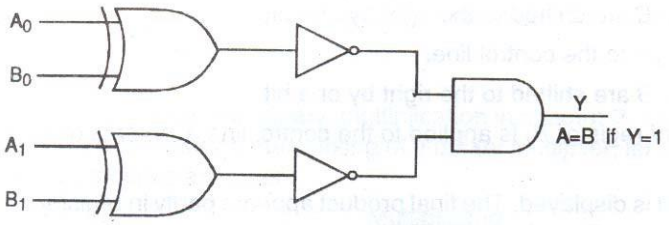
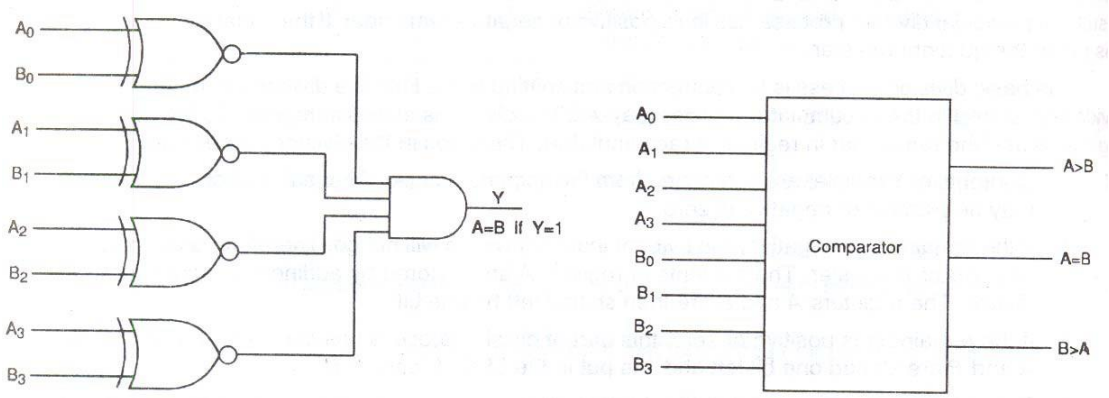


Fig. 3.42 Magnitude comparator for two bit numbers A_1 A_0 and B_1 B_0

exclusive NOR gates. As in the case of two bit numbers, a High output of AND gate indicates that A = B.



(a) (b)

Fig. 3.43 Magnitude comparator for two 4 bit numbers

IC magnitude comparators have an additional feature to test which of the two numbers A and B is bigger and indicate the result thereby, Fig. 6.25 shows a logical symbol of a 4 bit magnitude comparator with the facility to indicate whether A = B, A > B or B > A. Magnitude comparator for higher number of bits are also available in IC form. IC7485 is an 4 bit comparator. To test two 8 bit numbers, we need two such comparators.

3.16 PARITY DETECTOR

An exclusive OR gate gives an output of 1 if either of its inputs is 1. If both inputs are 0 or 1, the output is 0. Thus an exclusive OR gate can work as a parity detector. For morebits we use cascaded XOR gates as shown in Fig. 3.44. If the number of 1's on the input is even, the final output will be 0. On the other hand if the number of 1's on the input is odd,the final output will be 1.

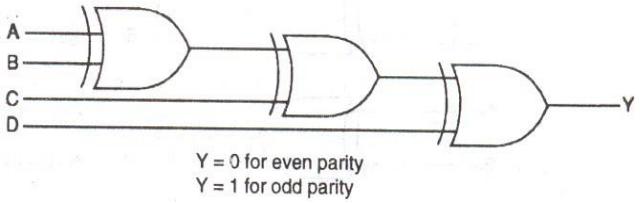


Fig. 3.44 Four bit parity detector/generator

A parity bit can be added to the code group either at the beginning or at the end depending on the system design. However, the total number of 1's, including parity bit is even for even parity and odd for odd parity. Table 3.20 lists the parity bits for various numbers on BCD code.

Table 3.20 BCD code with parity bits

Even Parity					Odd Parity				
P	8	4	2	1	P	8	4	2	1
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	1
1	0	0	1	0	0	0	0	1	0
0	0	0	1	1	1	0	0	1	1
1	0	1	0	0	0	0	1	0	0
0	0	1	0	1	1	0	1	0	1
0	0	1	1	0	1	0	1	1	0
1	0	1	1	1	0	0	1	1	1
1	1	0	0	0	0	1	0	0	0
0	1	0	0	1	1	1	0	0	1

The parity detector can detect a single error or an odd number of errors but cannot check for two errors.

Example 3.26 Add a parity bit to make even parity in the following binary words (a) 1110 (b) 1010 (c) 0111111000111001.

Solution : Parity Bit Data

(a) 1 1110

(b) 0 1010

(c) 0 0111111000111001

Example 3.27 A system with odd parity receives the followings data. Find out the words with error 10101, 110010, 1000111, 11111110000111.

Solution : In the following data, the numbers of 1s is not odd. These are in error 1000111, 11111110000111.

3.17 MULTIPLEXERS AND THEIR USE IN COMBINATIONAL LOGIC DESIGN

3.17.1 Multiplexer

The multiplexer is a special combinational circuit that is one of the most widely used standard circuits in digital design. The multiplexer (or data selector) is a logic circuit that gates one out of several inputs to a single output. The input selected is controlled by a set of select inputs. Figure 3.45 shows the block diagram of a multiplexer with n input lines and one output line. For selecting one out of n inputs for connection to the output, a set of m select inputs is required, where $2^m = n$. Depending upon the digital code applied at the select inputs one out of n data sources is selected and transmitted to a single output channel. Normally, a strobe (or enable) input (G) is incorporated which helps in cascading and it is generally active-low, which means it performs its intended operation when it is LOW.

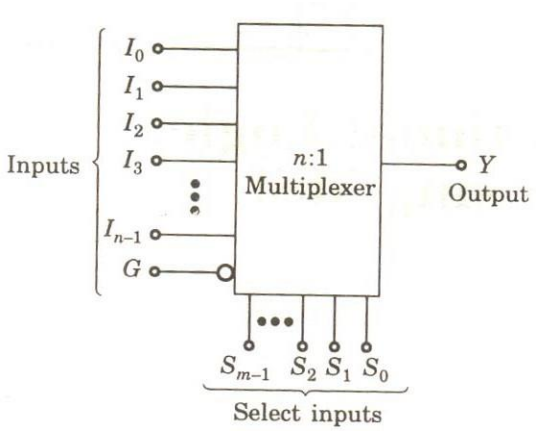


Fig. **3.45** Block diagram of a digital multiplexer.

Table 3.21 Truth table of a 4:1 multiplexer

Select inputs		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Table 3.21 gives the truth table of a 4:1 multiplexer. With LOW input at G, the output Y can be expressed as

$$Y = \overline{S}_1 \overline{S}_0 I_0 + \overline{S}_1 S_0 I_1 + S_1 \overline{S}_0 I_2 + S_1 S_0 I_3 \quad \ldots(3.36)$$

Equation (3.36) can be realized using NAND gates and the realization is given in Fig. 3.46.

3.17.2 Combinational Logic Design Using Multiplexers

The multiplexing function discussed above can conveniently be used as a logicelement in the design of combinational circuits. Standard ICs are available Table 3.22 for 2 : 1, 4 : 1, 8 : 1, and 16 : 1 multiplexers.

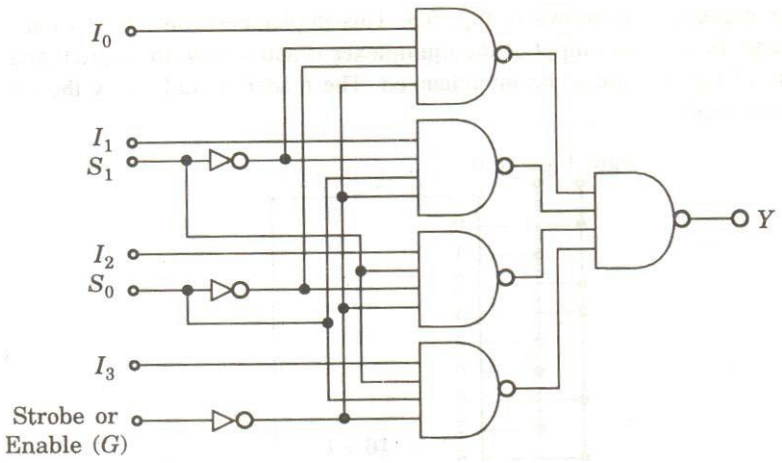


Fig. 3.46 A 4:1 multiplexer with strobe input using NAND gates.

Table 3.22 Available multiplexer ICs

IC No.	Description	Output
74157	Quad 2:1 Multiplexer	Same as input
74158	Quad 2 : 1 Multiplexer	Inverted input
74153	Dual 4 : 1 Multiplexer	Same as input
74352	Dual 4 : 1 Multiplexer	Inverted input
74151A	8 : 1 Multiplexer	Complementary output
74152	8 : 1 Multiplexer	Inverted input
74150	16:1 Multiplexer	Inverted input

Use of multiplexers offers the following advantages :

1. Simplification of logic expression is not required,
2. it minimizes the IC package count, and
3. logic design is simplified.

For using the multiplexer as a logic element, either the truth table or one of the standard forms of logic expression must be available. The design procedure is given below:

1. Identify the decimal number corresponding to each minterm in the expression. The input lines corresponding to these numbers are to be connected to logic 1 level.
2. All other input lines are to be connected to logic 0 level.
3. The inputs are to be applied to select inputs.

The following examples illustrate the above procedure.

Example 3.28 Implement the expression using a multiplexer.

$$f(A, B, C, D) = \sum m(0, 2, 3, 6, 8, 9, 12, 14)$$

Solution : Since there are four variables, therefore, a multiplexer with four select inputs is required. The circuit of 16:1 multiplexer connected to implement the above expression is shown in Fig. 3.47. This implementation requires only one IC package. In case the output of the multiplexer is active-low, the logic 0 and logic 1 inputs of Fig. 3.47 are to be interchanged. The reader should verify the validity of this statement.

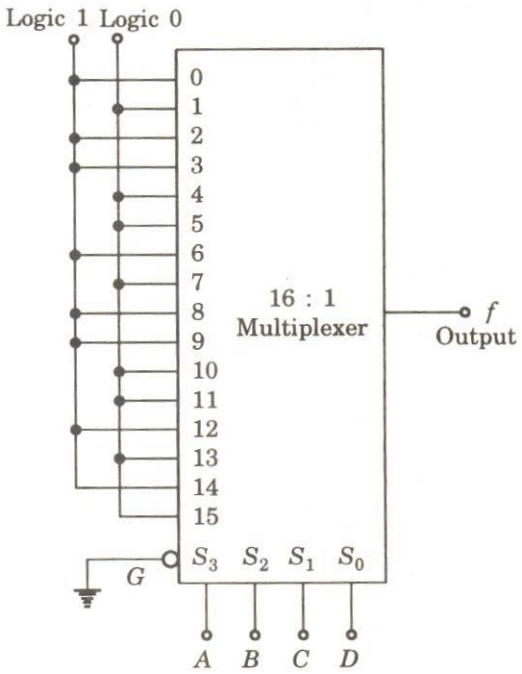


Fig. 3.47 Implementation of logic expression of Ex. 3.28

3.17.3 Multiplexer Tree

Since 16-to-1 multiplexers are the largest available ICs, therefore, to meet the larger input needs there should be a provision for expansion. This is achieved with the help of enable/strobe inputs and multiplexers tacks or trees are designed. Two commonly used methods for this purpose are illustrated in Figs (3.48) and (3.49)

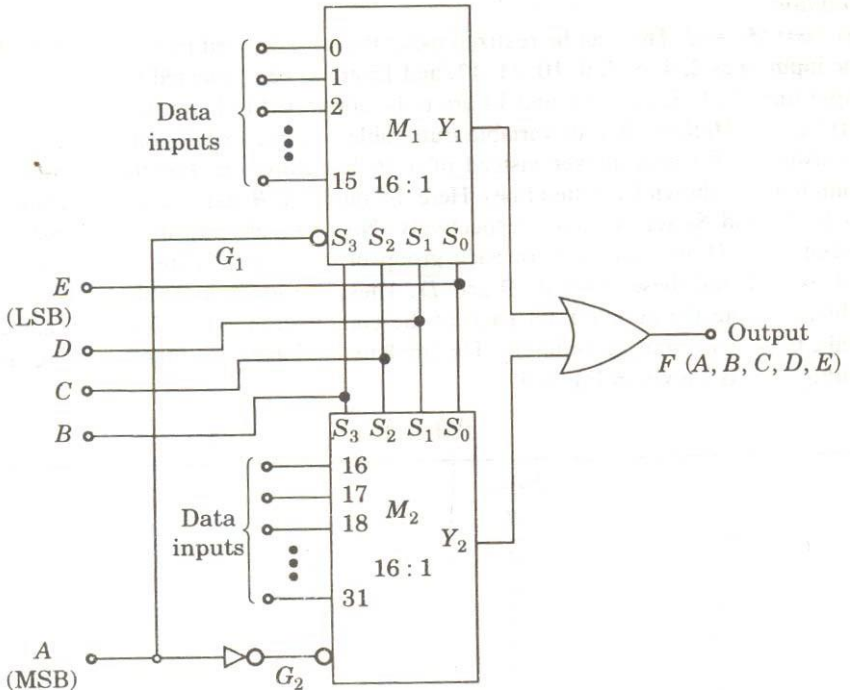


Fig. 3.48 32:1 multiplexer using two 16:1 multiplexers and one OR gate

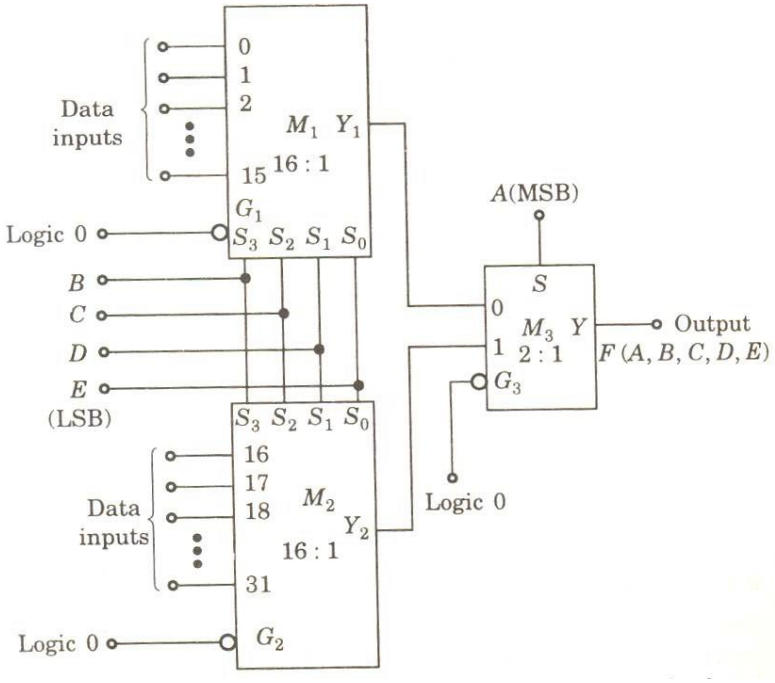


Fig. 3.49 32:1 multiplexer using two 16:1 multiplexers and one 2:1 multiplexer

These two general techniques can be used to expand to an n input multiplexer without any difficulty.

3.18 Demultiplexer

The demultiplexer performs the reverse operation of a multiplexer. It accepts a single input and distributes it over several outputs. Figure 3.50 gives the block diagram of a demultiplexer. The select input code determines to which output the data input will be transmitted.

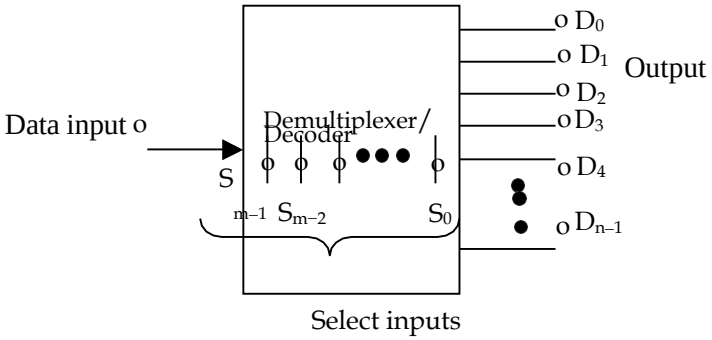


Fig. 3.50 Block diagram of a demultiplexer

The number of output lines in n and the number of select lines is m, where $n = 2^m$. This circuit can also be used as binary-to-decimal decoder with binary inputs applied at the select input lines and the output will be obtained on the corresponding line. The data inputline is to be connected to logic 1 level. This circuit can be designed using gates and it is left as an exercise for the reader. However, this device is available as an MSI IC and can conveniently be used for the design of combinational circuits. The device is very useful if multiple-output combinational circuit is to be designed, because this needs minimum package count. These devices are available (Table 3.23) as 2-line-to-line, 3-line-to-8-line, and 4-line- to-16-line decoders. The outputs of most of these devices are active-low, also there is an active low enable/data input terminal available.

Unlike the multiplexer, the decoder does require some gates in order to realizeBoolean expressions in the standard SOP form. The following example illustrates its use in combinational logic design.

Table 3.23 Available demultiplexer ICs

IC No.	Description	Output
74139	Dual 1:4 Demultiplexer (2-line-to-4-line decoder)	Inverted input
74155	Dual 1 : 4 Demltiplexer (2-line-to-4-line decoder)	1Y – Inverted input 2Y – Same as input
74156	– do –	Open – Inverted input 2Y – Same as input
74138	1 : 8 Demultiplexer (3-line-to-8line decoder)	Inverted input
74154	1:16 Demultiplexer (4-line-to-16-line decoder)	Same as input
74159	– do –	Same as input Open–collector

Example 3.29 Implement the following multi-output combinational logic circuit using a 4-to-16-line decoder.

$F_1 = \sum m (1, 2, 4, 7, 8, 11, 12, 13)$

$F_2 = \sum m (2, 3, 9, 11)$

$F_3 = \sum m(10, 12, 13, 14)$

$F_4 = \sum m (2, 4, 8)$

Solution : The realization is shown in Fig. 3.51

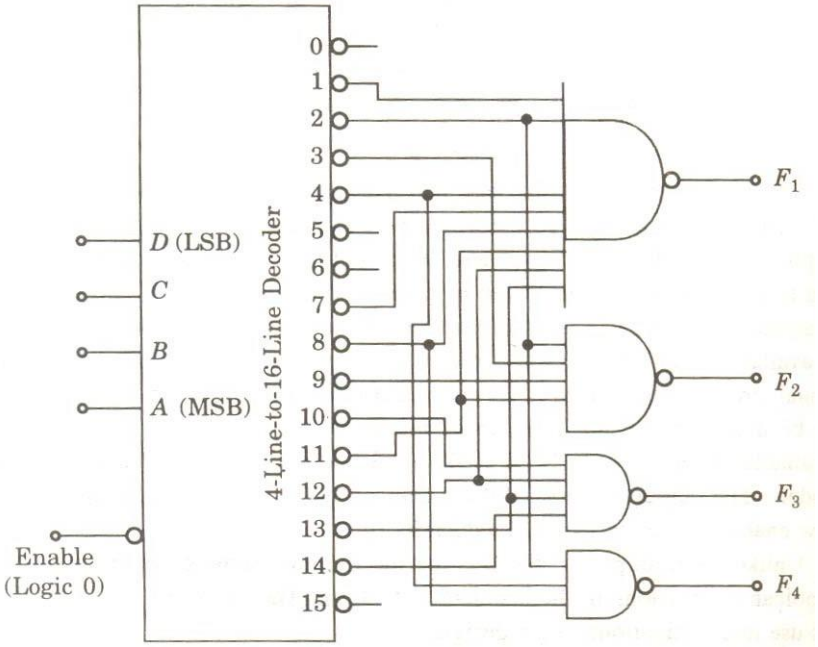


Fig. 3.51 Implementation of combinational logic circuit of Ex. 3.29

The four-bit input ABCD is applied at the Select input terminals S_3, S_2, S_1 and S_0. The output F_1 is required to be 1 for minterms 1, 2, 4, 7, 8, 11, 12 and 13. Therefore, a NAND gate is connected as shown. Similarly NAND gates are used for the outputs F_2, F_3 andF_4. Here, the decoder's outputs are active-low, therefore a NAND gate is required for every output of the combinational circuit.

In the combinational logic design using multiplexer, additional gates are not required, whereas design using demultiplexer requires additional gates. However, even with this disadvantage, the decoder is more economical in cases where non-trivial, multiple-output expressions of the same input variables are required. In such cases, one multiplexer is required for each output whereas it is likely that only one decoder will be required, supported with a few gates. Therefore, using a decoder could have advantages over using a multiplexer.

3.18.1 Demultiplexer Tree

Since 4–line-to-16-line decoders are the largest available circuits in ICs, to meet the larger inputs needs there should be a provision for expansion. This is made possible by using

enable input terminal. Figure 3.52 shows a 5-line-to-32-line decoder and Fig. 3.53 shows a 8-line-to-256-line decoder using 4-line-to-16-line decoders. In a similar way, any m-line-to-n-line decoder can be implemented. However, if only a few codes of a larger number need be recognized, the alternative approach, such as the one shown in Fig. 3.54, can be used. This is connected to detect the digital number 00011111. The most significant 4-bits are applied at A B C D inputs and the least-significant bits are applied at E F G H inputs. The output goes low when the most significant bits are 0 0 0 1 and the least-significant bits are 1 1 1 1. The circuit can be expanded to detect other 8-bit codes.

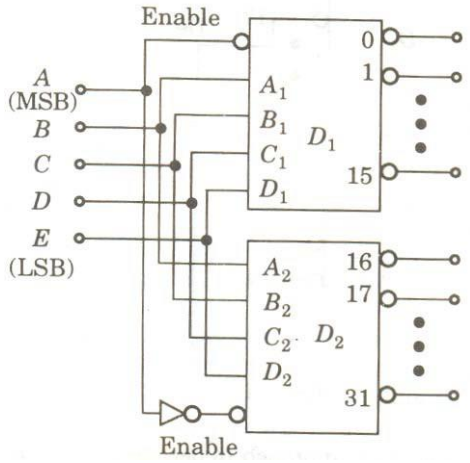


Fig. 3.52 5-line-to-32-line decoder using two 4-line-to-16-line decoders

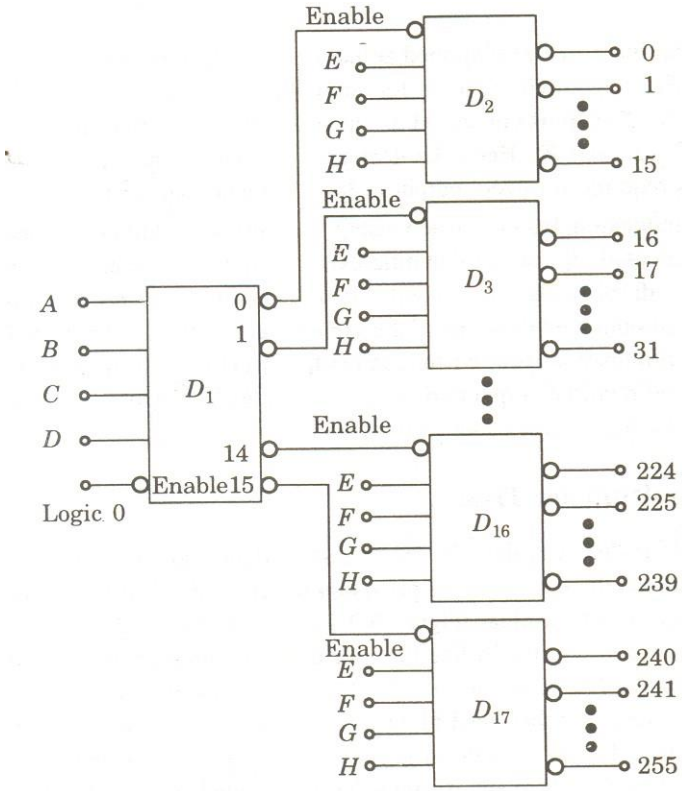


Fig. 3.53. 8-line-to-26-line decoder using 4-line-to-16-line decoder tree.

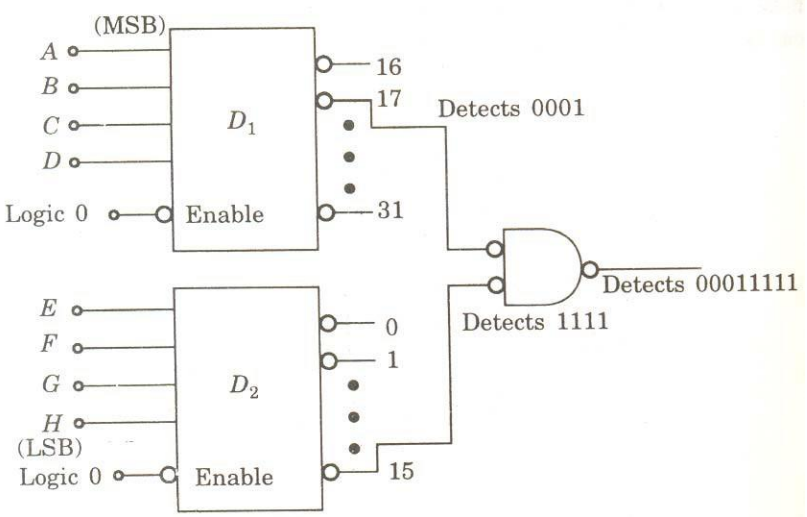


Fig. 3.54 An alternative approach to decode some combinations.

3.19 ENCODERS

3.19.1 Decimal to Binary Encoder

Digital computers operate on binary system. However, we work on decimal numbers and alphabets. The decimal numbers and alphabets are known s alphanumeric characters. An encoder is a device which converts alphanumeric characters to binary codes. An encoder may be decimal to binary, hexadecimal to binary, octal to BCD etc.

Fig. 3.55 shows a decimal to binary encoder. When push button 0 is pressed, none of the OR gates is energized and all outputs are low so that the output word is $Y_3\ Y_2\ Y_1\ Y_0 = 0\ 00\ 0$

If push button 4 is pressed, Y_2 OR gate has high input and the output word is

$$Y_3\ Y_2\ Y_1\ Y_0 = 0\ 1\ 0\ 0$$

When switch 7 is pressed, OR gates Y_2, Y_1 and Y_0 have high inputs and the output word is

$$Y_3\ Y_2\ Y_1\ Y_0 = 0\ \ 1\ \ 1\ \ 1$$

In a computer, it is required that a binary code be transmitted for every stroke of alphanumeric keyboard (a type writer or teletype). The keyboard of a computer has 26 capital alphabets, 26 lower case alphabets, 10 numerals (0 to 9) and about 22 special characters (+, – etc.). Thus, the total number of input codes is about 84. Encoder for generating an output binary word for these 84 inputs requires 7 bits (because $2^7 = 128$ and $2^6 = 64$). The block diagram of such an encoder is shown in Fig. 3.56. Whenever a key is pressed, the + 5 V supply is connected to one of the 84 input lines. Inside the box is an array of wiresinterconnected to generate the required code.

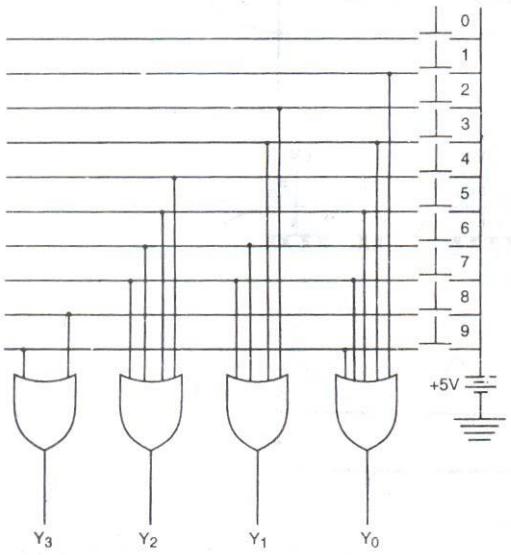


Fig. 3.55 Decimal to binary encoder

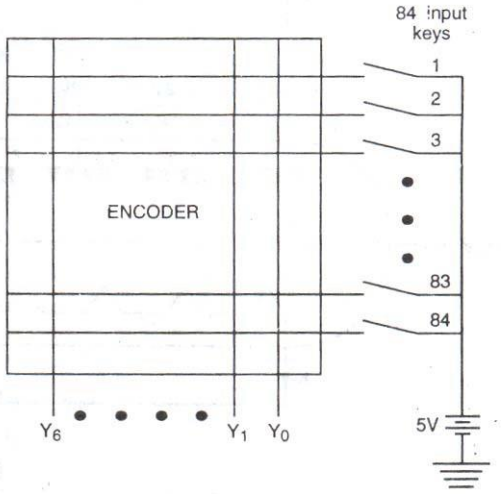


Fig. 3.56 Block diagram of decimal

3.19.2 Decimal to BCD Encoder

This encoder has 10 inputs (for decimal numbers 0 to 9) and 4 outputs for the BCD number. Thus it is a 10 line to 4 line encoder. Fig. 3.57 shows a block diagram of decimal to BCD encoder. Table 3.24 lists the decimal digits and the equivalent BCD numbers.

Table 3.24 BCD code

Decimal digit	BCD code			
	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0

5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

From Table 3.24, we can find the relationship between decimal digit and BCD bit. MSB of BCD bit is Y_3. For decimal digits 8 or 9, $Y_3 = 1$. Thus we can write OR expression for Y_3 bit as

$$Y_3 = 8 + 9 \quad \ldots(3.37)$$

Bit Y_2 is 1 for decimal digits 4, 5, 6, and 7. Thus we can write OR expression

$$Y_2 = 4 + 5 + 6 + 7 \quad \ldots(3.38)$$

Similarly,

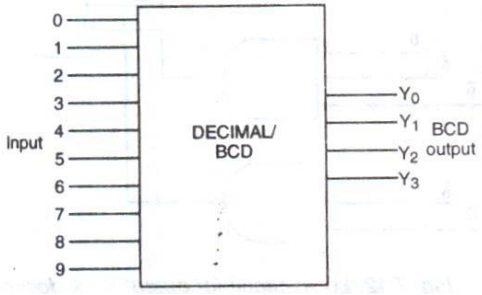
$$Y_1 = 2 + 3 + 6 + 7 \quad \ldots(3.39)$$

$$Y_0 = 1 + 3 + 5 + 7 + 9 \quad \ldots(3.40)$$

The logic circuit for above expression is shown in Fig. 3.58. It is seen that this circuit is the same as shown in Fig. 3.55. This is evidently due to the reason that for signle digit decimal numbers, the equivalent binary and BCD equivalents are the same. When a High appears on any of input lines the corresponding OR gates give the BCD output, e.g., if decimal input is B, High appears only on output 3(and LOW on Y_0, Y_1, Y_2), thus giving the BCD code for decimal 8 as 1000. Similarly, if decimal input is 7, then High appears on outputs Y_0, Y_1, Y_2 (and LOW on Y_3), thus giving BCD output as 0111.

3.19.3 Priority Encoder

When feeding data/program into a computer it is opposite that more than one key is pressed simultaneously. A priority function means tht the encoder will give priority to the



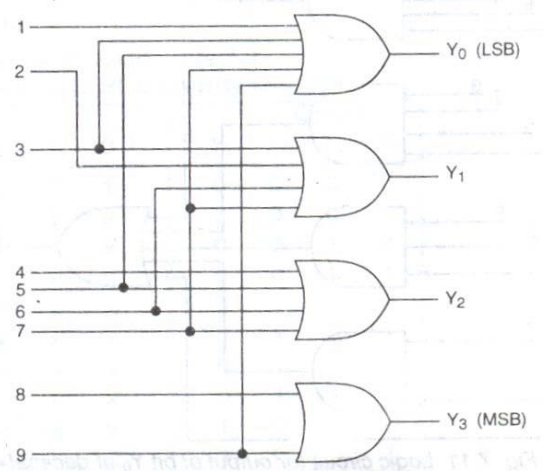


Fig. 3.57 Block diagram of decimal to BCD encoder **Fig. 3.58** Logic circuit for decimal to BCD encoder

highest order decimal digit in the inputs and ignore all other, e.g., in a priority encoder decimals 8 and 4 are pressed together (i.e., both 8 and 4 inputs are high), the encoder will convert the decimal 8 to the output and ignore 4.

3.19.4. Decimal to BCD Priority Encoder

The logic circuit for incorporating a priority in encoding must incorporate a feature to prevent a lower order digit input from disrupting the encoding of higher order digit. This is achieved by using inhibit (enable) gates. It is seen that in Fig. 7.10, Y_0 is High when inputs 1, 3, 5, 7, 9 are High. Priority can be added if digit 1 input activates gate Y_0 only if no higher digit (other than those which also activate Y_0) are high. This requirement can be expresses as under:

1. Y_0 is high if 1 is high and 2, 1, 6, 8 are low.
2. Y_0 is high if 3 is high and 4, 6, 8 are low.
3. Y_0 is high if 5 is high and 6, 8 are low.
4. Y_0 is high if 7 is high and 8 is low.
5. Y_0 is high if 9 is high.

The above 5 statements describe the priority for encoding for BCD bit Y_0. Thus Y_0 is high if any of above statements is true or in other words Y_0 is true if statement 1, statement 2, statement 3, statement 4 or statement 5 is true. Therefore, we can write the logic equation as

$$Y_0 = 1.\ \overline{4}.\overline{5}.\overline{8}.\overline{9} + 3\,.\,\overline{4}\,.\,\overline{5}\,.\,\overline{8}\,.\,\overline{9} + 6\,.\,\overline{8}\,.\,\overline{9} + 7\,.\overline{8}\,.\,\overline{9} \qquad \ldots(3.41)$$

The logic circuit to implement the logic equation (3.41) is shown in Fig. 3.59.

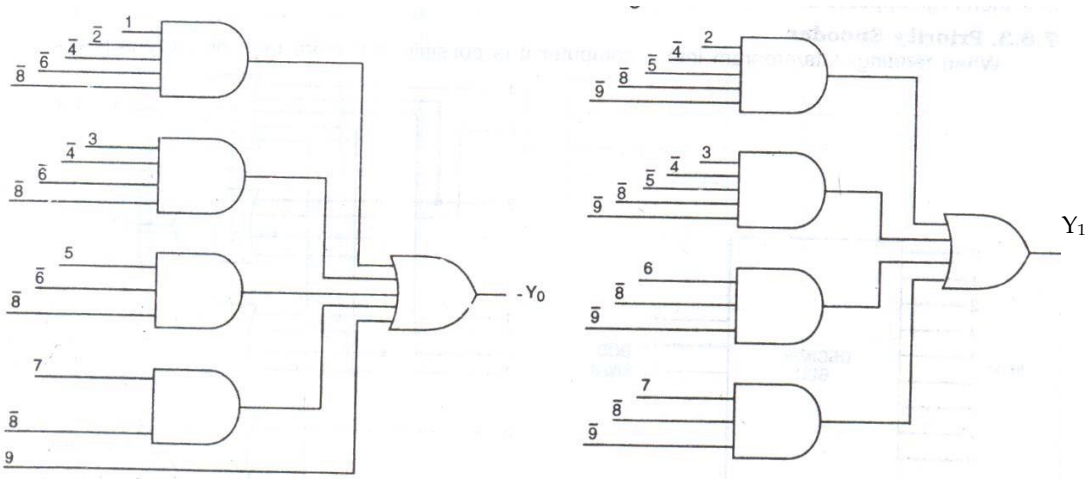
The logical statements for output Y_1 are as under :

1. Y_1 is high if 2 is high and 4, 5, 8, 9 are low.
2. Y_1 is high if 3 is high and 4, 5, 8, 9 are low.
3. Y_1 is high if 6 is high and 8, 9, are low.
4. Y_1 is high if 7 is high and 8, 9 are low.

The above statements can be written in the form of following equation

$$Y_1 = 2.\ \overline{4}\,.\,\overline{8}\,.\,\overline{9} + 3\,.\,\overline{4}\,.\,\overline{5}\,.\,\overline{8}\,.\,\overline{9} + 6\,.\,\overline{8}\,.\,\overline{9} + 7\,.\,\overline{8}\,.\,\overline{9} \qquad \ldots(3.42)$$

The logic circuit to implement Eqn. (3.42) is shown in Fig. 3.60



The output Y_2 can be described by the following statements :

1. Y_2 is high if 4 is high and 8, 9 are low.
2. Y_2 is high if 5 is high and 8, 9 are low.
3. Y_2 is high if 6 is high and 8, 9 are low.
4. Y_2 is high if 7 is high and 8, 9 are low.

Logical equation for above statements are

$$Y_2 = 4.\ \bar{8}\ .\ \bar{9} + 5\ .\ \bar{8}\ .\ \bar{9} + 6\ .\ \bar{8}\ .\ \bar{9} + 7\ .\ \bar{8}\ .\ \bar{9} \qquad \ldots(3.43)$$

Logical circuit for Eqn. (3.43) is shown in Fig. 3.61.

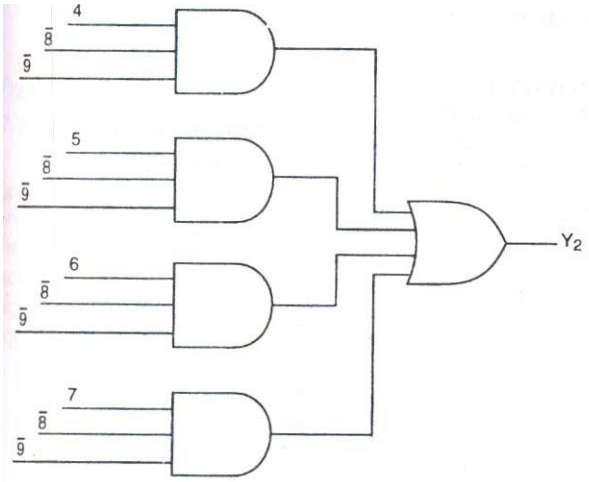


Fig. 3.61 Logic circuit for output Y_2 of decimal to BCD priority encoder

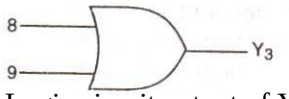


Fig. 3.62 Logic circuit output of Y_3 of decimal to BCD priority encoder

The statement for output Y_3 is

1. Y_3 is high if 8 is high and 9 is low.
2. Y_3 is high if 9 is high

Logical equation for above statements is

$$Y_3 = 8\ .\ \bar{9} + 9 \ = 8 + 9 \qquad \ldots(3.44)$$

Logic circuit for Eqn. (3.44) is shown in Fig. 3.62.

Truth table for decimal to BCD priority encoder is shown in Table 3.25.

Table 3.25 Truth table for decimal to BCD priority encoder

Inputs										Output			
9	8	7	6	5	4	3	2	1	0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	×	0	0	0	1
0	0	0	0	0	0	0	1	×	×	0	0	1	0
0	0	0	0	0	0	1	×	×	×	0	0	1	1
0	0	0	0	0	1	×	×	×	×	0	1	0	0
0	0	0	0	1	×	×	×	×	×	0	1	0	1
0	0	0	1	×	×	×	×	×	×	0	1	1	0
0	0	1	×	×	×	×	×	×	×	0	1	1	1
0	1	×	×	×	×	×	×	×	×	1	0	0	0
1	×	×	×	×	×	×	×	×	×	1	0	0	1

MSI chip 74LS147 is a decimal to BCD priority encoder.

Example 3.30 Fig. 3.63 shows a decimal to BCD priority encoder. As shown all inputs are active low.

(a) If pins 11, 3, 5 are low, find the state of outputs Y_0, Y_1.

(b) If pins 1, 4, 11 are low, find the state of outputs Y_0, Y_1, Y_2, Y_3

Solution : (a) Pin 5 is the highest order decimal input having a low level and represents decimal 8. Therefore, output is BCD equivalent of decimal 8 which is 1000.

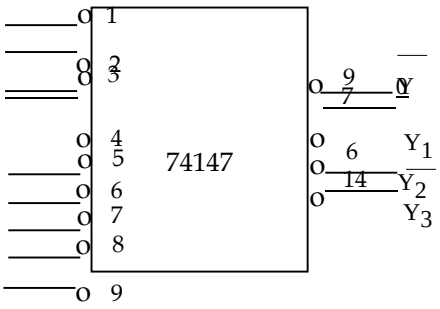


Fig. 3.63. 74147 decimal to BCD priority encoder

Therefore, $\overline{Y_3}$ is low and $\overline{Y_0}$, $\overline{Y_1}$, $\overline{Y_2}$ are High.

(b) Pin 4 is the highest order decimal input having a low level and represents decimal 7. Therefore, output is BCD equivalent for decimal 7 which is 0111. Therefore, $\overline{Y_3}$ is

high, $\overline{Y_2}$ is low, $\overline{Y_1}$ is low and $\overline{Y_0}$ is low.

Example 3.31 Design an octal to binary encoder.

Solution : The truth table for octal to binary is shown in Table 3.26. It has 8 inputs 0, 1, 2, 3, 4, 5, 6, 7 and three binary outputs (since $2^3 = 8$ weneed only 3 output lines). The rest of the implementation is similar to decimal to binary encoder and is shown in Fig. 3.64. The encoder consists of three OR gates. When any of inputs is high, the corresponding OR gates give highoutput, e.g., when we press 7, all the three OR gates give high output and the output is 111 which is theequivalent binary number. If we press 5, the OR gates number 3 and 1 give high output and OR gate 2 gives Low output thus giving the binary output as 101. Similarly we can check the outputs for the

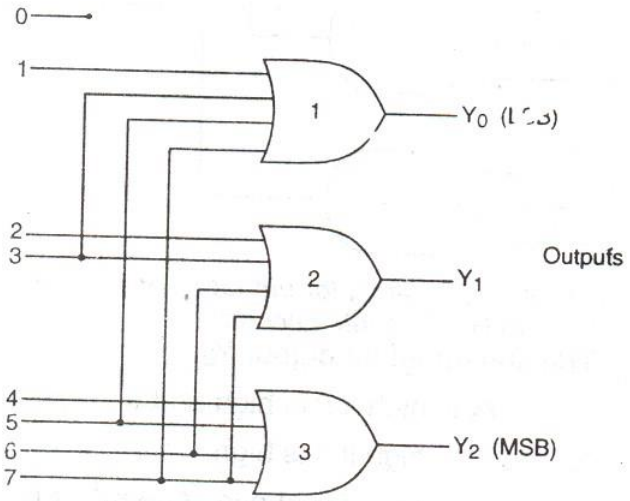


Fig. 3.64. Octal to binary encoder

remaining inputs. Inputs 0 is not connected to any gate because for 0 input all the three OR gates mustgive low output.

Table 3.26 Truth table for octal to binary encoder

Input	Output		
	Y_2	Y_1	Y_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

3.20 Decoder

Let the input be 1101. We need an AND gate as the basic decoding element because an AND gate gives high output only when all its inputs are high. In addition we need a NOT gate to convert 0 in the 2s bit position to 1. Thus the circuit will have one AND gate and one NOT gate as shown in Fig. 7.17(a).

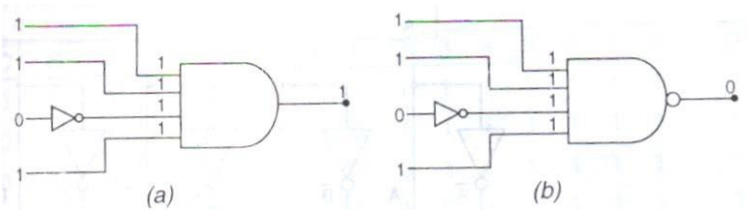


Fig. 3.65 Decoding for binary word 1101 (a) using AND gate (b) using NAND gate

The decoder for the above binary number 1101 can also be built with NAND gate and is shown in Fig. 3.65 (b). This logic circuit gives low active output.

3.20.1 Binary to Decimal Decoder

Fig. 7.18 shows a binary to decimal decoder. The input is from a 4 bit register. The inputs would be of the form 0101, 1001 etc. A particular input would cause one of the 10 output lines to be energized so that the corresponding decimal number would be the output, if the input is 0101, only the Y_5 AND gate has all high inputs and all other gates have at least one low input so that only Y_5 gate would give an output and number 5 would be displayed. If the input is 0 1 1 1, only AND gate Y_7 has all high input (all other AND gates have at least one low input) and number 7 is the output. The Boolean equations for different Y outputs are as under

$$Y_0 = \overline{A}\ \overline{B}\ \overline{C}\ \overline{D}$$

$$Y_1 = \overline{A}\ \overline{B}\ \overline{C}\ D$$

$$Y_2 = \overline{A}\ \overline{B}\ C\ \overline{D}$$

$$Y_3 = \overline{A}\ \overline{B}\ C\ D$$

$$Y_4 = \overline{A}\ B\ \overline{C}\ \overline{D}$$

$$Y_5 = \overline{A}\ B\ \overline{C}\ D$$

$$Y_6 = \overline{A}\ B\ C\ \overline{D}$$

$$Y_7 = \overline{A}\ B\ C\ D$$

$$Y_8 = A\ \overline{B}\ \overline{C}\ \overline{D}$$

$$Y_9 = A\ \overline{B}\ \overline{C}\ D$$

The circuit of Fig. 3.66 is 4 line to 10 line decoder (because it has 4 input lines and 10 output lines). It is also known as 1 of 10 decoder because only 1 out of 10 output lines is energized at one time. When a decoder is combined with a high current driving output, it is

known as decoder/driver. The device 7447 is used for decoding and driving a seven segment display. The decoder/driver turns on the LEDs so that the corresponding decimal number is displayed.

3.20.2 Four Bit Binary Decoder

This decoder is also called 4 line to 16 line decoder because there are 4 inputs (from the four bits) and 16 outputs (because $2^4 = 16$, there can be 16 outputs). Table 3.27 lists all the possible combinations of input and output words. It is seen from Table 3.27 that we require the complements $\overline{\overline{A}}, \overline{B}, \overline{C}, D$ a number of times. Therefore, it is desirable that these complements are generated once and then used for all the gates.

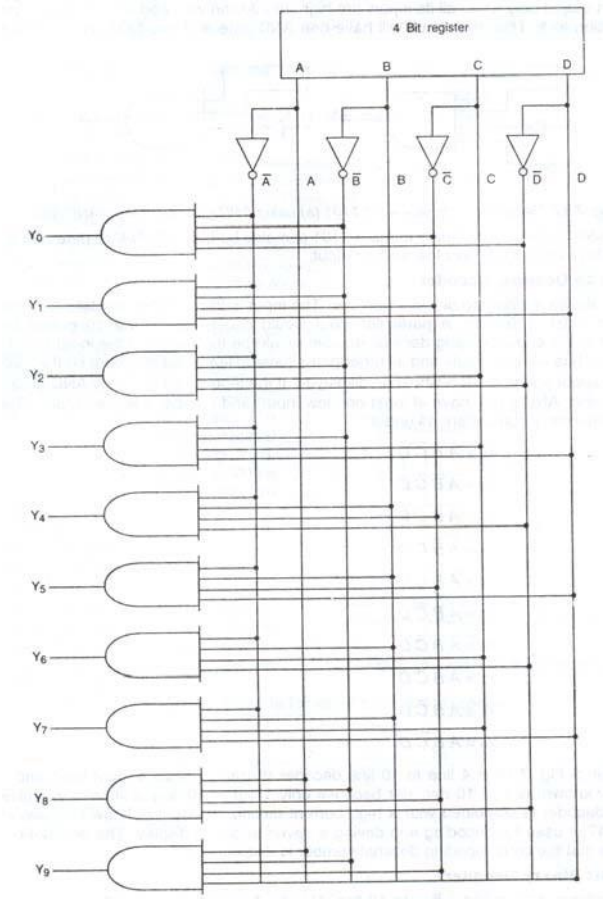


Fig. 3.66 Binary to decimal decoder (1 of 10)

TTL MSI 74154 is a 4 bit (or 4 line to 16 line decoder). It is seen that 4 additional NOT gates have been used on the input. These NOT gates prevent excessive loading of the driving source. Each input is connected to the input of only one NOT gate. In addition an enable function is also provided in this chip. This function is implemented with a NOR gate used as negative AND gate. A low level in each input $\overline{G_1}$ and $\overline{G_2}$ is required to make the output G high. The output G of enable is one of the inputs to all the 16 NAND gates. G must be high for the gates to be enabled. If the enable gate is not activated, then all sixteen decoder outputs will be High irrespective of the state of input variables A, B, C, D.

Table 3.27 Decoding functions and truth table for a 4-line-to-16 line decoder

Decimal Digit	Binarylnputs				Logic	Outputs														
	D	C	B	A	Function	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	$\overline{D}\,\overline{C}\,\overline{B}\,\overline{A}$	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	1	$\overline{D}\,\overline{C}\,\overline{B}A$	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	1	0	$\overline{D}\,\overline{C}B\,\overline{A}$	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
3	0	0	1	1	$\overline{D}\,\overline{C}BA$	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
4	0	1	0	0	$\overline{D}C\,\overline{B}\,\overline{A}$	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
5	0	1	0	1	$\overline{D}C\,\overline{B}A$	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
6	0	1	1	0	$\overline{D}CB\,\overline{A}$	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
7	0	1	1	1	$\overline{D}CBA$	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
8	1	0	0	0	$D\,\overline{C}\,\overline{B}\,\overline{A}$	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
9	1	0	0	1	$D\,\overline{C}\,\overline{B}A$	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
10	1	0	1	0	$D\,\overline{C}B\,\overline{A}$	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
11	1	0	1	1	$D\,\overline{C}BA$	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
12	1	1	0	0	$DC\,\overline{B}\,\overline{A}$	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
13	1	1	0	1	$DC\,\overline{B}A$	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
14	1	1	1	0	$DCB\,\overline{A}$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
15	1	1	1	1	DCBA	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

3.20.3 BCD to Decimal Decoder

BCD to decimal decoder is similar to binary to decimal decoder. It converts each BCD code into one of the 10 possible decimal digits. It is also called 4 line to 10 line decoder. The BCD decoding functions are given in Table 3.28. Since BCD code represents only 10 decimal digits, we use only 10 decoding gates.

The decoding can be done in two ways active low and active high.

The active low output decoding uses NAND gates. The symbol is shown in Fig. 3.67 (a). The small circles at the output lines indicate active low output.

The active high output decoding requires AND gates. The symbol in Fig. 3.67 (b) shows active High output decoding. For this decoding the logic circuit is the same as in Fig. 3.66.

Table 3.28 BCD decoding functions

BCD code				Logic function	Decimal digit
D MSB	C	B	A LSB		
0	0	0	0	$\overline{D}\,\overline{C}\,\overline{B}\,\overline{A}$	0
0	0	0	1	$\overline{D}\,\overline{C}\,\overline{B}\,A$	1
0	0	1	0	$\overline{D}\,\overline{C}\,B\,\overline{A}$	2
0	0	1	1	$\overline{D}\,\overline{C}\,B\,A$	3
0	1	0	0	$\overline{D}\,C\,\overline{B}\,\overline{A}$	4
0	1	0	1	$\overline{D}\,C\,\overline{B}\,A$	5
0	1	1	0	$\overline{D}\,C\,B\,\overline{A}$	6
0	1	1	1	$\overline{D}\,C\,B\,A$	7
1	0	0	0	$D\,\overline{C}\,\overline{B}\,\overline{A}$	8
1	0	0	1	$D\,\overline{C}\,\overline{B}\,A$	9

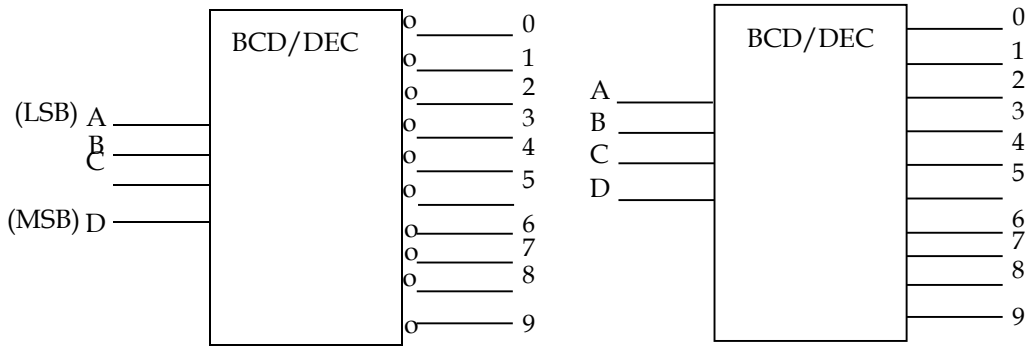


Fig. 3.67. BCD-decimal decoder (a) active low (b) active high

Example 3.32 The input waves of Fig. 3.68 (a) are applied to an BCD-DEC decoder having active high outputs. Draw output waveforms.

Solution : During interval t_0, all inputs are low which means input is 0000 and output is 0. Since it is active high output device, output will indicate High for decimal digit 0 and Low forall other decimal digits. The input and outputs for the remaining intervals are listed below:

	Inputs				Output
	D	C	B	A	
t_0	0	0	0	0	0
t_1	0	0	0	1	1
t_2	0	0	1	0	2
t_3	0	0	1	1	3
t_4	0	1	0	0	4
t_5	0	1	0	1	5
t_6	0	1	1	0	6
t_7	0	1	1	1	7
t_8	1	0	0	0	8
t_9	1	0	0	1	9

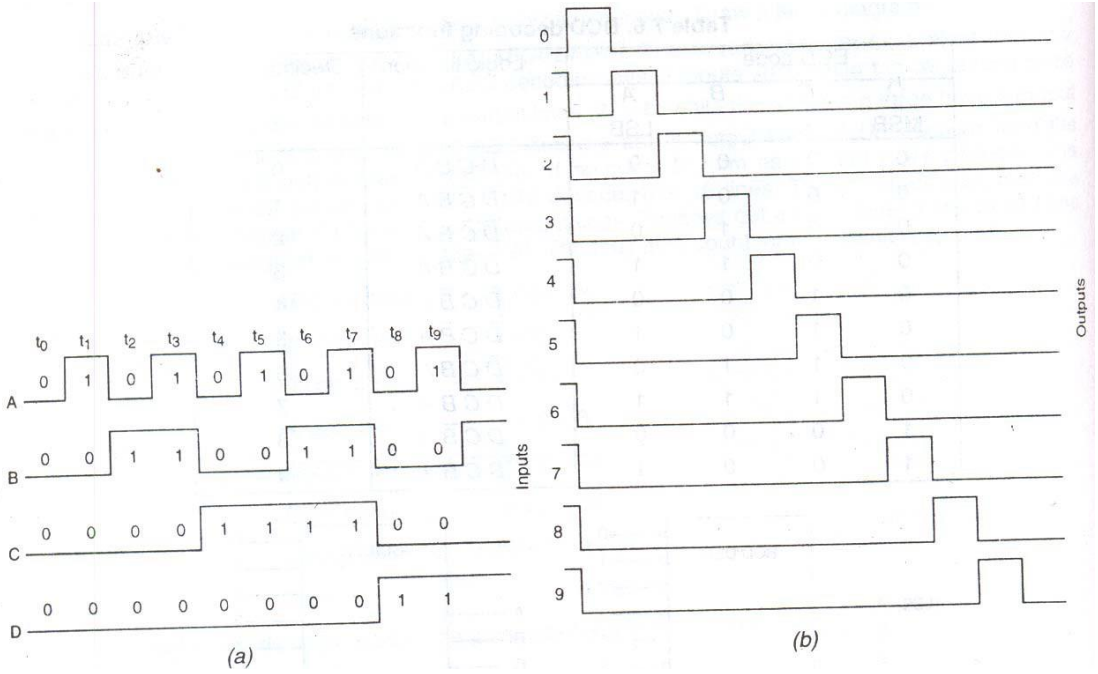


Fig. 3.68

The outputs are shown in Fig. 3.68 (b).

3.21 BCD DECODER/SEVEN SEGMENT DRIVER

It is a combination of BCD–decimal decoder and a seven segment display driver. The input is BCD. The outputs drive 7 segment display unit (LED or LCD etc.) and the decimal read out is displayed.

The seven segment display can be connected either in common anode or common cathode configuration.

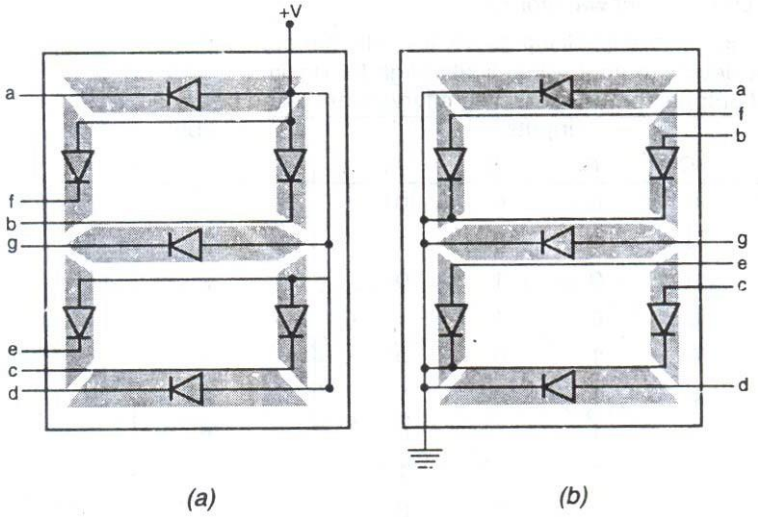


Fig. 3.69. Seven segment display (a) common anode connection suitable for active low output decoder (b) common cathode connection suitable for active high output decoder

When the decoder gives active low output, the common anode connection is used. Due to the low level output from decoder, the corresponding group of LEDs is forward biasedand current flows through them producing the display. When the decoder gives active High output, the common cathode connection is used. Due to High level output from the decoder, the corresponding group of LEDs is forward biased, current flows through them producingthe display. These two connections are shown in Fig. 3.69. It is seen that in both the connections the p-n junctions are forward biased because current can flow through a p-n junction only when it is forward biased.

We can now formulate the decoding logic required for the display. It is seen from Table 3.29 that segment a is activated when the output of decoder is 0, 2, 3, 5, 7, 8, 9. The Boolean expression for active segment a can be written as

$$a = \overline{D}\,\overline{C}\,\overline{B}\,\overline{A} + \overline{D}\,\overline{C}\,B\,\overline{A} + \overline{D}\,\overline{C}\,BA + \overline{D}\,C\,\overline{B}\,A + \overline{D}\,CBA, D\,\overline{C}\overline{B}\,A \qquad \ldots(3.45)$$

Eqn. (3.45) says that segment a is activated if BCD code is 0 or 2 or 3 or 5 or 7 or 8 or 9. In the same way we can write Boolean expression for all the other segments. These Boolean expressions for various segments are shown in Table 3.29.

Table 3.29 Boolean terms for seven segments

Segment	Used in digits	Boolean terms
a	0, 2, 3, 5, 7, 8, 9	$\overline{D}\,\overline{C}\,\overline{B}\,\overline{A} + \overline{D}\,\overline{C}\,B\overline{A} + \overline{D}\,\overline{C}\,BA + \overline{D}\,C\overline{B}A + \overline{D}\,CBA + D\overline{C}\,\overline{B}\,\overline{A} + D\overline{C}\,\overline{B}A$
b	0, 1, 2, 3, 4, 7, 8, 9	$\overline{D}\,\overline{C}\,\overline{B}\,\overline{A} + \overline{D}\,\overline{C}\,\overline{B}A + \overline{D}\,\overline{C}\,B\overline{A} + \overline{D}\,\overline{C}\,BA + \overline{D}C\,\overline{B}\,\overline{A} + \overline{D}CBA + D\overline{C}\,\overline{B}\,\overline{A} + D\overline{C}\,\overline{B}A$
c	0, 1, 3, 4, 5, 6, 7, 8, 9	$\overline{D}\,\overline{C}\,\overline{B}\,\overline{A} + \overline{D}\,\overline{C}\,\overline{B}A + \overline{D}\,\overline{C}BA + \overline{D}C\,\overline{B}\,\overline{A} + \overline{D}\,\overline{C}BA + \overline{D}CB\overline{A} + \overline{D}CBA + D\overline{C}\,\overline{B}\,\overline{A} + \overline{D}C\,\overline{B}A$
d	0, 2, 3, 5, 6, 8	$\overline{D}\,\overline{C}\,\overline{B}\,\overline{A} + \overline{D}\,\overline{C}\,B\overline{A} + \overline{D}\,\overline{C}BA + \overline{D}C\,\overline{B}A + \overline{D}CB\overline{A} + D\overline{C}\,\overline{B}\,\overline{A}$
e	0, 2, 6, 8	$\overline{D}\,\overline{C}\,\overline{B}\,\overline{A} + \overline{D}\,\overline{C}\,B\overline{A} + \overline{D}CB\overline{A} + D\overline{C}\,\overline{B}\,\overline{A}$
f	0, 4, 5, 6, 8, 9	$\overline{D}\,\overline{C}\,\overline{B}\,\overline{A} + \overline{D}C\,\overline{B}\,\overline{A} + \overline{D}C\,\overline{B}A + \overline{D}CB\overline{A} + D\overline{C}\,\overline{B}\,\overline{A} + D\overline{C}\,\overline{B}A$
g	2, 3, 4, 5, 6, 8, 9	$\overline{D}\,\overline{C}B\overline{A} + \overline{D}\,\overline{C}BA + \overline{D}C\,\overline{B}\,\overline{A} + \overline{D}C\,\overline{B}A + \overline{D}CB\overline{A} + D\overline{C}\,\overline{B}\,\overline{A} + D\overline{C}\,\overline{B}A$

The logic circuit for the various segments can be implemented from the Boolean terms listed in Table 3.29.

3.22 CODE CONVERTERS

There is a wide variety of binary codes used in digital systems. Some of these codes are binary-coded-decimal (BCD), Excess-3, Gray, octal, hexadecimal, etc. Often, it is required to convert from one code to another. For example the input to a digital system may be in natural BCD and the output may be 7-segment LEDs. The digital system used may be capable of processing the data in straight binary format. Therefore, the data has to be converted from BCD to binary at the output.

The BCD output has to be converted to 7-segment code before it can be used to drive the LEDs. Similarly, octal and hexadecimal codes are widely used in micro-processors and digital computers as inputs and outputs. The various code converters can be designed using gates, multiplexers or demultiplexers. However, there are some MSI ICs available for performing these conversions and are extremely useful in the design of digital systems. These devices have been discussed below.

3.22.1 BCD-to-Binary Converter

The block diagram of BCD-to-binary converter IC 74184 is given in Fig. 3.70 and Table 3.30 gives its truth table. This device can be used as a $1\frac{1}{2}$ decade BCD-to-binary converter as shown in Fig. 3.71.

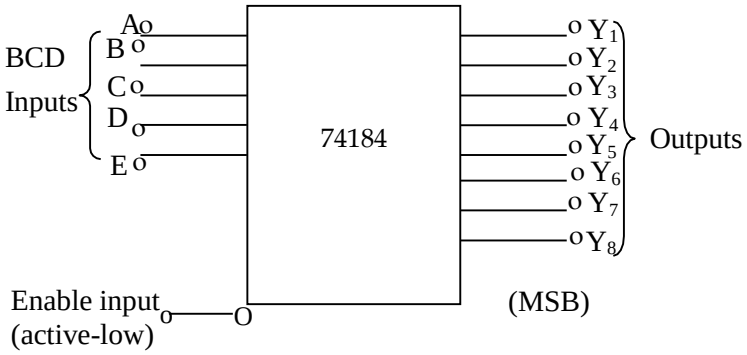
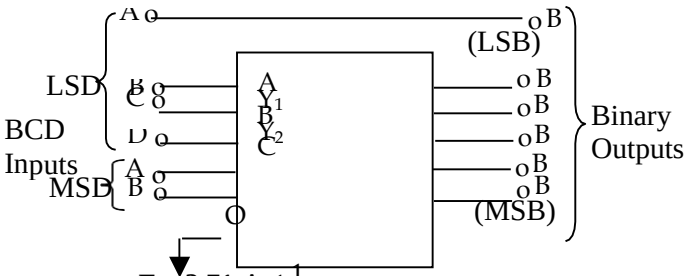


Fig. 3.70 Block diagram of 74184.



Fg. 3.71 A $1\frac{1}{2}$ decade BCD-to-binary

The BCD inputs are applied at the input terminals A through E and the LSB of the least-significant BCD digit bypasses the converter and appears as the LSB of the binary output. It accepts two BCD digits – a full digit D_1 C_1 B_1 A_1 and the two least-significant bits of a second digit B_2 A_2. This means that the BCD inputs 00 through 39 can be converted to corresponding binary output by this circuit. Terminals Y_6, Y_7, and Y_8 are not used for BCD- to-binary conversion. These terminals are used to obtain the 9's complement and the 10's

Table 3.30 Truth table of 74184 BCD-to-binary converter

BCD	Inputs						Outputs				
Words	E	D	C	B	A	G	Y_5	Y_4	Y_3	Y_2	Y_1
0–1	0	0	0	0	0	0	0	0	0	0	0
2–3	0	0	0	0	1	0	0	0	0	0	1
4–5	0	0	0	1	0	0	0	0	0	1	0
6–7	0	0	0	1	1	0	0	0	0	1	1
8–9	0	0	1	0	0	0	0	0	1	0	0
10–11	0	1	0	0	0	0	0	0	1	0	1
12–13	0	1	0	0	1	0	0	0	1	1	0
14–15	0	1	0	1	0	0	0	0	1	1	1
16–17	0	1	0	1	1	0	0	1	0	0	0
18–19	0	1	1	0	0	0	0	1	0	0	1
20–21	1	0	0	0	0	0	0	1	0	1	0
22–23	1	0	0	0	1	0	0	1	0	1	1
24–25	1	0	0	1	0	0	0	1	1	0	0
26–27	1	0	0	1	1	0	0	1	1	0	1
28–29	1	0	1	0	0	0	0	1	1	1	0
30–31	1	1	0	0	0	0	0	1	1	1	1
32–33	1	1	0	0	0	0	1	0	0	0	0
34–35	1	1	0	1	1	0	1	0	0	0	1
36–37	1	1	0	1	1	0	1	0	0	1	0
38–39	1	1	1	0	0	0	1	0	0	1	1
Any	×	×	×	×	×	1	1	1	1	1	1

Complement of BCD numbers, useful for BCD arithmetic operations. Figure 3.72 gives the block diagram of BCD 9's complement converter and Table 3.31 gives its truth table.

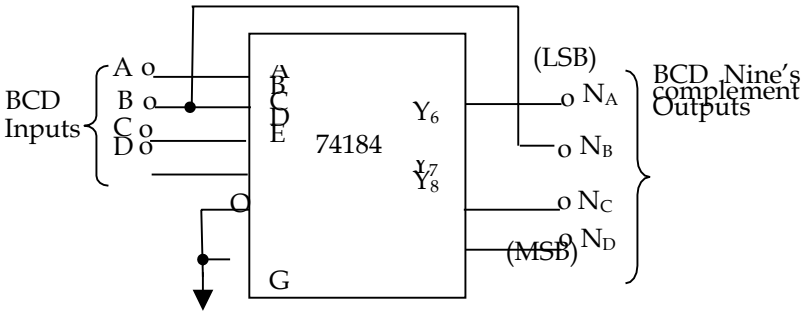


Fig. 3.72 Block diagram fo BCD 9's complement converter using 74184

Table 3.31 Truth table of 74184 as BCD 9's complement converter

BCD Word	Inputs						Outputs			BCD 9's complement			
	E	D	C	B	A	G	Y_8	Y_7	Y_6	N_D	N_C	N_B	N_A
0	0	0	0	0	0	0	1	0	1	1	0	0	1
1	0	0	0	0	1	0	1	0	0	1	0	0	0
2	0	0	0	1	0	0	0	1	1	0	1	1	1
3	0	0	0	1	1	0	0	1	0	0	1	1	0
4	0	0	1	0	0	0	0	1	1	0	1	0	1
5	0	0	1	0	1	0	0	1	0	0	1	0	0
6	0	0	1	1	0	0	0	0	1	0	0	1	1
7	0	0	1	1	1	0	0	0	0	0	0	1	0
8	0	1	0	0	0	0	0	0	1	0	0	0	1
9	0	1	0	0	1	0	0	0	0	0	0	0	0
Any	×	×	×	×	×	1	1	1	1				

BCD input is applied at DCBA input terminals and its 9's complement appears at N_D N_C N_B N_A terminals.

SUMMARY

The most popular and commonly available MSI ICs and their applications have been discussed in this chapter. Since the logic equations for thee circuits are very complex, therefore, their applicability must be recognized at the system level. Quite often, the system specifications are affected by the initial knowledge of available standard circuits. Each technique discussed can be thought of as a tool and each tool has its place. The designer hs tomake a proper choice of the tool for the job. Though more than one tool may work for a given job, the key is to select the right one. Although system design has been oriented aroundmaking use of higher levels of integration, a lot of little jobs of interfacing the MSI devices are still best done with discrete gates.

Active-low inputs and outputs have been indicated by small circles throughout. Recently, many authors have started using an alternative symbol (small right triangle) to represent the active-low input/output to avoid confusion (of inversion) associated with the small circle.

PROBLEMS

Realize the logic function of f{A, B, C, D} = ∑m (1, 3, 5, 9, 12, 15)

(a) a 16:1 multiplexer

(b) an 8:1 multiplexer.

Design a 32:1 multiplexer using two 16:1 multiplexer iCs.

Design a full adder using 8:1 multiplexer iCs. Compare the IC package count with the NAND-NAND realization.

Design a 4-bit ADDER/SUBTRACTOR circuit with ADD/SUB control line.

Design a 40:1 multiplexer using 8:1 multiplexers.

Design a 1:40 demultiplexer using BCD-to-decimal decoders.

Design a 4-digit BCD adder using 7483 adders.

Design a 2-bit comparator using gates.

Design an 8-bit comparator using only two 7485s.

3.10 Verify the operation of the 24-bit comparator of Fig. 3.41 for the following numbers:

A = 100110000111011001010010

B = 101110000111011000100011

3.11 Design an one digit-BCD-to-binary convergent using 74184.

3.12 Show that the hexadecimal-to-binary encoder can also work as a priority encoder.

3.13 Design a 6-bit odd/even parity checker.

3.14 Design a parity generator circuit to add an odd parity bit to a 7-bit word.

3.15 Design a parity generator circuit to add an even parity bit to a 14-bit word. Use two 74180 packages.

CHAPTER-4

FLIP FLOPS

4.1 INTRODUCTION

So far we have directed our studies towards the analysis and design of combinational digital circuits. Though very important, it constitutes only a part of digital systems. The other major aspect of digital systems is analysis and design of sequential circuits. However, sequential circuit design depends, to a large extent, on the combinational circuit design discussed earlier.

There are many applications in which digital outputs are required to be generated in accordance with the sequence in which the input signals are received. This requirement can not be satisfied using a combinational logic system. These applications require outputs to be generated that are not only dependent on the present input conditions but they also depend upon the past history of these inputs. The past history is provided by feedback form the outputback to the input.

A block diagram of a sequential circuit is shown in Fig. 4.1. It consists of combinational circuits which accept digital signals from external inputs and from outputs of

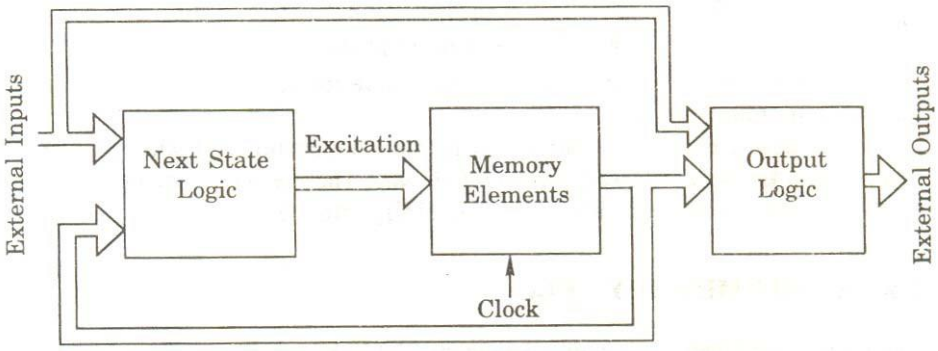


Fig. 4.1 Block diagram of sequential circuit.

memory elements and generates signals for external outputs and for inputs to memory elements referred to as excitation.

A memory element is some medium in which one bit of information (1 or 0) can be stored or retained until necessary, and thereafter its contents can be replaced by a new value. The contents of memory elements in Fig. 4.1 can be changed by the outputs of the combinational circuit which are connected to its input.

The combinational circuit performs certain operations, some of which are used to determine the digital signals to be stored in memory elements. The other operations are performed on external inputs and memory outputs to generate the external outputs.

The above process demonstrates the dependence of the external outputs of a sequential circuit on the external inputs and the present contents of the memory elements (referred to as the *present state* of memory elements). The new contents of the memory elements, referred to as the *next state;* depend on the external inputs and the present state, hence, the output of a sequential circuit is a function of the time sequence of inputs and the internal states.

Sequential circuits are classified in two main categories, known as *Sychronous* and *Asynchronous* sequential circuits depending on timing of their signals.

A sequential circuit whose behaviour depends upon the sequence in which the input signals change is referred to as an asynchronous sequential circuit. The outputs will be affected whenever the inputs change. The commonly used memory elements in these circuits are time delay devices. These can be regarded as combinational circuits with feedback.

A sequential circuit whose behaviour can be defined from the knowledge of its signal at discrete instants of time is referred to as a synchronous sequential circuit. In these systems, the memory element are affected only at discrete instants of time. The synchronization is achieved by a timing device known as a system clock which generates a periodic train of clock pulses as shown in Fig. 4.2. The outputs are affected only with the application of a clock pulse.

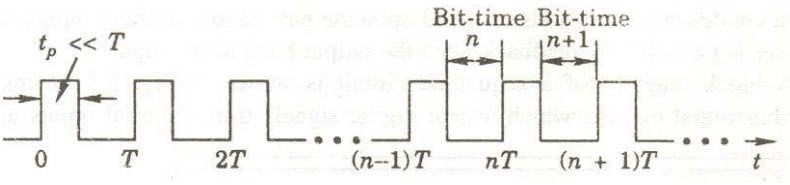


Fig. 4.2 A train of pulses.

Since the design of asynchronous circuits is more-tedious and difficult, therefore their uses are rather limited.

Synchronous circuits have gained considerable domination and wide popularity and are also known as *clocked-sequential circuits.* The memory elements used are FLIP-FLOPs which are capable of storing binary information.

4.2 1-BIT MEMORY CELL

The basic digital memory circuit is known as FLIP-FLOP. It has two stable states which are known as the 1 state and the 0 state. It can be obtained by using NAND or NOR gates. We shall be systematically developing a FLIP-FLOP circuit starting from the fundamental circuit shown in Fig. 7.3. It consists of two inverters G_1 and G_2 (NAND gates used as inverters). The output of G_1 is connected to the input of G_2 (A_2) and the output of G_2 is connected to the inputof G_1 (A_1).

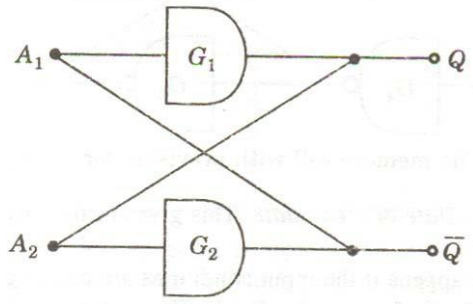


Fig. 4.3 Cross- coupled inverters as a memory element.

Let us assume the output of G_1 to be Q = 1, which is also the input of G_2 (A_2 = 1). Therefore, the output of G_2 will be $Q = \overline{0}0$, which makes A_1= 0 and consequently Q = 1 which confirms our assumption.

In a similar manner, it can be demonstrated that if Q = 0, then $\overline{Q}$ =1 and this is also consistent with the circuit connections.

From the above discussion we note the following

1. The outputs Q and $\overline{Q}$ are always complementary.
2. The circuit has two stable states; in one of the stable state Q = 1 which is referred to as the 1 state (or set state) whereas in the other stable state Q = 0 which is referred to as the 0 state (or reset state).
3. If the circuit is in 1 state, it continues to remain in this state and similarly if it is in 0 state, it continues to remain in this state. This property of the circuit is referred to as *memory,* i.e. it can store 1- bit of digital information.

Since this information is locked or latched in this circuit, therefore, this circuit is also referred to as a latch.

In the latch of Fig. 4.3, there is no way of entering the desired digital information to be stored in it. In fact, when the power is switched on, the circuit switches to one of the stable states (Q = 1 or 0) and it is not possible to predict the state. If we replace the inverters G_1 and G_2 with 2-input NAND gates, the other input terminals of the NAND gates can be used to enter the desired digital information. The modified circuit is shown in Fig. 4.4. Two additional inverters G_3 and G_4 have been added for reasons which will become clear from the following discussion.

If S = R = 0, the circuit is exactly the same as that of Fig. 4.3. If S = 1 and R = 0, the output of G_3 will be 0 and the output of G_4 will be 1. Since one of the inputs of G_1 is 0, its output will certainly be 1. Consequently, both the inputs of G_2 will be giving an output $\overline{Q}$ = 0. Hence, for this input condition, Q = 1 and $\overline{Q}$ = 0. Similarly, if S = 0 and R = 1 then the outputs will be Q = 0 and $\overline{Q}$ = 1. The first of these two input conditions (S = 1, R =0) makes Q = 1 which is referred to as the set state, whereas the second input condition (S = 0, R = 1) makes Q = 0 which is

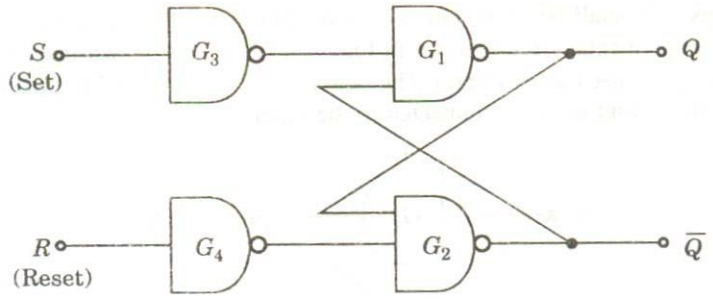


Fig. 4.4. The memory cell with provision for entering data.

referred to as the reset state or clear state. This gives us the means for entering the desired bit in the latch.

Now we see what happens if the input conditions are changed from S = 1, R = 0 to S = R = 0 or from S = 0, R = 1 to S = R = 0. The output remains unaltered. This shows the basic difference between a combinational circuit and a sequential circuit, even though the sequential circuit is made up of combinational circuits.

The two input terminals are designated as set (S) and reset (R) because S = 1 brings the circuit in set state and R = 1 brings it to reset or clear state.

If S = R = 1, both the outputs Q and $\overline{Q}$ will try to become 1 which is not allowed and therefore, this input condition is prohibited.

4.3 CLOCKED S-R FLIP-FLOP

It is often required to set or reset the memory cell (Fig. 4.4) in synchronism with a train of pulses (Fig. 4.2) known as clock (abbreviated as CK). Such a circuit is shown in Fig. 4.5, and is referred to as a *clocked set-reset* (S-R) FLIP- FLOP.

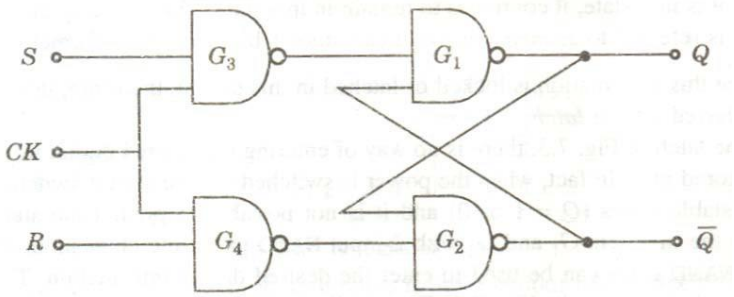


Fig. 4.5 A clocked S-R FOIP-FLOP.

In this circuit, if a clock pulse is present (CK = 1), its operation is exactly the same as that of Fig. 4.4 On the other hand, when the clock pulse is not present (CK =0), the gates G_3 and G_4 are inhibited, i.e. their outputs are 1 irrespective of the values of S or R. In other words, the circuit responds to the inputs S and R only when the clock is present.

Assuming that the inputs do not change during the presence of the clock pulse, we can express the operation of a FLIP-FLOP in the form of the truth table in Table 4.1 for the S-R FLIP-FLOP. Here S_n and R_n denote the inputs and Q_n the output during the bit time n (Fig. 4.2) Q_{n+1} denotes the output Q after the pulse passes, i.e. in the bit time n + 1.

Table 4.1 Truth table of S-R FLIP-FLOP

Inputs		Output
S_n	R_n	Q_{n+1}
0	0	Q_n
1	0	1
0	1	0
1	1	?

If $S_n = R_n = 0$, and the clock pulse is applied, the output at the end of the clock pulse is same as the output before the clock pulse, i. e. $Q_{n+1} = Q_n$. This is indicated in the first row of the truth table.

If $S_n = 1$ $R_n = 0$, the output at the end of the clock pulse will be 1, where if $S_n = 0$ and $R_n = 1$, then $Q_{n+1} = 0$. These are indicated in the second and third rows of the truth table respectively.

In the circuit of Fig. 4.4, it was mentioned that S = R = 1 is not allowed. Let us see what happens in the S-R FLIP-FLOP of Fig. 4.5 if $S_n = R_n = 1$. When the clock is present the outputs of gates G_3 and G_4 are both 0, making one of the inputs of G_1 and G_2 NAND gates 0. Consequently, Q and $\overline{Q}$ both will attain logic 1 which is inconsistent with our assumption of complementary outputs. Now, when the clock pulse has passed away (CK = 0), the outputs of

G_3 and G_4 will rise from 0 to 1. Depending upon the propagation delays of the gates, either the stable state $Q_{n+1} = 1(\overline{Q}_{n+1} = 0)$ or $Q_{n+1} = 0$ ($\overline{Q}_{n+1} = 1$) will result. That means the state of the circuit is undefined, indeterminate or ambiguous and therefore is indicated by a question mark (fourth row of the truth table).

The condition $S_n = R_n = 1$ is forbidden and it must not be allowed to occur.

The logic symbol of clocked S-R FLIP-FLOP is given in Fig. 4.6.

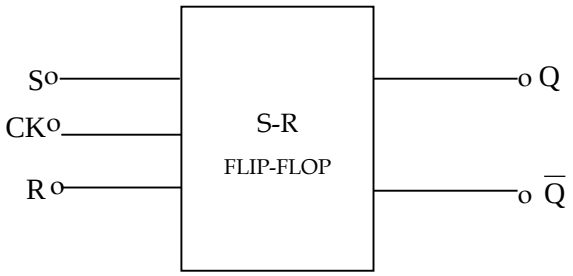


Fig. 4.6 Logic symbol of clocked S–R FLIP-FLOP

4.3.1 Preset and Clear

In the FLIP-FLOP of Fig. 4.5, when the power is switched on, the state of the circuit is uncertain. It may come to set (Q = 1) or reset (Q = 0) state. In many applications it is desired to initially set or reset the FLIP-FLOP, i.e. the initial state of the FLIP-FLOP is to be assigned. This is accomplished by using the direct, or asynchronous inputs, referred to as preset (Pr) and clear (Cr) inputs. These inputs may be applied at any time between clock pulses and are not in synchronism with the clock. An S-R FLIP-FLOP with preset and clear is shown in Fig. 4.7. If Pr = Cr = 1 the circuit operates in accordance with the truth table of S- R FLIP-FLOP given in Table 4.1.

In Pr = 0 and Cr = 1, the output of G_1 (Q) will certainly be 1. Consequently, all the three inputs to G_2 will be 1 which will make $\overline{Q}$ = 0, Hence, making Pr = 0 sets the FLIP-FLOP.

Similarly, if Pr = 1 and Cr = 0, the FLIP-FLOP is reset. Once the state of the FLIP-FLOP is established asynchronously, the asynchronous inputs Pr and Cr must be connected to logic 1 before the next clock is applied.

The condition Pr = Cr= 0 must not be used, since this leads to an uncertain state

In the logic symbol of Fig. 4.7b, bubbles are used for Pr and Cr inputs, which means these are active-law, i.e. the intended function is performed when the signal applied to Pr or Cr is LOW. The operation of Fig. 4.7 is summarized in Table 4.2.

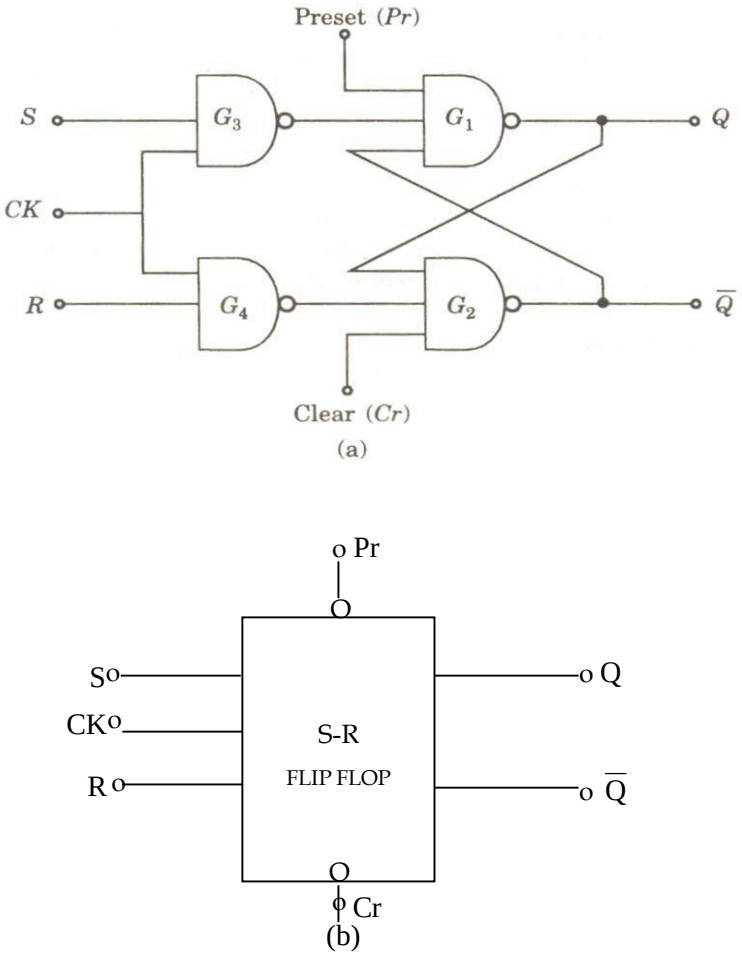


Fig. 4.7 (a) An S-R FLIP-FLOP with preset and clear,

(b) its logic symbol.

Table 4.2 Summary of operation of S-R FLIP-FLOP

Inputs			Output	Operation performed
CK	Cr	Pr	Q	
1	1	1	Q_{n+1} (Table 4.1)	Normal FLIP-FLOP
0	0	1	0	Clear
0	1	0	1	Preset

The circuit can be designed such that the asynchronous inputs override the clock, i.e. the circuit can be set or reset even in the presence of the clock pulse.

4.4. J-K FLIP-FLOP

The uncertainty in the state of an S-R FLIP-FLOP when $S_n = R_n = 1$ (fourth row of the truth table) can be eliminated by converting it into a J-K FLIP-FLOP. The data inputs are J and K which are ANDed with $\overline{Q}$ and Q, respectively, to obtain S and R inputs, i.e.

$$S = J.\ \overline{Q} \qquad (4.1a)$$

$$R = K.\ Q \qquad (4.1b)$$

A J-K FLIP-FLOP thus obtained is shown in Fig. 4.8. Its truth table is given in Table 4.3a which is reduced to Table 4.3b for convenience. Table 4.3a has been prepared for all the possible combinations of Land K inputs, and for each combination both the states of the output have been considered.

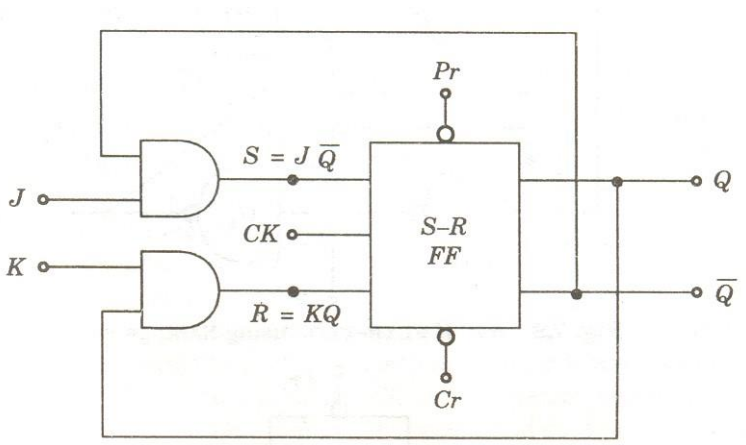


Fig. 4.8 An S-R FLIP-FLOP converted into J-K FLIP-FLOP.

It is not necessary to use the AND gates of Fig. 4.8, since the same function can be performed by adding an extra input terminal to each NAND gate G_3 and G_4 of Fig. 4.7. With this modification incorporated in Fig. 747, we obtain the J-K FLIP-FLOP using NAND gates as shown in Fig. 4.9. The logic symbol of J-K FLIP-FLOP is given in Fig. 4.10.

Table 4.3a Truth table for Fig. 4.8

Data inputs		Outputs		Inputs to S-R FF		Output
J_n	K_n	Q_n	$\overline{Q}_n$	S_n	R_n	Q_{n+1}
0	0	0	1	0	0	0] = Q_n
0	0	1	0	0	0	1
1	0	0	1	1	0	1] = 1
1	0	1	0	0	0	1
0	1	0	1	0	0	0] = 0
0	1	1	0	0	1	0
1	1	0	1	1	0	0] = $\overline{Q}_n$
1	1	1	0	0	1	1

Table 4.3b Truth table of J–K FLIP-FLOP

Inputs		Output
J_n	K_n	Q_{n+1}
0	0	Q_n
1	0	1
0	1	0
1	1	$\overline{Q}_n$

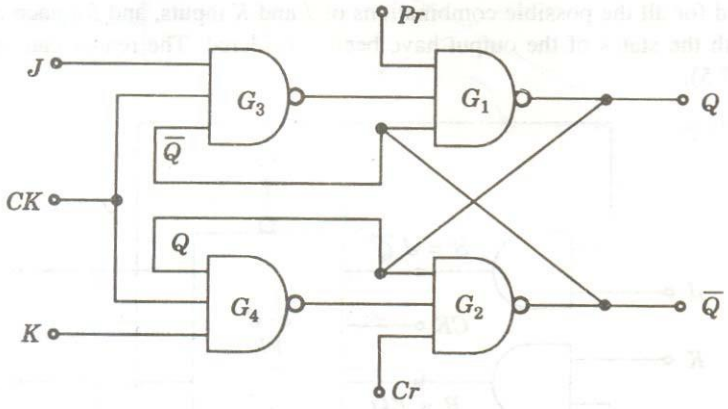


Fig. 4.9 A J–K FLIP-FLOP using NAND gates.

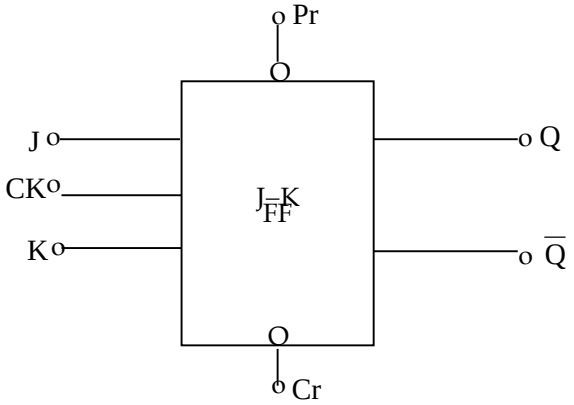


Fig. 4.10 Logic symbols of J–K FLIP-FLOP

4.4.1 The Race-Around Condition

The difficulty of both inputs 1 (S = R = 1) being not allowed in an S-R FLIP-FLOP is eliminated in a J-K FLIP-FLOP by using the feedback connection from outputs to the inputs of the gates G_3 and G_4 (Fig. 4.9). Table 4.3 assumes that the inputs do not change during the clock pulse (CK = 1), which is not true because of the feedback connections. Consider, for example, that the clock input, after a time interval Δt equal to the propagation delay through two NAND gates in series, the output will change to Q = 1 (see fourth row of Table 4.3b). Now we have J = K = 1 and Q =1 and after another interval of Δt the output will change back to Q = 0. Hence, we conclude that for the duration t_p of the clock p0ulse, the output will oscillate back and forth between 0 and 1. At the end of the clock pulse, the value of Q is uncertain. This situation is referred to as the race-round condition.

The race-around condition can be avoided if $t_p < \Delta t < T$. However, it may be difficult to satisfy this inequality because of very small propagation delays in ICs. A more practical method for overcoming this difficulty is the use of the master-slave (M-S) configuration discussed below.

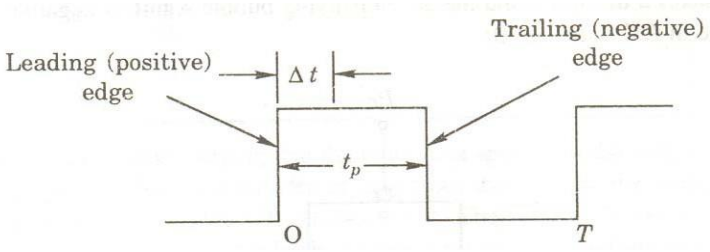


Fig. 4.11 A clock pulse

4.4.2 The Master-Slave J-K FLIP-FLOP

A master-slave J-K FLIP-FLOP is a cascade of two S-R FLIP-FLOPs with feedback from the outputs of the second to the inputs of the first as illustrated in Fig. 4.12. Positive clock pulses are applied to the first FLIP-FLOP and the clock pulses are inverted before these are applied to the second FLIP-FLOP.

When CK = 1, the first FLIP-FLOP is enabled and the outputs Q_M and $\overline{Q}_M$ respond to the inputs J and K according to Table 4.3. At this time, the second FLIP-FLOP is inhibited because its clock is LOW ($\overline{CK}$ = 0). When CK goes LOW ($\overline{CK}$ = 1), the first FLIP-FLOP is inhibited and the second FLIP-FLOP is enabled, because now its clock is HIGH ($\overline{CK}$ = 1). Therefore, the outputs Q and $\overline{Q}$ follow the outputs Q_M and $\overline{Q}_M$, respectively (second and third rows of Table 4.3b). Since the second FLIP-FLOP simply follows the first one, it is referred to as the slave and the first one as the master. Hence, this configuration is referred to as master-slave (M–S) FLIP-FLOP.

In this circuit, the inputs to the gates G_{3M} and G_{4M} do not change during the clock pulse, therefore the race-around condition does not exist. The state of the master-slave FLIP-FLOP changes at the negative transition (trailing edge) of the clock pulse. The logic symbol of a M-S FLIP-FLOP is given in Fig. 4.13.

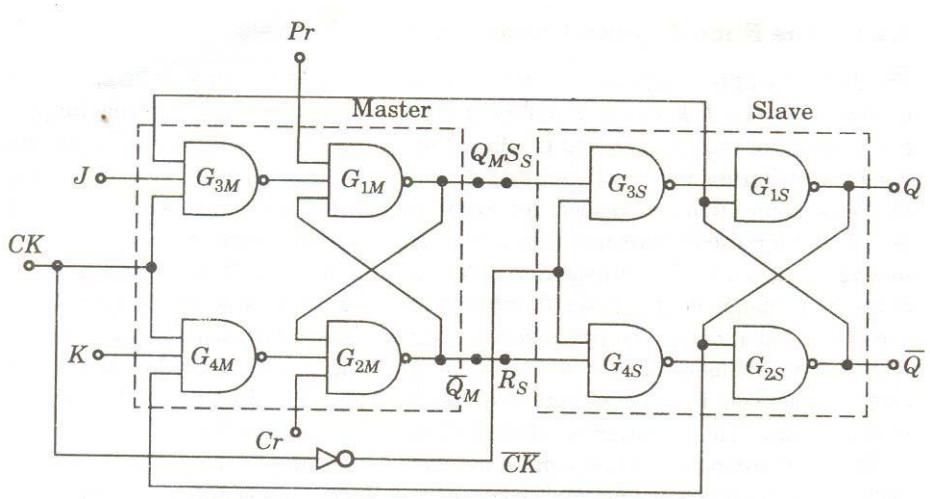


Fig. 4.12 A master-salve J-K FLIP-FLOP.

At the clock input terminal, the symbol > is used to illustrate that the output changes when the clock makes a transition and the accompanying bubble signifies negative transition (change in DK from 1 to 0).

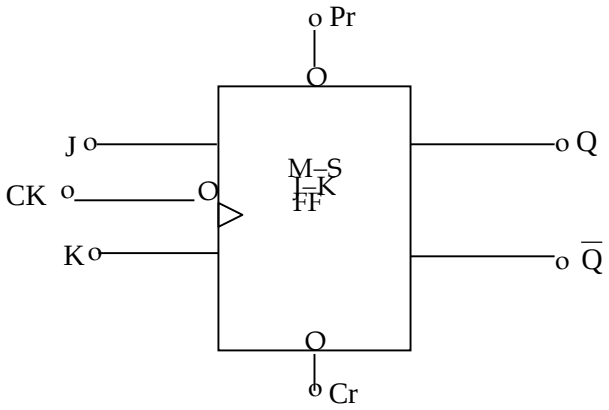


Fig. 4.13 A master–slave J–K FLIP-FLOP logic symbol

4.5 D-TYPE FLIP-FLOP

If we use only the middle two rows of the truth table of the S-R (Table 4.1) or J-K (Table 4.3b) FLIP-FLOP, we obtain a D-type FLIP-FLOP as shown in Fig. 4.14. It has only one input referred to as D-input or data input. Its truth table is given in Table 4.4 from which it is clear that the output Q_{n+1} at the end of the clock pulse equals the input D_n before the clock pulse.

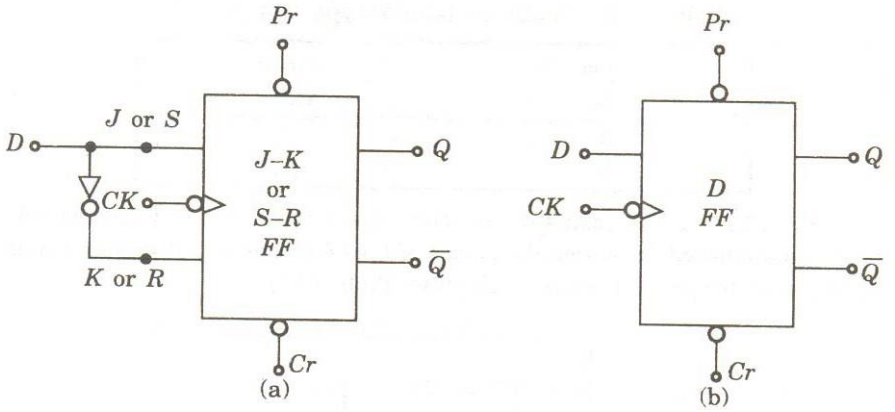


Fig. 4.14 (a) A J–K or S–R FLIP-FLOP converted into a D-type FLIP-FLOP (b) its logic symbol

Table 4.4 Truth table of a D-type FLIP-FLOP

Input D_n	Output Q_{n+1}
0	0
1	1

This is equivalent to saying that the input data appears at the output at the end of the clock pulse. Thus, the transfer of data from the input to the output is delayed and hence the name delay (D) FLIP-FLOP. The D-type FLIP-FLOP is either used as a delay device or as a latch to store 1-bit of binary information.

4.6 T-TYPE FLIP-FLOP

In a J-K FLIP-FLOP, if J = K, the resulting FLIP-FLOP is referred to as a T-type FLIP-FLOP and is shown in Fig. 4.15. It has only one input, referred to as T-input. Its truth table is given in Table 4.5 from which it is clear that if T = 1 it acts as a toggle switch. For every clock pulse, the output Q changes.

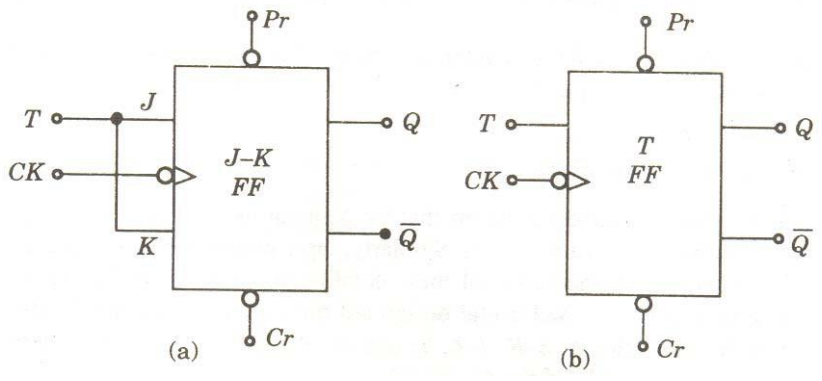


Fig. 4.15 (a) A–K FLIP-FLOP converted into a T-type FLIP-FLOP, (b) its logic symbol

Table 4.5 Truth table of T-type FLIP-FLOP

Input T_n	Output Q_{n+1}
0	Q_n
1	$\overline{Q}_n$

As S-R FLIP-FLOP cannot be converted into a T-type FLIP-FLOP since S = R = 1 is not allowed. However, the circuit of Fig. 4.16 acts as a toggle switch, i.e. the output Q changes with every clock pulse.

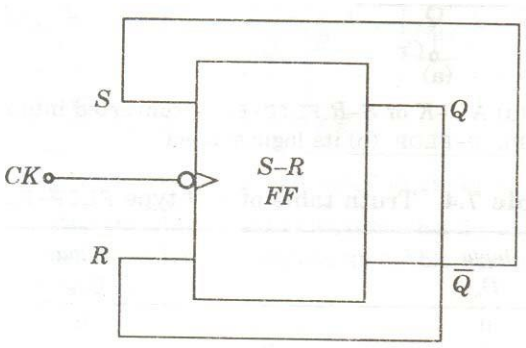


Fig. 4.16 An S-R FLIP-FLOP as a toggle switch

4.7 EXCITATION TABLE OF FLIP-FLOP

The truth table of a FLIP-FLOP is also referred to as the *characteristic table* and specifies the operational characteristic of the FLIP-FLOP.

In the design of sequential circuits, we usually come across situations in which the present state and the next state of the circuit are specified, and we have to find the input conditions that must prevail to cause the desired transition of the state. By the present state and the next state we mean the state of the circuit prior to and after the clock pulse respectively. For example, the output of an S-R FLIP-FLOP before the clock pulse is $Q_n = 0$ and it is desired that the output dies not change when the clock pulse is applied. What input conditions (S_n and R_n values) must exist to achieve this?

From the truth table (or the characteristic Table) of an S-R FLIP-FLOP (Table 7.1) we obtain the following conditions:

1. $S_n = R_n = 0$ (first row)

2. $S_n = 0$, $R_n = 1$ (third-row)

We conclude from the above conditions that the S_n input must be 0, whereas the R_n input may be either 0 or 1 (don't-care). Similarly, input conditions can be found for all possible situations. A tabulation of these conditions is known as the *excitation table*. It is a very important and useful design aid for sequential circuits. Table 7.6 gives the excitation tables of S-R, J-K, T, and D FLIP-FLOPs. This is derived from the characteristic table of the FLIP- FLOP.

Table 4.6 Excitation table of FLIP-FLOPs

Present State	Next State	S-R S_n	FF R_n	J-K J_n	FF K_n	T-FF T_n	D-FF D_n
0	0	O	×	0	×	0	0
0	1	1	0	1	×	1	1
1	0	0	1	×	1	1	0
1	1	×	0	×	0	0	1

4.8 CLOCKED FLIP-FLOP DESIGN

In earlier sections, we defined or specified the operation of different FLIP-FLOPs assuming a circuit without regard to where the circuit came from or how it was designed. In this section, the design of a FLIP-FLOP is given. The design philosophy illustrated is, in fact, a general approach for the design of sequential circuits and systems.

Consider the general model of the FLIP-FLOP shown in fig. 4.17. Basically, aclocked FLIP-FLOP is a sequential circuit which stores the bits 0 and 1. This operation is accomplished by using a binary cell coupled with some combinational set/reset decoding

logic to allow some input control over the set and reset operations of the cell. The steps for the design of FLIP-FLOP are given below.

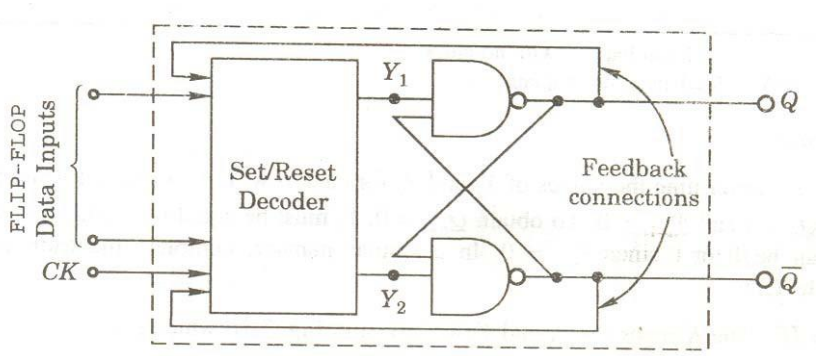


Fig. 4.17 The general model of the FLIP-FLOP

Step I: Examine each row of the given characteristic table, specifying the desired inputs and outputs, and answer the following questions and make a truth table with Y_1 and Y_2 as output variables.

1. Does the cell need to be set ($Q_{n+1} = 1$) for this condition?
2. Does the cell need to be reset ($Q_{n+1} = 0$) for this condition?
3. Does the cell need to be left as it is?

Step II: Prepare the K-map for Y_1 and Y_2 output variables, minimize it and determine the logic for Y_1 and Y_2, respectively. Draw the complete circuit using gates.

The above design steps are illustrated in Example 4.1.

Example 4.1 Using the technique described above, design a clocked S-R FLIP-FLOP whose characteristic table is given in Table 4.7.

Table 4.7 Truth table

Characteristic table					Truth table for decoder	
CK	S	R	Q_n	Q_{n+1}	Y_1	Y_2
0	0	0	0	0	1	×
0	0	0	1	1	×	1
0	0	1	0	0	1	×
0	0	1	1	1	×	1
0	1	0	0	0	1	×
0	1	0	1	1	×	1
0	1	1	0	0	1	× *
0	1	1	1	1	×	1
1	0	0	0	0	1	×
1	0	0	1	1	×	1
1	0	1	0	0	1	×
1	0	1	1	0	1	0
1	1	0	0	1	0	1
1	1	0	1	1	×	1
1	1	1	0	×	×	× *
1	1	1	1	×	×	×

*S = R = 1 can happen with no clock.

**S = R = 1 must not happen.

Solution

Step I Determine the values of Y_1 and Y_2 for each row. For example, for the first row $Q_n = 0$ and $Q_{n+1} = 0$. To obtain $Q_{n+1} = 0$,. Y_1 must be equal to 1, since $\overline{Q}_n = 1$, Y_2 can be 0 or 1 since $Q_n = 0$. In a similar manner, complete the truth table (Table 4.7).

Step II the K-maps for Y_1 and Y_2 are given in Fig. 4. 18 which give

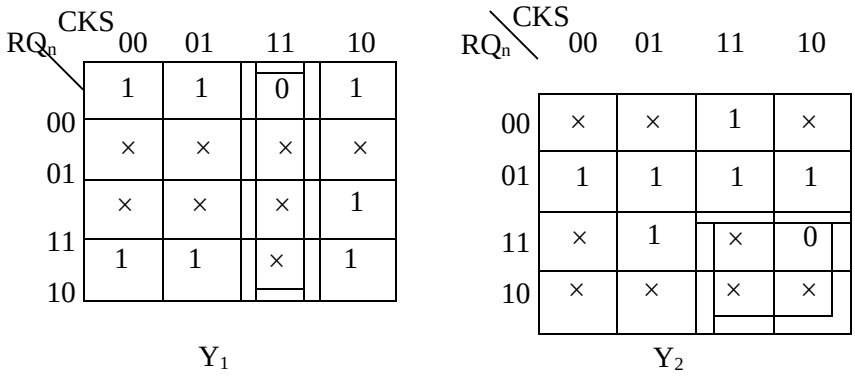


Fig. 4.18 K-maps for Ex. 4.1

$$Y_1 = \overline{CK} + \overline{S} = \overline{CK \cdot S}$$

$$Y_2 = \overline{R} + \overline{CK} = \overline{CK \cdot R}$$

Thus, we see that the circuit resulting from this design is the same as that shown in Fig. 4.5.

4.8.1 Conversion From One Type of FLIP-FLOP to Another Type

In earlier sections, we have discussed conversion from S-R to J-K, S-R (or J-K) to D-type, and J-K to T-type FLIP-FLOPs. Now, we shall effect the conversion from one type of FLIP-FLOP to another type by using a formal technique which is similar to the one used above and will be useful in the design of clocked sequential circuits.

Consider the general model for conversion from one type of FLIP-FLOP to another type (Fig.4.19). In this, we are required to design the combinational logic decoder (conversion logic) for converting new input definitions into input codes which will cause the given FLIP-FLOP to perform as desired.

To design the conversion logic we need to combine the excitation tables for both FLIP-

FLOPs and made a truth table with data input (s) and Q as the inputs and the input (s) of

the given FLIP-FLOP as the output (s). The conventional method of combinational logic design then follows as usual. The conversion is illustrated in Example 4.2.

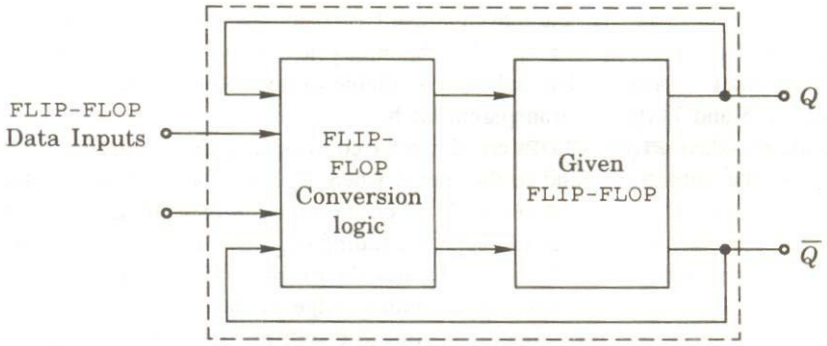


Fig. 4.19 The general model used to convert one type of FLIP-FLOP

Example 4.2 Convert as S-R FLIP-FLOP to a J-K FLIP-FLOP.

Solution : The excitation tables of S–R and J–K FLIP-FLOPs are given in Table 4.6 from which we make the truth table given in Table 4.8

$$S = J \cdot \overline{Q} \qquad \text{and} \qquad R = K \cdot Q$$

Thus, we see that the circuit resulting from this design is the same as that shown in Fig. 4.8.

Table 4.8 Truth table of conversion logic

Row	FF data inputs		Output	S–R FF inputs	
	J	K	Q	S	R
1	0	0	0	0	×
2	0	1	0	0	×
3	1	0	0	1	0
4	1	1	0	1	0
5	0	1	1	0	1

6	1	1	1	0	1
7	0	0	1	×	0
8	1	0	1	×	0

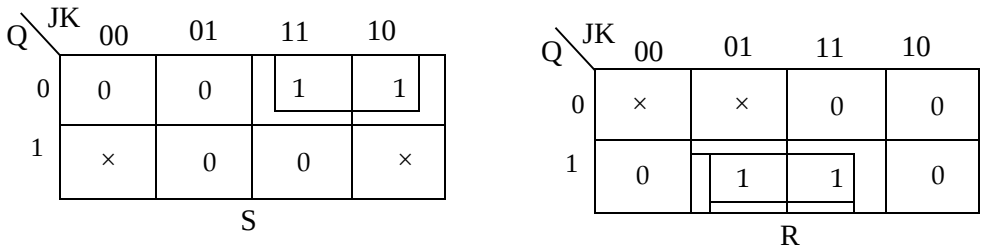


Fig. 4.20 K-maps for Ex. 4.2

4.9. LEVEL CLOCKING AND EDGE TRIGGERING

In a clocked flip flop, the output can change state when CLK is high. When CLK is low, the output remains in the same state. Thus, the output can change state during the entire half cycle when CLK is high. This may be a disadvantage in many situations. It is necessary that the output should change state only at one instant in the positive half cycle of the clock. This is known as edge triggering and the resulting flip flop is known as edge triggered flip flop.

Edge triggering is possible using an RC circuit. The time constant RC is made much smaller than the width of the clock pulse. Therefore, the capacitor can charge fully when CLK is high. The exponential charging produces a narrow positive voltage spike across the resistor. The input gates are activated at the instant of this positive spike.

Edge triggering using RC circuit is not very convenient for a computer because it is difficult to fabricate a capacitor on a chip. Instead, additional NAND gates are used to produce positive spike for edge triggering.

4.10. PULSE TRIGGERED FLIP FLOP

Another type of triggering is pulse triggering. The term pulse triggering means that the data are entered into the flip flop on the leading edge of the clock pulse, but the output does not reflect the input state until the trailing edge of the pulse. Thus the input must be set up prior to the leading edge of the clock pulse, but the output is postponed until the trailing edge of the pulse. A major restriction in pulse triggered flip flop is that the data input should not change when clock pulse in High because the flip flop is sensitive to any change of input level during the period when clock pulse in High.

All the three edge triggered flip flop, viz., SR, D and JK can be pulse triggered also. JK flip flop is the most commonly available one in IC form.

4.11. PARAMETERS OF FLIP FLOPS

When a flip flop is to be selected for a specific application, it is necessary to consider its parameters. These parameters are given in data sheets supplied by manufacturers and are common to all the flip flops.

(a) Propagation Delay Time

It is the time interval between data input signal and the resulting change in output. It is further categorized into different types

Propagation delay t_{PLH} is measured from the triggering edge of clock pulse to Low to High transition of output and is shown in Fig. 4.21 (a).

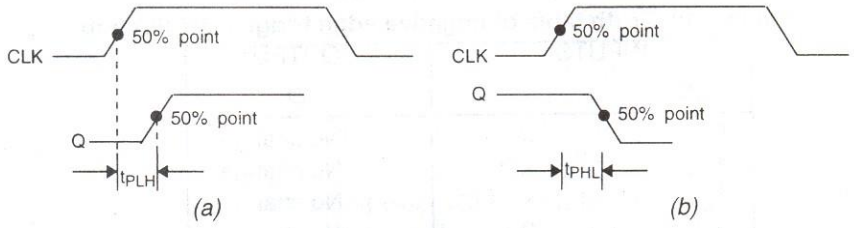


Fig. 4.21 Propagation delay times of flip flop (clock to output) (a) t_{PLH} (b) t_{PHL}

Propagation delay t_{PLH} is measured from the triggering edge of clock pulse to high to low transition of output and is shown in Fig. 4.21 (b).

(b) Set-up Time (t_s)

It is the minimum time required for the control levels to be maintained constantly on the inputs (D or SR or JK) prior to the triggering edge of the clock pulse so that for the levels be reliably clocked into the flip flop. It is shown in Fig. 4.22 for a D flip flop. It is denoted by t_a.

(c) **Hold Time (t_h)**

It is the minimum time required for the control levels to remain on the inputs after the triggering edge of the clock pulse so that the levels are reliably clocked into the flip flop. It is denoted by t_h and is shown in Fig. 4.23 for a D flip flop.

(d) **Maximum Clock Frequency (f_{max})**

It is the highest rate at which the flip flop can be reliably triggered. It clock frequency is more than the maximum specified value, the flip flop may not be able to reliably.

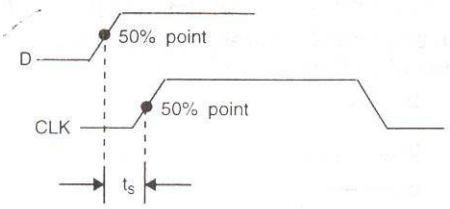


Fig. 4.22 Set up time of flip flop

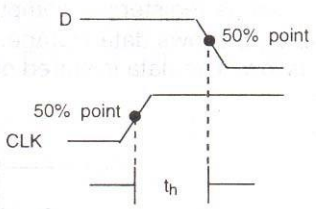


Fig. 4.23 Hold time of flip flop

(e) Pulse Width (t_w)

Minimum pulse widths are specified for clock, preset and clear inputs. These are denoted by t_w.

(f) **Power Dissipation**

It is the total power dissipation of the flip flop. It is the product of voltage and current. If voltage is + 5 V dc and current is 25 mA, the power dissipation is $5 \times 25 = 125$mW.

This parameter assumes importance because a system may have a large number of flip flops (say 10). The dc source must have a current rating to supply rated current to all the flip flops in the system.

(g) Other Parameters

Flip flop is also a digital gate. The parameters of gates viz., fan out, noise margin, input voltage, output voltage are applicable for flip flops also.

4.12. Contact Bounce Elimination

When a mechanical switch is closed the holes of the switch vibrate or bounce a number of times before finally making a solid contact. These bounces may produce voltage spikes. This is shown in Fig. 4.24 (a). Such voltage spikes are not desirable in a digitalsystem.

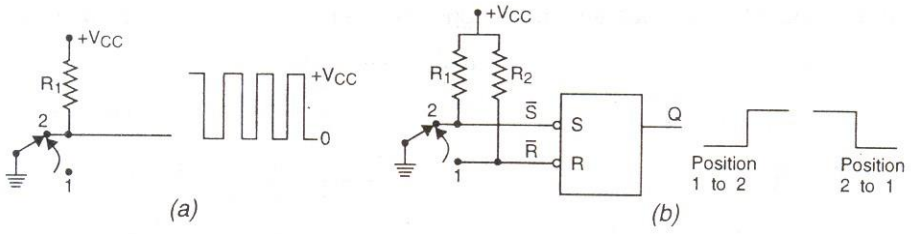


Fig. 4.24 Use of SR latch to eliminate contact bounce and voltage spikes

As SR latch can be used to eliminate the contact bounce as shown in Fig. 9.25 (b). The normal position of switch is 1. Thus $\overline{R}$ input is Low and latch is in RESET position. When switch is thrown to position 2, R goes High due to presence of pull up resistor to Vcc and $\overline{S}$ goes to Low on first contact. This SETS the latch. Any further voltage spikes on the $\overline{S}$ input (due to contact bounce) do not affect the latch and it remains SET. The Q output makes a clean transition from LOW to HIGH and voltage spikes due to contact bounce are eliminated. A similar clean transition occurs when the switch is brought back to position 1.

4.13. Parallel Data Storage in Registers

Data storage is an important aspect of digital systems. Many data hits are taken on parallel lines and stored simultaneously in a group of flip flops. Fig. 4.25 (a) shows this

parallel data input. The data inputs are connected to D_0, D_1, D_2. The clock inputs of all the three flip flops are tied together and connected to common clock input. This ensures that all the three flip flops are triggered together. As shown in Fig. 4.25 (a) positive edge triggering isused in this case. Therefore, the data is stored in flip flops on the positive edge (i.e., Low to High edge) of the clock. Moreover, all CLR* terminals are also connected together so that all the three flip flops are reset together. Groups of flip flops used to store data are known as registers in computer terminology. Fig. 4.25 (b) shows data inputs D_0, D_1, D_2 as 101, Fig. 4.25 (c) shows data storage. The flip flops have low CLR. Therefore, the flip flops are cleared when CLR is low. The data is stored on the positive edge of CLK as shown.

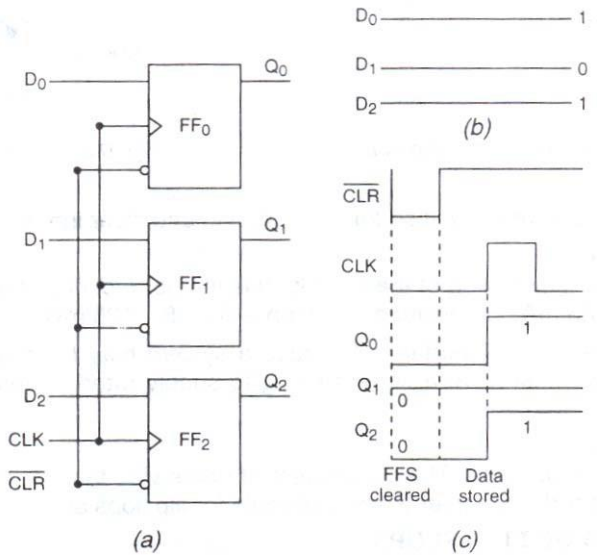


Fig. 4.25. Data storage by flip flops (a) logic circuit (b) data (c) waveforms of $\overline{CLR}$, CLK, Q_0, Q_1 and Q_2

4.14. Transfer of Data

Many times it is necessary to transfer data from one bit to another. This can be done by using flip flops as illustrated in Fig. 4.26 (a). The data is to be initially stored in FF_1 and is then required to be transferred to FF_2. Both the flip flops are pulse triggered. Their clock inputs are connected together.

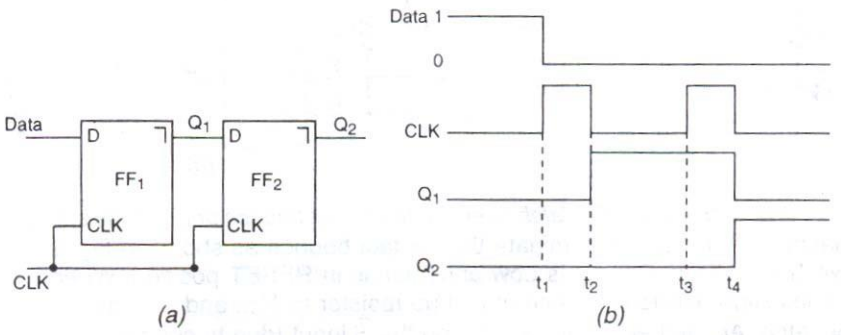


Fig. 4.26. Data transfer using flip flops (a) connection of flip flops (b) instants of data transfer.

The process of data transfer is shown in Fig. 4.26 (b). Since the flip flops are pulse triggered the data appears on the output on trailing edge of the clock pulse. At the trailing edge of first clock pulse data output appears at Q_1. At the trailing edge of second clock pulse data appears at output Q_2 of second flip flop. As in the pulse triggered operation, the data is clocked into the master of flip flop at the leading clock edge but appears at the output at the trailing edge of the clock pulse. Thus at t_1 data (i.e., 1) is clocked into master of FF_1. At t_2 this data appears at Q_1. At t_3 this data is clocked into master of FF_2 and at t_4 this data appears at Q_2.

4.15. Counting

Digital counting is an important application of flip flops. This is illustrated in Fig. 4.27. These two flip flops are for counting two bit binary numbers (for bigger binary numbers, the number of flip flops will be more). As shown the flip flops are negative edge triggered*. At trailing edge of first clock pulse Q_1 becomes 1. At trailing edge of second clockpulse Q_2 becomes 1 and Q_1 becomes 0 (indicating binary number 10 or decimal number 2). Attrailing edge of third clock pulse, $Q_1 = 1$ and $Q_2 = 1$ indicating the binary number as 11 (equalto decimal number 3). This binary sequence is repeated every 4 clock pulses as shown in Fig.
4.27 (b). Thus the flip flops count 00, 01, 10 and 11 (i.e., from decimal 0 to 3) and then return to 00 to start the next sequence.

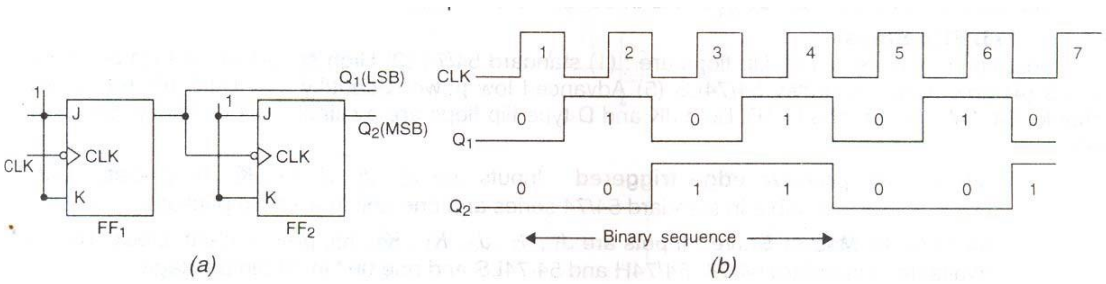


Fig. 4.27. Use of flip flops as counters (a) connections of flip flops (b) counting process

4.15.1. Frequency Division

Flip Flops can be used to reduce the frequency of clock pulses. When a pulsewaveform is applied to clock input of JK flip flop connected to toggle, the frequency of output at Q is half of that at clock input. This is shown in Fig. 4.28. Thus one flip flop is a divide by 2 device. The flip flop changes state at every positive clock edge to give at the output a square wave with half the frequency of that at dock input.

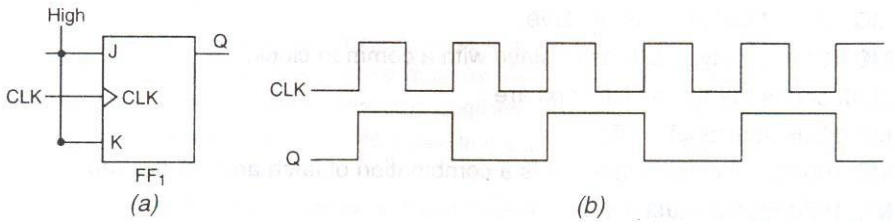


Fig. 4.28. Frequency division by 2 using JK flip flop

More flip flops can be added to reduce the frequency further. In Fig. 4.29, two flip flops 1 and 2 are connected in tandem. The frequency of output at Q_1 is half of that at clock input. The frequency of output at Q_2 is half of that at Q_1 or one fourth of that at clock input. If the number of flip flops is n, the frequency can be reduced by factor 2^n.

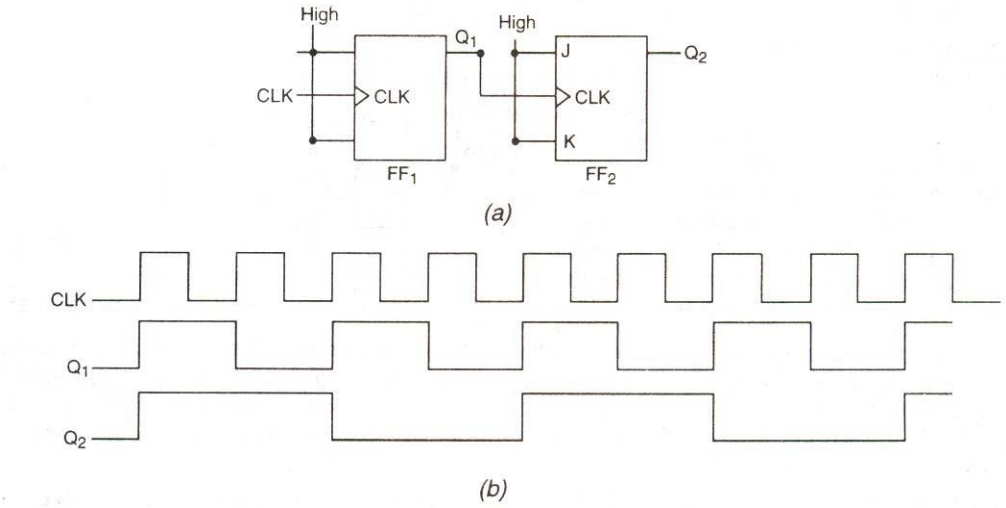


Fig. 4.29. Frequency division by 4 using two flip flops

If n = 5, the frequency is reduced to 1/32 of the original frequency.

SUMMARY

1. Latches and flip flops are bistable elements with two stable states. The main difference between a latch and flip flop is in the method used for changing state.
2. Latches are bistable elements whose state depends on asynchronous inputs. On the other hand, edge triggered flip flops are bistable elements with synchronous inputs whose state depends on inputs only at the triggering transition of a clock pulse. Edge triggering can be positive edge triggering or negative edge triggering.
3. RS latch can be built with NOR gates or NAND gates. In this latch a High on both R and S inputs leads to race condition.
4. A clocked RS latch has a clock input in addition to R and S inputs. A state change can occur only when clock signal becomes high.
5. In a D latch a common D input drives the R and S inputs, S input directly and R input through inverter. A clocked D latch uses a clock signal to enable and disable the latch.

6. Flip flops often have present (PR) and clear (CLR) inputs also. A clear signal is the same as reset signal. A preset is equivalent to set the flip flop before the computer run.
7. JK master slave flip flop has two clocked SR flip flops, one known as master and the second known as slave.
8. Edge triggered JK flip flop is used in digital counters.
9. A T flip flop is obtained by connecting the J and K inputs of JK edge triggered flip flop.
10. Pulse triggering means that data is entered into the flip flop at the leading edge of the clock pulse but the output occurs only on the trailing edge of clock pulse.
11. Data lock out flip flop has dynamic clock input but the data inputs are disabled after the leading edge transition.
12. Various parameters of flip flops are : propagation delay, set up time, hold time, maximum clock frequency, pulse width and power dissipation.
13. Flip flops are used for contact bounce elimination, data storage, data transfer, counting and frequency division.
14. A Schmitt trigger is a very suitable device for waveshaping. It can convert a sine wave or a distorted signal into a square wave.

PROBLEMS

4.1 What are flip flops? Why they are called electronic switch?
4.2 What are different parameters of flip flops?
4.3 What is J-K flip flop and how it is converted from S-R flip flop?
4.4 Explain the concept of d and t flip flop.
4.5 What is the significance of excitation table?
4.6 Enumerate the advantages of flip flops.
4.7 How data is transferred from one place to another with the help of digital techniques?
4.8 Differentiate between a latch and a flip flop.
4.9 Differentiate between edge triggered and level triggered flip flops with help of examples.
4.10 How can we convert a JK flip flop to a D type flip flop?

CHAPTER-5

REGISTERS

5.1 INTRODUCTION

A register is device capable of storing a bit. The data can be serial or parallel. A register can convert a data from serial to parallel and vice versa. Shifting the digits to left and right is an important aspect of arithmetic operations. In this chapter we discuss these concepts.

5.2 BUFFER REGISTER

A register is used for storing and shifting data entered into it from an external source. We know that a flip flop is the basic storage element in digital system. Fig. 5.1 explains the concept of storing a 1 or 0 in a flip flop. A 1 is applied to the input of flip flop and a clock pulse is applied. 1 is stored by setting the flip flop. Even when 1 is removed from the input, the flip flop remains in the set state storing 1. If a 0 is applied at the input, 0 is stored in the flip flop on the application of clock pulse. For storing more bits we need more flip flops. Eachstate of a flip flop has one bit of storage capacity. Thus the number of stages is equal to the storage capacity.

A buffer register is the simplest type of register. It can only store a digital word.

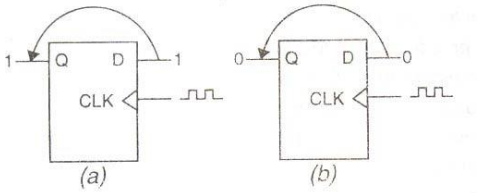


Fig. 5.1. Flip flop as storage element
(a) Q = 1 on positive edge of block or remains 1 if already in that state
(b) Q = 0 on positive edge of clock or remain 0 if already in that state

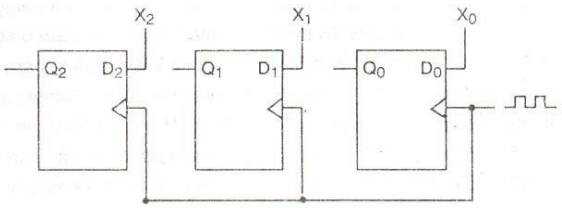


Fig. 5.2. 3 bit buffer register

Fig. 5.2 shows a 3 bit buffer register. The X bits set the flip flops for loading. When the first positive clock edge arrives the output becomes

$$Q_2\ Q_1\ Q_0 = X_2\ X_1\ X_0$$

The buffer register is too primitive to be of any use. A control over the X bits is needed.

5.3 CONTROLED BUFFER REGISTER

Fig. 5.3 shows a controlled buffer register. It has an active High CLR. When CLR goes High, all flip flops set and $Q_2\ Q_1\ Q_0 = 000$. When CLR returns Low, the register is ready for working

Load is the control input. When Load is Low, the X bits cannot reach the flip flops. When Load is Low, Load is High. This forces each flip flop output to feedback its data input. When positive clock edge arrives, data bits are circulated or retained, i.e., when Load is Low, the register contents remain the same.

When Load is High, X bits can reach the flip flops. When positive clock edge arrives, the X bits are stored so that $Q_2\ Q_1\ Q_0 = X_2\ X_1\ X_0$. If Load now returns to Low, this word is stored indefinitely and even if X bits change, the stored contents remain the same.

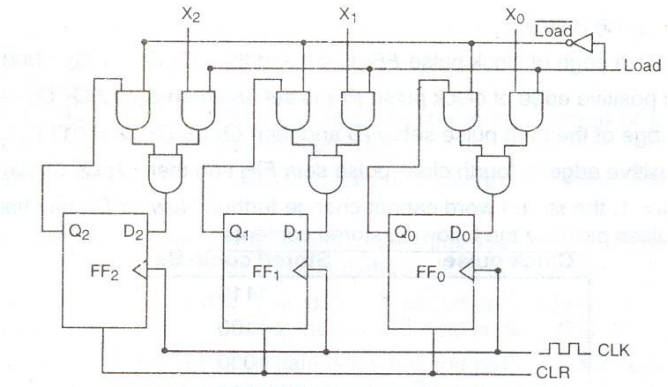


Fig. 5.3 bit controlled buffer register

5.4 BASIC SHIFT OPERATIONS

A simple example of shift operation is that in a calculator. Suppose we have to enter 356 in the calculator. First we press and release 3. The digit 3 appears in the display. Next we press and release 5. The digit 3 is shifted one place to the left and 5 appears on the extreme right. As we press and release 6, the digit 3 and 5 shift to the left and 6 appears in the extreme right position. This simple example illustrates two characteristics of a shift register. (1). It is a temporary memory and holds the number displayed. (2). When we press a new digit on the keyboard, the earlier number is shifted to the left. Thus, shift register has memory and shifting characteristics.

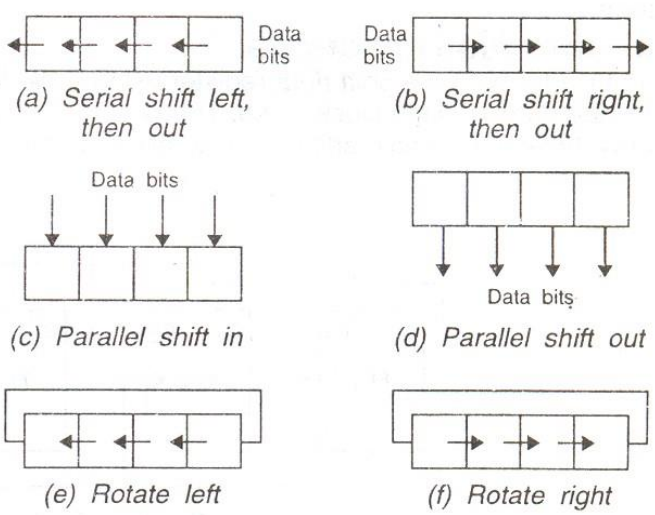


Fig. 5.4 Basic shift operations in digital registers

The basic shift operations are :

(a) serial shift left, then out

(b) serial shift right, then out

(c) parallel shift in

(d) parallel shift out

(e) rotate left

(f) rotate right.

These operations are shown in Fig. 5.4.

5.5 SHIFT LEFT REGISTER

Fig. 5.5 shows a shift left register. It uses D flip flops. The circuit shown has positive edge triggering. It is a 4 bit register using 4 flip flops FF_0, FF_1, FF_2 and FF_3. D_{in} is the input to FF_0. Output of FF_0 is Q_0 and is fed to D_1. Similarly Q_1 is fed to D_2 and Q_2 to D_3. All the flip flops are clocked together by the clock pulse.

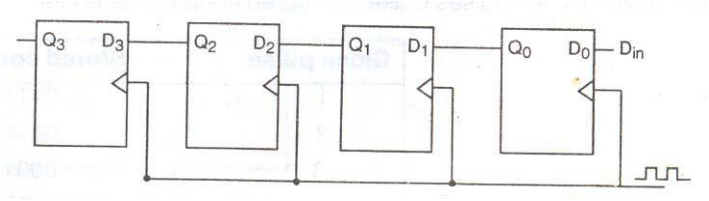


Fig. 5.5 Shift left register

Initially $Q_3\ Q_2\ Q_1 = 0000$

At the first positive edge of clock pulse FF_0 is set and then $Q_3\ Q_2\ Q_1\ Q_0 = 0001$

At the second positive edge of clock pulse FF_1 is set and then $Q_3\ Q_2\ Q_1\ Q_0 = 0011$

The positive edge of the third pulse sets FF_2 and then $Q_3\ Q_2\ Q_1\ Q_0 = 0111$

Finally, the positive edge of fourth clock pulse sets FF_3 and then $Q_3\ Q_2\ Q_1\ Q_0 = 1111$

As long as $D_{in} = 1$, the stored word cannot change further. Now let D_{in} be changed to 0. Then the successive clock pulses produce the following stored contents.

Clock pulse	Stored contents
1	1110
2	1100
3	1000
4	0000

As long as $D_{in} = 0$, the stored contents cannot change further.

5.6 SHIFT RIGHT REGISTER

Fig. 5.6 shows a shift right register using D flip flops. As the name suggests the stored contents are shifted right on each clock pulse. The Q output is connected to the D input of preceding flip flop. At the arrival of each positive edge of clock shift right operation occurs. Let $D_{in} = 1$ and

$$Q_3\ Q_2\ Q_1\ Q_0 = 0$$

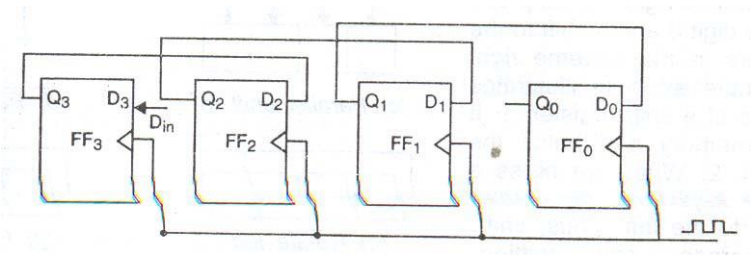


Fig. 5.6 Shift right register

The positive edge of first clock pulse sets up flip flop FF_3 and

$$Q_3\ Q_2\ Q_1\ Q_0 = 1000$$

The positive edge of second clock pulse makes the stored contents as

$$Q_3\ Q_2\ Q_1\ Q_0 = 1100$$

The positive edge of third clock pulse gives

$$Q_3\ Q_2\ Q_1\ Q_0 = 1110$$

and the positive edge of fourth clock pulse gives

$$Q_3\ Q_2\ Q_1\ Q_0 = 1111$$

After this the stored contents remain the same till $D_{in} = 1$. Let D_{in} be changed to 0 now. Then successive clock pulses make the stored contents as under:

Clock pulse	Stored contents
1	0111
2	0011
3	0001
4	0000

As long as $D_{in} = 0$, subsequent clock pulse do not cause any further change in stored contents.

5.7 SHIFT REGISTER OPERATIONS

One method to describe the operation of shift register is the method of loading in and reading from the storage bits. There could be 4 such operations :

(a) **Serial in – Serial out :** The data is loaded into and read from the shift register serially. [Fig. 5.7 (a)]

(b) Serial in – **Parallel out** : The data is loaded into the register serially but read in parallel (i.e., data is available from all bits simultaneously. [Fig. 5.7(b)].

(c) **Parallel in – Serial out :** The data is loaded in parallel, i.e., the bits are entered simultaneously in their respective stages and read serially. [Fig. 5.7 (c)]

(d) **Parallel in – Parallel out** : The data is loaded and read from the register in paralle, i.e., all bits are loaded simultaneously and read simultaneously. [Fig. 5.7(d)].

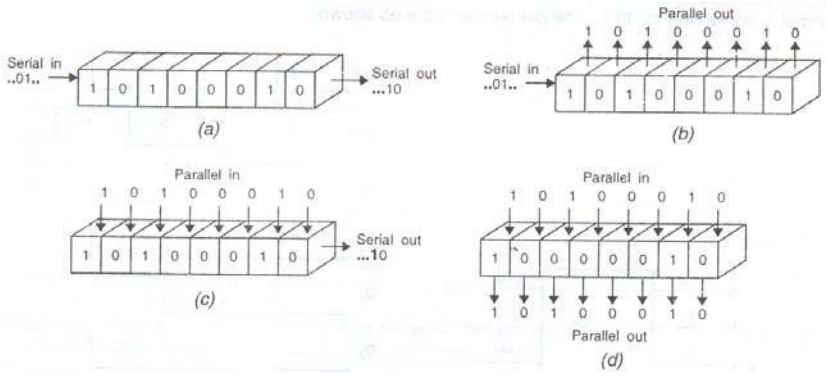


Fig. 5.7 Shift register operations (a) serial in-serial out (SISO) (b) serial in – parallel out (SIPO) (c) parallel in – serial out (PISO) (d) parallel in – parallel out (PIPO)

5.8 SERIAL IN – SERIAL OUT SHIFT REGISTER

Fig. 5.8 shows a 4 bit serial in – serial out shift register consisting of four D flip flops FF_0, FF_1, FF_2 and FF_3. As shown it is a positive edge triggered device. We study the workingof this register for the data 1010 in the following steps :

1. Bit 0 is entered into data input line. $D_0 = 0$, first clock pulse is applied, FF_0 is reset and stores 0.
2. Next bit 1 is entered. $Q_0 = 0$, since Q_0 is connected to D_1, D_1 becomes 0.
3. Second clock pulse is applied, The 1 on the input line is shifted into FF_0 because FF_0 sets. The 0 which was stored in FF_0 is shifted into FF_1.

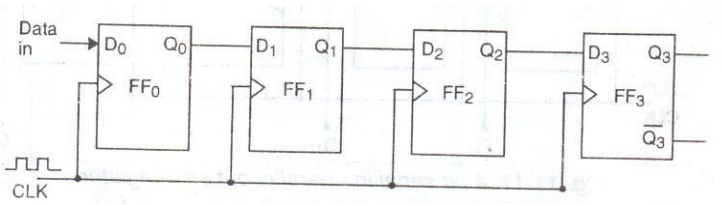


Fig. 5.8 Serial in – serial out shift register

4. Next bit 0 is entered and third clock pulse applied. 0 is entered into FF_0 1 stored in FF_0 is shifted to FF_1 and 0 stored in FF_1 is shifted to FF_2.
5. Last bit 1 is entered and 4^{th} clock pulse applied. 1 is entered into FF_0, 0 stored in FF_0 is shifted to FF_1, 1 stored in FF_1 is shifted to FF_2 and 0 stored in FF_2 is shifted to FF_3. This completes the serial entry of 4 bit data into the register. Now the LSB 0 is on the output Q_3.
6. Clock pulse 5 is applied. LSB 0 is shifted out. The next bit 1 appears on Q_3 output.
7. Clock pulse 6 is applied. The 1 on Q_3 is shifted out and 0 appears on Q_3 output.
8. Clock pulse 7 is applied. 0 on Q_3 is shifted out. Now 1 appears on Q_3 output.
9. Clock pulse 8 is applied. 1 on Q_3 is shifted out.

When the bits are being shifted out (on CLK pulse 5 to 8) more data bits can be entered in.

IC 7491A is a serial in – serial out shift register. It uses eight clocked SR flip flops. Thus it is an 8 bit register. It is designated as SRG8 (shift register 8 bits). A and B are two input lines. When data is entered into A, B must be High and vice versa. The output is Q_H and $\overline{Q_H}$ is complement of Q_H. Its logic symbol is shown in Fig. 5.9. The pin numbers are as shown.

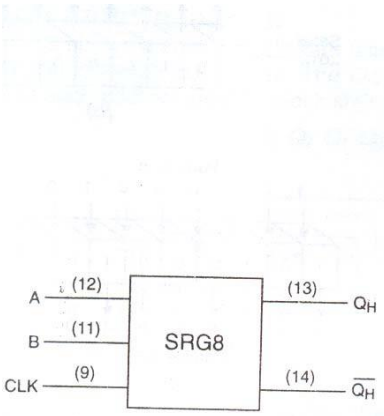


Fig. 5.9 Logic symbol of 8 bit serial in-serial out IC shift register 7491 A

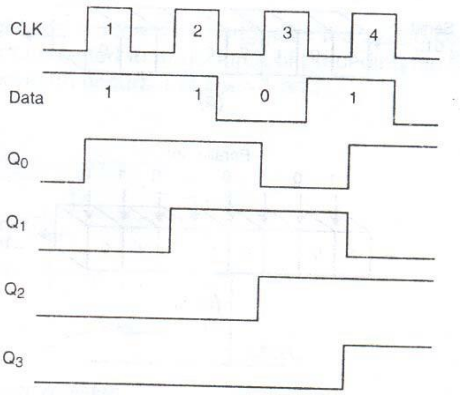


Fig. 5.10.

Example 5.1 Data 1101 is fed into 4 bit serial in – serial out shift register. Show the status of register at various clock pulses.

Solution : Fig. 5.10 shows the diagram depicting the status of register.

5.9 SERIAL IN – PARALLEL OUT SHIFT REGISTER

The data bits are entered serially but they are available at their respective positions simultaneously. Fig. 5.11 shows the circuit of 4 bit serial in – parallel out shift register using D flip flops. Q_0, Q_1, Q_2 and Q_3 are the output terminals and data is available at all of them together.

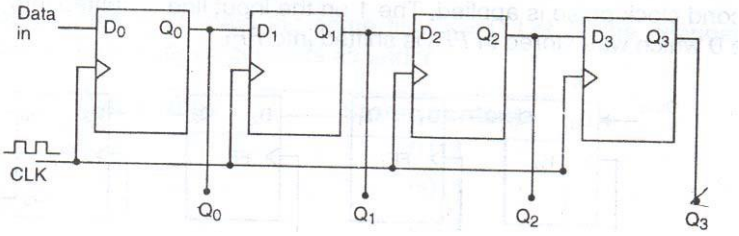


Fig. 5.11 bit serial in – parallel out shift register

IC 74164 is an 8 bit serial in-parallel out shift register. It has two serial inputs A and B, active Low clear $\overline{CLR}$ and parallel outputs Q_A to Q_H. it uses SR flip flops. Fig. 5.12 shows its logical symbol and pins.

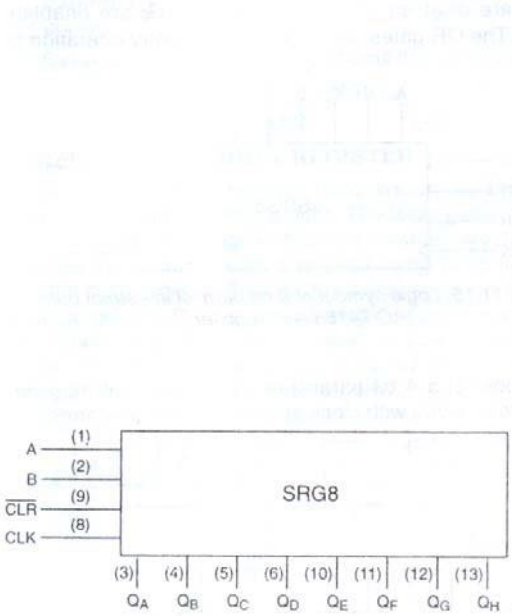


Fig. 5.12. 8 bit serial in –parallel out shift register IC 74164

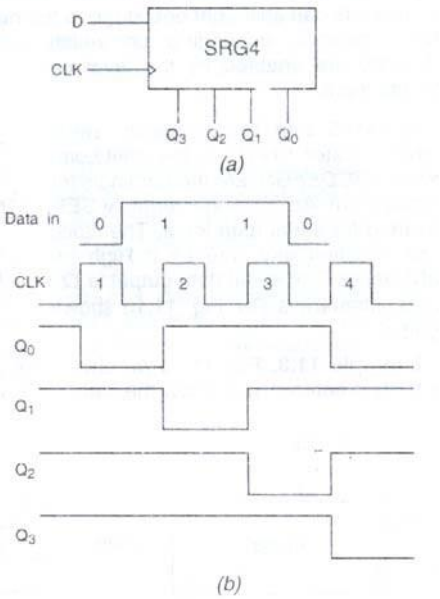


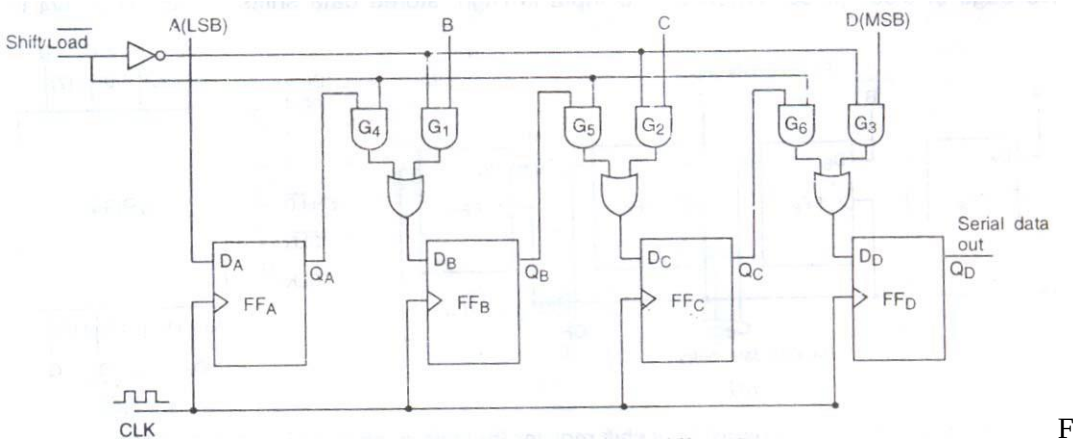
Fig. 5.13.

Example 5.2 Fig. 5.13 (a) shows a 4 bit serial in – parallel out shift register. Show the status of 4 registers for the data 0110.

Solution : Fig. 5.13 (b) shows the diagram depicting the status of register.

5.10 PARALLEL IN –SERIAL OUT SHIFT REGISTER

The data bits are entered simultaneously and taken out serially. Fig. 5.14 shows such a shift register with 4 bits. It uses D flip flops and 4 data input lines A, B, C, D. Moreover, it has a shift/$\overline{\text{Load}}$ input which allows 4 bits of data to be entered into the register simultaneously. When shift/$\overline{\text{Load}}$ is Low,



F

Fig. 5. 14. 4 bit parallel in-serial out shift register

gates G_1, G_2, G_3. are enabled and data can be entered into the D input of the four flip flops. When clock pulse is applied, the flip flop with data bit 1 will SET and flip flops with data bits 0 will reset and thus the 4 bit data will be stored.

When shift/ $\overline{\text{Load}}$ is High, gates G_1, G_2, G_3 are disabled while gates G_4, G_5.G_6 are enabled. Thus datra bits can shift right one stage to the next. The OR gates allow parallel data entry operation or shifting operation depending on which AND gates are enabled by the level on shift/$\overline{\text{Load}}$ input.

IC 74165 is an 8 bit parallel in-serial out shift register SH/$\overline{\text{LD}}$ is the shift/$\overline{\text{Load}}$ terminal. ABCDEFGH are the terminals for 8 bit data input. An additional terminal SER is provided for serial data input. The clock can be

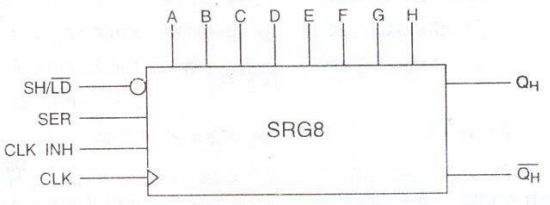


Fig. 5.15. Logic symbol of 8 bit parallel in – serial out IC 74165 shift register

inhibited any time by a High on CLKINH input. The serial data output is $\overline{Q_H}$ Fig. 5.15 shows its symbol.

Example 5.3. Fig. 5.16 (a) shows the symbol of a 4 bit parallel in –serial out shift register. Data 1010 is entered in it. Show the data and waveform along with clock and shift/ Load waveforms.

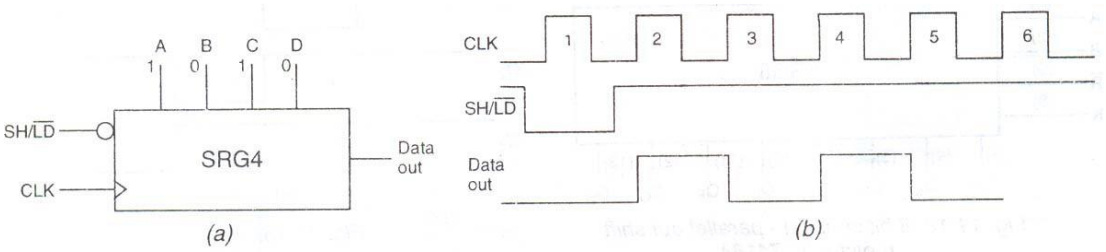


Fig. 5.16

Solution: The clock, shift/ $\overline{\text{Load}}$ and data (1010) waveforms are shown in Fig. 5.16 (b).

5.11. PARALLEL IN – PARALLEL OUT SHIFT REGISTER

Fig. 11.17 (a) shows a 4 bit parallel in – parallel out shift register. ABCD are parallel data bits and Q_A Q_B Q_C Q_D are parallel data outputs.

IC 74165 is a 4 bit parallel in – parallel out shift register. Its symbol is shown in in Fig. 5.17 (b). when SH/ $\overline{\text{LD}}$ input is 'low, data on parallel lines A, B, C, D can be entered synchronously on the positive edge of clock pulse. When SH/ $\overline{\text{LD}}$ input is High, stored data shifts to right (i.e., Q_A to Q_D) synchronously with the clock.

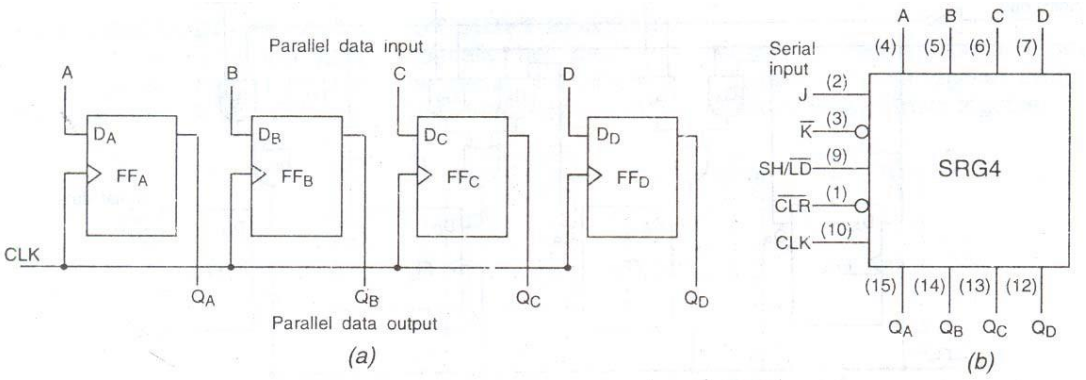
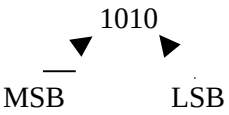


Fig. 5.17(a) Parallel in – parallel out shift register (b) logic Symbol of IC 74195 4 bit shift register

This IC can also be used for serial input – serial output. J and $\overline{K}$ are serial data inputs to the first stage of IC. Q_D can be used for serial data output. The active Low clear is asynchronous.

Example 5.4. In Fig. 5.17 (a), A = 0, B = 1, C = 0 and D = 1. What are data outputs after 2 clock pulses.

Solution : The data output remains the same as input, i.e.,



5.12. UNIVERSAL SHIFT REGISTER

A universal shift register (also known as bidirectional shift register) can shift data in both directions, i.e., left as well as right. The logic gates are arranged in such a way that data bits can be transferred from one stage to the next in either direction depending on the control line input. Fig. 5.18 (a) shows the circuit of such a register using D flip flops. When Right/ $\overline{\text{Load}}$ control input is High, it acts as a shift right register. This occurs because gates G_1, G_2, G_3, G_4 are enabled and the Q output of one stage goes to D input of next stage. At positive edge of each clock, data bits shift one place to the right. When Right/ $\overline{\text{Left}}$ control input is Low, it acts as a shift left register. In this case gates G_5, G_6, G_7, G_8 are enabled and Q

output of each stage goes to D input of preceding stage. At the positive edge of each clock data bits shift one place to the left.

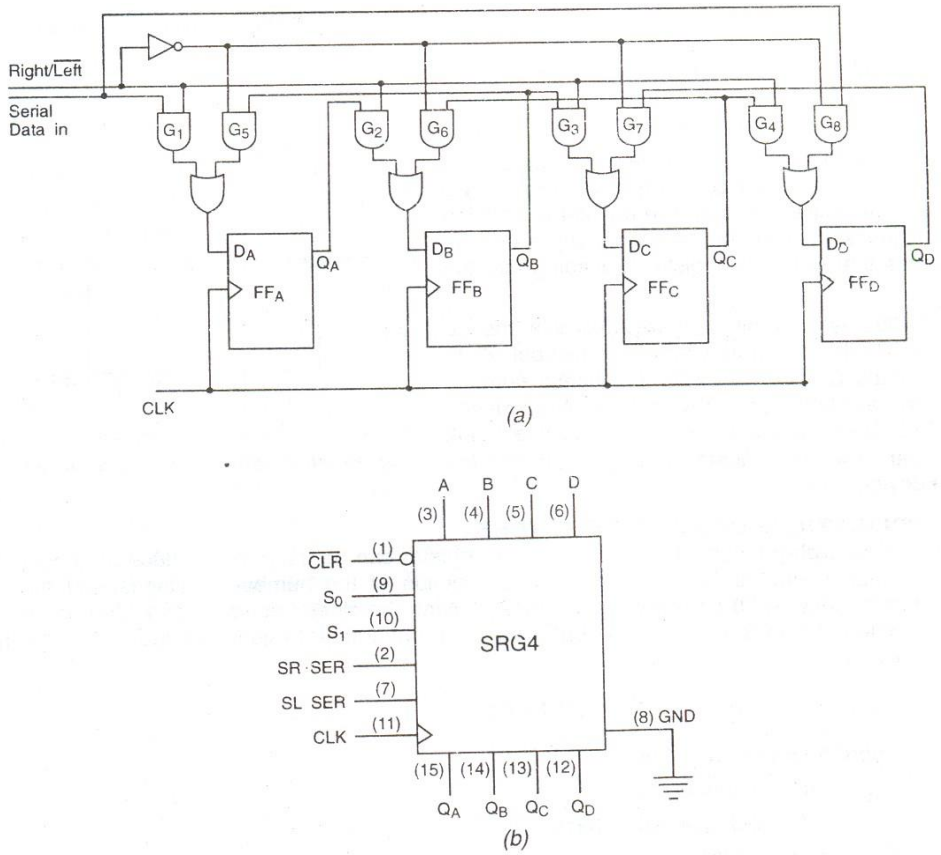


Fig. 5.18. (a) 4 bit universal shift register (b) symbol of IC 74194 4 bit universal shift register

IC 74194 is a 4 bit universal shift register. Its symbol is shown in Fig. 5.18 (b). ABCD are the inputs and Q_A Q_B Q_C Q_D are outputs. Parallel loading is achieved by applying 4 bits of data simultaneously to the inputs ABCD and High to S_0 ,S_1 inputs. At positive edge of clock, parallel data is loaded into the register.

When S_0 is High and S_1 is Low, shift right operation occurs at positive edge of clock. In this mode serial data can be entered as shift right serial input (SR SER). When S_0 is Low and S_1 is High, shift left operation occurs at positive edge of clock. In this mode serial data can be entered at shift serial input (SL SER).

5.13. TRISTATE Buffer register

The term tristate means three stages. In digital systems refers to three levels, viz., High Low and High Impedance. Table 5.1 shows its truth table.

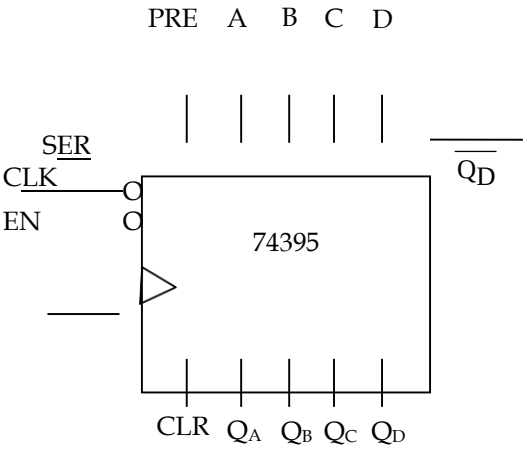
Table 5.1: Truth table of Tristate buffer register

$\overline{EN}$	Data in	Output
0	1	1
0	0	0
0	1	Float (High impedance)
0	0	Float (High impedance)

When $\overline{EN}$ is Low the register works as a buffer register. When EN is High, the output is placed in high impedance (i.e., open circuit) level.

Many times it is necessary to feed the output from who or more registers to a common device. In such a situation it is essential that only one register is enabledand the other is in float (high impedance) condition. This ensures that only one register is feeding into the device.

IC 74395 is a 4 bit shift register with



register with tristate feature. Fig. 5.19 shows its when logic symbol. A, B, C, D are inputs

Q_A, Q_B, Q_C, Q_D are tristate outputs. Outputs are available only EN is Low. Non tristate

Fig. 5.19 Tristate buffer register IC 74395

output $\overline{Q_D}$ is also available. SER is serial

data input. $\overline{CLR}$ is clear Low. As indicated
it is a negative edge triggered device.

5.14. APPLICATIONS OF SHIFT RECISTERS

(a) **Time Delay :** Serial in – serial out shift register can be used to introduce time delay from input to output. This time delay is a function of the number of stages and the clock frequency. An 8 bit register (IC 7491A) having the clock frequency of 1 MHz will mean a time delay of 8×1, i.e., 8 μs between input and output. In general time delay Δt is given by

$$\Delta t = n \times \frac{1}{f}$$

where Δt = time delay, μs

n = number of stages

f = clock frequency, MHz

(b) **Ring Counter :** If the output of a shift register is fed back to serial input, the shift register can be used as a ring counter.

(c) **Serial to parallel data conversion :** In many digital systems, serial data is used to reduce the number of input lines. By using serial in – parallel out shift register, serial data can be converted to parallel data.

SUMMARY

1. Shift registers have two functions: to store data and to shift data (for arithmetic operations etc.).
2. Flip flops are connected together to for shift registers. IC shift registers are also available.
3. A buffer register is the simplest of all registers. It can only store data.
4. Basic shift operations are : shift left, shift right.
5. Data entry and output can be : serial in – serial out, serial in – parallel out, parallel in – serial out, parallel in – parallel out. ICs are available for all the above configurations.

6. A universal shift register (also called bidirectional shift register) can shift left as well as shift right.

7. A tri state buffer register has three states, viz., High, Low and High impedance.

PROBLEMS

5.1. What are the applications of shift registers.

5.2. A parallel in – serial out shift registers can be used to convert parallel data to serial data. Is this statement true or false, comment?

5.3. A parallel in – parallel out shift register can be used to introduce delay in digital signals. Is this statement true or false, comment?

5.4. A serial in – serial out shift register can be used o introduce delay in digital signals. Evaluate the statement and comment.

5.5. A universal shift register can shift left or right, How?

5.6. Explain the concept of UP – DOWN counter.

5.7. How shift left and shift right operations occur in a calculator also.

5.8. How parallel in – parallel out shift register all bits are loaded and read simultaneously.

5.9. Expalin, will serial in – serial out shift registers can be fabricated from D flip flops

5.10. In IC 74165, only parallel data can be loaded. Justify the statement.

B. Fill in the blanks with suitable words :

5.1. Serial loading means entering_____bit per clock pulse.

5.2. When load input of a buffer register is active, the input word is stored on the next _____ of clock pulse.

5.3. In parallel loading only _____ _____ _____ is required to load all the bits.

5.4. IC 74194 is_____shift register.

5.5. The two principal functions of a shift register are_____and _____

CHAPTER-6
COUNTERS

6.1. INTRODUCTION

In computer terminology a group of memory elements is called a register. A counter is a register capable of counting the number of clock pulses which have arrived at clock input. Counters are used for a variety of counting purposes, e.g., counting the number of revolution of a motor in a given time, frequency division required to produce the hour, minute and seconds output in a digital watch etc.

6.2 TERMINOLOGY AND CLASSIFICATION

Binary counters generate 0s and 1s. A counter for counting 4 bits will start at decimal number 0 and count upto decimal number 3. The binary sequence is 00, 01, 10, 11. The counter may not have this sequence but may follow an arbitrary sequence also. If a decimal output of the count is required, the output of counter can be converted to decimal by adecoder. In counting from 0 to 3, the counter goes through 4 states and then recycles. This number is called modulus of counter. This counter is mod-4 counter.

We know that each flip flop has 2 possible states. If a counter has N flip flops, it has a maximum of 2^N states. Each of these 2^N states has a unique combination of 0s and 1s stored in the different flip flops. A counter with N flip flops can have a maximum modulus of 2^N. However, the actual number of states used may be less than maximum. If 4 flip flops are used, the maximum modulus can be 2^4 or 16. But a counter for counting upto 10 is known as decade or mod 10 counter though it has 4 flip flops and has a maximum modulus of 16.

Counters are classified as a synchronous and asynchronous. Synchronous counter is clocked such that all the flip flops are clocked at the same time. For this purpose the clock terminal is connected to each stage of the counter. Asynchronous means that flip flops, in the counter, do not change state exactly at the same instant. In asynchronous counter the clock pulses are not connected directly to clock input of each flip flop in the counter. The counters mostly use JK flip flops connected properly.

The word binary counter means a counter which counts and produces binary output 0000, 0001, 0010, 0011 etc. It goes through a binary sequence depending on its modulus.

6.3 CIRCUIT AND WORKING OF RIPPLE COUNTER:

Fig. 6.1 (a) shows the circuit of a bit ripple counter consisting of 4 edge triggered JK flip flop. As indicated by small circles at the CLK input of flip flops, the triggering occurs when CLK input gets a negative edge. Q_0 is the least significant bit (LSB) and Q_3 is the most significant bit (MSB). The flip flops are connected in series. The Q_0 output is connected to CLK terminal of second flip flop. The Q_1 output is connected to CLK terminal of third flip flop and so on. By adding more flip flop, a counter of any length can be built. It is known asa ripple counter because the carry moves through the flip flops like a ripple on water. Initially, CLR is made low and all flip flops reset giving an output Q = 0000. When CLR becomes high, the counter is ready to start. As LSB receives its clock pulse, its output changes from 0 to 1 and the total output Q = 0001. When second clock pulse arrives, Q_0 resets and carries (i.e., Q_0 goes from 1 to 0 and, second flip flop will receive CLK input). Now the output is Q = 0010. The third CLK pulse changes Q_0 to 1 giving a total output Q = 0011. The fourth CLK pulse causes Q_0 to reset and carry and Q_1 also resets and carries givinga total output Q = 0100 and the process goes on. The action is shown in Table 6.1. The number of output states of a counter are known as modulus (or mod). A ripple counter with 4 flip flops can count from 0 to 15 and is, therefore, known as mod-16 counter while one with 6 flip flops can count from 0 to 63 and is a mod-64 counter and so on.

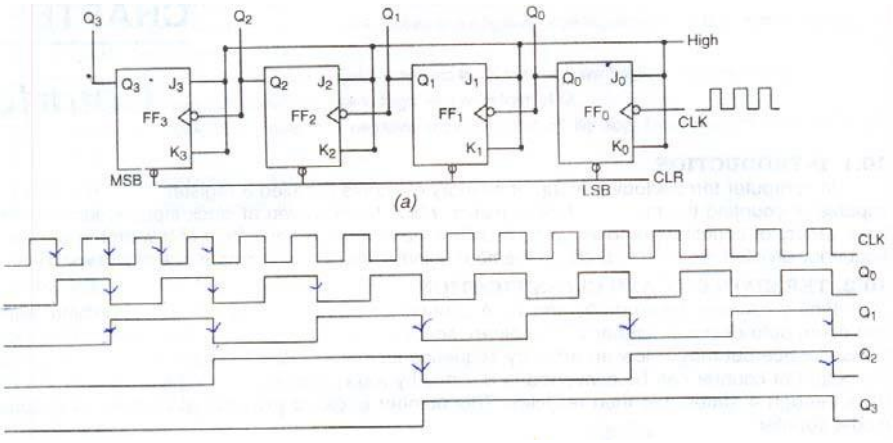


Fig. 6.1 Ripple counter (a) circuit (b) timing diagram

Table 6.1 4 Bit Ripple Counter

Count	Q_3	Q_2	Q_1	Q_0	Count	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	8	1	0	0	0
1	0	0	0	1	9	1	0	0	1
2	0	0	1	0	10	1	0	1	0
3	0	0	1	1	11	1	0	1	1
4	0	1	0	0	12	1	1	0	0
5	0	1	0	1	13	1	1	0	1
6	0	1	1	0	14	1	1	1	0
7	0	1	1	1	15	1	1	1	1

Ripple counters are simple to fabricate but have the problem that the carry has to propagate through a number of flip flops. The delay times of all the flip flops are added. Therefore, they are very slow for some applications. Another problem is that unwanted pulses occur at the output of gates.

The timing diagram is shown in Fig. 6.1 (b). FF_0 is LSB flip flop and FF_3 is the MSB flip flop. Since FF_0 receives each clock pulse, Q_0 toggles once per negative clock edge as shown in Fig. 6.1 (b). The remaining flip flops toggle less often because they receive negative clock edge from preceding flip flops. When Q_0 goes from 1 to 0, FF_1 receives a negative edge and toggles. Similarly, when Q_1 changes from 1 to 0, FF_2 receives a negative edge and toggles. Finally when Q_2 changes from 1 to 0, FF_3 receives a negative edge and toggles. Thus whenever a flip flop resets to 0, the next higher flip flop toggles.

From the above discussion it is evident why this counter is known as ripple counter. As the 16^{th} clock pulse is applied, the trailing edge of 16^{th} pulse causes a transition in each flipflop. Q_0 goes from High to Low, this causes Q_1 go from High to Low which causes Q_2 to go from High to Low which causes Q_3 to go from High to Low. Thus the effect ripples through the counter. It is the delay caused by this ripple which results in a limitation on the maximum frequency of the input signal.

6.3.1 DIVIDE BY N EFFECT

The waveform in Fig. 6.1 (b) shows that the frequency of output Q_0 is one half of that of input, frequency of output Q_1 is one fourth of input frequency, frequency of output Q_2 is

one eighth of input frequency and frequency of output Q_3 is one-sixteenth of input frequency. Thus the circuit acts as a frequency divider. It we have N flip flops, the input frequency is divided by 2^N in steps of 2.

6.4 SYNCHRONOUS COUNTER

In a synchronous counter, all the flip flops are clocked together. Fig. 6.2 shows a synchronous counter having positive edge triggered JK flip flops. Since all the flip flops are clocked together, the delay time is less. The flip flop corresponding to lest significant bit (LSB) has its input JK fed from voltage + V_{CC}. Therefore, it responds to each positive clock edge. However, the other three flip flops can respond to the positive clock pulse under some certain conditions. The Q_1 flip flop toggles on positive clock edge only if Q_0 is 1. The Q_2 flipflop toggles on positive clock edge only when Q_0 and Q_1 are (due to presence of AND circuit)and so on.

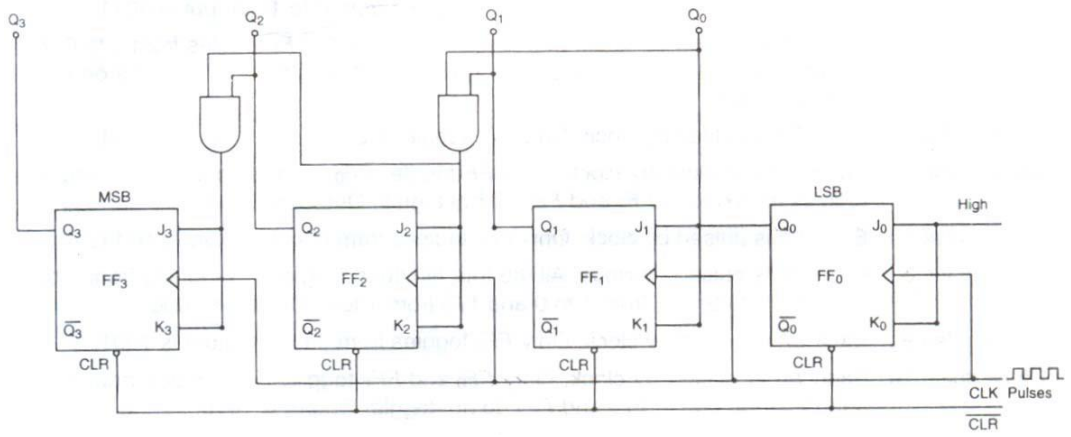


Fig. 6.2 Synchronous counter

Thus, each flip flop toggles on the next positive clock edge if all lower bits are 1.

A low $\overline{CLR}$ signal resets the counter so that Q = 0000. When $\overline{CLR}$ goes high, the counter is ready to start. The first positive clock edge sets Q_0 to 1 so that Q = 0001. At second positive clock edge Q_1 and Q_0 toggle and Q = 0010. The third positive clock edge increases the count by 1 so that Q = 0011.

The successive Q outputs are 0100, 0101 and so on upto 1111 (i.e., decimal 15). The next positive clock edge resets the counter to 0000 and the cycle is repeated. More flip flops can be added to increase the count.

The counting sequence is given in Table 6.2.

The circuit action and output for various clock pulses are as under:

Clock pulse 1 : Each FF is pulsed by clock. However, only FF_0 can toggle because it is the only one with 1s applied to both J and K inputs. FF_0 goes from 0 to 1. Output is 0001.

Clock pulse 2 : Each FF is pulsed by clock. FF_0 and FF_1 have 1s applied to both J and K inputs. Therefore, FF_0 and FF_1 toggle. FF_1 goes from 0 to 1 and FF_0 goes from 1 to 0. Output is 0010.

Table 6.2 Counting sequence of 4 bit Synchronous Counter

Number of clock pulses	Counting sequence				Equivalent decimal
	Q_3	Q_2	Q_1	Q_0	
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	2
3	0	0	1	1	3
4	0	1	0	0	4
5	0	1	0	1	5
6	0	1	1	0	6
7	0	1	1	1	7
8	1	0	0	0	8
9	1	0	0	1	9
10	1	0	1	0	10
11	1	0	1	1	11
12	1	1	0	0	12
13	1	1	0	1	13
14	1	1	1	0	14
15	1	1	1	1	15

Clock pulse 3 : Each FF is pulsed by clock. Only FF_0 toggles from 0 to 1. Output is 0011.

Clock pulse 4 : Each FF is pulsed by clock. FF_0 toggles from 1 to 0, FF_1 toggles from 1 to 0. FF_2 toggles from 0 to 1. FF_3 does not toggle. (Its JK inputs do not have 1s on then). Output is 0100.

Clock pulse 5 : Each FF is pulsed by clock. Only FF_0 toggles from 0 to 1. Output is 0101.

Clock pulse 6 : Each FF is pulsed by clock. Two FF toggle. FF_0 toggle from 1 to 0 and FF_1 toggles from 0 to 1. FF_2 and FF_3 do not toggle. Output is 0110.

Clock pulse 7 : Each FF is pulsed by clock. Only FF_0 toggles from 0 to 1. Output is 0111.

Clock pulse 8 : Each FF is pulsed by clock. All the four flip flops toggle. FF_3 toggles from 0 to 1, FF_2 from 1 to 0, FF_1 from 1 to 0 and FF_0 from 1 to 0. Output is 1000.

Clock pulse 9 : Each FF is pulsed by clock. Only FF_0 toggles from 0 to 1. Output is 1001. Clock pulse 10 : Each FF is pulsed by clock. Only FF_0 and FF_1 toggle. FF_0 toggles from 1 to 0 and FF_2 from 0 to 1. FF_2 and FF_3 do not toggle. Output is 1010.

Clock pulse 11 : Each FF is pulsed by clock. Only FF_0 toggles from 0 to 1. FF_3, FF_2 and FF_1 do not toggle. Output is 1011.

Clock pulse 12 : Each FF is pulsed by clock. FF_3 does not toggle. FF_2 toggles from 0 to 1, FF_1 from 1 to 0 and FF_0 from 1 to 0. Output is 1100.

Clock pulse 13 : Each FF is pulsed by clock. FF_3, FF_2, FF_1 do not toggle. Only FF_0 toggles from 0 to 1. Output is 1101.

Clock pulse 14 : Each FF is pulsed by clock. FF_3 and FF_2 do not toggle. FF_1 toggles from 0 to 1 and FF_0 toggles from 1 to 0. Output is 1110.

Clock pulse 15 : Each FF is pulsed by clock. FF_3, FF_2, FF_1 do not toggle. Only FF_0 toggles from 0 to 1. Output is 1111.

Clock pulse 16 : Each FF is pulsed by clock. All FFs toggle from 1 to 0. Output is 0000.

Thus 1 cycle of operation is completed. The 4 bit synchronous counter of Fig. 10.13 is Mod-16 counter.

6.5 UP-DOWN COUNTER

As the name suggests an UP-DOWN counter can count either upwards from 0000 onwards or from the highest number (equal to modulus of counter) to 0000. Thus a 4 bit up-down counter can count from 0000 to 1111 and also from 1111 to 0000.

Fig. 6.3 shows a 3 bit up-down counter. When UP signal is High, Q_1, Q_0 drive the clock inputs and the circuit counts upwards from 000 to 111. When UP signal is Low, $\overline{Q_1}$, $\overline{Q_0}$ drive the clock inputs and the circuit counts downwards from 111 to 000.

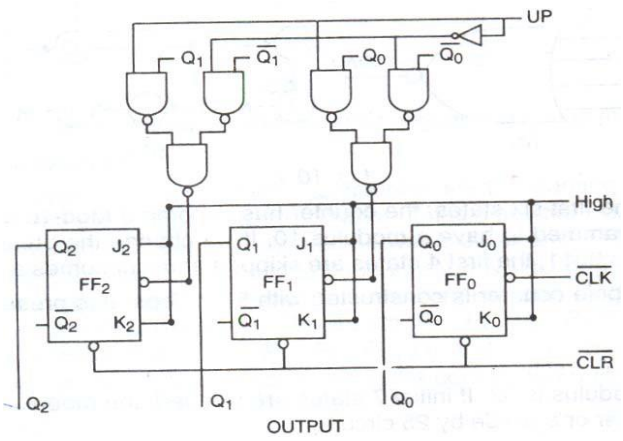


Fig. 6.3 Up-down counter

Example 6.1 A ripple counter is constructed with 5 flip flops. It is preset to skip initial 7 states. Find the modulus.

Solution : $2^5 = 32$

Thus the natural modulus is 32. If initial 7 states are skipped, the modulus is 32 – 7 = 25. Thus it becomes a Mod-25 counter or a divide by 25 circuit.

6.6 DIFFERENCE BETWEEN ASYNCHRONOUS AND SYNCHRONOUS COUNTER

The term asynchronous means that the counter is clocked in such a way that all the flip flops of the counter do not receive clock pulses at the same time. Ripple counter is an asynchronous counter. The clock pulses drive the clock input of the first flip flop. But the clock input of the second flip flop is driven by Q output of the first flip flop. The clock input of third flip flop is driven by the Q output of the second flip flop and so on. This results in propagation delay in each flip flop. In a 4 bit ripple counter having flip flops with propagation delay of 10 ns each, there will be a total propagation delay of 40 ns. This may be undesirable in many cases.

The term synchronous means that all flip flops are clocked simultaneously. The clock pulses drive the clock input of all the flip flops together so that there is no propagation delay.

6.7 RING COUNTER

If the serial output Q_0 of the shift register of Fig. 8.2 is connected back to the serial input, then an injected pulse will keep circulating. This circuit is referred to as a *ring counter*. The pulse is injected by entering 00001 in the parallel form after clearing the FLIP-FLOPs. When the clock pulses are applied, this 1 circulates around the circuit. The waveforms at the Q outputs are shown in Fig. 6.5. The outputs are sequential non-overlapping pulses which are useful for control-state counters, for stepper motor (which rotates in steps) which require sequential pulses to rotate it from one position to the next, etc.

Fig 6.4: A five bit register (7496)

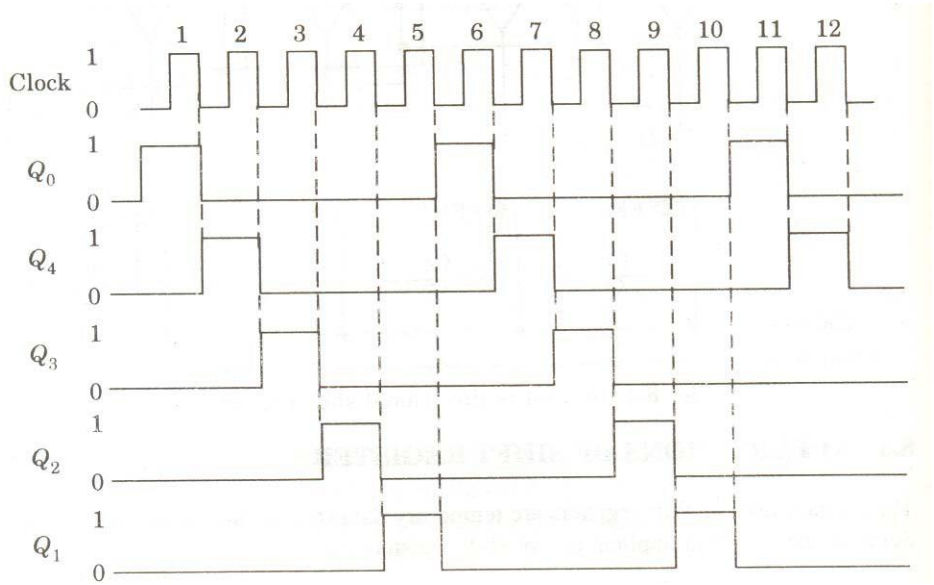


Fig. 6.5 Output waveforms of ring counter.

This circuit can also be used for counting the number of pulses. The number of pulses counted is read by noting which FLIP-FLOP is in 1 state. No decoding circuitry is required. Since there is one pulse at the output for each of the N clock pulses, this circuit is referred to as a *divide-by-N counter* or an N : 1 *scalar*.

6.8 TWISTED-RING COUNTER

In the shift register of Fig. 6.4, if $\overline{Q}_0$ is connected to the serial input, the resulting circuit is referred to as a *twisted-ring Johnson, or moebius* counter. If the clock pulses are applied after clearing the FLIP-FLOPs, square waveform is obtained at the Q outputs as shown in Fig. 6.6.

Similar to ring-counter sequence, the moebious is also useful for control- state counters. It is also useful for the generation of multiphase clock.

The moebious counter is a divide-by-2N counter. For decoding the count, two input AND gates are required. The decoder circuit for a five-stage counter is shown in Fig. 6.7.

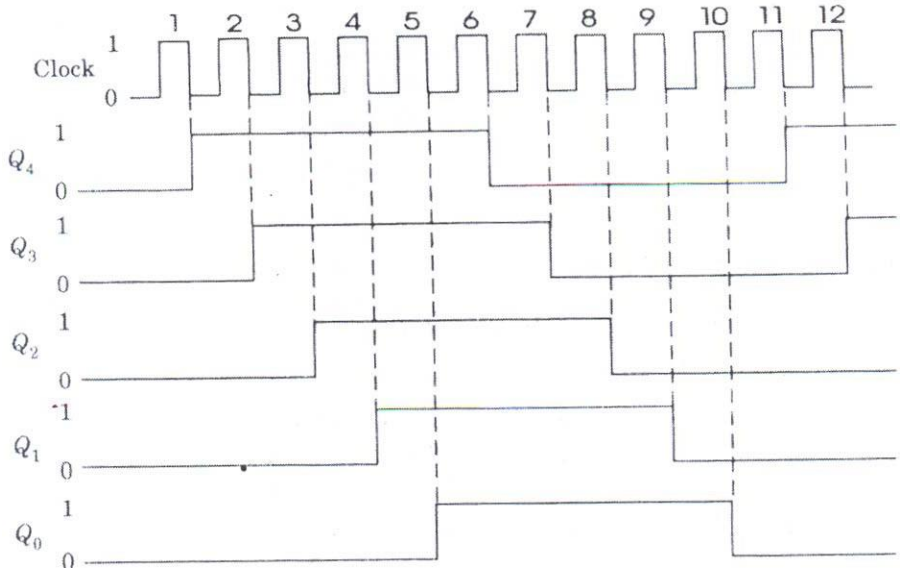


Fig. 6.6 Output wave forms of twisted ring counter

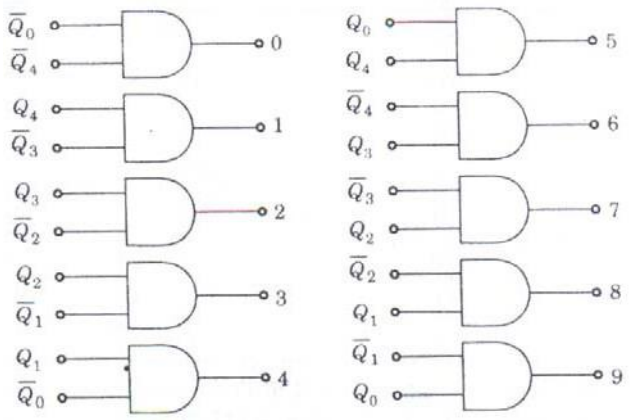


Fig. 6.7 The decoding logic for a 5-stage twisted-ring counter.

6.9 RIPPLE OR ASYNCHRONOUS COUNTER DESIGN

Consider the count sequence shown in Table 6.3. The number of states in this sequence is 8 which requires 3 FLIP-FLOPs ($2^3 = 8$) and Q_2, Q_1, and Q_0 are the outputs of these FLIP-FLOPs. Assume master-slave FLIP-FLOPs.

Table 6.3 Counting sequence of a 3-bit binary counter

Counter state	Count		
	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

The output Q_0 of the least-significant FLIP-FLOP, changes for every clock pulse. This can be achieved by using T-type FLIP-FLOP with $T_0 = 1$. The output Q_1 makes a transition (from 0 to 1 or 1 to 0) whenever Q_0 changes from 1 to 0. Therefore, if Q_0 is connected to the clock input of next T-type FLIP-FLOP, FF1 with $T_1 = 1$, Q_1 will change whenever Q_0 goes from 1 to 0 (falling edge of clock pulse). Similarly, Q_2 makes a transition whenever Q_1 goes from 1 to 0 and this can be achieved by connecting Q_1 to the clock input of the most-significant FLIP-FLOP, FF2 (and $T_2 = 1$). The resulting circuit is shown in Fig. 6.8. The wave-forms of the outputs of the FLIP-FLOPs are shown in Fig. 6.9

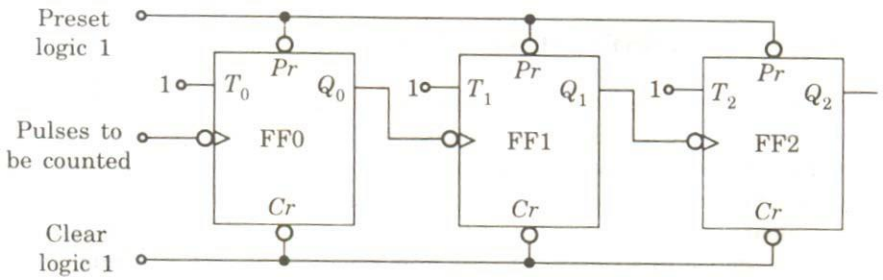


Fig. 6.8 A 3-bit binary counter.

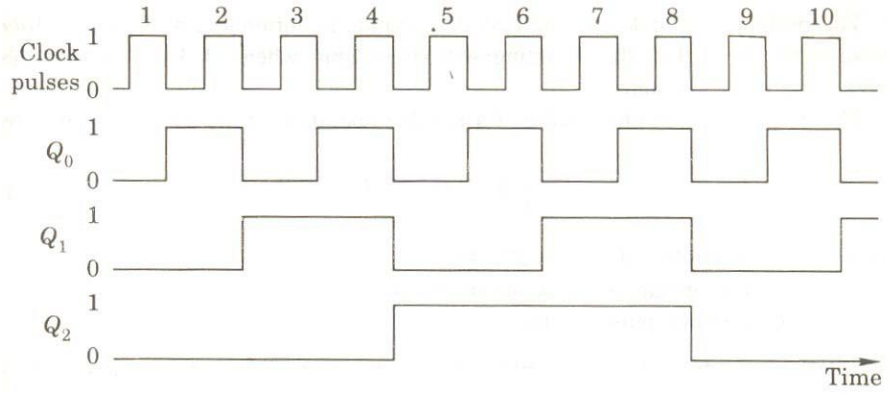


Fig. 6.9 Output waveforms of counter of Fig. 6.8

A decoder circuit for decoding the count is shown in Fig. 6.10. In this, the output corresponding to the number counted goes low (active-low).

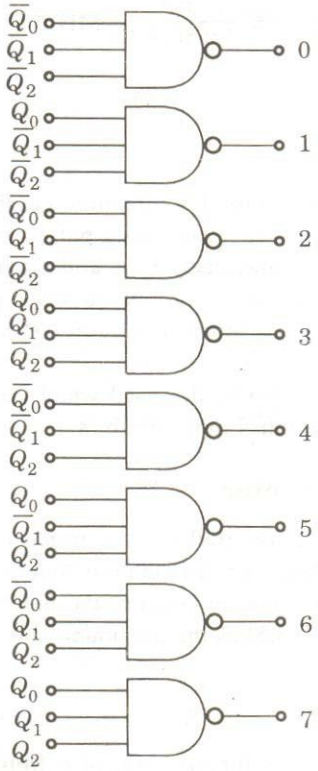


Fig. 6.10 A decoder circuit for a 3-bit binary counter.

At the decoder outputs, false pulses of a short duration, known as spikes, occur as counter FLIP-FLOPs change state. This is because of the propagation delay of the FLIP- FLOPs due to which either all the FLIP-FLOPs do not change sate at exactly the same time oronly one FLIP-FLOP changes state for any clock pulse.

The problems of spikes at the decoder outputs is eliminated by using a strobe pulse input. With this, the decoding will occur only when all the FLIP-FLOPs have come to steady state.

The frequency, f, of clock pulses for reliable operation of the count is given by

$$\frac{1}{f} \geq N \cdot (t_d) + T_N \qquad \ldots(6.1)$$

where N = number of FLIP-FLOPs

t_d = propagation delay of one FLIP-FLOP

T_s = strobe pulse width

Example 6.2 In a 4-stage ripple counter, the propagation delay of a FLIP-FLOP is 50 ns. If the pulse width of the strobe is 30 ns, find the maximum frequency at which the counter operates reliably.

Solution : The maximum frequency is

$$f_{max} = \frac{10^3}{4 \times 50 + 30} \text{MHz}$$

$$= 4.35 \text{ MHz.}$$

6.10 SYNCHRONOUS COUNTER DESIGN

Synchronous counters for any given count sequence and modulus can eb designed in the following way:

1. Find the number of FLIP-FLOPs required using Eq. $m \leq 2^N$
2. Write the count sequence in the tabular form similar to Table 6.3.
3. Determine the FLIP-FLOP inputs which must be present for the desired next state from the present state using the excitation table of the FLIP-FLOPs.
4. Prepare K-map for each FLIP-FLOP input in terms of FLIP-FLOP outputs as the input variables. Simplify the K-maps and obtain the minimized expressions.

5. Connect the circuit using FLIP-FLOPs and other gates corresponding to the minimized expressions.

The above design steps can be clearly understood from the following examples.

Example 6.3 Design a 3-bit synchronous counter using J–K FLIP-FLOPs.

Solution : The number of FLIP-FLOPs required is 3. Let the FLIP-FLOPs be FF0, FF1, and FF2 and their inputs and outputs are given below:

FLIP-FLOP	Inputs	Output
FF0	J_0, K_0	Q_0
FF1	J_1, K_1	Q_1
FF2	J_2, K_2	Q_2

Table 6.4

Counter state			FLIP-FLOP inputs					
			FF0		FF1		FF2	
$\overline{Q_2}$	Q_1	Q_0	J_0	K_0	J_1	K_1	J_2	K_2
0	0	0	1	×	0	×	0	×
0	0	1	×	1	1	×	0	×
0	1	0	1	×	×	0	0	×
0	1	1	×	1	×	1	1	×
1	0	0	1	×	0	×	×	0
1	0	1	×	1	1	×	×	0
1	1	0	1	×	×	0	×	0
1	1	1	×	0	×	1	×	1
0	0	0						

The count sequence and the required inputs of FLIP-FLOPs are given in Table 6.4. The inputs to the FLIP-FLOPs are determined in the following manner :

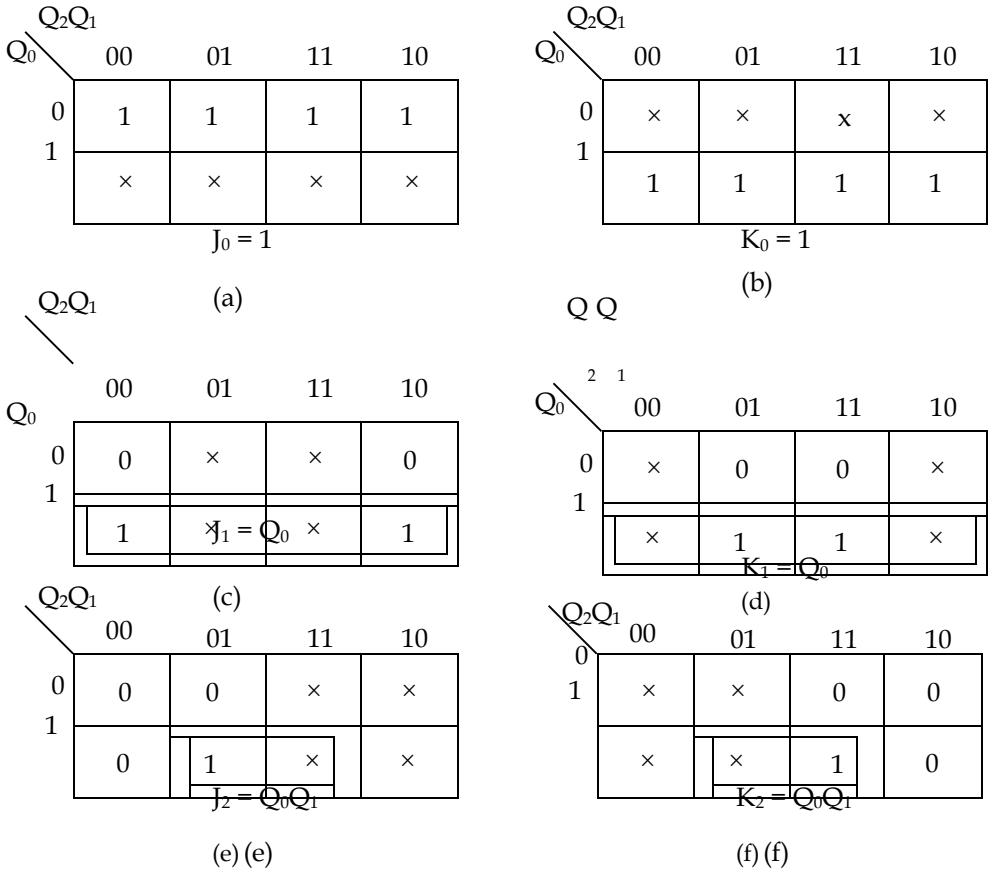


Fig. 6.11 K-maps of Ex. 6.3

Consider one column of the counter state at a time and start from the first row, for example, consider Q_0. Before the first pulse is applied, $Q_0 = 0$ and it is required to be 1 at the end of the first clock pulse. Therefore, to achieve this condition, the values of J_0 and K_0 are 1 and × respectively (from the excitation table 4.6). These are entered in the table in the row corresponding to 0 pulse. When the second clock pulse is applied Q_0 is to change from 1 to 0, therefore, the required inputs are

$$J_0 = \times, K_0 = 1$$

In a similar manner inputs of each FLIP-FLOP are determined.

Now, we prepare the K-maps (Fig. 6.11) with Q_2, Q_1 and Q_0 as input variables and FLIP-FLOP inputs as output variables. We then minimize the K-maps and the resulting minimized expressions are :

$J = 1, \quad K_0 = 1$

$J_1 = Q_0, \quad K_1 = Q_0$

$J_2 = Q_0\,Q_1, \quad K_2 = Q_0\,Q_1$

The circuit can be implemented using above equations.

Example 6.4 Design a 3-bit binary UP/DOWN counter with a direction control M. Use J–K FLIP-FLOPs.

Solution : The count sequence is given in Table 6.5. For M = 0, it acts as an UP counter and for M = 1 as a DOWN counter. The number of FLIP-FLOPs required is 3. The inputs of the FLIP-FLOPs are determined in a manner similar to the one employed in Ex. 6.3.

Table 6.5

Direction control	Counter state			FLIP-FLOP inputs					
M	Q_2	Q_1	Q_0	J_0	K_0	J_1	K_1	J_2	K_2
0	0	0	0	1	×	0	×	0	×
0	0	0	1	×	1	1	×	0	×
0	0	1	0	1	×	×	0	0	×
0	0	1	1	×	1	×	1	1	×
0	1	0	0	1	×	0	×	×	0
0	1	0	1	×	1	1	×	×	0
0	1	1	0	1	×	×	0	×	0
0	1	1	1	×	1	×	1	×	1
1	0	0	0	1	×	1	×	1	×
1	1	1	1	×	1	×	0	×	0
1	1	1	0	1	×	×	1	×	0
1	1	0	1	×	1	0	×	×	0
1	1	0	0	1	×	1	×	×	1
1	0	1	1	×	1	×	0	0	×
1	0	1	0	1	×	×	1	0	×
1	0	0	1	×	1	0	×	0	×
	0	0	0						

$J_0 = K_0 = 1$

The K-maps for J_1, K_1, J_2 and L_2 are shown in Fig. 6.12. From the k-maps, the minimized expression are obtained as

$$J_1 = K_1 = Q_0 \overline{M} + \overline{Q}_0 M$$

$$J_2 = K_2 = \overline{M} Q_1 Q_0 + M \overline{Q}_1 \overline{Q}_0$$

The counter circuit can be drawn using the above expressions

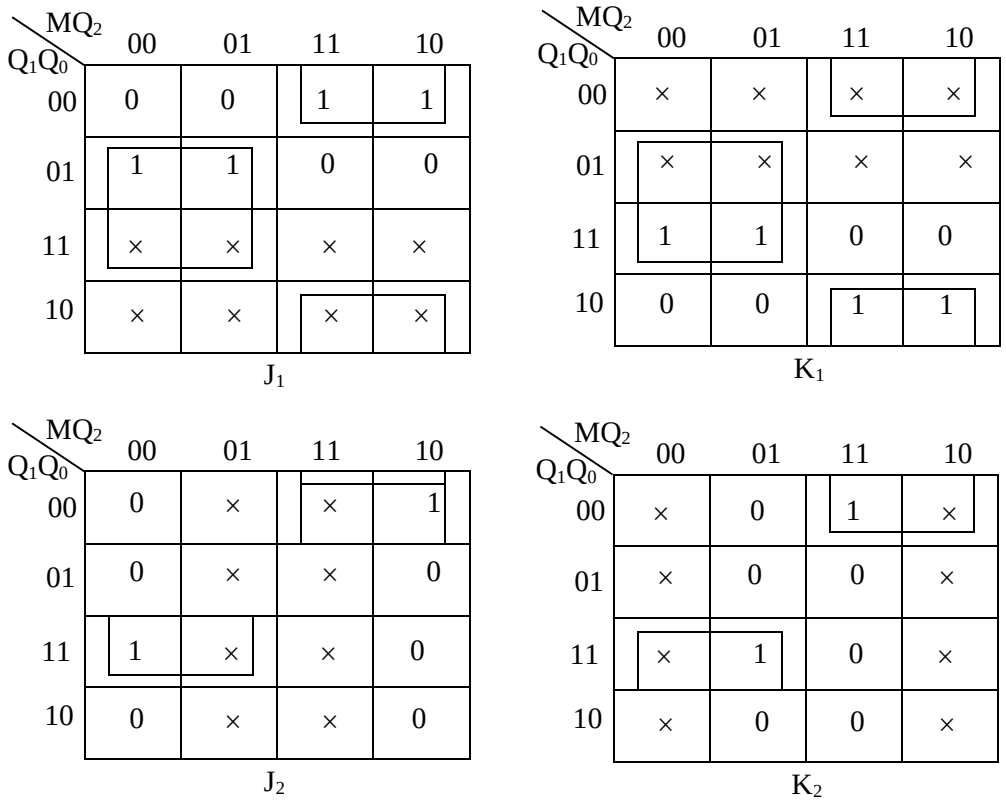


Fig. 6.12 –maps for Ex. 6.4

Example 6.5 Design a decade UP counter. Use J-K FLIP-FLOPs.

Solution : There are ten states in a decade counter, which requires four FLIP-FLOPs. The remaining six states are unused states. The count sequence and the FLIP-FLOP inputs are given in Table 6.6.

Table 6.6

Counter state				FLIP-FLOP inputs							
Q_3	Q_2	Q_1	Q_0	J_0	K_0	J_1	K_1	J_2	K_2	J_3	K_3
0	0	0	0	1	×	0	×	0	×	0	×
0	0	0	1	×	1	1	×	0	×	0	×
0	0	1	0	1	×	×	0	0	×	0	×
0	0	1	1	×	1	×	1	1	×	0	×
0	1	0	0	1	×	0	×	×	0	0	×
0	1	0	1	×	1	1	×	×	0	0	×
0	1	1	1	1	×	×	0	×	0	0	×
0	1	1	1	×	1	×	1	×	1	1	×
1	0	0	0	1	×	0	×	0	×	×	0
1	0	0	1	×	1	0	×	0	×	×	1
0	0	0	0								

The K-maps are drawn for J_0, K_0, J_1, K_1, J_2, K_2 and J_3, K_3 from which the minimized expressions are obtained as

$$J_0 = 1, \qquad K_0 = 1$$

$$J_1 = Q_0 \overline{Q}_3, \qquad K_1 = Q_0$$

$$J_2 = Q_0Q_1, \qquad K_2 = Q_0Q_1$$

$$J_3 = Q_0Q_1Q_2, \qquad K_3 = Q_0$$

The counter circuit can be drawn using the above expressions.

SUMMARY

1. Counters consist of flip flops. They are classified as synchronous and asynchronous. In a synchronous counter the clock terminal is connected to each stage of the counter so that all the flip flops are triggered together. In asynchronous counter all the flip flops are not triggered together.
2. A counter having N flip flops has 2^N states. The actual number of states may be equal or less than 2^N. The actual number of states is called modulus of counter.

3. A ripple counter consists of JK flip flops. The Q output of each stage feeds the clock input of next stage. In this counter the carry moves through the flip flops like a ripple on water.
4. In a controlled ripple counter the COUNT signal controls the action of counter. The counter works only when COUNT is High.
5. A decade counter has a modulus of 10. It uses 4 flip flops but only 10 states are used.
6. A Mod-m counter has m states. It uses a NAND gate to skip the remaining states.
7. A presettable counter counts, not from zero, but from any specified number (say 3). It has a special LOAD terminal and counting begins with P_3 P_2 P_1 P_0 which can be any number between 0000 and 1111 (in a 4 bit counter).
8. In a programmable counter the modulus of the counter can be programmed.
9. An up counter counts from 0000 upwards. A down counter starts with the highest state and counts downwards to 0000. An up-down counter can count in both directions.
10. A self stopping down counter stops at 0000 and does not start the next cycle.
11. In synchronous counter all the flip flops are clocked simultaneously. Therefore, there is no propagation delay. A synchronous counter all the flip flops are clocked simultaneously. Therefore, there is no propagation delay. A synchronouscounter can also have any modulus.
12. A ring counter has D flip flops. The Q output of any stage feeds the D input of next stage. In this counter the stored bits follow a circular path.
13. A Johnson counters also uses D flip flops. The $\overline{Q}$ output of last stage feeds the D input of first stage.
14. Counters can be connected in cascade to achieve higher modulus. When a Mod-4 counter is cascaded with a Mod-8 counter, we get Mod-32 counter.
15. Counters have innumerable application in scientific field, engineering field and every day household, commercial and industrial devices.
16. IC counters are available in all the three families, i.e., TTL, CMOS and ECL.

PROBLEMS

6.1 Explain the operation of the bi-directional shift register.

6.2 Explain the operation of a twisted-ring counter and give its state diagram.

6.3 Explain the decoder logic.

6.4 A stepper-motor drive circuit requires four signal waveforms given in Fig. 6.14 Design a sequence generator to provide the necessary signals for this stepper-motor drive.

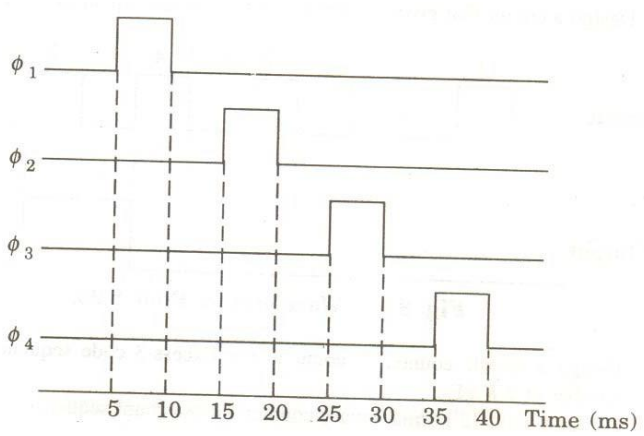


Fig. 6.14 Waveforms

6.5 Write the count sequence of a 3-bit binary DOWN counter. Design a ripple counter using FLIP-FLOPs for this sequence.

6.6 Design a 4-bit binary UP/DOWN ripple counter with a control for UP/DOWN counting.

6.7 Design a 4-bit asynchronous counter with provision for asynchronous loading.

6.8 The latch is used for eliminating resetting difficulties due to unequal internal delays of FLIP-FLOPs. Explain the operation of this latch and show how the resetting difficulties are eliminated.

6.9 Convert 7493 into a 4-bit DOWN counter.

6.10 What is meant by modulus of a counter?

6.11 Differentiate between asynchronous and synchronous counters.

6.12 Explain the terms: up counter, down counter, up-down counter.

6.13 Which type of flip flops are used in counters?

6.14 What is the function of counter in a computer?

6.15 Name applications of counters.

6.16 How are counters classified?

6.17 Discuss the design procedure for Mod-m asynchronous counter.

6.18 Explain the terms: Presettable counter, programmable counter.

6.19 Draw the circuit of a 4 bit up-down counter. How are both up and down features obtained?

6.20 Draw the circuit of a 3 bit synchronous counter and explain its working.

6.21 What is a ring counter? Draw its circuit and explain its working.

CHAPTER-7

INTRODUCTION OF DIGITAL LOGIC FAMILY

7.1. INTRODUCTION

The design of a logic system starts with the statement of logic problems. This problem is then translated into a truth table and then into a logic equation. The logic equation yields the logical blocks and their interconnection so that we get the desired output for the given input conditions. From the logical block we realize an actual digital circuit.

The basic building block in a digital system is logic gate logic circuits have evolved into families each of which has its own advantages and disadvantages. Generally a digital system is designed with circuits from one family only. When circuits from more than one family are used. It is necessary to ensure that the output of one is compatible with input to the other.

The different logic functions are available in integrated circuit (IC) form. All digital systems are built with digital ICs. The present day ICs have many advantages in terms of size, power, reliability *and const over discrete circuits*. A monolithic integrated circuit is an electronic circuit constructed entirely on a single chip of silicon. All the components, i.e., transistors, diodes, resistors etc. are an integral part of this chip. Typical chip sizes range from 40 × 40 mils** to about 300 × 300 mils and it contains both active and passive components. Many processes like wafer preparation, impurity diffusion, ion implantation, oxide growth, photolithography are used in their manufacture.

7.2. CLASSIFICATION OF LOGIC FAMILIES

7.2.1. Classification as per Level of Integration

As per this classification digital integrated circuits can be classified as small scale integration (SSI), Medium scale integration (MSI), Large scale integration (LSI), Very large scale integration (VLSI) and ultra large scale integration (ULSI). This classification is given in Table 7.1.

Table 7.1. Levels of integration of digital ICs

Category	Numbers of equivalent basic gates on a single chip	Total number of components on single chip
SSI	Less than 12	Less than 100
MSI	12-99	100-999
LSI	100-999	1000-9999
VLSI	1000-9999	10,000 to 99,999
ULSI	10,000 and more	100,000 and more

SSI are the least complex and include basic gates and flip flops. They are available in dual in package (DIP) or flat package and 14 or 16 pin versions.

Small digital sub systems form the MSI category. The complex logic functions, e.g., adders, registers, comparators, code converters, counter, multiplexers etc., are fabricated in MSI. They are also available in dual in package (KIP), flat package and carrier package with 24 or 28 pins.

LSI chips are small digital systems, e.g., digital clocks, calculators, microprocessors, ROM, RAM*** etc.

VLSI is a digital system on a chip. Large memory chips and advanced microprocessors fall in this category.

ULSI is the most complex of digital ICs.

In 1960s, digital ICs manufactured were SSI and MSI only. LSI technology was developed in late 1960s and single chip calculators and memories become available. After this microprocessors were introduced in 1973. A microprocessor has the architecture of a computer on a single chip. In 1980s many sections of computers (central processing unit or CPU, RAM, ROM etc.,) were combined into a single chip. At present the latest version of microprocessor has equivalent of millions of transistors on a single chip.

7.2.2. Classification as per Technology

Digital ICS are manufactured by two technologies viz., Bipolar and MOS.

The bipolar family uses transistor fabricated on a chip. This family includes DTL (Diode transistor logic using diodes and transistor), TTL (transistor-transistor logic which

uses transistors only) and ECL (emitter coupled logic). TTL is the most poplar family in SSI and MSI category.

MOS family includes PMOS (p-channel MOSFET), NMOS (n0channel MOSFET) and CMOS (complementary MOSFET). PMOS is almost obsolete. NMOS is dominating the LSI field. CMOS is the most commonly used technology for digital wrist watches, pocket calculators etc.

The technologies being used now-a-days are TTL, ECL and CMOS. Before discussing these technologies we discuss the important specifications of digital ICs.

7.3 CHARACTERISTICS OF DIGITAL ICs

With the widespread use of ICs in digital systems and with the development of various technologies for the fabrication of ICs, it has become necessary to be familiar with the characteristics of IC logic families and their relative advantages and disadvantages. Digital ICs are classified either according to the complexity of the circuit, as the relative number of individual basic gates (2-input NAND gates) it would require to build the circuit to accomplish the same logic function or the number of components fabricated on the chip.

The various characteristics of digital ICs used to compare their performances are:

1. Speed of operation,
2. Power dissipation,
3. Figure of merit,
4. Fan-out,
5. Current and voltage parameters,
6. Noise immunity,
7. Operating temperature range,
8. Power supply requirements, and
9. Flexibilities available.

7.3.1 Speed of Operation

The speed of a digital circuit is specified in terms of the propagation delay time. The input and output waveforms of a logic gate are shown in Fig. 7.1. The delay times are measured between the 50 per cent voltage levels of input and output waveforms. There are two delay times: t_{pLH} , when the output goes from the HIGH state to the LOW state and t_{pLH} ,

corresponding to the output making a transition from the LOW state to the HIGH state. The propagation delay time of the logic gate is taken as the average of these two delay times.

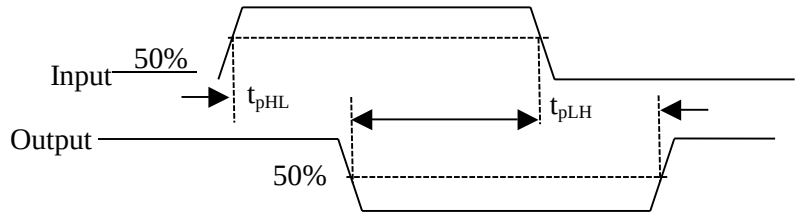


Fig. 7.1 Input and output voltage waveforms to define propagation delay tines.

7.3.2 Power Dissipation

This is the amount of power dissipated in an IC. It is determined by the current, $1_{CC,}$ that it draws from the V_{CC} supply, and is given by $V_{CC} \times 1_{CC}$. 1_{CC} is the average value of 1_{CC} (0) and 1_{CC}(1). This power is specified in milliwatts.

7.3.3 Figure of Merit

The figure of merit of a digital IC is defined as the product of speed and power. The speed is specified in terms of propagation delay time expressed in nanoseconds.

Figure of merit = propagation delay time (ns) × power (mW)

It is specified in pico joules (ns × mW = p^J)

A low value of speed-power product is desirable. In a digital circuit, if it is desired to have high speed, i.e. low propagation delay, then there is a corresponding increase in the power dissipation and vice-versa.

7.3.4 Fan-Out

This is the number of similar gates which can be driven by a gate, High fan-out is advantageous because it reduces the need for additional drivers to drive more gates.

7.3.5 Current and Voltage Parameters

The following currents and voltages are specified which are very useful in the design of digital system.

High-level input voltage, V_{IH}: This is the minimum input voltage which is recognized by the gate as logic 1.

Low-level input voltage, V_{IL}: This is the maximum input voltage which is recognized by the gate as logic 0.

High-level output voltage, V_{OH} : This is the maximum input voltage available at the output corresponding to logic 1.

Low-level output voltage, V_{OL}: This is the maximum input voltage available at the output corresponding to logic 0.

High-level input current, I_{IH}: This is the maximum current which must be supplied by a driving source corresponding to 1 level voltage.

Low-level input current, I_{IL}: This is the maximum current which must be supplied by a driving source corresponding to 0 level voltage.

High-level output current, I_{OH}: This is the maximum current which the gate can sink in 1 level.

Low-level output current, I_{OL}: This is the maximum current which the gate can sink in 0 level.

High-level supply current, I_{CC} (1): This is the supply current when the output of the gate is at logic 1.

Low-level supply current, I_{CC} (0) : This is the supply current when the output of the gate is at logic (0).

The current directions are illustrated in Fig. 7.2.

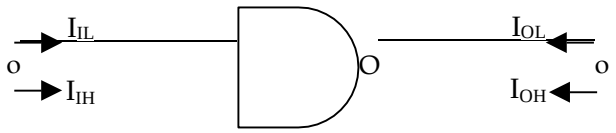


Fig. 7.2 A gate with current directions marked

7.3.6. Noise Immunity

The input and output voltage levels defined above are shown in Fig. 4.3. Stray electric and magnetic fields may induce unwanted voltages, known as noise, on the connecting wires between logic circuits. This may cause the voltage at the input to a logic circuit to drop below V_{IH} or rise above V_{IL} and may produce undesired operation.

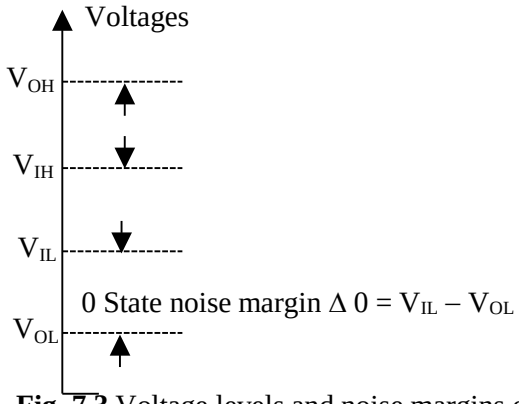


Fig. 7.3 Voltage levels and noise margins of ICs.

The circuit's ability to tolerate noise signals is referred to as the no8ise immunity, a quantitative measure of which is called noise margin. Noise margins are illustrated in Fig. 7.3.

The noise margins defined above are referred to as dc noise margins. Strictly speaking, the noise is generally thought of as an a.c. signal with amplitude and pulse width. For high speed ICs, a pulse width of a few microseconds is extremely long in comparison to the propagation delay time of the circuit and therefore, may be treated as d.c. as far as the response of the logic circuit is concerned. As the noise pulse width decreases and approaches the propagation delay time of the circuit, the pulse duration is too short for the circuit to respond. Under this condition, a large pulse amplitude would be required to produce a change in the circuit output. This means that a logic circuit can effectively tolerate a large noise amplitude if the noise is of a very short duration. This is referred to as ac noise margin and is substantially greater than the dc noise margin. It is generally supplied by the manufacturers in the form of a curve between noise margin and noise pulse width.

7.3.7 Operating Temperature

The temperature range in which an IC functions properly must be known. The accepted temperature ranges are: 0 to + 70 ^{0}C for consumer and industrial applications and --55 ^{0}C to + 125 ^{o}C for military purposes.

7.3.8 Power Supply Requirements

The supply voltage (s) and the amount of power required by an IC are important characteristics required to choose the proper power supply.

7.3.9 Flexibilities Available

Various flexibilities are available in different IC logic families and these must be considered while selecting a logic family for a particular job. Some of the flexibilities available are:

1. *The breadth of the series*: Type of different logic functions available in the series.
2. *Popularity of the series*: The cost of manufacturing depends upon the number of ICs manufactured. When a large number of ICs of one type are manufactured, the cost per function will be very small and it will be easily available because of multiple sources.
3. *Wired-logic capability:* the outputs can be connected together to perform additional logic without any extra hardware.
4. *Availability of complement outputs*: This eliminates the need for additional inverters.
5. *Type of output*: Passive pull-up, active pull-up, open-collector/drain, and tristate. These will be explained in subsequent sections.

7.4. TRANSISTOR-TRANSISTOR LOGIC (TTL)

Fig. 7.4 shows a TTL NAND gate with a totem pole output. The totem pole output means that transistor T_4 sits atop T_3 so as to give low output impedance. The low output impedance implies a short time constant RC so that the output can change quickly from one state to another. T_1 is a multiple emitter transistor. This transistor can be thought of as a combination of many transistors with a common base and collector. Multiple emitter transistors with about 60 emitters have been developed. In Fig. 7.4, T_1 has 3 emitters so that there can be three inputs A, B, C. The transistor T_2 acts as a phase splitter because the emitter voltage is out of phase with the collector voltage. The transistors T_3 and T_4 form the totem pole output. The capacitance C_L represents the stray capacitance etc. The diode D is added to ensure that T_4 is cut off when output is low. The voltage drop of diode D keeps the base- emitter junction of T_4 reverse biased so that only T_3 conducts when output is low. The operation can be summed up as under:

Condition I : At least one input is low (i.e.,0). Transistor T_1 saturates. Therefore, the base voltage of T_2 is almost zero. T_2 is cut off and forces T_3 to cut off. T_4 acts like an emitter follower and couples a high voltage to load. Output is high (i.e., Y = 1).

Condition II : All inputs are high. The emitter base junctions of T_1 are reverse biased. Thecollector base junction of T_1 is forward biased. Thus, T_1 is in reverse active mode.The collector current of T_1 flows in reverse direction. Since this current is flowing into the base of T_2, the transistors T_2 and T_3 saturate and output Y is low.

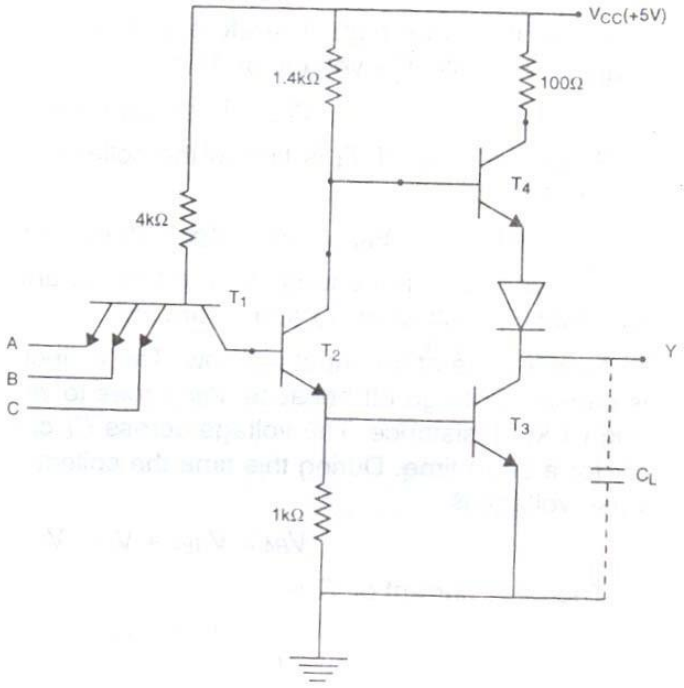


Fig. 7.4. TTL NAND gate with totem pole output

Condition II : The circuit is operating under condition II when one of the input becomes low. The corresponding emitter base junction of T_1 starts conducting and its base voltage drops to a low value. Therefore, T_1 is in forward active mode. The high collector current of T_1 removes the stored charge inT_2 and T_3 and therefore, T_2 and T_3 go cutoff and T_1 saturates and output Y returns to high.

TTL is the most popular logic in the SSI and MSI category. Table 7.2 summarises the input and output conditions.

Table 7.2. Three input NAND gate

A	B	C	$Y = \overline{ABC}$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Two important characteristics of a digital IC are power dissipation per gate and propagation delay time. TTL 5400/7400 series is the most popular and commonly used. The power dissipation of TTL series varies from 1 to 20mW per gate. The propagation delay time is the time taken by the output to change from one state to the other. Thus, this determines the speed of operation. For the different ICs in TTL series, the propagation delay time varies from about 1.5 ns to 10 ns.

An improved version of TTL logic uses Schottky transistors and is designated as 74 S series. In this series, bipolar junction transistors have been replaced by Schottky transistors. In this series, the propagation delay time is reduced by a factor of 3 but the power dissipation is double than that in 5400/7400 series.

7.4.1. Active Pull Up

The capacitance C_L is the stray capacitance, capacitances of driven states and output capacitance of transistor T_3. This capacitance limits the speed of operation of the gate. The totem pole output has the advantage of low output impedance and hence short time constant. This is known as active pull up. The action is an under.

Let all inputs be high. Therefore, output Y is low (about 0.2 V). In this state both T_2 and T_3 are saturated. The collector voltage of T_2 is

$$V_{C2} = V_{CE2\ sat} + V_{BE3} = 0.7 + 0.2 = 0.9\ V$$

Since the base of T_4 is tied to the collector of T_2, $V_{B2} = 0.9$ V. The voltage from base of T_4 to collector of T_3 is

$$V_{BE2} + V_D = V_{C2} - V_{CE3} = 0.9 - 0.2 = 0.7 \text{ V}$$

This voltage is not enough to turn both T_4 and diode D on. Diode D provides the offset voltage to ensure that T_4 is off when T_2 and T_3 are on.

Now let one of the input go low. The collector current of T_1 becomes zero and T_2 becomes off. This causes T_3 to go off because V_{BE3} goes to zero after the charge stored in base of T_3 is removed through 1 KΩ resistance. The voltage across C_L cannot change instantaneously* and remains close to 0.2 V for a short time. During this time the collector voltage of T_2 rises and T_4 turns on. When T_4 is on, its base voltage is

$$V_{B4} = V_{BE4} + V_D + V_0 = 0.7 + 0.7 + 0.2 = 1.6 \text{ V}$$

The base current of T_4 is

$$I_{B4} = \frac{V_{CC} - V_{B4}}{1.4 \times 10^3} = \frac{5 - 1.6}{1.4 \times 10^3} = 2.43 \text{ mA}$$

The collector current of T_4 is

$$I_{C4} = \frac{V_{CC} - V_{CE4,\ sat} - V_D - V_0}{100} = \frac{5 - 0.2 - 0.7 - 0.2}{100} = 39 \text{ mA}$$

The ratio $\frac{I_{C4}}{I_{B4}} = \frac{39}{2.43} = 16$. Since transistor β for T_4 is about 50, T_4 will be in saturation. T_4 supplies the current to charge C_L. Thus T_4 acts as a source. With T_4 in saturation, the output voltage rises exponentially with time constant equal to RC_L where R = (100 + saturation resistance of T_4 forward resistance of diode). Typical value of saturation resistance of T_4 is 10 Ω and forward resistance of diode less than 10 Ω. Therefore, time constant RC_L is very small about 7 times lesser than passive pull up (when T_4 and diode are not used). In high speed TTL the resistance of 100 Ω is replaced by 50 Ω so that time constantreduces further by a factor of 2. As output Y reaches a steady state when T_4 is at the edge of saturation. Then output voltage Y is

$$Y = V_{CC} - 1.4 \times 10^3 \, I_{B4} - V_{BE4} - V_D = 5 - 0 \ \ 0.7 - 0.7 = 3.6 \text{ V}$$

The voltage drop $1.4 \times 10^3 \times I_{B4}$ is negligible.

7.4.2. Wired AND Connection

* A basic characteristic of a capacitance is that voltage across it cannot change instantaneously.

TTL gates with active pull up (as in Fig. 7.4) should not be used in wired AND connection. If one gate output is HIGH when the other gate output is LOW, the gate with HIGH output tends to dissipate a large amount of power.

7.3.3. Open Collector Connection

TTL gates without active pull up and with collector open and brought out can be connected for wired AND connection. Fig. 7.5 shows one such connection. In this case all the open collector terminals share one common pull up resistance R. The size of this resistance R depends on the number of open collector gates, required noise margin, fan out etc.

In any of the input (say A) of TTL gate is not used (and left unconnected or open) the corresponding emitter base junction of T_1 will not be forward biased and behave as if logical 1 is applied to this input. Therefore, the unused input terminal of any TTL gate should preferably be

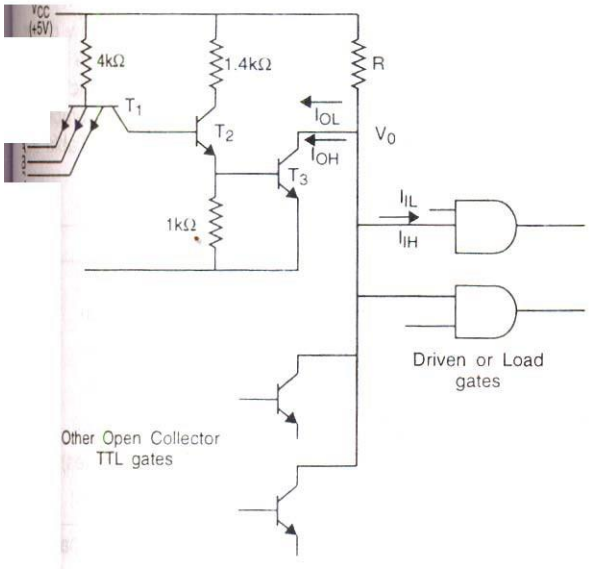


Fig. 7.5 Open collector TTL gate

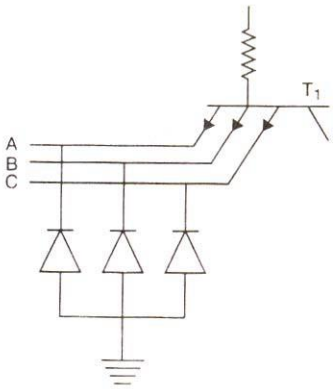


Fig. 7.6 Clamping diodes connected at inputs to TTL gate

TTL gates are used with clamping diodes connected between each input terminal and ground as shown in Fig. 7.6. These diodes clamp the input to – 0.7 V and minimize undesired negative noise transients.

Example 7.1. The circuit of Fig. 7.5 contains three 5401/7401 NAND gates with open collector outputs driving four 5400/7400 load gates. Find the value of pull up resistance R. Assume V_{CC} = 5 V, $V_{OH, min}$ = 2.4 V, $V_{OL, max}$ =0.4 V.

For each driving gate, I_{OH} = 0.25 mA, I_{OL} = 16 mA.

For each load gate, I_{IH} =0.04 mA, I_{IL} = -- mA.

Solution : When output V_O is HIGH, there should be sufficient available load current for load gates and off condition current for driving gates. This condition gives the maximum value of R. When V_O is LOW, the current through R and currents from load gates should not cause the voltage V_O to cross V_{OL} , $_{max}$, even if only one driving gate is sinking all the currents. This condition gives minimum value of R.

Maximum of value of R : V_O is HIGH, all driving gates are off. The voltage drop across R should be less than

$$V_{R,\ max} = V_{CC} - V_{OH,\ min,} = 5—2.4 = 2.6 \text{ V}$$

The total current through R is sum of current I_{IH} of load gates and off currents I_{OH} of driving gates. This current is $4 \times 0.04 + 3 \times .25 = 0.91$ mA.

Therefore,
$$R_{max} = \frac{2.6}{0.91} = 3.956\ \text{k}\Omega$$

Minimum value of R : V_0 is LOW. All load gates are ON. The worst situation occurs when only one driving gate is ON. The current through R should be such that current I_{OH} flowing through OH driving gate.

$$V_{R,\ min} = V_{CC} \text{ -- } V_{OL,\ max} = 5 - 0.4 = 4.6 \text{ V}$$

$$R_{min} = \frac{4.6}{16 + 4(-1.6)} = 0.479\ \text{k}\Omega$$

Hence minimum value of R is 479 ohm and maximum value is 3956 ohm.

7.5 CMOS LOGIC

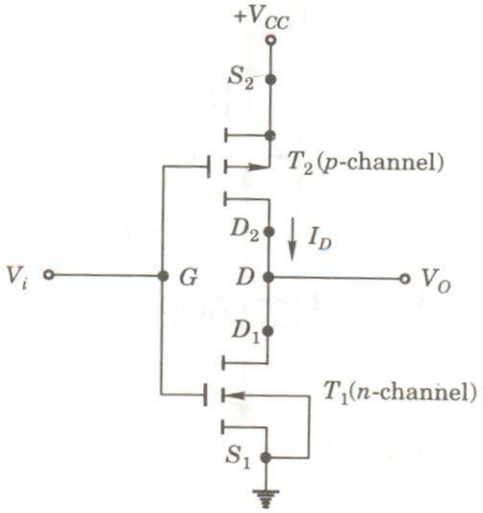
A complementary MOSFET (CMOS) is obtained by connecting a p-channel and an

n-channel MOSFET is series, with drains tied together and the output is taken at the common

drain. Input is applied at the common gate formed by connecting the two gates together. (Fig. 7.7). In a CMOS, p-channel and n-channel enhancement MOS devices are fabricated on the same chip, which makes its fabrication more complicated and reduces the packing density. But because of negligibly small power consumption, CMOS is ideally suited for battery operated systems.

Its speed is limited by substrate capacitances. To reduce the effect of these substrate capacitances, the latest technology known as silicon on sapphire (SOS) is used in microprocessor fabrication which employs an insulating substrate (sapphire). CMOS is becoming very popular in MSI and LSI areas.

Fig. 7.7 CMOS Inverter



7.5.1 CMOS Inverter

The basic CMOS logic circuit is an inverter shown in Fig. 7.7. For this circuit the logic levels are 0 V (logic 0) and V_{CC} (logic 1). When $V_i = V_{CC,}$ T_1 turns On and T_2 turns OFF. Therefore $V_0 = 0$ V, and since the transistors are connected in series the current I_D is very small. On the other hand, when $V_i = 0$ V, T_1 turns OFF and T_2 turns ON giving an output voltage $V_0 = V_{CC}$

and I_D is again very small. In either logic state, T_1 or T_2 is OFF and the quiescent power dissipation which is the product of the OFF leakage current and V_{CC} is very low. More complex functions can be realized by combinations of inverters.

7.5.2 CMOS NAND and NOR Gates

A 2-input CMOS NAND gate is shown in Fig. 7.8 and NOR gate in Fig. 7.9. In the NAND gate, the NMOS drivers are connected in series, where as the PMOS loads are connected in parallel. On the other hand, the CMOS NOR gate is obtained by connecting the NMOS drivers in parallel and PMOS loads in series. The operation of NAND gate can be understood from Table 7.3. The operation of the NOR gate can be verified in the similar manner.

Table 7.3 Operation of CMOS NAND gate

Inputs		State of MOS devices				Output
A	B	T_1	T_2	T_3	T_4	Y
0	0	OFF	OFF	ON	ON	V_{CC}
0	V_{CC}	ON	OFF	ON	OFF	V_{CC}
V_{CC}	0	OFF	ON	OFF	OF	V_{CC}
V_{CC}	V_{CC}	ON	ON	OFF	OFF	0

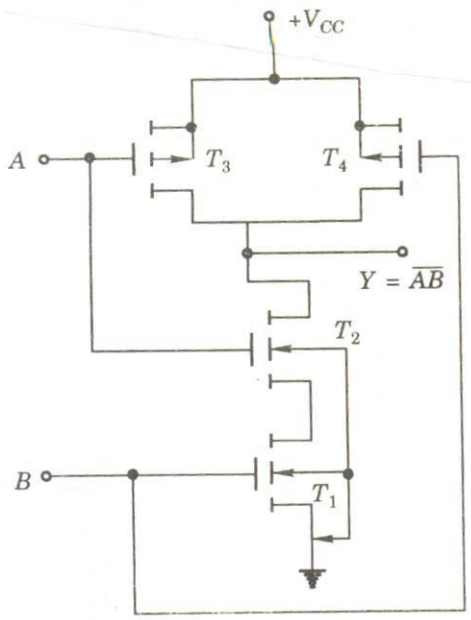


Fig. 7.8 A 2-input CMOS NAND gate.

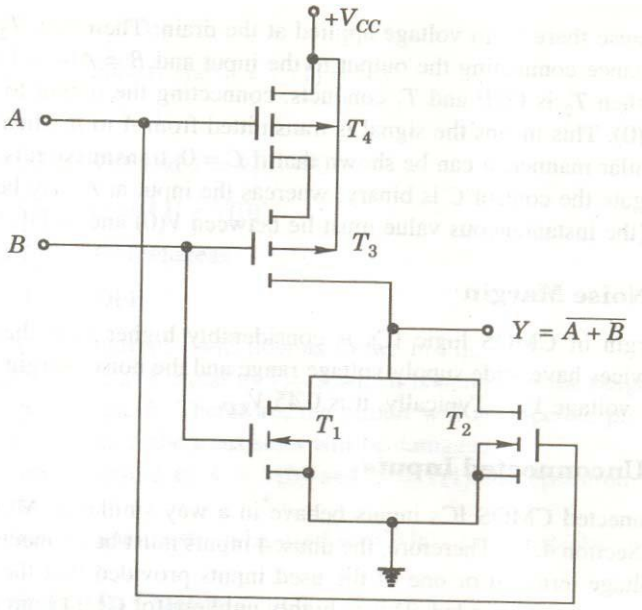


Fig. 7.9 A 2-input CMOS NOR gate.

7.5.3. CMOS Transmission Gate

A CMOS transmission gate controlled by gate voltages C and $\overline{C}$ is shown in Fig. 7.10. Assume C – 1. If A = V(1), then T_1 is OFF and T_2 conducts in the ohmic

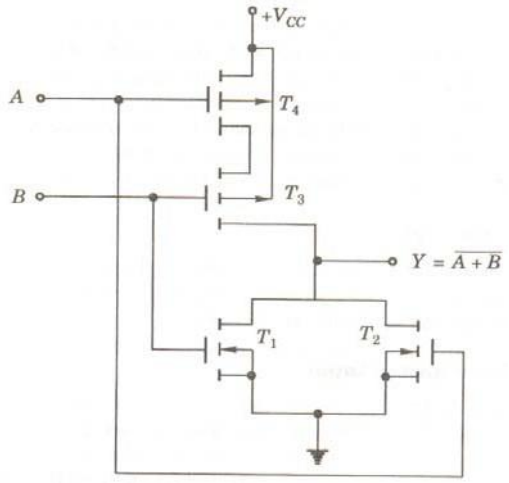


Fig. 7.10 (a) A CMOS transmission gate.

region because there is no voltage applied at the drain. Therefore, T_2 behaves as a small resistance connecting the output to the input and B = A = V(1). Similarly, if A = V(0), then T_2 is OFF and T_1 conducts, connecting the output to the input and B = A = V(0). This means the signal is transmitted from A to B when C = 1.

In a similar manner, it can be shown that if C = 0, transmission is not possible.

In this gate the control C is binary, whereas the input at A may be either digital or analog [the instantaneous value must lie between V(0) and V(1)].

7.5.4 Noise Margin

Noise margin of CMOS logic ICs is considerably higher than that of TTL ICs. CMOS devices have wide supply voltage range and the noise margin increases with the supply voltage V_{CC} Typically, it is 0.45 $V_{CC.}$

7.5.5 Unconnected Inputs

The unconnected CMOS ICs inputs behave in a way similar to MOS devices. Therefore, the unused inputs must be connected to either the supply voltage terminal or one of the used inputs provided that the fan-out of the signal source is not exceeded. This is highly unlikely for CMOS circuits because of their high fan-out.

Some CMOS ICs have Zener diodes connected at the inputs for protection against high input voltages.

7.5.6 Wired-Logic

Figure 7.11 shows two CMOS inverters with their outputs connected together. In this circuit,

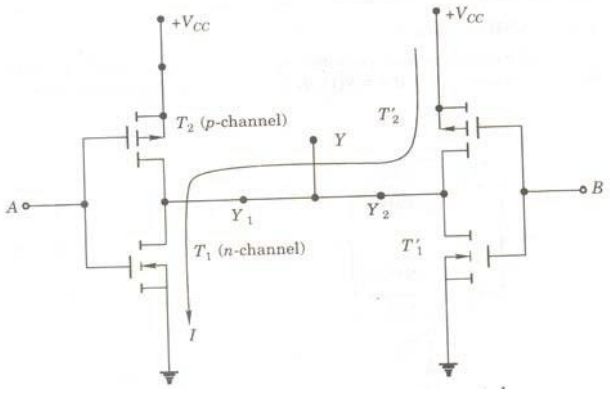


Fig. 7.11 CMOS inverters with outputs connected.

(i) When A = B = V(0)

T_1 and T'_1 are cut-off and Y = V (1) = V_{CC}

(ii) When A = B = V(1)

T_1 and T'_1 are ON and Y = V(0) = 0

(iii) When A =V(1) and B = V(0)

T_1 and T'_2 are ON whereas

T'_1 and T_2 are OFF

Therefore, a large current I will flow as shown in Fig. 7.11.

This will make voltage at Y equal to $V_{CC}/2$ which is neither in the range of logic 0 nor in the range of logic 1. Therefore, the circuit will not operate properly. Also because of large current I, the transistors will be damaged.

Similarly, corresponding to A = V(0) and B = V(1) the operation will not be proper.

Therefore, wired-logic must not be used for CMOS logic circuits.

7.5.7 Open-Drain Outputs

CMOS gates with open-drain output are available which useful for wired-AND operation. In this the drain terminal of the output transistor (n-channel) is available outside and the load resistor is to be connected externally since p-channel load does not exist.

7.5.8 54C00/74C00 CMOS Series

There are two commonly used CMOS series ICs. These are the 4000 series and 54C/74C series. 54C/74C CMOS series is pin-for-pin, function-for-function equivalent to the 54/74 TTL family and has, therefore, become very popular. The temperature range for 54 series is - 55 ^{o}C to + 125 ^{o}C and for 74C series is – 40 ^{o}C to 85 ^{o}C. It has a wide supply voltage range, 3 V to 15 V. A person can take full advantage of his knowledge of the 54?74 TTL series for the effective use of 54C/74C series.

There have been significant improvements in 54C/74C series. The 74HC/74HTC have higher speed and better current capabilities. 74HC is known as *high-speed* CMOS and 74HCTisknown as *high-speed*, TTL *compatible* CMOS series. 74AC/74ACT are very fast and have very high current sinking capabilities. There are known as *advanced* CMOS and *advanced, TTL compatible* CMOS, respectively. The 74 HC/74HCT/74AC/74ACT series can be operated at supply voltages in the range of 2—6 volts.

The voltage and current parameters of various 74 CMOS series with 5V supply voltage are given in Table 4.9. From the table, we observe that the output currents and voltages for 74HC/74HCT/75AC/74ACT are different when gates of these series are driving CMOS circuits and TTL circuits. 74 HCT and 74 ACT series are compatible with TTL series for input as well as output and therefore, can easily be used along with TTL ICs for optimum system design from the point of view of speed, power dissipation noise margins, cost, etc.

The fan-out of 74 HC/74HCT series is 20, whereas for 74AC/74ACT series it is 50 while driving these CMOS series. The fan-out of these gates while driving various TTL series gates can be determined using the specifications of TTL (Table 4.3) and CMOS (Table 7.4).

Table 7.4 Specifications of CMOS IC families

Parameter	Load	74C	74HC	74HCT	74AC	74ACT	Units
V_{IH}		3.5	3.85	2.0	3.85	2.0	volts
V_{IL}		1.5	1.35	0.8	135	0.8	volts
V_{OH}	CMOS	4.5	4.4	4.4	4.4	4.4	volts

	TTL		3.84	3.84	3.76	3.76	volts
V_{OL}	CMOS	0.5	0.1	0.1	0.1	0.1	volt
	TTL		0.33	0.33	0.37	0.37	volt
			1	1	1	1	μA
I_{IH}		1	–1	–1	–1	–1	μA
I_{IL}		–1	–0.02	–0.02	–0.05	–0.05	mA
I_{OH}	CMOS	–0.1	–4.0	–4.0	–24.0	–24.0	mA
	TTL		0.02	0.02	0.05	0.05	mA
	CMOS	0.36	4.0	4.0	24.0	24.0	mA
I_{OL}	TTL						

7.6 TRI-STATE LOGIC

In normal logic circuits there are two states of the output, LOW and HIGH. If the output isnot in the LOW state, it is definitely in the other state (HIGH). Similarly, if the output isnot inthe HIGH state, it is definitely in the LOW state. In complex digital systems like microcomputers and microprocessors, a number of gate outputs may be required to be connected to a common line which is referred to as a bus which, in turn, may be required to drive a number of gate inputs. When a number of gate outputs are connected to the bus, we encounter some difficulties. These are:

1. Totem-pole outputs cannot be connected together because of very large current drain from the supply and consequent heating of the ICs which may get damaged.
2. Open-collector outputs can connected together with a common collector-resistor connected externally. This causes the problems of loading and speed of operation.

To overcome these difficulties, special circuits have been developed in which there is one more state of the output, referred to as the *third state* or *high-impedance state,* in addition to the LOW and HIGH states. These circuits are known as TRI-STATE, *tri-state logic* (TSL) or *three-state logic.* TRI-STATE, is a registered Trade Mark of National Semiconductor Corporation of USA.

There is a basic functional difference between wired-OR and the TSL. For the wired-OR connection of two functions Y_1 and Y_2 is

$$Y = Y_1 + Y_2$$

whereas for TSL, the result is not a Boolean function but an ability to multiplex many functions economically.

SUMMARY

1. The different logic functions are available in IC form.
2. Digital ICs are categorized as SI, MSI LSI, VLSI and ULSI depending on the number of gates and components on a single clip.
3. The most popular technologies being used now-a-days are TTL, ECL and CMOS. In each technology a number of different series having different parameters are available.

4. Specifications of digital ICs include input and output voltage and current levels (for high and low conditions), power dissipation, propagation delay, noise margin, operating temperature, power supply requirements, fan in, fan out etc.
5. Voltage and current characteristics are important factors when interfacing one logic family with another or when interfacing logic family with other devices.
6. The fan out and fan in are determined by the output drive and input loading conditions.
7. CMOS family has higher noise margin than TTL family.
8. CMOS has slow speed but very small physical size and simple geometry. Its power dissipation is very low. It is widely used in electronic wrist watches, calculators, portable computers etc.
9. Special ICs for interfacing TTL-CMOS, CMOS-TTL etc., are available.
10. When a logic gate is interfaced with devices like relays, motors, solenoids etc., it is desirable to have separate power supplies for the gate and these devices.
11. Tri state logic refers to the conditions that output can exist in three states, i.e., Low, High and High impedance states. This logic has another input called control input.
12. Open input and open output are the most common faults on ICs. These can be detected by a logic pulser and logic probe.

PROBLEMS

7.1 What are the different logic functions available in the IC's ?

7.2 What is tri state logic? What is its significance?

7.3 Why CMOS devices are more suitable for IC as compared to transistors?

7.4 Enumerate the different parameters and characteristics of the logic families.

7.5 Categorize the different IC's.

7.6 How we can calculate the FAN OUT and FAN IN of the different families?

7.7 List the advantages of the TTL families.

7.8 Explain the working of TTL NAND gate.

7.9 What is the significance of the totem pole configuration?

7.10 Why do we require different logic families?