Srinivas Institute of Management Studies

Pandeswar, Mangalore – 575 001

BACKGROUND STUDY MATERIAL

Computer Organization and Architecture

BCA I Semester

Compiled by

Mr. Shylesh S

SIMS, Mangalore

2015-2016

UNIT - I

Contents

Page No

UNIT – I

Chapter1. Applications and Structure of Digital Computers

The widespread acceptance of digital computers in payroll offices is largely due to the repetitious type of work normally done there. Mechanization of such tasks is straightforward, although often complicated; but the additional accuracy and speed, as well as the lower operating costs, which electronic business machines make possible, has made their use especially popular in this field

1.1 <u>BUSINESS APPLICATIONS</u>

The main difference between the use of digital machines in business and in scientific work lies in the ratio of operations performed to total data processed. While the business machine performs only a few calculations using each datum, a great volume of data must be processed. The scientific problem generally starts with fewer data, but a great many calculations are performed using each datum. Both types of machines still fall under the heading of digital computers, and either type of work may be done on all computers, although some machines may be better adapted to one or the other type of problem.

The description of the use of a computer in figuring payrolls is an example of a business application and illustrates the similarity in programming the operation of a computer and figuring out employee office procedures. First, the problem to be solved is reduced to a series of simple operations: finding the name of the next employee whose wages are to be computed figuring how many hours he or she has worked, and multiplying this figure by the hourly rate of pay. After the procedure to be used has been worked out and explained to the clerk, the clerk is provided with the necessary numerical information, such as pay rates, insurance rates, etc. If the operations are further simplified, each step in Fig. 1.1 may be performed by a different clerk. For instance, the first clerk may find the employee’s record and send it to the second clerk, who computes the total number of hours worked and presents it to the next clerk, who multiplies by the wage rate, and so on, until all the operations have been performed. Thus the breaking down of business procedures into basic steps is a very old practice indeed.

The procedure for preparing a list of instructions for a digital computer is basically the same. All the operations the computer is to perform are written in flowchart form (Fig. 1.1). Then the problem is broken down into a list of instructions to the computer which specify exactly how the solution is to be obtained. After the problem has been programmed, the list of instructions is read into the computer. The computer automatically performs the required steps. Notice that once the procedure has been established and the programmed steps have been read in, the programming is finished until a change in procedure is desired. Changes in rates, for instance, can be inserted by simply reading the new pay rates into the machine. This does not affect the procedure.

1.2 SCIENTIFIC APPLICATIONS

Modem science and engineering use mathematics as a language for expressing physical laws in precise terms. The electronic digital computer is a valuable tool for studying the consequences of these laws. Often the exact procedure for solving a problem has been found, but the time required to perform the necessary calculations manually is prohibitive. Sometimes it is necessary to solve the same problem many times with different sets of parameters, and the computer is especially useful for solving problems of this type. Not only is the computer able to evaluate types of mathematical expressions at high speeds; but also if a set of calculations is performed repeatedly on different sets of numerical values, the computer can compare the results and determine the optimum values that were used.

1.3 Computers in Control Systems

The ability of digital computers to make precise calculations and decisions at high speeds has made it possible to use them as parts of control systems. The Air Traffic Control System used at airports is an example of computer usage in a control system. In this system, data from a network of radar stations, which are used to detect the positions of all aircraft in the area, are fed via communication links into a high-speed computer. The computer stores all the incoming positional information from the radar stations and from this calculates the future positions of the aircraft, their speed and altitude, and all other pertinent information. A number of other types of information are also relayed into the computer, including information from picket ships, AEW aircraft, Ground Observer Corps aircraft spotters, flight plans for both military and civilian aircraft, and weather information.

A computer receives all this information and from it calculates a composite picture of the complete air situation. The computer then generates displays on special oscilloscopes which are used by air traffic controllers. By means of radio links the computer automatically guides aircraft in and out of airports.

Other examples of real-time control applications include oil refineries and other manufacturing areas which use the computer to control the manufacturing processes automatically. Digital computers are used to guide machine tools which perform precision-machining operations automatically

The basic elements of a control system using a digital computer consist of (_e data-gathering devices which perform measurements on the external environment and, if necessary, also perform analog-to-digital conversion on the data from the system to be controlled. The computer itself performs calculations on the data supplied and makes the necessary decisions; and the means of communication with, or control over, certain of the elements in the external environment.

Because of their high speed, computers, often operating unattended, can measure, test, analyze, and control manufacturing functions as they occur. Computers can handle shop floor control, quality control testing, materials handling, and production monitoring.

1.4 Basic Components of a Digital Computer

The block diagram in Fig. 1.1 illustrates the five major operational divisions of an electronic digital computer. Although presently available machines vary greatly in the construction details of various component, the overall system concepts remain roughly the same.

A digital computer may be divided into the following fundamental units:

1. ***Input:*** The input devices read the necessary data into the machine. In most general-purpose computers, the instructions that constitute the program must be read into the machine. along with all the data to be used in the computations. Some of the more common input devices are keyboards, punched-card and punched paper-tape readers, magnetic-tap_ readers, and various manual input devices such as toggle switches and pushbuttons.

2.*Control:* The control section of a computer sequences the operation of the computer, controlling the actions of all other units. The control circuitry interprets

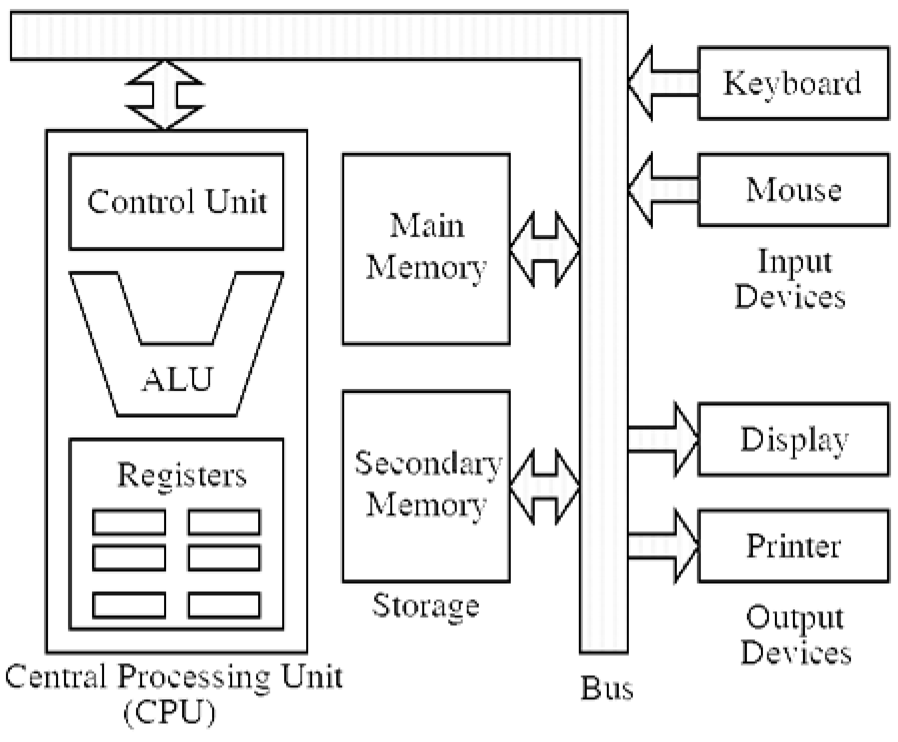


Fig 1.1 Basic components of a digital computer

3.*Memory:* The memory, or storage, section of the computer consists of the devices used to store the information that will be used during the computations. The memory section of the computer is also used to hold both intermediate and final results as the computer proceeds through the program. Memory devices are constructed so that it is possible for the control unit to obtain any information in the memory. The time required to obtain information may vary somewhat, however, and is determined by the type of device used to store the information. Common storage devices are integrated-circuit memories, magnetic tape, and magnetic disks.

4.*Arithmetic-logicunit:*The arithmetic-logic units of most computers are capable of performing addition, subtraction, division, and multiplication as well as some "logical operations" which are described later. The control unit tells the arithmetic-logic unit which operation to perform and then sees that the necessary numbers are supplied. The arithmetic element can be compared to the calculating machines described previously in that the numbers to be used are inserted, and it is then directed to perform the necessary operations.

5.*Output :* The output devices are used to record the results obtained by the computer and present them to the outside world. Most output devices are directed by the control element, which also causes the necessary information to be supplied to them. Common output devices are CRT displays, printers, card-punching machines, and magnetic-tape drives. There are also many other types of output devices, such as lights, buzzers, and loudspeakers.

Chapter 2. Number Systems and Codes

2.1 Introduction

Number system is a basis for counting various items. On hearing the word 'number', all of us immediately think of the familiar decimal number system with its 10 digits: 0,1, 2, 3, 4, 5, 6, 7, 8 and 9.

Modern computers communicate and operate with binary numbers which use only the digits 0 and 1. Let us consider decimal number 18. This number is represented in binary as 10010. In the example, if decimal number is considered, we require only two digits to represent the number, whereas if binary number is considered we require five digits. Therefore we can say that, when decimal quantities are represented in the binary form, they take more digits. This fact gave rise to three new number systems: Octal, Hexadecimal and Binary Coded Decimal (BCD). These number systems represent binary number in a compressed form. Therefore, these number systems are now widely used to compress long strings of binary numbers.

In this chapter, we discuss binary, octal, hexadecimal, and BCD number systems, and we will see how to convert from decimal to binary, octal and hexadecimal, and vice versa. In the later section of this chapter we are going to see binary arithmetic such as binary addition, subtraction, multiplication and division.

2.2 Decimal Number System

Before considering any number system, let us consider familiar decimal number system. In decimal number system we can express any decimal number in units, tens, hundreds, thousands and so on. When we write a decimal number say, 5678.9, we know it can be represented as
5000 + 600 + 70 + 8 + 0.9 = 5678.9

In power of 10, we can write as

$5 \times 10^3 + 6 \times 10^2 + 7 \times 10^1 + 8 \times 10^0 + 9 \times 10^{-1} = 5678.9$

This says that, the position of a digit with reference to the decimal point determines its value/weight. The sum of all the digits multiplied by their weights gives the total number, being represented. The leftmost digit, which has the greatest weight is called the most significant digit and the rightmost digit, which has the least weight, is called the least significant digit. Fig. 2.1 shows binary digits and its weights expressed as a power of 10.

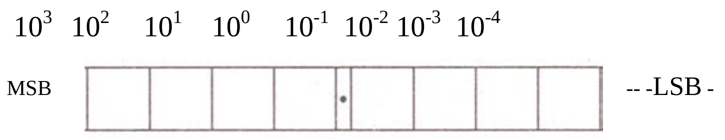


Fig. 2.1 Decimal position values as powers of 10

Ex 34 = 3xl0 + 4x1 = 30 + 4

The digit 3 has a weight of 10, as indicated by its position, and the digit 4 has a weight of 1, as indicated by its position.

Ex. 2.2 98.72 = 9x10+8x1+7x0.1+2x 0.01

The digit 9 has a weight of 10, the digit 8 has a weight of 1, the-digit 7 has a weight of 1/10 and the digit 2 has a weight of 1/100.

2.3 Binary Number System

We know that decimal system with its ten digits is a base-ten system. Similarly, binary system with its two digits is a base-two system. The two binary digits (bits) are 1 and 0. Like digital system, in binary system each binary digit commonly known as bit, has its own value or weight. However in binary system weight is expressed as a power of 2, as shown ir_ Fig. 2.2.

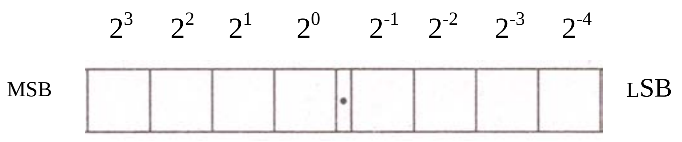


Fig. 2.2 Decimal position values as powers of 2

2.3.1 Binary Codes and Binary Logic

We know that in decimal system we start at 0 and count upto 9 before we run out of digits. We then take another digit to the left and continue counting 10 through 99. We can carryon counting 100 through 999 by taking one more digit to the left. In the same fashion we can count binary numbers. But in binary numbers we have only two digits. We begin counting, 0,1. At this point we have used both digits, so we take another digit position and continue. Table 2.1 shows the binary numbers from' 0000 to 1111 and its decimal equivalents.

Binary	Decimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7

1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

Table 2.1 Binary numbers

2.3.2 Number Base Conversion

2.3.2.1 Binary-to-Decimal Conversion

Integers

Just as in a decimal number each digit position has a weight, so also, each digit position in a binary number has a value or weight. In a binary number, the Least Significant Digit (the right most one) has a weight of 1. The second position from the right has a weight of 2, the next weight is 4, then 8, 16, 32, and so on. Diagrammatically, we can represent the weights of the digit positions of a binary number as follows:

32	16	8	4	2	1
2^5	2^4	2^3	2^2	2^1	2^0

It may be noted that the weights of the digit positions of a binary number also be expressed in ascending powers of 2.

Examples of binary-to-decimal conversion

(a) Binary Number 1 0 0

Positional Weight 4 2 1

Decimal equivalent 1 x 4 + 0 x 2 + 0 = 4 + 0 + 0 = 4

(b) Binary Number 1 0 1 0

Positional Weight 8 4 2 1

Decimal equivalent 1 x 8 + 0 x 4 +1 x 2 + 0 x 1 = 8 + 0 + 2 + 0 = 10

2.3.2.2 Fractions

In this case, the weights of digit positions to the right of the binary point are given by 1/2,1/4, 1/8 , 1/16, and so on. Diagrammatically

1/2	1/4	1/8	1/16
2^{-1}	2^{-2}	2^{-3}	2^{-4}

$\uparrow$ Binary Point

Therefore, the binary fraction 0.101 has a decimal equivalent of

Binary Fraction	0	.	1	0	1	
Positional Weight			1/2	1/4	1/8	
Decimal equivalent			0.5 +	0 +	0.125	= 0.5 + 0.125 = 0.625

As another example, the decimal equivalent of 0.1001 is

Binary Fraction	0	.	1	0	0	1	
Positional Weight			1/2	1/4	1/8	1/16	
Decimal equivalent			0.5 +	0 +	0 +	0.0625	= 0.5625

2.3.4 Decimal to Binary Conversion

Converting a decimal number to its binary equivalent is exact reverse the process described in the previous section, In the conversion process we have to first represent the number in the sum of power of 2, and then it is necessary to write Is and 0s in the appropriate digit position.

There are two ways of converting from a decimal number to a binary number. These are:

- Sum-of Weights Method
- Repeated Division by 2 Method

2.3.4.1 Sum-of-Weights Method

One way to find the binary number equivalent to a given decimal number is to determine the set of binary weights values whose sum is equal to the decimal number. This is illustrated in the examples 2.7, 2.8.

Ex. 2.7 : Convert decimal 10 to its binary equivalent.

Sol. :10 = 8 + 4 + 2 + 1 (Binary weights)

= 1 0 1 0 (Binary number) = $1010_{(2)}$

Note: The subscript 2 identifies this as a binary number.

Ex. 2.8 : Convert following decimal numbers to binary (a) 23 (b)37 (c) 94

(a)$23_{10} = 16 + \cancel{8} + 4 + 2 + 1 \rightarrow 10111_2$

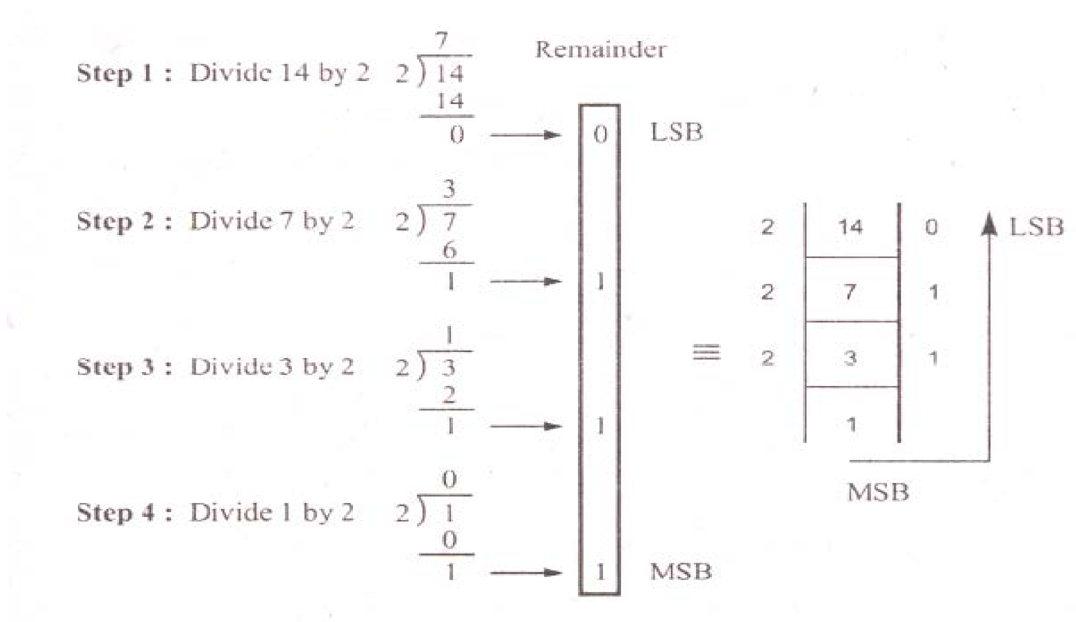
(b) $37_{10} = 32 + \cancel{16} + \cancel{8} + 4 + \cancel{2} + 1 \rightarrow 100101_2$

(c)$94_{10} = 64 + \cancel{32} + 16 + 8 + 4 + 2 + \cancel{1} \rightarrow 1011110_2$

2.3.4.2 Repeated Division by 2 or Double Dabble Method

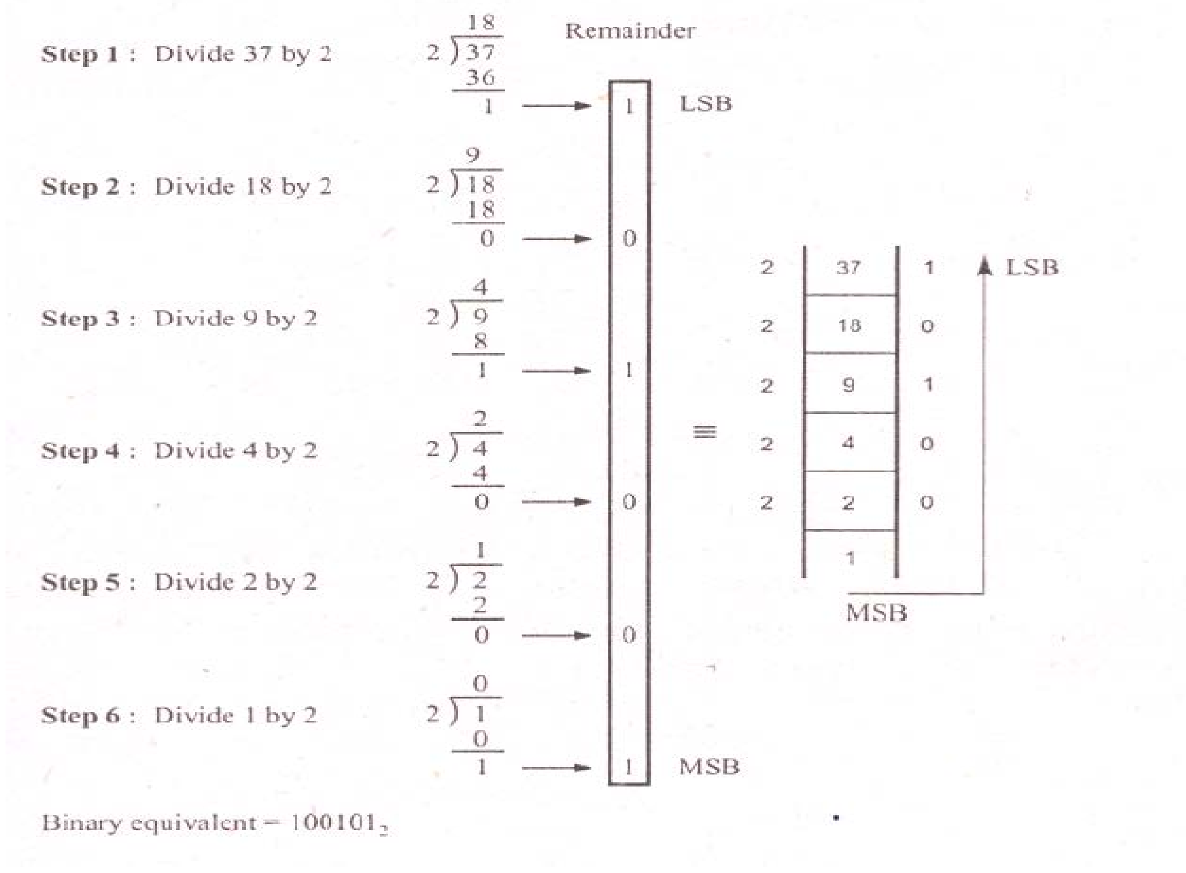
In this method we progressively divide the decimal number by 2 until quotient is zero and writing down the remainder after each division. The remainders are taken in the reverse order to form the binary number. This means that the first remainder is the least significant bit (LSB) and the last remainder is the most significant bit (MSB). This is illustrated in the examples 2.9 and 2.10.

Ex. 2.9 : Convert decimal number 14 to its binary equivalent.



Binary equivalent = 1110_2

Ex.: Convert decimal number 37 to its binary equivalent.



Review Questions

1. *Convert* 6710 *to binary using both methods.*

2. *Convert* 81310 *to binary using both methods and check your answer by converting back to decimal.*

2.3.5 Fractional Decimal to Binary Conversion

In case of fractional decimal numbers, numbers are multiplied by 2 and carries are 1 recorded from the integer position. These carries when read downward represent binary equivalent to fractional decimal number. This means that the first carry produced is the MSB and the last carry produced is the LSB. This procedure is repeated until the required no. of decimal points is obtained or the numbers start repeating or a there are only zeros left after the decimal point.

Ex: Convert 0.8125 decimal number to its binary equivalent.

Fraction		Base		Result		Recorded Carries	
0.8125	X	2	=	1.625 = 0.625	with a carry of	1	MSB
0.625	X	2	=	1.25 = 0.25	with a carry of	1	
0. 25	X	2	=	0.5 = 0.5	with a carry of	1	
0.5	X	2	=	1.0 = 0.0	with a carry of	1	LSB

Reading carries downward we get,

Binary fraction = 0.1101, which is equivalent to 0.8125 decimal.

Ex: Convert 0.95 decimal number to its binary equivalent

In this case, 0.8 is repeated and if we multiply further, we will get repeated sequence. If we stop here, we get 7 binary digits,. 1111001. This answer is an approximate answer. To get more accurate answer we have to continue multiplying by 2 until we have as many digits as necessary for our application.

Fraction		Base		Result		Recorded Carries	
0.95	X	2	=	1.9 = 0.625	with a carry of	1	MSB
0.9	X	2	=	1.8 = 0.25	with a carry of	1	
0. 8	X	2	=	1.6 = 0.6	with a carry of	1	
0.6	X	2	=	1.2 = 0.2	with a carry of	1	LSB

Reading carries downwards upto 4 decimal points we get,
Binary fraction = 0.111 which is equivalent to 0.95 decimal

Review Questions

1. *Convert each decimal number to binary.*

(a) 11.125 *(b) 0.625*

2. *Convert each decimal number to binary by using the repeated division by* 2 *method for integers and repeated multiplication by* 2 *for fractions.* .

(a) 578.3125 *(b) 3015.8125*

2.3.6 Binary Arithmetic

Computer circuits do not process decimal numbers; they process binary numbers. In this section, we are going to learn binary arithmetic. Binary addition is the key to binary subtraction, multiplication, and division. So, let us see rules for binary addition.

Rules for Binary Addition

A +B	Sum	Carry
0 + 0	0	0
0 + 1	1	0
1 + 0	1	0
1 + 1	0	l

Example: Add $(1\ 0\ 1\ 0)_2$ and $(0\ 0\ 1\ 1)_2$

```
Sol:   1 0 1 0     Decimal 10
     + 0 0 1 1     Decimal 3
     ---------     ----------
       1 1 0 1     Decimal 13
```

Example 2: Add 28 and 15 in binary

Sol. : First we find the binary equivalent of 28 and 15.

Addition of 28 and 15

```
1 1 1 0 0     Decimal 28
0 1 1 1 1     Decimal 15
---------     ----------
1 0 1 0 1 1   Decimal 43
```

In case of unsigned 8-bit binary numbers the decimal range is 0 to 255. For signed magnitude 8-bit binary numbers the largest magnitude is reduced from 255 to 127 because we need to represent both positive and negative numbers.

Maximum positive number	01111111 = +127
Maximum negative number	1111 1111 = -128

2.3.7 Complements

2.3.7.1 <u>1's complement Representation</u>

The l's complement of a binary number is the number that results when we change all ones to zeros and the zeros to ones.

```
. Sol.: 1 1 0 1  ←  Number
        0 0 1 0  ←  1's complement
```

Ex 2: Find l's complement of 1011 1001

```
Sol. :    1 0 1 1 1 0 0 1     number
          0 1 0  0 0 1 1 0     l's complement
```

2.3.7.2 <u>1's Complement Subtraction</u>

In 1's complement subtraction, the 1's complement of the subtrahend is added to the minuend. The last carry (called the end-around carry), if any, is then added to the partial result to get the final answer.

Whenever there is an end-around carry, the final answer is positive. But when there is no end-around carry, the partial result is in 1's complement form. The final answer is negative, obtained by taking the 1's complement of the partial result. For instance, when we subtract 1 1 0 1 from 1 0 1 0, we proceed as follows:

```
   1 0 1 0
 + 0 0 1 0     (Add the 1's complement of 1101)
 ---------
   1 1 0 0     (No end-around carry. This partial result is in 1's complement form)
 ---------
 └──►-0 0 1 1  (Final answer obtained by taking the 1's complement of 1100 with
                a minus sign prefixed.)
```

The use of the 1's complement to subtract binary numbers is quite popular in digital computers, because in such cases, only the adders are needed ; also it is easy with digital circuits to get the 1's complement of a binary number.

Let us now subtract 0 1 1 0 1 from 1 1 0 1 1. Using 1's complement method

```
   1 1 0 1 1
 + 1 0 0 1 0    (Add the 1's complement of the subtrahend, i.e. 01101))
 -----------
┌1 0 1 1 0 1    (Partial result)
└──────────►1   (Add the end-around carry)
 -----------
 + 0 1 1 1 0    (Final answer)
 -----------
```

Let us discuss some other examples.

(a) Subtract 11011 from 01101, using the 1's complement subtraction.

```
  0 1 1 0 1
+ 0 0 1 0 0   (Add the 1's complement of 11011)
-----------
  1 0 0 0 1
-----------
```

There is no end-around carry. Take the l's complement of the above result and prefix the negative sign, to get the final answer as

−01110

(b) Subtract 0 1 1 0 from 1 0 0 1, using the 1's complement subtraction.

```
   1 0 0 1
   1 0 0 1    (Add the 1's complement of 0110)
 ---------
 1 0 0 1 0    (Partial result)
 |
 └------→1    (Add the end-around carry)
 ---------
   0 0 1 1    (Final answer)
 ---------
```

2.3.7.2 <u>2's Complement Representation</u>

The 2's complement is the binary number that results when we add 1 to the 1's complement. It is given as

2's complement = l's complement + 1

The 2's complement form is used to represent negative numbers.

Ex: Find 2's complement of (1 0 0 1}$_2$

```
Sol : 1 0 0 1    Number

      0 1 1 0    1's complement
            1
     ---------
      0 1 1 1    2's complement
```

2.3.7.3 Subtraction using 2's complement

In the 2's complement subtraction, as in the 1's complement subtraction, the 2's complement of the subtrahend is added to the minuend, but the end-around carry, if generated , is disregarded. For instance, to subtract 101 from 111, we proceed as follows

$$\begin{array}{rl}
111 & \\
+\,011 & \text{(Add the 2's complement of 101)} \\
\hline
\boxed{1}\,010 & \text{(Discard the end-around carry)} \\
\hline
 & 010 \text{ (Final answer)}
\end{array}$$

As another example, subtract 1 0 1 0 from 1 1 0 1.

$$\begin{array}{rl}
1101 & \\
0110 & \text{(Add the 2's complement of 1010)} \\
\hline
\boxed{1}\,0011 & \text{(Discard the end-around carry)} \\
\hline
 & 0011 \text{ (Final answer)}
\end{array}$$

When there is no end-around carry, the answer is negative and in the 2's complement form. Therefore, we take the 2's complement of the answer and prefix the minus sign with the final answer.

Let us take the help of a few more examples to illustrate the 2's complemen subtraction.

(a) Subtract 1 0 1 1 from 1 1 1 1.

$$\begin{array}{lrl}
 & 1111 & \\
 & +\;\;0101 & \text{(Add the 2's complement of 1011)} \\
\cline{2-2}
 & \boxed{1}\,0100 & \text{(Discard the carry in the fifth bit position)} \\
\cline{2-2}
\therefore\ \textbf{Ans:} & 0100 &
\end{array}$$

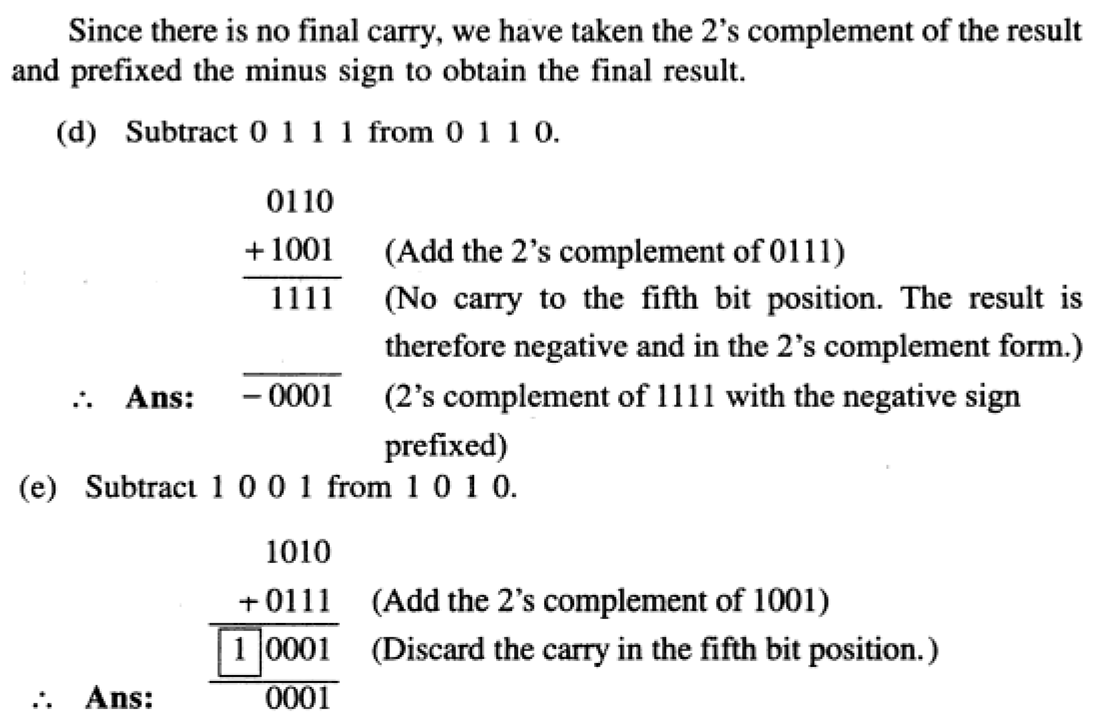
(b) Subtract 1 0 0 1 0 1 from 1 1 0 0 1 1.

$$\begin{array}{lrl}
 & 110011 & \\
 & +\,011011 & \text{(Add the 2's complement of 100101)} \\
\cline{2-2}
 & \boxed{1}\,001110 & \text{(Discard the carry in the seventh bit position)} \\
\cline{2-2}
\therefore\ \textbf{Ans:} & 001110 &
\end{array}$$

(c) Subtract 1 1 1 0 1 0 from 1 0 0 0 1 1.

(c) Subtract 1 1 1 0 1 0 from 1 0 0 0 1 1.

$$\begin{array}{lrl}
 & 100011 & \\
 & +\,000110 & \text{(Add the 2's complement of 111010)} \\
\cline{2-2}
 & 101001 & \text{(No carry to the seventh bit position. The result is} \\
 & & \text{therefore negative and in the 2's complement form.)} \\
\cline{2-2}
\therefore\ \textbf{Ans:} & -010111 & \text{(2's complement of 101001 with the negative} \\
 & & \text{sign prefixed)}
\end{array}$$



Since there is no final carry, we have taken the 2's complement of the result and prefixed the minus sign to obtain the final result.

(d) Subtract 0 1 1 1 from 0 1 1 0.

```
             0110
            +1001    (Add the 2's complement of 0111)
            -----
             1111    (No carry to the fifth bit position. The result is
                     therefore negative and in the 2's complement form.)
            -----
∴  Ans:     -0001    (2's complement of 1111 with the negative sign
                     prefixed)
```

(e) Subtract 1 0 0 1 from 1 0 1 0.

```
             1010
            +0111    (Add the 2's complement of 1001)
           ------
           [1]0001   (Discard the carry in the fifth bit position.)
           ------
∴  Ans:      0001
```

Review Questions

1. Determine the 1's and 2's complement of each number.
(a) 639 (b) 243 (c) 576
2. Determine the 1's and 2's complement of each binary number.
(a) 100_2 (b) 10000_2 (c) 111001_2
3. Perform the following subtraction using the 1's complement and 2' s complement method.
(a) 235 – 476 (b) 100111_2-100010_2

2.3.7.5 10's Complement Representation

To convert a number to its 10's complement form, you have to first check the total no. of digits in the given number. For example, consider a positive number 'N' with 'n' digits. Then you can find the 10's complement of the number with the formula 10^n– N.

Example: Find the 10's complement of 845

Solution: The no. of digits in 845 is 3

Therefore the 10's complement of 845 = 10^3– 845
= 1000 – 845
= 155

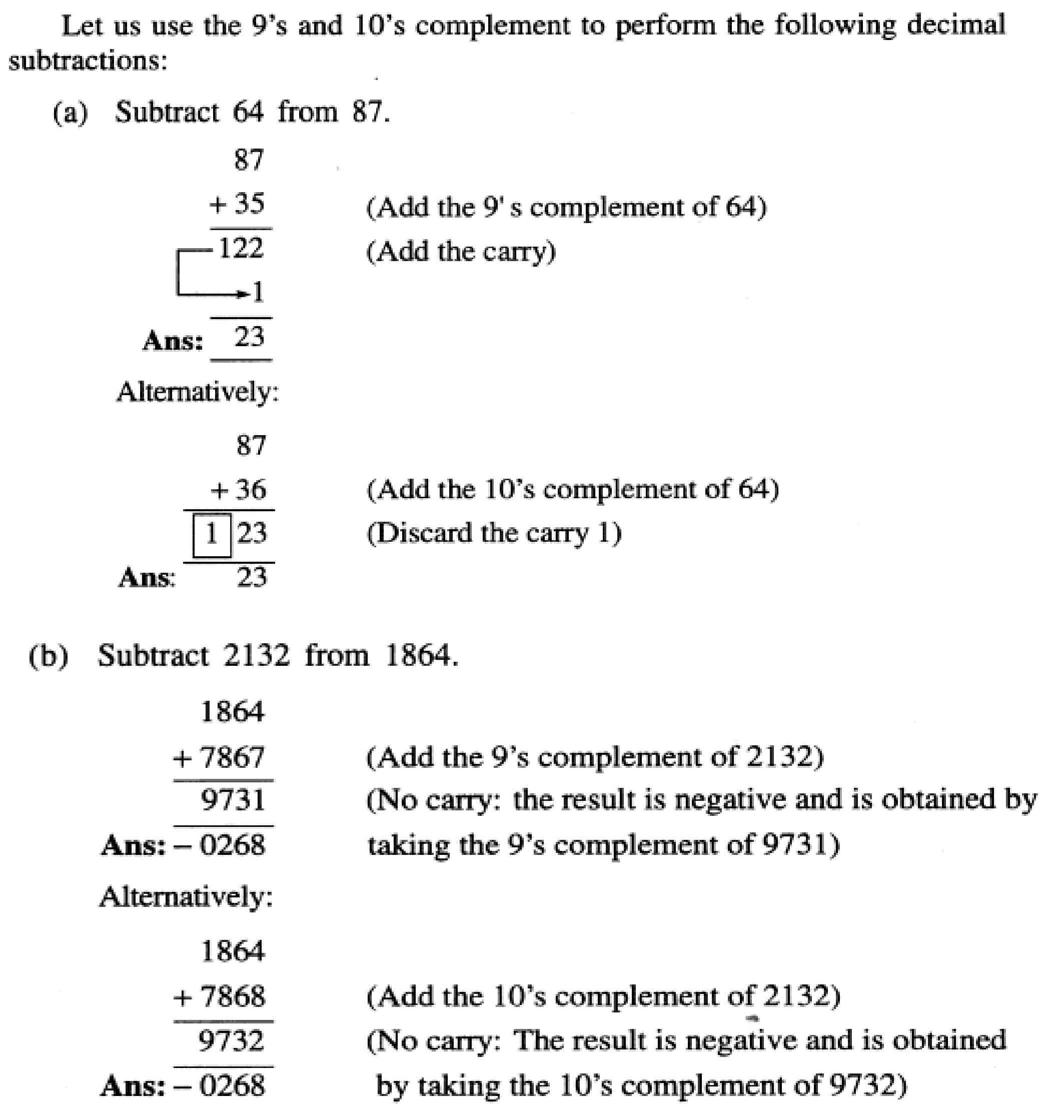
2.3.7.6 9's Complement Representation

To convert a number to its 9’s complement form, you have to first check the total no. of digits in the given number. For example, consider a positive number ‘N’ with ‘n’ digits. Then you can find the 9’s complement of the number with the formula $10^n – 1 – N$. That is, write ‘n’ no. of 9’s and then subtract the given number from it.

Example: Find the 10’s complement of 639

Solution: The no. of digits in 639 is 3
Therefore the 10’s complement of 639 = $10^3 – 1 – 639$
= 999 – 639
= 360

2.3.13 Subtraction using 9’s and 10’s complement



Let us use the 9’s and 10’s complement to perform the following decimal subtractions:

(a) Subtract 64 from 87.

$$\begin{array}{rl} 87 & \\ +35 & \text{(Add the 9' s complement of 64)} \\ \hline 122 & \text{(Add the carry)} \\ \to 1 & \\ \hline \textbf{Ans: } 23 & \\ \hline \end{array}$$

Alternatively:

$$\begin{array}{rl} 87 & \\ +36 & \text{(Add the 10's complement of 64)} \\ \hline \boxed{1}\,23 & \text{(Discard the carry 1)} \\ \hline \textbf{Ans: } 23 & \end{array}$$

(b) Subtract 2132 from 1864.

$$\begin{array}{rl} 1864 & \\ +7867 & \text{(Add the 9's complement of 2132)} \\ \hline 9731 & \text{(No carry: the result is negative and is obtained by} \\ \hline \textbf{Ans: } -0268 & \text{taking the 9's complement of 9731)} \end{array}$$

Alternatively:

$$\begin{array}{rl} 1864 & \\ +7868 & \text{(Add the 10's complement of 2132)} \\ \hline 9732 & \text{(No carry: The result is negative and is obtained} \\ \hline \textbf{Ans: } -0268 & \text{by taking the 10's complement of 9732)} \end{array}$$

Review Questions

1. Determine the 10's and 9’s complement of each number.
(a) 548 (b) 73 (c) 3425
2. Perform the following subtraction using the 10’s complement and 9’s complement method.
(a) 416 - 273 (b) 635- 2458

2.4 Octal Numbers System

We know that the base of the decimal number system is 10 because it uses the digits a to 9, and the base of binary number system is 2 because it uses digits a and 1. The octal number system uses first eight digits of decimal number system: 0, 1, 2, 3, 4, 5, 6, and 7. As it uses 8 digits, its base is 8.
When we write an octal number say, 567, it can be represented in power of 8 as

$5 \text{ x } 8^2 + 6 \text{ x } 8^1 + 7 \text{ x } 8^0$

$= 5 \text{ x } 64 + 6 \text{ x } 8 + 7 \text{ x } 1$

$= 320 + 48 + 7$

$= 375_{10}$

2.4.1 Octal to Decimal Conversion

The octal number can be converted into decimal number using following steps.
1. Multiply most significant digit (leftmost digit) by 8.
2. Add next adjacent digit to the result.
3. Multiply result by 8.
4. Add next adjacent digit to the result.
5. Repeat steps 3 and 4 until all the digits are over.

Ex: Convert octal number 714 8 to its decimal equivalent
Sol. :
Step 1 : 7 x 8 = 56
Step 2: 56 + 1 = 57
Step 3: 57 x 8 = 456
Step 4: 456 + 4 = 460 decimal

2.4.2 Fractions

In the above section we have considered only octal integer numbers. In case of octal fraction numbers, we have to see the weights of digit positions to the right of the decimal point. These are given as 1/8, 1/64, 1/512 and so on. In power of 8, the weights are $8^{-1}, 8^{-2}, 8^{-3}, 8^{-4}$ and so on. These weights may be written in decimal form as 0.125, 0.015625, $1.95312 \text{ x } 10^{-3}$ and so on.

Ex: Convert octal number 0.512 to its decimal equivalent

Sol.: 05128 = 5x 8^{-1} + 1 x 8^{-2} + 2x 8^{-3}
= 0.625 + 0.015625 + 3.90625 x 10-3 = 0.6445312

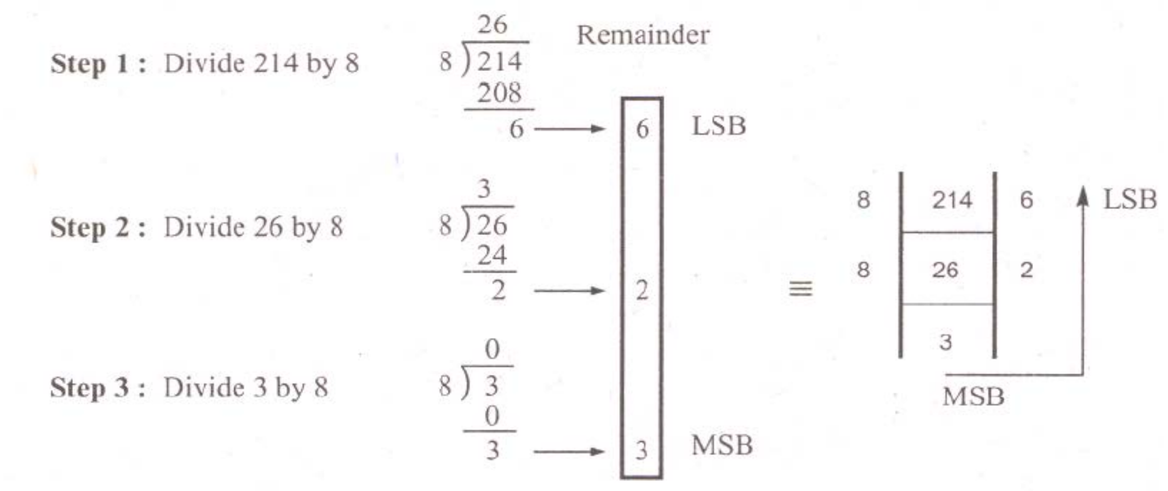
Ex: Convert $(475.25)_8$to its decimal equivalent

Sol.: N = 4 X 8^2+ 7 x 8^1 + 5 x 8^0 + 2 x 8^{-1} + 5 x 8^{-2}
= 256 + 56 + 5 + 0.25 + 0.078125
= 317.3281310

2.4.3 Decimal-to Octal Conversion

To convert decimal number to its octal equivalent, we have to progressively divide the decimal number by 8 and write down remainders. By taking the remainders in the reverse order we get the octal number.

Ex: *Convert decimal number* 214 *to its octal equivalent*



The conversion is over when quotient is 0, and we get $(326)_8$ as an octal equivalent to decimal number 214.

2.4.4 Fractional Decimal to Octal Conversion

In case of fractional decimal numbers, numbers are multiplied by 8 and carries are recorded from the integer position. These carries when read downward represents octal equivalent to fractional decimal number.

Ex: Convert 0.640625 decimal number to its octal equivalent.

Fraction		Base		Result	Recorded Carries	
0.640625	X	8	=	5.125 = 0.125 with a carry of	2	MSB
0.125	X	8	=	1.0 = 0 with a carry of	1	LSB

Reading carries downward we get octal fraction = 0.51, which is equivalent to 0.640625 decimal.

2.4.5 Binary to Octal Conversion

We know that base for octal numbers is 8 and the base for binary numbers is 2. The base for octal number is the third power of the base for binary numbers. Therefore, by grouping 3 digits of binary numbers and then converting each group digit to its octal equivalent we can convert binary number to its octal equivalent.

Ex. 2.34: *Convert* (1 1 1 1 0 1 1 0 0) 2 *to octal equivalent.*

$$\underbrace{111}_{7}\quad \underbrace{101}_{5}\quad \underbrace{100}_{4}$$

Therefore Octal form of 1 1 1 1 0 1 1 0 0 $_2$= $(754)_8$

2.4.6 Octal to Binary Conversion

Conversion from octal to binary is a reversal of the process explained in the previous section. Each digit of the octal number is individually converted to its binary equivalent to get octal to binary conversion of the number.

Ex: Convert $(634)_8$to binary.

```
Sol. :          6    3     4
               110 011 100
              Binary number = 110 011 100
```

Ex: Convert (725.63) $_8$to binary.

```
Sol. :     7    2     5       6     3
          111 010 101 . 110 011
   :.     Binary number = 111010101.110011
```

2.4.7 Usefulness of Octal System

When dealing with a large quantity of binary numbers of many bits, it is convenient to use octal numbers as a "shorthand" means of expressing large binary numbers. But it is necessary to keep in mind that the digital circuits and systems work strictly in binary; we are using octal only as a convenience for the operators of the system.

Review Questions

1. Convert following octal numbers to decimal
   (a) $305_{(8)}$ (b) $712.25_{(8)}$ (c) $416.72_{(8)}$
2. Convert following decimal numbers to octal
   (a) 129$(_{10)}$ (b) $71.472_{(10)}$ (c) $368.95_{(10)}$
3. Convert following octal numbers to binary
   (a) $512_{(8)}$ (b) $634.23_{(8)}$ (c) $41.6_{(8)}$
4. Convert following binary numbers to octal
   (b) $10111011_{(2)}$ (b) $10011011.1011011_{(2)}$

2.5 Hexadecimal Number System

The hexadecimal number system has a base of 16 having 16 digits: 0, 1,2,3,4,5,6,7,8,9, A, B, C, D, E and F. It is another number system that is particularly useful for human communications with a computer. Although it is somewhat more difficult to interpret than the octal number system, it has become the most popular. Since its base is a power of 2 (16), it is easy to convert hexadecimal no's to binary & vice versa.

Table 2.5 shows the relationship between decimal, binary and hexadecimal. Note that each hexadecimal digit represents a group of four binary digits, called nibbles, which are fundamental parts of larger binary words.

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

Table 2.5 Relation between decimal, binary and hexadecimal numbers'

When we write a hexadecimal number say 3FD (16), it can be represented in power of 16 as

$3 \times 16^{2} + F \times 16^{1} + D \times 16^{0}$

$= 3 \times 256 + F(15) \times 16 + D(13) \times 1$

$= 768 + 240 + 13$

$= 1021_{(10)}$

2.5.1 Hexadecimal to Decimal Conversion

The hexadecimal number can be converted into decimal number using following steps.
1. Multiply most significant digit (leftmost digit) by 16.
2. Add next adjacent digit to the result.
3. Multiply result by 16.
4. Add next adjacent digit to the result.
5. Repeat steps 3 and 4 until all the digits are over.

Ex: Convert hexadecimal number $7E4_{(16)}$to its decimal equivalent
7×16^2 + E (decimal 14) $\times 16^1 + 4 \times 16^0$
$= 7 \times 256 + 14 \times 16 + 4 \times 1$
$= 1792 + 224 + 4 = 2020_{(10)}$

2.5.2 Fractions

In the above section we have considered only hexadecimal integer numbers. In case of hexadecimal fraction numbers, we have to see the weights of digit positions *to* the right of the decimal point. These are given as 1/16, 1/256, 1/4096 and so on. In power of 16, the weights are 16^{-1}, 16^{-2},16^{-3},16^{-4} and so on. These weights maybe written in decimal form as 0.0625, 0.0039 and so on.

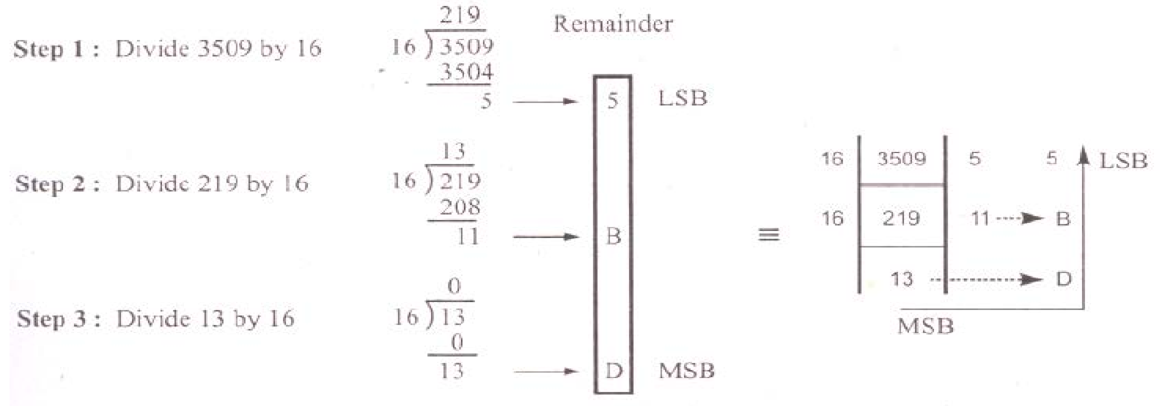
Ex: Convert hexadecimal number 21.582(16) to its decimal equivalent

$= 2x16^2 + 1x16 + 5x16^{-1} + 8x16^{-2} + 2x16^{-3}$
$= 2 \times 256 + 16 + 0.3125 + 0.03125 + 4.8828x10^{-4}$
= 528.34424(10)

2.5.3 Decimal to Hexadecimal Conversion

To convert decimal number to its hexadecimal equivalent, we have to progressively divide the decimal number by 16 and write down remainders. By taking the remainders in the reverse order we get the hexadecimal number.

Ex: Convert decimal number 3509 to its hexadecimal equivalent.



The conversion is over when quotient is 0, and we get DB5H as hexadecimal equivalent to decimal number 3509.

2.5.5 Fractional Decimal to Hexadecimal Conversion

In case of fractional decimal numbers, numbers are multiplied by 16 and carries are recorded from the integer position. These carries when read downward represents hexadecimal equivalent to fractional decimal number. This procedure is repeated until the required no. of decimal points is obtained or the numbers start repeating or a there are only zeros left after the decimal point.

Ex: Convert 0.1289062 decimal number to its hex equivalent

Fraction		**Base**		**Result**	**Recorded Carries**
0.1289062	X	16	=	2.0625	2 MSB
0.0625	X	16	=	1.0	1 LSB

Reading carries downward we get hexadecimal fraction = $0.21_{(16)}$,which is equivalent to 0.1289062 decimal.

2.5.6 Binary to Hexadecimal Conversion

We know that base for hexadecimal numbers is 16 and the base for binary numbers is 2. The base for hexadecimal number is the fourth power of the base for binary numbers. Therefore, by grouping 4 digits of binary numbers and then converting each group digit to its hexadecimal equivalent we can convert binary number to its hexadecimal equivalent.

Ex: Convert $(110110001001\ 1011)_2$to hexadecimal equivalent.

$$\underbrace{1101}_{D}\ \ \underbrace{1000}_{8}\ \ \underbrace{1001}_{9}\ \ \underbrace{1011}_{B}$$

Thus Hexadecimal number = $D89B_{(16)}$

2.5.7 Hexadecimal to Binary Conversion

Conversion from hexadecimal to binary is a reversal of the process explained in the previous section. Each digit of the hexadecimal nuil1ber is individually converted to its 4 bit binary equivalent to get hexadecimal to binary conversion of the number.

Eg: Convert 3FD(16) to binary.

Sol. : 3 F D

0011 1111 1101

Binary number = 001111111101

Convert 5A.B4(16) to binary. .

5 A 9 B 4

0101 1010 1001.1011 0100

Binary number = 0101 1010 1001 . 1011 0100

2.5.8 Octal to Hexadecimal Conversion

The easiest way to convert octal number to hexadecimal number is given below.

1. Convert octal number to its binary equivalent
2. Convert binary number to its hexadecimal equivalent.

Ex. 2.54: Convert $(615)_8$to its hexadecimal equivalent.

Step 1 : Octal to binary

```
6     1     5
110   001   101
```

Binary number = 110001101

Binary to hexadecimal

```
0001 1000 1101
1      8     D
```

Hexadecimal Number is $18D_{(16)}$

2.5.9 Hexadecimal to Octal Conversion

The easiest way to convert hexadecimal number to octal number is given below

1. Convert hexadecimal number to its binary equivalent
2. Convert binary number to its octal equivalent

Ex. 2.55: Convert $25B_{(16)}$ to its octal equivalent.

Step 1 : Hexadecimal to binary

```
2      5      B
0010   0101   1011
```

Binary number = 001001011011

Step 2 :Binary to octal

001 001 011 011

Octal number = 1133

Review Questions

1. Convert following hexadecimal numbers to decimal. (a) $F28_{(16)}$ (b) $BC2_{(16)}$
2. Convert following decimal numbers to hexadecimal.(a) 1259 (b) 5768
3. Convert 10 0 1 0 0 1 11 0 1 0 1 1 0 1 $_{(2)}$to hexadecimal.
4. Convert BED(16)to octal and binary.
5. Convert 674(8) to hexadecimal.

2.6 BCD Numbers

The BCD numbers contains 10 digits: 0,1,2,3,4,5,6,7,8 and 9.

The table below shows the relationship between decimal, binary and BCD numbers.

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

2.6.1 Decimal to BCD Conversion

The easiest way to convert a decimal number to BCD number is simple. Just Convert Each digit of the decimal number to its four bit equivalent binary number and then combine these digits together, then you will get the BCD equivalent of the given decimal number.

Ex 1: Convert 265 to its BCD form.

First take each digits separately and then convert each digit to its four bit equivalent binary no.

2 – 0010

6 – 0110

5 – 0101

Now combine all these 4 bit binary groups together – 001001100101

Therefore $265_{(10)}$ =001001100101 in BCD

Ex 2: Convert 9723 to its BCD form.

9 – 10017 – 01112 – 00103 – 0011

Therefore $9723_{(10)}$ = 1001 0111 0010 0011 in BCD

Review Questions

1. Convert following numbers to its BCD form.

(a) 728 (b)46 (c) 583

2.7. Boolean Algebra

Boolean Algebra is a useful mathematical system used to simplify/rearrange Boolean equation to make simple logic circuit. In this section we will see, logic expressions used for different gates in Boolean algebra, rules, laws and theorems used in the Boolean Algebra.

2.7.1 Rules in Boolean Algebra

In Boolean Algebra certain rules are followed. They are as follows:

1. We will use capital letters for representing variables and functions of variables. Any single variable or a function of several variables can have either a 1 or a value. Using positive logic, a binary 1 will represent a HIGH level, and a binary a will represent a LOW level, in Boolean equations.

2. The complement of a variable is represented by a "bar" over the letter. For example, the complement of a variable A will be denoted by A. So if A = 1,$\overline{A}$ =0 and if A = 0, $\overline{A}$ = 1. Sometimes a prime symbol (') is used to denote the complement. For example, the complement of A can be written as A'.

3. The logical AND function of two variables is represented either by writing a "dot" between the two variables, such as A**.**B. Many a times, this dot is omitted and written as AB.

4. The logical OR function of two variables is represented by the sign "+" between the two variables such as A + B (read as A or B).

5. Addition in Boolean algebra involves variables having values of either a binary 1 or a binary 0. The basic rules for Boolean addition are :

0 + 0 = 0	1 + 0 = 1
0 + 1 = 1	1 + 1 = 1

We note that Boolean addition is the same as the logical OR function.

6. Multiplication in Boolean algebra follows the same basic rules- governing binary multiplication, and are as follows:

0 . 0 = 0	1 . 0 = 0
0 . 1 = 0	1 . 1 = 1

2.8 Duality Form (Complements)

The duality or complement theorem says that, starting with a Boolean relation, you can derive dual form or complement of another boolean relation by
1. Changing each OR sign to an AND sign
2. Changing each AND sign to an OR sign
3. Complementing any 0 or 1 appearing in the expression
For instance, Rule 1 says that A + 0 = A. The dual relation is A. 1 = A
This dual property is obtained by changing the OR sign to an AND sign, and by complementing the 0 to get a 1. The duality theorem is useful because it sometimes produces a new Boolean relation.
For example, Distributive Law states that A (B + C) = AB + AC
By changing each OR and AND operation, we get the dual relation
A + BC = (A + B) (A + C)
(For, proof, construct the truth table for each side of the equation. The truth tables will be identical, which means the Boolean relation is true.)

Some examples for duals/complements

Given expression	**Dual**
$x \bullet (y + z)$	$(x \bullet y) + (x \bullet z)$
$x + (y \bullet z)$	$(x + y) \bullet (x + z)$
$x \bullet 0 = 0$	$x + 1 = 1$
$x + 1 = 1$	$x . 0 = 0$
$\boldsymbol{x \bullet (x + y) = x}$	$\boldsymbol{x + (x . y) = x}$
$A (A + B) = A$	$A + AB = A$
$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\,\overline{B}$
$(A + C) (\overline{A} + B) = AB + \overline{A}C$	$AC + \overline{A}B = (A + B) (\overline{A} + C)$

2.9 Laws, Postulates and Theorems of Boolean Algebra

Three of the basic laws of Boolean algebra are the same as in ordinary algebra: the commutative laws, associative laws, and the distributive law.

Commutative Laws

LAW 1 : A + B = B + A : This states that the order in which the variables are OR ed makes no difference in the output. The truth tables are identical. Therefore, A OR B is same as B OR A.

Truth table for commutative law for OR gates

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

=

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

LAW 2 : AB =BA : The commutative law of multiplication states that the order in which the variables are AND ed makes no difference in the output The truth tables are identical. Therefore, A AND B is same as B AND A.

A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

=

A	B	B.A
0	0	0
0	1	0
1	0	0
1	1	1

It is important to note that the commutative laws can be extended to any number of variables. For example, since A + B = B + A, it follows that A + B + C = B+A + C , and since A + C = C + A, it is true that B + A + C = B + C + A. Similarly, ABCD= BACD = BADC = ABDC, and so on.

Associative Laws

LAW 1 : A + (B + C) =(A + B) + C : This law states that in the ORing of several variables, the result is the same regardless of the grouping of the variables. For three variables, A OR B ORed with C is the same as A OR ed with B OR C.

Truth tables for associative law for OR gates

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

LAW 2 : (AB) C = A (BC) : The associative law of multiplication states that it makes no difference in what order the variables are grouped when ANDing several variables. For three variables, A AND B ANDed with C is the same as A ANDed with Band C.

Truth table for associative law for AND gates

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

Distributive Law

LAW 1: A (B + C) =AB + AC : The distributive law states that ORing several variables and ANDing the result with a single variable is equivalent to ANDing the result with a single variable with each of the several variables and then ORing the products.

Truth Table

x	y	z	y + z	x.(y + z)	x.y	x.z	(x.y)+(x.z)
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

By changing each OR and AND operation, we get the 2nd law of distribution by duality
LAW 2: A + BC = (A + B) (A + C)

The following are the **postulates and theorems** that are useful in manipulating & simplifying Boolean algebra expressions.

1) **Identity Element**
   - x • 1 = x x + 0 = x

2) **Complement**
   - x • x' = 0 x + x' = 1

3) Duality

The dual of a Boolean algebraic expression is obtained by interchanging the AND and the OR operators and replacing the 1’s by 0’s and the 0’s by 1’s.

- x • (y + z) = (x • y) + (x • z)
- x + (y • z) = (x + y) • (x + z)

Theorem 1: Idempotence

- x • x = x x + x = x

Theorem 2

- x • 0 = 0 x + 1 = 1

Theorem 3: Involution

- (x’)’ = x (x) = x

Theorem 4: Associative & Distributive

- (x • y) • z = x • (y • z) (x + y) + z = x + (y + z)
- x • (y + z) = (x • y) + (x • z)
- x + (y • z) = (x + y) • (x + z)

Theorem 5: DeMorgan

- (x • y)’ = x’ + y’ (x + y)’ = x’ • y’
- (x • y) = x + y (x + y) = x • y

Theorem 6: Absorption

- x • (x + y) = x x + (x • y) = x

The following table summarizes the postulates and axioms used in boolean algebra

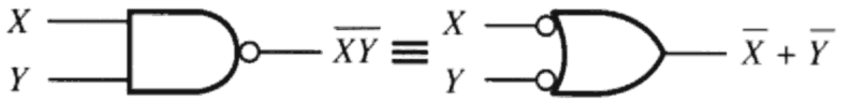
1.	$X + 0 = X$	2.	$X \cdot 1 = X$	Identity element
3.	$X + 1 = 1$	4.	$X \cdot 0 = 0$	
5.	$X + X = X$	6.	$X \cdot X = X$	Idempotence
7.	$X + \overline{X} = 1$	8.	$X \cdot \overline{X} = 0$	Complement
9.	$\overline{\overline{X}} = X$			Involution
10.	$X + Y = Y + X$	11.	$XY = YX$	Commutative
12.	$(X + Y) + Z = X + (Y + Z)$	13.	$(XY)Z = X(YZ)$	Associative
14.	$X(Y + Z) = XY + XZ$	15.	$X + YZ = (X + Y)(X + Z)$	Distributive
16.	$\overline{X + Y} = \overline{X} \cdot \overline{Y}$	17.	$\overline{X \cdot Y} = \overline{X} + \overline{Y}$	DeMorgan’s

2.10. DeMorgan's Theorems

DeMorgan suggested two theorems that form 'an important part of Boolean algebra. In the equation form, they are: -

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

The complement of a product is equal to the sum of the complements. This is illustrated by the gate equivalency in Fig. below and truth table

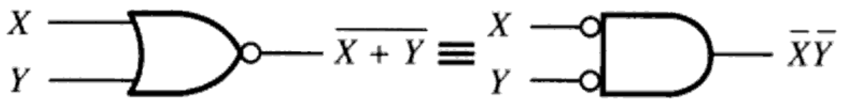


<u>Truth Table</u>

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

2) $\overline{A+B} = \overline{A} . \overline{B}$

The complement of a sum is equal to the product of the complements. This is illustrated by the gate equivalency in Fig. below and truth table



<u>Truth Table</u>

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

Using Boolean theorems and postulates, prove the following

1) x+x=x

<u>Solution</u>:
LHS: x+x
= x(1+1)
= x(1)
= x = RHS

2) x+xy=x

<u>Solution</u>:
LHS: x+xy
= x(1+y)
= x(1)
= x = RHS

3) x+1=1

<u>Solution</u>:
LHS: 1.(x+1)
= (x+x’).(x+1)
= xx + x.1 + x’x + x’.1
= x + x + 0 + x’
= x + x’
= 1 = RHS

4) x+x’y=x+y

<u>Solution</u>:
LHS: x+x’y
= (x+x’).(x+y) [By distributive law]
= 1.(x+y)
= x+y = RHS

5) x’y’z+x’yz+xy’=x’z+xy’

<u>Solution</u>:
LHS: x’y’z+x’yz+xy’
= x’z(y’+y)+xy’
= x’z(1)+xy’
= x’z+xy’= RHS

6) ABC+A’B+ABC’=B

<u>Solution</u>:
LHS: ABC+A’B+ABC’

Re-arranging the terms according to common terms

ABC+ABC'+A'B
= AB(C+C')+A'B
= AB(1)+A'B
= AB+A'B
= B(A+A')
= B(1)
= B= RHS

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

<u>Solution</u>:
LHS: (A+B)(A+C)
= AA+AC+BA+BC
= A+AC+AB+BC
= A(1+C)+AB+BC
= A(1)+ AB+BC
= A+ AB+BC
= A(1+B)+BC
= A(1)+BC
= A+BC= RHS

8) x(x'+y)=xy

<u>Solution</u>:
LHS: x(x'+y)
= xx'+xy
= 0+xy
= xy = RHS

9) xy+x'z+yz=xy+x'z

<u>Solution</u>:
LHS: xy+x'z+yz
= xy+x'z+yz(x+x')
= xy+x'z+xyz+x'yz
Re-arranging the terms according to common terms
= xy+xyz+x'z+x'yz
=xy(1+z)+x'z(1+y)
= xy(1)+x'z(1)
= xy+x'z= RHS

2.11 Assignment Questions

Short answer questions

1) Convert binary number 1010.1001 to its decimal equivalent (April/May 2009)
2) Convert decimal 82.45 to its binary equivalent (May/June 2010)
3) Differentiate between 1's and 2's complement (April/May 2011, 2010)
4) Find l's and 2's complement of 1011 1101 (April/May 2008)
5) Find 1's and 2's complement of 1100.1010
6) Find 9's and 10's complement of 3125 (April/May 2011)
7) Find binary and octal equivalent of $BCD_{(16)}$ (April/May 2012)
8) Convert 1101.1011(2) to hexadecimal
9) What is a BCD code? Give an example (April/May 2008)
10) Find BCD and binary equivalent of $98_{(10)}$ (April/May 2009)
11) Convert $BCA_{(16)}$ into binary and octal (April/May 2011)
12) What do you mean by duality of boolean algebra? Give some examples (April/May 2011)
13) State and prove any 2 postulates of boolean algebra (April/May 2010)
14) Write the boolean functions for equivalence and implication (April/May 2013)
15) Prove that ABC + A'B + ABC' = B (April/May 2012)
16) Prove (A + B) (A + C) = A + BC (April/May 2011)
17) What is meant by the principle of duality? Write the dual of given expression

    F = (X+Y) (X+Y') (X+Y+Z) (April/May 2009)

Long answer questions

1) Convert 493.68(10) to binary, octal and hexadecimal forms (April/May 2011)
2) Perform the following Conversion (April/May 2012)

    i) $(1101.11)_2 = (?)_{10}$

    ii) $(37)_8 = (?)_{16}$

    iii) $(45)_{10} = (?)_2$.
3) 5137.46 $_{(8)}$= ? $_{(10)}$= ?$_{(16)}$= ? $_{(2)}$ (April/May 2012)
4) F9B.75 $_{(16)}$= ? $_{(10)}$ = ? $_{(8)}$
5) $(4096)_{10}$= ( ) $_2$ = ( ) $_8$ = ( ) $_{16}$ (April/May 2007)
6) Perform the following conversions (May/June 2010, 2012)

    i) $(1101.11)_2 = (\ )_{10}$

    ii) $(37)_8 = (\ )_{16}$

    iii) $(45)_{10} = (\ )_2$

    iv) (BCD.A5)16 = $(\ )_2$

    v) $(125.48)_{10} = (\ )_8$
7) Convert 225.225 to binary octal and hexadecimal numbers. (May/June 2010)
8) Convert each of the following binary numbers to octal and hexadecimal forms:

    (a)1011_2(b) 1001.001_2(c) 10110.00101_2

9) Perform the following subtraction using 1's and 2's complement (April/May 2012)
   i) $10011_2 - 1001_2$
   ii) $(1011.11)_2 - (1100.11)_2$

10) Perform the following subtraction using 1's and 2's complement (April/May 2011)
   i) $(6753)_{10} - (3310)_{10}$ ii) $(7156)_{10} - (3739)_{10}$

11) Using 9'scomplement method perform 345 – 257

12) Using 10'scomplement method perform 1426 – 738

13) Perform the following subtraction using 9's and 10's complement (April/May 2012)
   i) $(834)_{10} - (325)_{10}$
   ii) $(267)_{10} - (23)_{10}$.

14) State and prove any 5 postulates of boolean algebra. (May/June 2010)

15) How do you get complement or dual form of a function? Find the complement of
   F1 = x'yz' + x'y'z (April/May 2013)
   F2 = x(y'z' + yz)

16) Using Boolean theorems and postulates, prove the following (April/May 2013)
   i) x + x'y = x + y
   ii) x'y'z + x'yz + xy' = x'z + xy'

17) Prove the following theorems of boolean algebra: (May/June 2010)
   i) x + x = x ii) x + xy = x iii) x + x'y = x + y

18) State and prove De Morgan's Theorem for 2 or 3 variables (2009, 2010, 2011, 2013)

UNIT – II

Chapter 3.Digital Logic Gates

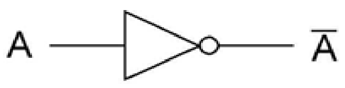
3.1 Logic Gates

A **logic gate** is an idealized or physical device implementing a boolean function. That is, it performs a logical operation on one or more logical inputs, and produces a single logical output.

There are 3 basic logic gates

1. **NOT gate**: The operation of an inverter (NOT circuit) can be expressed as follows: If the input variable is A and the output variable is Y; then Y= $\overline{A}$. The output is complement of the input.

Logic Diagram

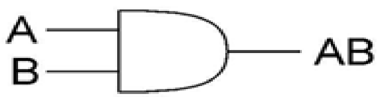


Truth Table

X	$\overline{X}$
0	1
1	0

2. **AND gate:** Let the two input variables be A and B and the output variable Y; then the boolean expression is Y = AB or A.B. If there are four input variables A, B, C, D, then the output Y = ABCD.
The output is 1 (HIGH) only when all the inputs are 1s (HIGH).

Logic Diagram

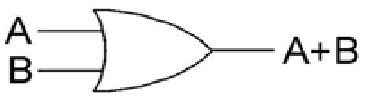


Truth Table

A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

OR gate: If one input is A, the other input is B, and the output is Y, then the Boolean expression for OR function is Y = A+B. If there are four input variables A, B, C, D; then the output is Y = A + B+ C + D. The output is a 1 (HIGH) when anyone or more of the inputs are 1 (HIGH).

Logic Diagram



Truth Table

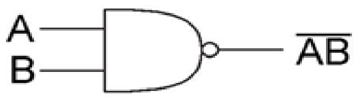
A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

3.2 Universal Gates

A universal gate is the gate using which you can implement all the other gates in the logical system. There are 2 universal gates

1. **NAND gate**: The Boolean expression for NAND function is Y = $\overline{AB}$. This expression tells that the two input variables, A & B, are first ANDed and then complemented, as indicated by the bar over the AND expression. The NAND expression can be extended to more than two input variables by including additional letters to represent all the variables.

Logic Diagram

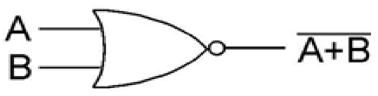


Truth Table

A	B	$\overline{(A.B)}$
0	0	1
0	1	1
1	0	1
1	1	0

2. **NOR gate**: The Boolean expression for NOR function is Y = $\overline{A + B}$. This equation says that the two input variables are first ORed and then complemented.

Logic Diagram



Truth Table

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

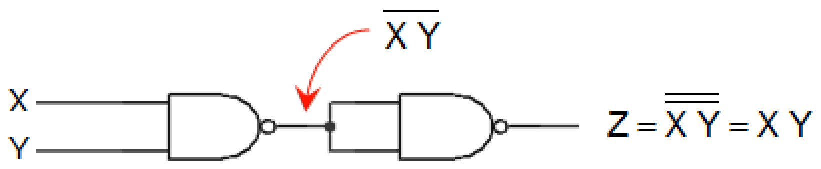
3.3 NAND and NOR Implementation

**** Show that NAND gate and NOR gate are universal gates**

To show that the NAND gate and the NOR gate are universal gates, we have to show that all the basic gates can be implemented using only NAND gates or using only NOR gates.
The diagrams given in the figure below shows the realization of AND, OR and NOT functions using either only NAND gates or only NOR gates.

I) Showing that NAND gate is a universal gate

(a) Implementation of AND function using only NAND gates

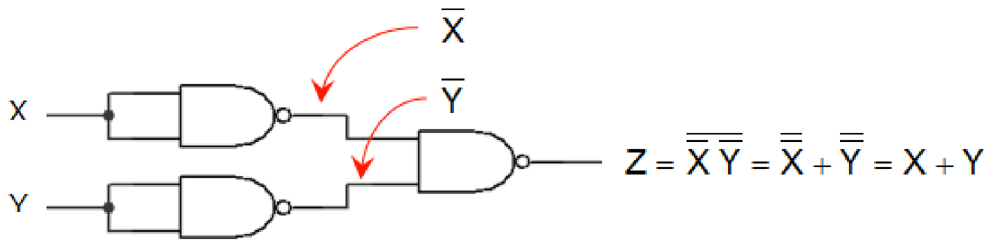


Truth Table

X	Y	X.Y	$\overline{X.Y}$	$\overline{\overline{X.Y}}$
0	0	0	1	0
0	1	0	1	0
1	0	0	1	0
1	1	1	0	1

Equivalent to AND Gate

(b) Implementation of OR function using only NAND gates

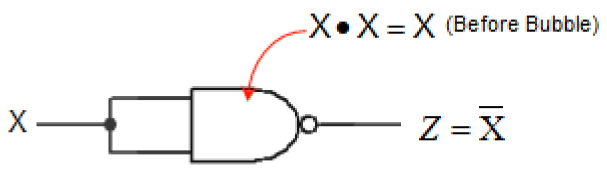

$\overline{X}$
$\overline{Y}$
X
Y
$Z = \overline{\overline{X}\,\overline{Y}} = \overline{\overline{X}} + \overline{\overline{Y}} = X + Y$

<u>Truth Table</u>

X	Y	$\overline{X}$	$\overline{Y}$	$\overline{\overline{X}}+\overline{\overline{Y}}$
0	0	1	1	0
0	1	1	0	1
1	0	0	1	1
1	1	0	0	1

Equivalent to OR Gate

(c) Implementation of NOT function using only NAND gates

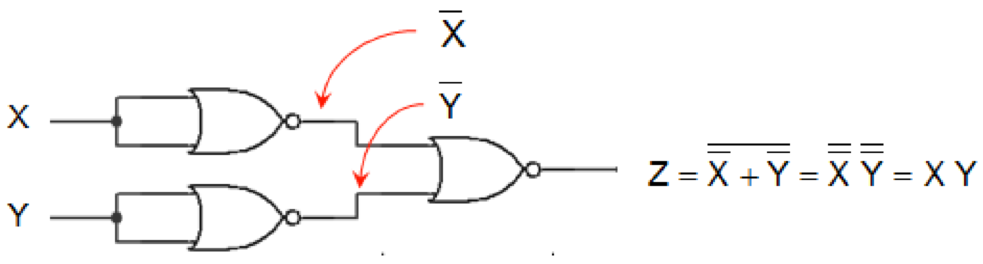


<u>Truth Table</u>

X	$\overline{Z}$
0	1
1	0

II) Showing that NOR gate is a universal gate

(a) Implementation of AND function using only NOR gates

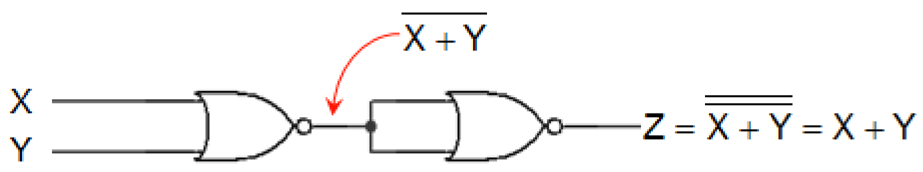

$\overline{X}$
$\overline{Y}$
X
Y
$Z = \overline{\overline{X} + \overline{Y}} = \overline{\overline{X}}\,\overline{\overline{Y}} = X\,Y$

Truth Table

X	Y	XY	$\overline{\overline{X}}\,.\,\overline{\overline{Y}}$
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

Equivalent to AND Gate

(b) Implementation of OR function using only NOR gates

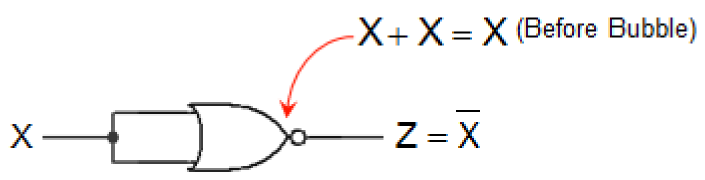


Truth Table

X	Y	X+Y	$\overline{X+Y}$	$\overline{\overline{X+Y}}$
0	0	0	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	0	1

Equivalent to OR Gate

(c) Implementation of NOT function using only NOR gates



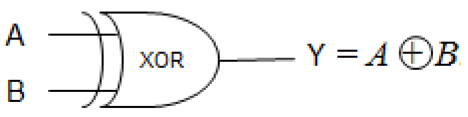
Truth Table

X	$\overline{Z}$
0	1
1	0

3.4 Other Gate Types

1. **X-OR gate**: If one input is A, the other input is B, and the output is Y. Then the Boolean Expression for X-OR function is $A \oplus B = \overline{A}B + A\overline{B}$

Logic Diagram



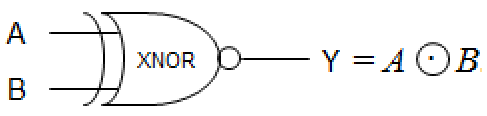
Truth Table

A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

2. **X-NOR gate**: If one input is A, the other input is B, and the output is Y. Then the Boolean expression for X-OR function is $A \odot B = \overline{A \oplus B} = \overline{A}\,\overline{B} + AB$

This equation says that the two input variables are first EX-ORed and then complemented.

Logic Diagram



Truth Table

A	B	A ⊙ B
0	0	1
0	1	0
1	0	0
1	1	1

3.5 Representation of Boolean Expressions

Let us consider a Boolean expression Y = AB + BC. In this Boolean expression A, B and C are literals. A literal is a primed or an 'unprimed variable. When a Boolean function is implemented with Boolean gates, each literal in the function designates an input to a gate, as illustrated in Fig 3.5.

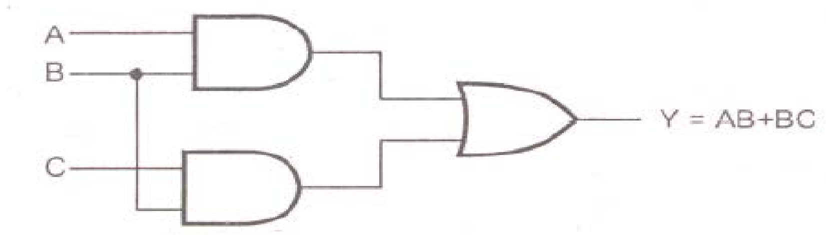


Fig. 3.5 Representation of Boolean Expression Y = AB + BC

3.6 Standard Forms of Boolean Expressions

To implement a Boolean function with a less number of gates we have to minimize literals and the number of terms. Usually, literals and terms are arranged in one of the two standard forms:
Sum of product form (SOP) and Product of sum form (POS).

3.6.1 Sum of Products or Minterms

The words sum and product are derived from the symbolic representations of. The OR and AND functions by + and .(addition and multiplication), respectively. But we realize that these are not arithmetic operators in the usual sense. A product term is any group at literal that are ANDed together. For example, ABC, XY and so on. A sum term is any group of literals that are ORed together such as A + B+ C + X + Y and so on. A sum of products (SOP) or **Minterm** is a group of product terms ORed together. Some examples of this form are:

1. $\bar{A}BC + A\bar{B} + C$
2. $\bar{X}Y + X\bar{Y}Z + Y\bar{Z}$
3. $P\bar{Q} + \bar{Q}R + P\bar{Q}\bar{R}$

Each of these sum of products expressions consists of two or more product terms (AND) that are ORed together. Each product term consists of one or more literals appearing in either complemented or uncomplemented form. For example, in the sum of products expression ABC + ABC, the first product term contains literals A, Band C in their uncomplemented form. The second product term contains Band C in their complemented (inverted) form.

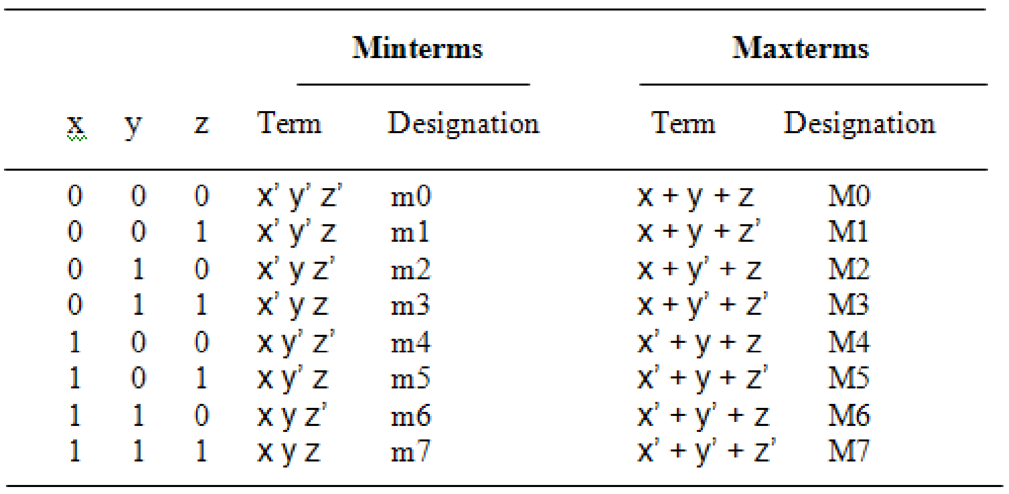
3.6.2 Product of Sums or Maxterms

A product of sums or **Maxterm** is any groups of sum terms ANDed together. Some examples of this form are

1. $(\bar{A} + B)(\bar{B} + C)$
2. $(A + \bar{B} + C)(B + \bar{C} + \bar{D})$

3. $(\bar{P} + Q)(Q + \bar{R} + S)$
Each of these product of sums expressions consists of two or more sum terms (OR) that are ANDed together. Each sum term consists of one or more literals appearing in either complemented or uncomplemented form.

Following is the table which shows Minterms and Maxterms for three binary variables



			Minterms		**Maxterms**	
x	y	z	Term	Designation	Term	Designation
0	0	0	x' y' z'	m0	x + y + z	M0
0	0	1	x' y' z	m1	x + y + z'	M1
0	1	0	x' y z'	m2	x + y' + z	M2
0	1	1	x' y z	m3	x + y' + z'	M3
1	0	0	x y' z'	m4	x' + y + z	M4
1	0	1	x y' z	m5	x' + y + z'	M5
1	1	0	x y z'	m6	x' + y' + z	M6
1	1	1	x y z	m7	x' + y' + z'	M7

A Boolean function may be expressed algebraically from a given truth table by forming a minterm for each combination of the variables which produce a 1 in the function, and then taking the OR of all those terms. For example consider a function f1 in the table above which is determined by expressing the combinations 001, 100 and 111as x'y'z, xy'z', and xyz respectively. Since each one of these minterms results in f1= 1, we should have:

f1=x'y'z +xy'z' +xyz =ml+m4+m7

x	y	z	Function f1	Function f2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

These examples demonstrate an important property of Boolean algebra: Any Boolean function can be expressed as a sum of minterms.
Now consider the complement of a Boolean function. It may be read from the truth table by forming a minterm for each combination that produces a 0 in the function and then ORing those terms. The complement of fl is read as:
F1 =x'y'z' +x'yz' +x'yz +xy'z +xyz'

If we take the complement of f1' we obtain the function f1:

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

=Mo.M2.M3.M5.M6

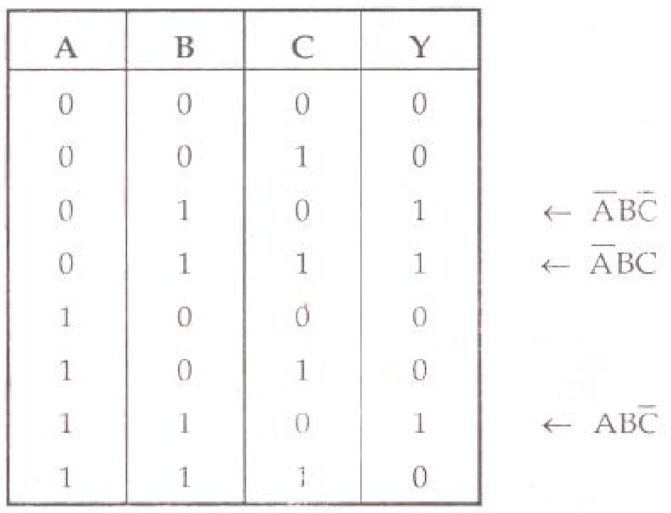
Similarly, it is possible to read the expression for f2 from the table:

f2 =(x + y +z) (x + y +z') (x + y' +z) (x' + y +z)

=Mo M1 M2 M4

3.7 Standard SOP and POS Forms

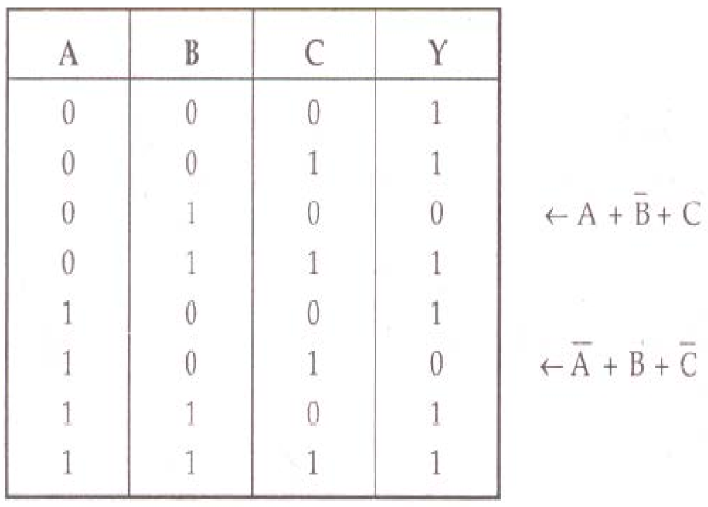
We can realize that in the SOP form and in the POS form, all the individual terms do not involve all literals. For example, in expression AB + A’ B’ C the first product term does not contain literal C. If each term in SOP and POS forms contain all the literals then these are known as standard (or canonical) SOP and POS, respectively. We know that logical expression can be represented in the truth table form. It is possible to write logic expression in standard SOP or POS form corresponding to a given truth table. The logic expression corresponding to a given truth table can be 0 written in a standard sum of products form by writing one product term for each a1 input combination that produces an output of 1. These product terms are ORed together to create the standard sum of products. Consider, for example, the following truth table:



A	B	C	Y	
0	0	0	0	
0	0	1	0	
0	1	0	1	← $\overline{A}B\overline{C}$
0	1	1	1	← $\overline{A}BC$
1	0	0	0	
1	0	1	0	
1	1	0	1	← $AB\overline{C}$
1	1	1	0	

We can note that Y is 1 when A’ = 0, B = 1 and C’ =0, so this particular combination makes this output of an AND gate equal to 1 when A and Band C are all equal to 1. Thus A B C is one ***product*** term. Similarly, the other two product terms are ABC and ABC. Thus, the standard sum of products form is

The logic Y = A’ B C’ + A’ B C + A B C’ expression corresponding to a truth table can also be written in a standard product of sums form by writing one sum term for each output O. The sum terms are expressed by writing complement of a variable when it appears as an input 1 and the variable itself when it appears as an input O. Consider, for example, the following truth table:



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

The sum corresponding to input combinations 010 is A + B’ + C, and the sum corresponding to input 101 is A’ + B + C’. Thus, the standard product of sums form is

Y = (A’ + B + C’) (A + B’ + C)

A product of sums form derived from a truth table is logically equivalent to a sum of products form derived from the same truth table.

3.8 Simplification of Boolean Expressions

We can reduce required hardware for implementation of boolean expression by simplifying it. Boolean expressions can be simplified using boolean algebra and boolean theorems. In this section we will use of basic laws, rules and theorems of boolean algebra for simplification of boolean expressions.

The steps for simplification of boolean expression using boolean algebra can be generalized as follows:
1. The original expression is put into the sum of products form by repeated application of DeMorgan's theorems and multiplication of terms.
2. Once it is in the sum of products form, the product terms are checked for common factors and factoring is performed whenever possible. Boolean rules are used simultaneously which results in the elimination of one or more terms.

3.9 Assignment Questions

Short answer questions

1) What is a logic gate? Name the 3 basic logic gates (April/May 2012)
2) List the universal gates and why are they called so? (April/May 2008)
3) Explain NAND gate with truth table
4) Explain NOR gate with truth table (April/May 2011)
5) Explain XOR gate with truth table and logic diagram (April/May 2007, 08)
6) Explain XNOR gate with truth tableland logic diagram (May/June 2010)
7) Implement XOR and XNOR gates using NAND gate (April/May 2011)
8) Implement XOR and XNOR gates using NOR gate
9) What are minterm and maxterm? (April/May 2013, 2008)
10) Define Sum of Product and Product of Sum with an example for each (April/May 2011)

Long answer questions

1) What is a logic gate? Explain all the basic gates with logic diagram and truth table (April/May 2011, 2009)
2) What are universal gates? Explain any one universal gate with example and truth table
3) Prove that NAND gate is a universal gate (April/May 2011, 2009)
4) Prove that NOR gate is a universal gate
5) Implement the following function with NAND gate (April/May 2011)
   $F(x, y, z) = \sum (0, 6)$
6) Implement boolean function F=x'y'z + x'yz + yz' with basic gates (May/June 2010)
7) Express the boolean function F = XY' + X'Z as sum of minterm and product of maxterm (April/May 2012)
8) Implement the following functions using gates (May/June 2010)
   i) F = XYZ'
   ii) F = XY' + X'Z
9) Implement the following function using NAND and NOR gates. (April/May 2012)
   F (A, B, C, D) = A (B + CD) + BC'
10) Simplify the expression $AB + \overline{A}B \,.\, (\overline{\overline{A}.\overline{C}})$ (April/May 2011)
11) Explain XOR gate with logic diagram and truth table (April/May 2007, 2008)
12) Explain XNOR gate with logic diagram and truth table (May/June 2010)

Chapter 4: Canonical and Standard Forms

4.1 Introduction

We know that Boolean expressions are implemented by connecting specific logic gates. These logic gates produce a specific output for certain specified combinations of input variables, with no storage involved. These circuits are commonly known as combinational circuits. In combinational circuits, the output level is always dependent on the combinations of the input levels. The combinational circuits can be specified in one of the following ways:
. A set of statements,
. Boolean expression, and
. Truth table.
In this section we will continue our study of combinational circuits and we will further study two methods of simplifications of logic circuits. These include K-maps and other methods.

4.2 Karnaugh Map (K-Map) Simplifications

We have seen that for simplification of boolean expressions by boolean algebra we need better understanding of boolean laws, rules and theorems. During the process of simplification we have to predict each successive step. For these reasons, we can never be absolutely certain that an expression .simplified by Boolean algebra alone is the simplest possible expression. On the other hand, the map method gives us a systematic approach for simplifying a Boolean expression. The map method, first proposed by Veitch and modified by Karnaugh, hence it is known as the Veitch diagram or the Karnaugh map.

4.2.1 The Karnaugh Map

The basis of this method is a graphical chart known as Karnaugh map (K-map). It Contains boxes called cells. Each of the cells represents one of the 2n possible products that can be formed from n variables. Thus, a 2-variable map contains 22= 4 cells, a 3-variablemap contains 23 = 8 cells, and so forth. Fig. 4.1 shows outlines of 2, 3 and 4 variable maps.

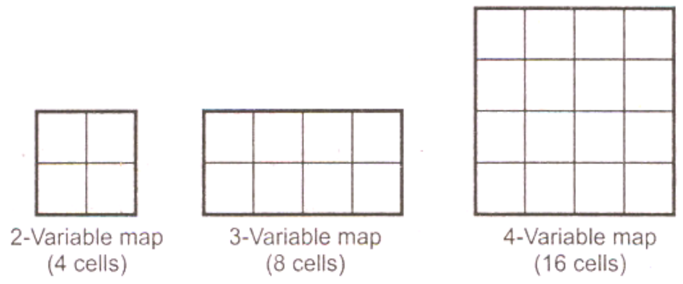
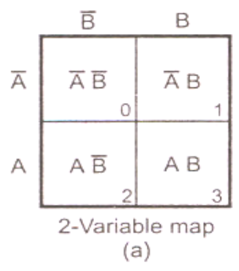
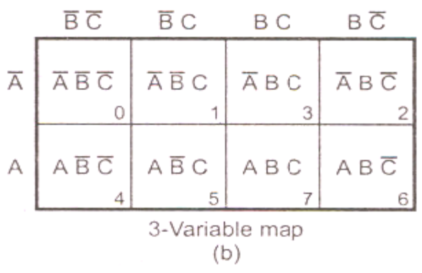


Fig. 4.1 Outlines of 2, 3 and 4 variable Karnaugh maps

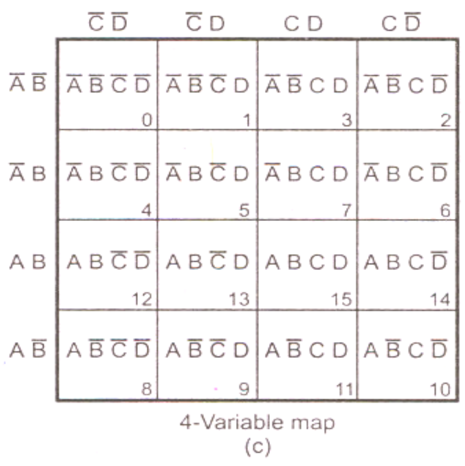
Product terms are assigned to the cells of a Karnaugh map by labeling each row and each column of the map with a variable, with its complement, or with a combination of variables and complements. The product term corresponding to a given cell is then the product of all variable in the row and column where the cell located.

Fig. 4.2 shows the way to label the rows and columns of a 2, 3 and 4 variable maps and the product terms corresponding to each cell.





$\bar{B}\bar{C}$ $\bar{B}C$ BC $B\bar{C}$
$\bar{A}$ $\bar{A}\bar{B}\bar{C}$ 0 $\bar{A}\bar{B}C$ 1 $\bar{A}BC$ 3 $\bar{A}B\bar{C}$ 2
A $A\bar{B}\bar{C}$ 4 $A\bar{B}C$ 5 ABC 7 $AB\bar{C}$ 6
3-Variable map
(b)

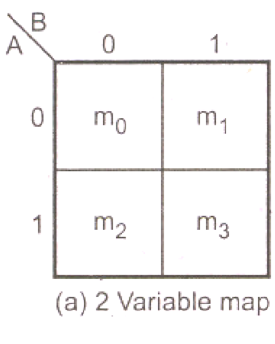


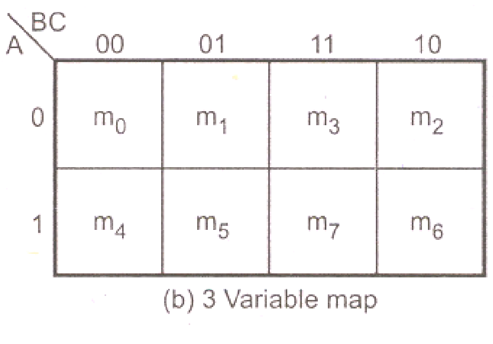
$\bar{C}\bar{D}$ $\bar{C}D$ CD $C\bar{D}$
$\bar{A}\bar{B}$ $\bar{A}\bar{B}\bar{C}\bar{D}$ 0 $\bar{A}\bar{B}\bar{C}D$ 1 $\bar{A}\bar{B}CD$ 3 $\bar{A}\bar{B}C\bar{D}$ 2
$\bar{A}B$ $\bar{A}B\bar{C}\bar{D}$ 4 $\bar{A}B\bar{C}D$ 5 $\bar{A}BCD$ 7 $\bar{A}BC\bar{D}$ 6
AB $AB\bar{C}\bar{D}$ 12 $AB\bar{C}D$ 13 $ABCD$ 15 $ABC\bar{D}$ 14
$A\bar{B}$ $A\bar{B}\bar{C}\bar{D}$ 8 $A\bar{B}\bar{C}D$ 9 $A\bar{B}CD$ 11 $A\bar{B}C\bar{D}$ 10
4-Variable map
(c)

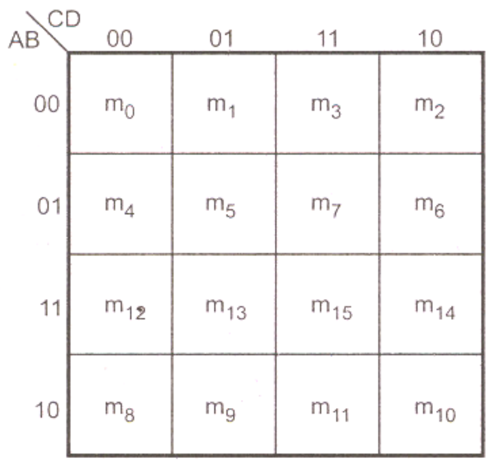
Fig. 4.2 2, 3 and 4 variable maps with product terms

It is important to note that when we move from one cell to the next along and row or from one cell to the next along any column, one and only one variable in the product term changes (to a complemented or to an uncomplemented form). For example, in Fig. 4.2 (a) the only change that occurs in moving along the bottom row from AB to AB is the change from B to B. Similarly, the only change that occurs in moving down is, the right column from A B to AB is the change from A to A. Irrespective of number of variables the labels along each row and column must conform to the single-change rule.

The Fig. 4.3 shows another way to label the rows and columns of a 2, 3 and 4 Variable maps and the product terms corresponding to each cell. Here, instead of writing actual product terms, corresponding shorthand min term notations are written in the cell, and row and columns are marked with 0s and 1s instead of variables.







AB \ CD	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

(c) 4 Variable map

Fig.4.3 Another way to represent 2, 3 and 4 variable maps

4.3 Plotting a Karnaugh Map

We know that logic function can be represented in various forms such as truth table, SOP boolean expression and POS boolean expression. In this section we will see the procedures to plot the given logic function in any form on the Karnaugh map.

4.4 Representation of Truth table on Karnaugh map

Fig. 4.4 shows K-maps plotted from truth tables with 2, 3 and 4 variables. Looking at the Fig. 4.4 we can easily notice that the terms which are having output I, have the corresponding cells marked with Is. The other cells are marked with zeros.

4.4.1 Representation of 2 variable K-map

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

A \ B	0	1
0	0	1
1	1	0

OR

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

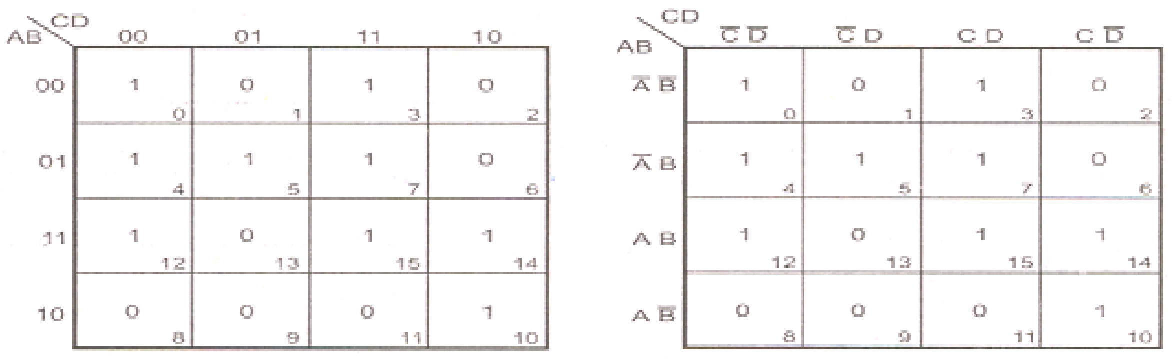
4.4.2 Representation of 3 variable K-map

A \ BC	00	01	11	10
0	0	0	0	1
1	1	0	1	0

OR

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

4.4.3 Representation of 4 variables K-map



AB \ CD	00	01	11	10
00	1_{0}	0_{1}	1_{3}	0_{2}
01	1_{4}	1_{5}	1_{7}	0_{6}
11	1_{12}	0_{13}	1_{15}	1_{14}
10	0_{8}	0_{9}	0_{11}	1_{10}

AB \ CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1_{0}	0_{1}	1_{3}	0_{2}
$\bar{A}B$	1_{4}	1_{5}	1_{7}	0_{6}
AB	1_{12}	0_{13}	1_{15}	1_{14}
$A\bar{B}$	0_{8}	0_{9}	0_{11}	1_{10}

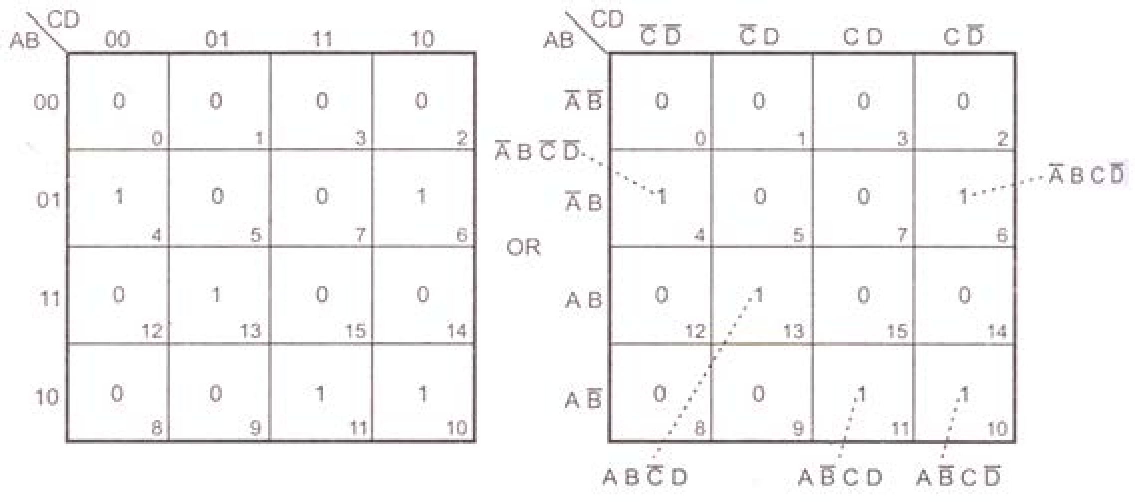
Fig. 4.4 plotting truth table on K-map

4.5 Representing standard SOP on K-map

A Boolean expression in the sum of products form can be plotted on the Karnaugh map by placing a 1 in each cell corresponding to a term (minterm) in the sum of products expression. Remaining cells are filled with zeros. This is illustrated in the following examples.

Example:

Sol.: The expression has 4-variables and hence it can be plotted using 4-variable map as shown below. -



AB \ CD	00	01	11	10
00	0_{0}	0_{1}	0_{3}	0_{2}
01	1_{4}	0_{5}	0_{7}	1_{6}
11	0_{12}	1_{13}	0_{15}	0_{14}
10	0_{8}	0_{9}	1_{11}	1_{10}

OR

AB \ CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0_{0}	0_{1}	0_{3}	0_{2}
$\bar{A}B$	1_{4}	0_{5}	0_{7}	1_{6}
AB	0_{12}	1_{13}	0_{15}	0_{14}
$A\bar{B}$	0_{8}	0_{9}	1_{11}	1_{10}

$\bar{A}B\bar{C}\bar{D}$
$\bar{A}BC\bar{D}$
$AB\bar{C}D$
$A\bar{B}CD$
$A\bar{B}C\bar{D}$

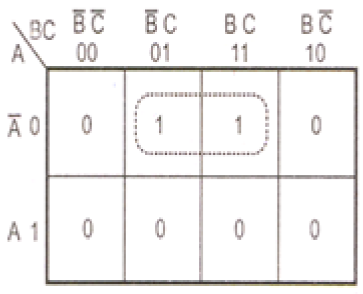
Representing Standard SOP and POS forms on K-map

4.6 Grouping Cells for Simplification

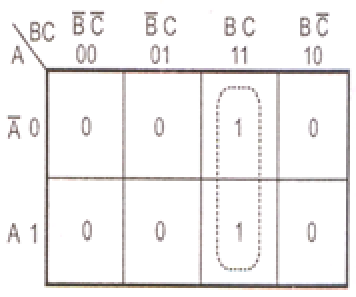
In the last section we have seen representation of logic function on the Karnaugh map. We have also seen that minterms are marked by 1s and maxterms are marked by 0s. Once the logic function is plotted on the Karnaugh map we have to use grouping technique to simplify the logic - function. The grouping is nothing but combining terms in adjacent cells. Two cells are said to be adjacent if they conform the single change rule. The simplification is achieved by grouping adjacent 1s or 0s in groups of 2^i, where i = 1, 2,.. -, n and n is the number of variables. When adjacent 1s are grouped then we get result in the sum of products form; otherwise we get result in the product 0f sums form. Let us see the various grouping rules. Grouping two adjacent ones (Pair)

Fig. 4.6 (a) shows the Karnaugh map for a particular three variable truth table. This K-map contains a pair of 1s that are horizontally adjacent to each other; the first represents ABC and the second represents A Be. Note that in these two terms only the B variable appears in both normal and complemented form (A and C remain unchanged). Thus these two terms can be combined to give a resultant that eliminates the B variable since it appears in both uncomplemented and complemented form.

This same principle holds true for any pair of vertically or horizontally adjacent1s.



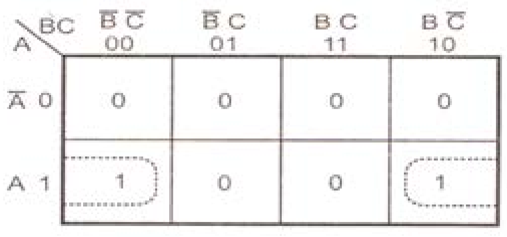
4.6 (a)



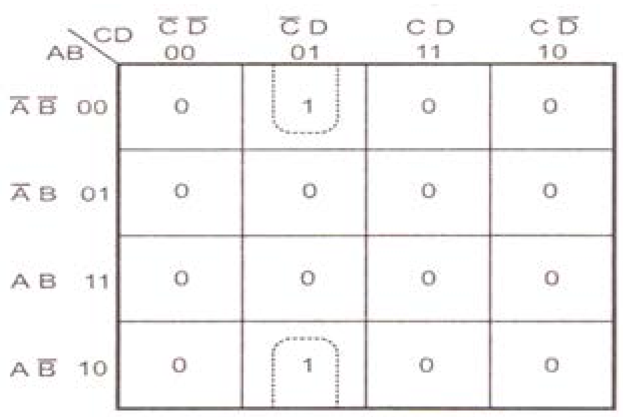
4.6 (b)

Fig. 4.6 (b) shows an example of two vertically adjacent 1s. These two can be combined to eliminate a variable since it appears in both its uncomplemented and complemented forms. This gives result

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



Here variable B has appeared in both its forms and hence eliminated as follows; --



Let us see another example shown above. Here two 1s from top row and bottom row of some column are combined to eliminate the variable and also the leftmost and rightmost columns are combined to eliminate the variable, since in a K map the top row and bottom row and the leftmost and rightmost columns are considered to be adjacent.

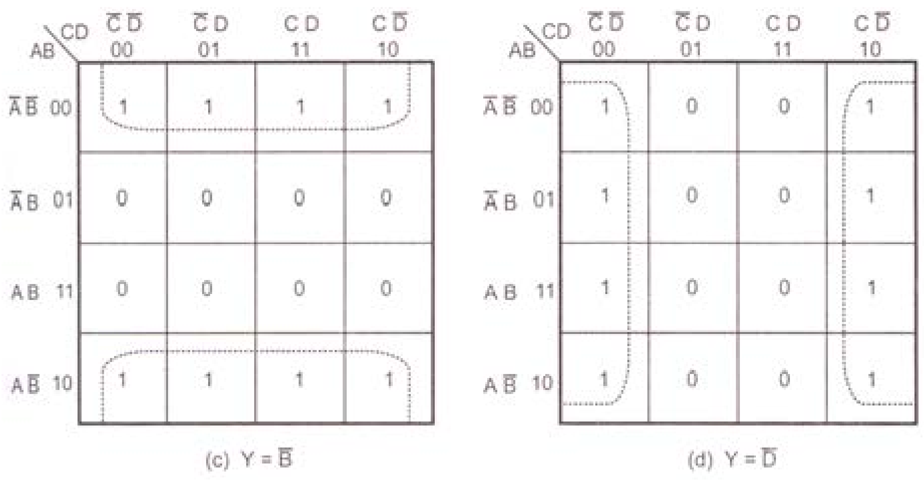
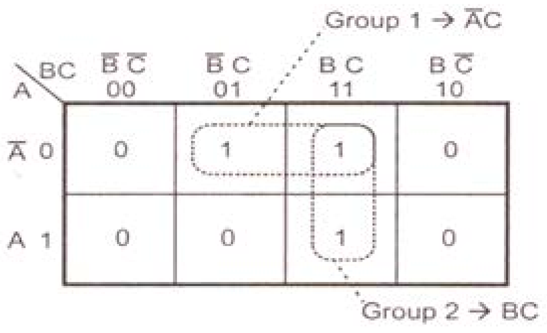
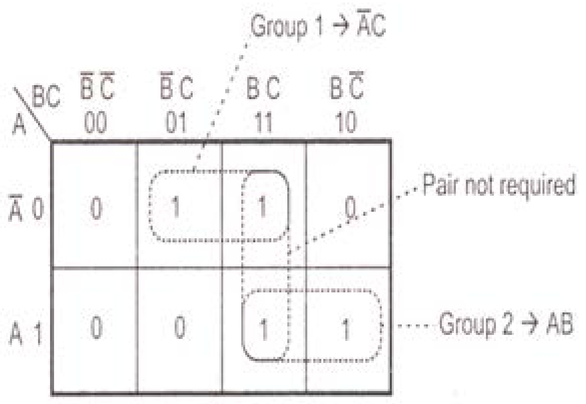


Fig. 4.6 (c) and (d) shows a Karnaugh map that has two overlapping pairs of 1s. This shows that we can share one term between two pairs.



4.6 (e)

Fig.4.6 (e) shows a K-map where three group of pairs can be formed. But only two pairs are enough to include all 1s present in the K-map. In such cases third pair is not required as shown in fig. 4.6 (f).



4.6 (f)

4.7 Grouping four Adjacent ones (Quad)

In a Karnaugh map we can group four adjacent 1s; The resultant group is called Quad. Fig. 4.9 (a) shows the four 1s are horizontally adjacent and Fig. 4.9 (b) shows they are vertically adjacent.

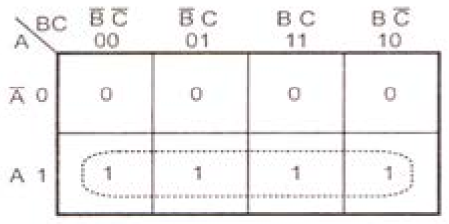


Fig. 4.9 (a) Y = A

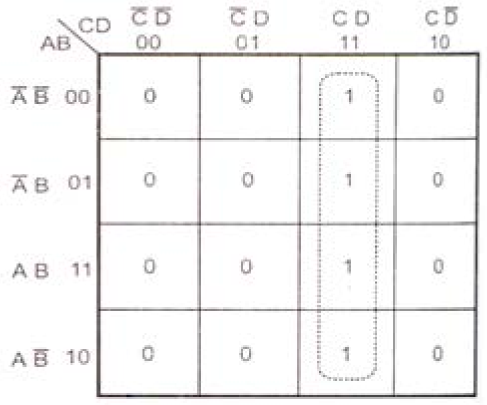


Fig. 4.9 (b) Y = CD

A K-map in Fig. 4.9 (c) contains four 1s in 9 square, and they are considered adjacent to each other. The four Is in Fig. 4.9 (d) are also adjacent, as are those in FigA.9 (e) because, as mentioned earlier, the top and bottom rows are considered to be adjacent to each other and the leftmost and rightmost columns are also adjacent to each other.

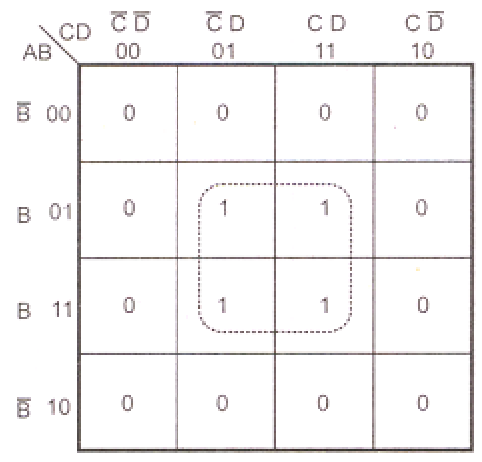


Fig. 4.9 (c) Y = BD

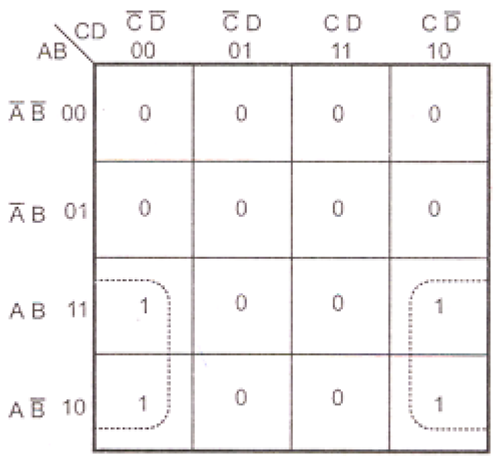


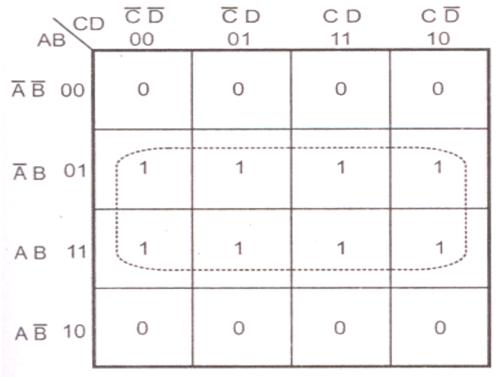
Fig. 4.9 (d) Y = AD

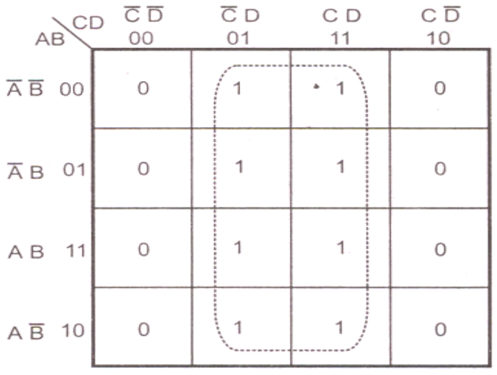
From the above Karnaugh maps we can easily notice that when a quad is combined two variables are eliminated.

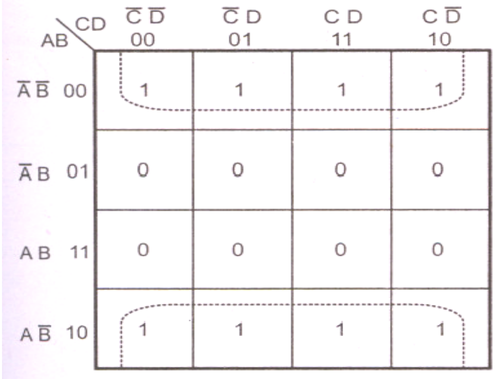
4.8 Grouping Eight Adjacent Ones (Octet)

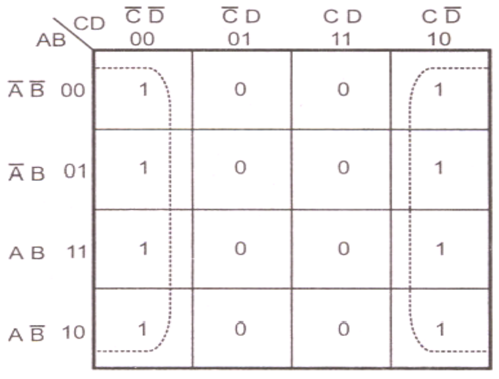
In a Karnaugh map we can group eight adjacent 1s. The resultant group is called as octet. Fig. 4.10 shows, several examples of octets. Fig. 4.10 (a) shows the eight 1s are horizontally adjacent and Fig. 4.10 (b) shows they are vertically adjacent. From the figures 4.10 we can easily observe that when an octet is combined in a four variable map, three of the four variables are eliminated because only one variable remains unchanged. For example, in K-map shown in Fig. 4.10 (a) we have following terms:

$$
\begin{aligned}
Y &= \overline{A}\,B\,\overline{C}\,\overline{D} + \overline{A}\,B\,\overline{C}\,D + \overline{A}\,B\,C\,D + \overline{A}\,B\,C\,\overline{D} \\
&\quad + A\,B\,\overline{C}\,\overline{D} + A\,B\,\overline{C}\,D + A\,B\,C\,D + A\,B\,C\,\overline{D} \\
&= \overline{A}\,B\,\overline{C}\,(\overline{D} + D) + \overline{A}\,B\,C\,(D + \overline{D}) \\
&\quad + A\,B\,\overline{C}\,(\overline{D} + D) + A\,B\,C\,(D + \overline{D}) \\
&= \overline{A}\,B\,\overline{C} + \overline{A}\,B\,C + A\,B\,\overline{C} + A\,B\,C \\
&= \overline{A}\,B\,(\overline{C} + C) + A\,B\,(\overline{C} + C) \\
&= \overline{A}\,B + A\,B \\
&= B\,(\overline{A} + A) \\
&= B
\end{aligned}
$$


(a) Y = B


(b) Y = D


(c) Y = $\overline{B}$


(d) Y = $\overline{D}$

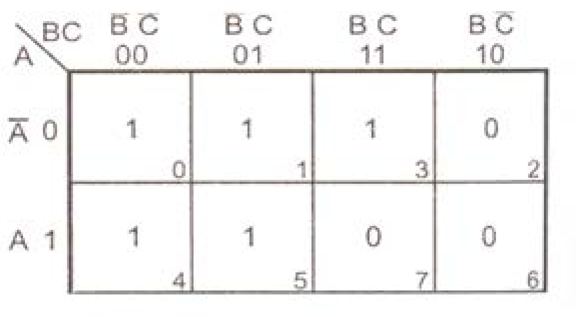
4.9 Simplification of Sum of Products Expressions

We have seen how combination *of* pairs, quads and octets on a Karnaugh map can be used to obtain a simplified expression. A pair *of* 1s eliminates one variable, a quad of1s eliminates two variables and an octet of1s eliminates three variables, In general, when a variable appears in both complemented and uncomplemented form within a group, that variable is eliminated from the resultant expression. Variables that are same in all with the group must appear in the final expression. From the above discussion we can outline generalized procedure to simplify Boolean expressions as follows:

1. Plot the K-map and place Is in those cells corresponding to the 1s in the truth table or sum *of* product expression. Place 0s in other cells.
2. Check the K-map for adjacent 1s and encircle those 1s which are not adjacent to any other Is. These are called isolated 1s.
3. Check for those 1s which are adjacent to only one other 1 and encircle such pairs.
4. Check for quads and octets of adjacent Is even if it contains some Is that have already been encircled. While doing this, make sure that there are minimum number of groups.
5. Combine any pairs necessary to include any 1s that have not yet been grouped.
6. Form the simplified expression by summing product terms of all the groups.

Ex. 4.5: ***Minimize the expression***

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



A \ BC	$B\overline{C}$ 00	BC 01	BC 11	$B\overline{C}$ 10
$\overline{A}$ 0	1 (0)	1 (1)	1 (3)	0 (2)
A 1	1 (4)	1 (5)	0 (7)	0 (6)

Solution:

Step 1: Fig 4.11 (a) shows the K-map for three variables and it is plotted according to the given expression.

Step 2: There are no isolated 1's

Step 3: There are no isolated 1's, 1 in the cell 3 is adjacent only to 1 in the cell 1.

This pair is combined and referred to as group 1. Hence we get A**'** from the row and common element C from the 2 columns. Therefore the term obtained is A**'**C

Step 4: The 1's in the cells 0,1,4,5 can be combined to form group 2 and we obtain the term B as common element in the 2 columns.

Therefore the final reduced expression is: A**'**C + B**'**

Review Questions:

Simplify the following SOP forms using K-Map

$F1(X,Y,Z)=\sum_m(3,4,6,7)$

$F2(X,Y,Z)=\sum_m(0,2,4,5,6)$

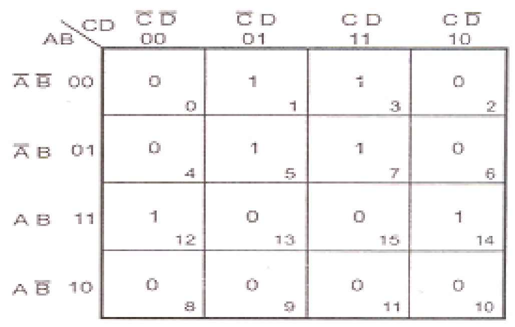
Reduce the following function using Karnaugh map technique.

$F(A, B, C, D) = \overline{A}\,\overline{B}D+AB\overline{C}\,\overline{D}+\overline{A}BD+ABC\overline{D}$

Sol.: The given function is not in the standard sum of products form. It is converted into standard SOP form as given below.

$$\begin{aligned} f(A, B, C, D) &= \overline{A}\,\overline{B}D+AB\overline{C}\,\overline{D}+\overline{A}BD+ABC\overline{D} \\ &= \overline{A}\,\overline{B}D(C+\overline{C})+AB\overline{C}\,\overline{D}+\overline{A}BD(C+\overline{C})+ABC\overline{D} \\ &= \overline{A}\,\overline{B}CD+\overline{A}\,\overline{B}\,\overline{C}D+AB\overline{C}\,\overline{D}+\overline{A}BCD+\overline{A}B\overline{C}D+ABC\overline{D} \end{aligned}$$

Step 1: Fig. 4.10 (a) shows the K-map for four variables and it is plotted according to the expression in standard SOP form

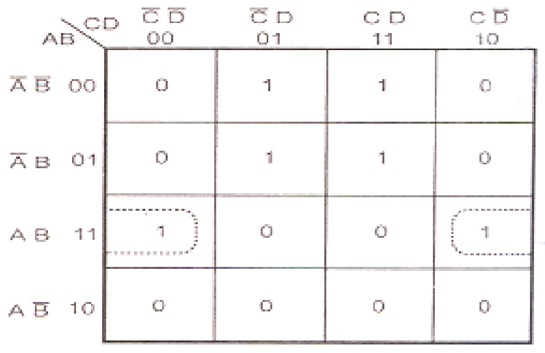


AB \ CD	$\bar{C}\bar{D}$ 00	$\bar{C}D$ 01	CD 11	$C\bar{D}$ 10
$\bar{A}\bar{B}$ 00	0_{0}	1_{1}	1_{3}	0_{2}
$\bar{A}B$ 01	0_{4}	1_{5}	1_{7}	0_{6}
AB 11	1_{12}	0_{13}	0_{15}	1_{14}
$A\bar{B}$ 10	0_{8}	0_{9}	0_{11}	0_{10}

Fig. 4.10 (a)

Step 2: There are no isolated 1's

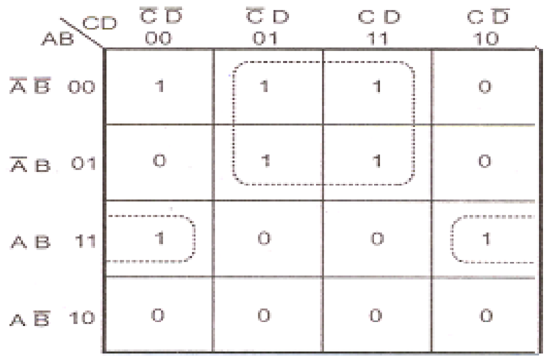
Step 3: The 1 in cell 12 is adjacent only to the one in cell 14. The pair is combined and referred as group 1.



AB \ CD	$\bar{C}\bar{D}$ 00	$\bar{C}D$ 01	CD 11	$C\bar{D}$ 10
$\bar{A}\bar{B}$ 00	0	1	1	0
$\bar{A}B$ 01	0	1	1	0
AB 11	1	0	0	1
$A\bar{B}$ 10	0	0	0	0

Step 4: There is a Quad here. Cells 1, 3, 5 and 7 form a quad. This quad is referred as group 2

Step 5: All 1s have already been grouped



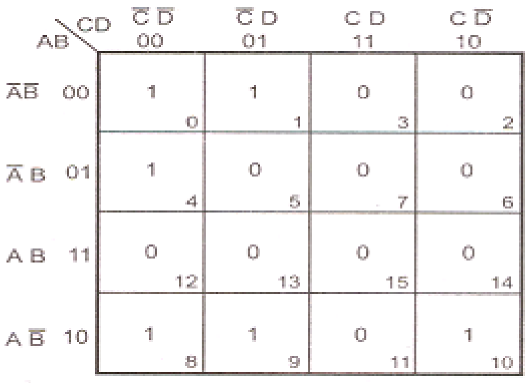
AB \ CD	$\bar{C}\bar{D}$ 00	$\bar{C}D$ 01	CD 11	$C\bar{D}$ 10
$\bar{A}\bar{B}$ 00	1	1	1	0
$\bar{A}B$ 01	0	1	1	0
AB 11	1	0	0	1
$A\bar{B}$ 10	0	0	0	0

Step 6: In group 1 variable C is eliminated and in group 2, variables B and C are eliminated. We get simplified equation as Y = $\bar{A}\ B\ \bar{D} + A\ D$

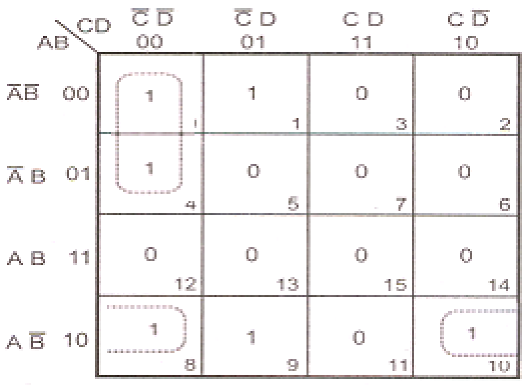
Reduce the following function using K-map technique

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

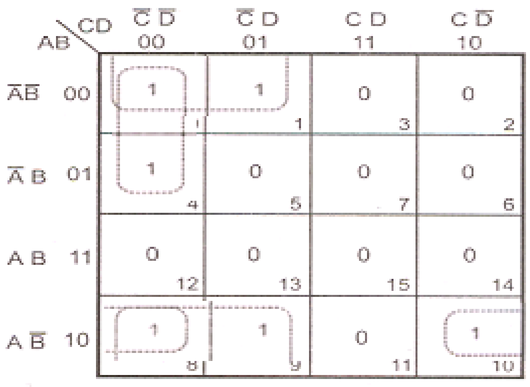
Solution:



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



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



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

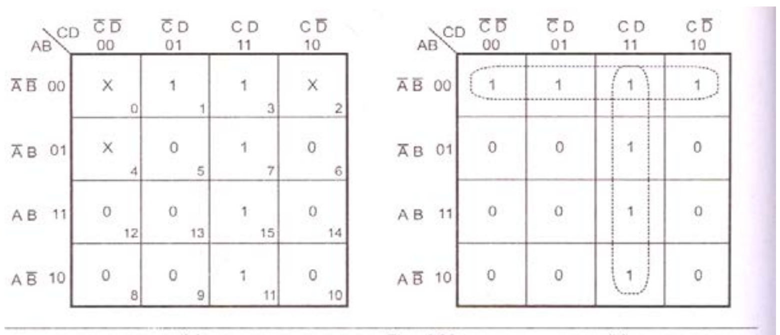
4.10 Don’t Care Conditions

In some logic circuits, certain input conditions never occur, therefore the corresponding output never appears, In such cases the output level is not defined, it can be either HIGH or LOW. These output levels are indicated by 'X' or ‘d’ in the truth tables and are called don't care outputs or don't care conditions. Let us see the output levels in the truth table as shown in the Table 4.2. Here outputs are defined for input conditions from 0 0 0 to 1 0 1. For remaining two conditions of input, output is not defined, hence these are called don't care conditions for this truth table.

A circuit designer is free to make the output for any "don't care" condition either a '0' or a '1' in order to produce the simplest output expression. For example, the K-map for above truth table is shown in Fig. 4.22 (a) with X placed in the ABC and ABC cells.

Example: Solve the following using K-Map

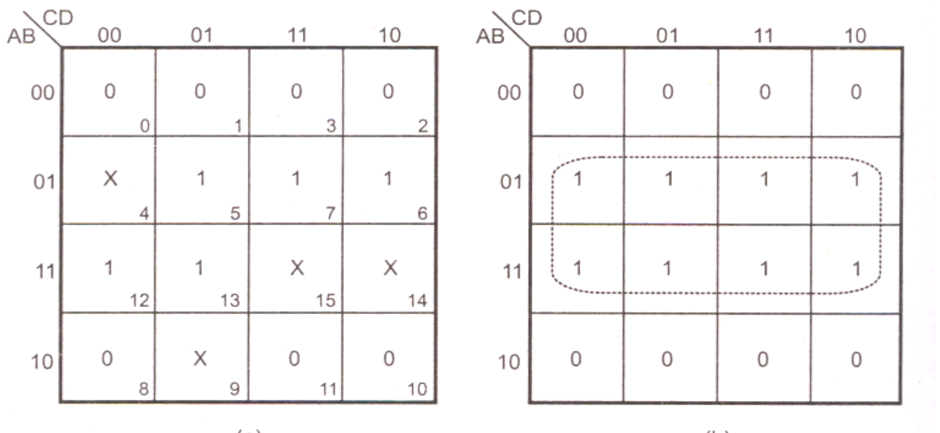
$F(A, B, C, D) = \Sigma(1, 3, 7, 11, 15) + d(0, 2, 4)$



(a) Fig. 4.23 (b)

To form a quad of cells 0, I, 2 and 3 the don't care conditions 0 and 2 are replaced by Is. The remaining don't care condition is replaced by 0 since it is not required to form any group. With these replacements we get the simplified equation as

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

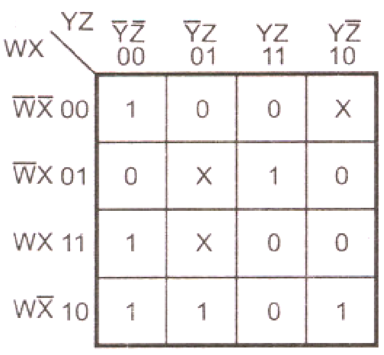


(a) (b)

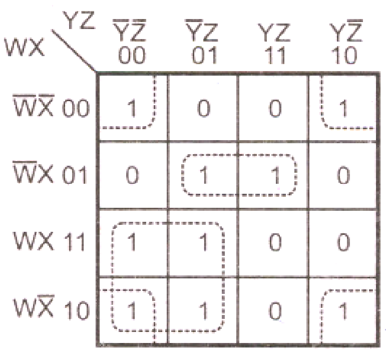
To form an octet of cells 4, 5, 6, 7, 12, 13, 14 and 15 the don't care conditions 4, 14 and 15 are replaced by Is. The remaining don't care condition 9 is replaced by 0 to get simplified function as
f (A, B, C, D) = B

Ex: Find the reduced SOP form of the following function

F(W, X, Y, Z) = ∑ m (0,7, 8, 9, 10, 12) + d (2, 5, 13)



WX \ YZ	$\bar{Y}\bar{Z}$ 00	$\bar{Y}Z$ 01	YZ 11	$Y\bar{Z}$ 10
$\bar{W}\bar{X}$ 00	1	0	0	X
$\bar{W}X$ 01	0	X	1	0
WX 11	1	X	0	0
$W\bar{X}$ 10	1	1	0	1



WX \ YZ	$\bar{Y}\bar{Z}$ 00	$\bar{Y}Z$ 01	YZ 11	$Y\bar{Z}$ 10
$\bar{W}\bar{X}$ 00	1	0	0	1
$\bar{W}X$ 01	0	1	1	0
WX 11	1	1	0	0
$W\bar{X}$ 10	1	1	0	1

In the above examples all don't cares are replaced by 1's and we get, -- - -

F (W, X, Y, Z) = $\bar{X}\,\bar{Z} + \bar{W}X\,Z + W\bar{Y}$

4.11 Simplification of Product of Sums Expressions

In the above discussion, we have considered the Boolean expression in sum of products form and grouped 2, 4, and 8 adjacent ones to get the simplified Boolean expression in the same form. In practice, the designer should examine both the sum of products and product of sums reductions to ascertain which is more simplified. We have already seen the representation of product of sums on the K-Map, the expression is plotted on the K-map instead of making the groups to make groups of zeros. The technique for using maps for POS reductions is a simple step by step process and it is similar to the one used earlier.

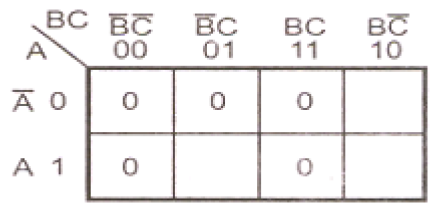
1. Plot the K-map and place as in those cells corresponding to the as in the truth table or maxterms in the products of sum expression.
2. Check the K-map for adjacent as and encircle those as which are not adjacent to any other as. These are called isolated as.
3. Check for those as which are adjacent to only one other a and encircle such pairs.
4. Check for quads and octets of adjacent as even if it contains some as that have already been encircled. While doing this make sure that there are minimum number of groups.
5. Combine any pairs necessary to include any as that have not yet been grouped,
6. Form the simplified SOP expression for F by summing product terms of all the groups.

Use DeMorgan's theorem on F to produce the simplified expression in POS form. To get familiar with these steps we will solve some examples

Example: Minimize the following in POS form using K-Map

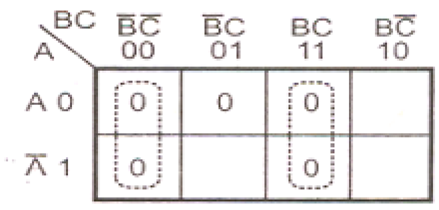
f (A, B, C) = ∏m (0, 1, 3, 4, 7)

Step 1: Fig. 4.11 (a) shows the K-map for 3 variable and it is plotted according to given maxterms



Step 2: There are no isolated 0s.

Step 3: 0 in the cell 4 is adjacent only to 0 in the cell 0 and 0 in the cell 7 is adjacent only to 0 in the cell 3. These two pairs are combined and referred to as group 2 respectively



Step 4: There are no quads and octets

Step 5: The 0 in the cell 1 can be combined with 0 in cell 3 to form a pair of group 3.

Step 6: In group 1 and 2, A is eliminated, whereas in group 3 variable B is eliminated and we get

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

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

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

Reduce the following in POS form using 4 variable K-Map

f (A, B, C, D) = ∏m (0, 1, 2, 3, 8, 9,12, 13, 15)

Step 1: Fig 4.11 (a) shows the K-map for four variables and it is plotted according to given maxterms

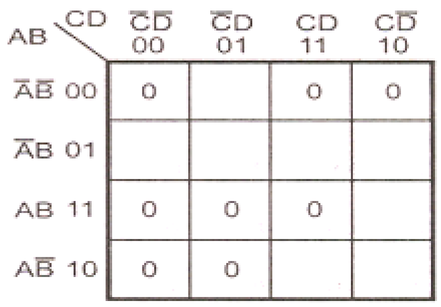
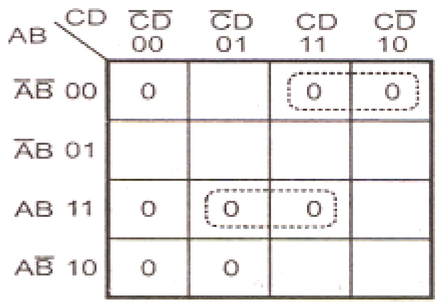


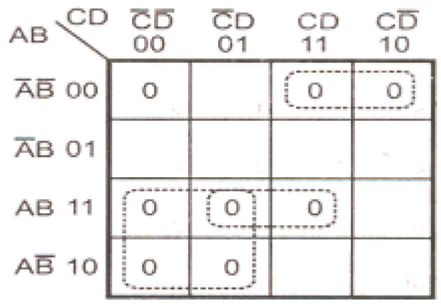
Fig 4.11 (a)

Step 2: There are no isolated 0s.

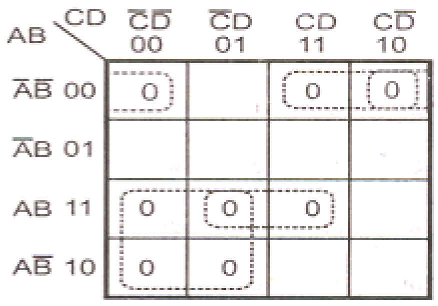
Step 3: The 0 in the cell 15 is adjacent only to 0 in the cell 13 and 0 in the cell 3 is adjacent only to 0 in the cell 2. These two pairs are combined and referred to as group 2 respectively



Step 4: The cells 8, 9, 12, 13 form a quad which forms the 3rd group



Step 5: The remaining 0 in the cell 0 is combined with the 0 in the cell 2 to form a pair, which is referred to as group 4.



Step 6: In group 1 and 4 variable C is eliminated. In group 2 variable D is eliminated and in group 3 variables B and D are eliminated. Therefore, we get simplified expression in SOP form as

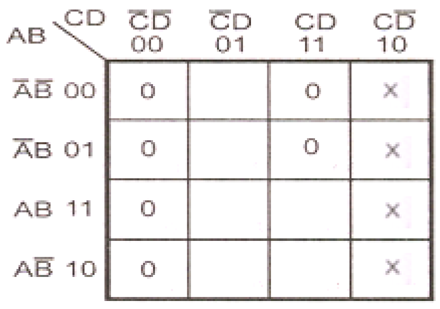
$\overline{F} = A\,B\,D + \overline{A}\,\overline{B}\,C + A\,\overline{C} + \overline{A}\,\overline{B}\,\overline{D}$

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

Reduce the following function using K-map

f (A, B, C, D) = ∏ m (0, 3, 4, 7,8, 10, 12, 14) + d (2, 6)

Step 1: Fig 4.11 (b) shows the K-map for four variables and it is plotted according to given maxterms and don’t care terms



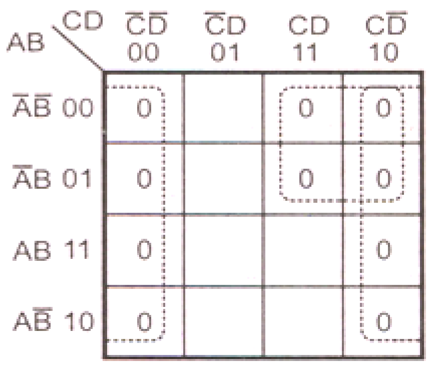
AB \ CD	$\overline{C}\,\overline{D}$ 00	$\overline{C}D$ 01	CD 11	$C\overline{D}$ 10
$\overline{A}\,\overline{B}$ 00	0		0	x
$\overline{A}B$ 01	0		0	x
AB 11	0			x
$A\overline{B}$ 10	0			x

Fig 4.11 (b)

Step 2: There are no isolated 0s.

Step 3: There is no such 0 which is adjacent to only one other 0

Step 4: By taking don’t cares as 0s we have one quad and one octet which forms group 1 and 2



AB \ CD	$\overline{C}\,\overline{D}$ 00	$\overline{C}D$ 01	CD 11	$C\overline{D}$ 10
$\overline{A}\,\overline{B}$ 00	0		0	0
$\overline{A}B$ 01	0		0	0
AB 11	0			0
$A\overline{B}$ 10	0			0

Step 5: Group 1 eliminates two variables B and D and group 2 eliminates 3 variables A, B and C. Therefore, we get simplified expression in SOP form as

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

$F = \overline{\overline{F}} = \overline{\bar{A}\,C + \bar{D}}$

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

4.12 Assignment Questions

Short answer questions

1) Define K-map? How is it useful in reducing a boolean expression
2) Using 2 variable K-Map reduce $AB + \bar{A}B$ (April/May 2010)
3) Using 3 variable K map reduce $A\bar{B}C + \bar{A}B\bar{C} + A\bar{C} + \bar{A}B$
4) Give the general structure of 3 and 4 variable K-maps (April/May 2007)
5) Define a QUAD and QCTET in a K-Map (April/May 2008)
6) What are minterm and maxterm? (April/May 2013)
7) Define SOP and POS forms with an example to each (April/May 2011, 2013)
8) What are don't care conditions? How are they useful in reducing an expression in K-map

Long answer questions

1) Simplify the boolean functions using Karnaugh Map method (April/May 2012)
   $F(x,y,z) = \sum(2,3,4,6)$
2) Simplify the boolean functions using Karnaugh Map method (April/May 2013)
   $F(x, y, z) = \Pi(0, 1, 4, 5)$
3) Using K-Map reduce the following expressions (April/May 2009)
   i) $\bar{A}\bar{B}C+AB\bar{C}+AB\bar{C}+AB\bar{C}+ABC\,\overline{}$
   ii) $(A + B+C)\,(\bar{A} + B+\bar{C})\,(\bar{A} + \bar{B}+C)\,(A + \bar{B}+\bar{C})\,(\bar{A} + \bar{B}+C)$
4) Using K-Map reduce the following expression (April/May 2012)
   $F(a, b, c, d) = \sum (0, 1, 2, 4, 5, 6, 8, 9, 12, 13 ,14)$
5) Using K-Map reduce the following expression
   $F(a, b, c, d) = \prod (1, 3, 5, 7, 8,11, 12, 14 ,15)$
6) Simplify the following expressions using K-Map (April/May 2009)
   i) $f = \sum (0, 3, 4, 7, 8) + d\,(10, 11, 12, 13, 14, 15)$
   ii) $f = \Pi\,(2, 3, 5, 7, 8, 14) + d\,(4, 6, 11)$
7) Express the boolean function $F = XY' + X'Z$ as sum of minterm and product of maxterm (April/May 2013)
8) Simplify the expression $AB + A\bar{B}\,.\,(\overline{\bar{A}.\bar{C}})$ (April/May 2011)

9) Obtain the simplified expression in (April/May 2008)

i) Sum of Products

ii) Product of sums for the xepression

x’z + y’z’+xyz

10) Obtain the simplified SOP expression for the following truth table and draw the logic circuit (April/May 2007)

Input			Output
A	B	C	Z
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

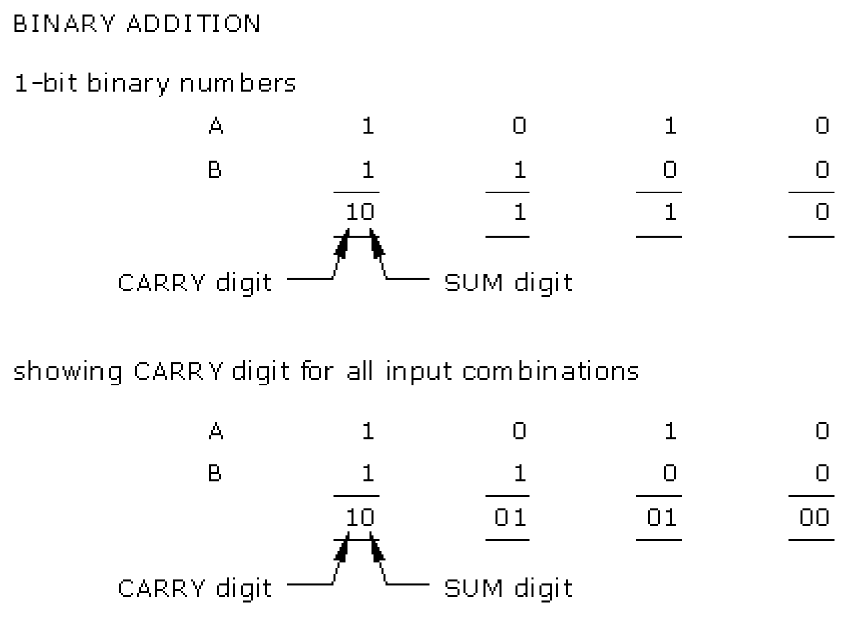
UNIT – III

Chapter 5. COMBINATIONAL LOGIC DESIGN

5.1 Adder circuits

Adder circuits are essential inside microprocessors as part of the **ALU**, or **arithmetic logic unit**, where the processing and manipulation of binary numbers takes place.

How does binary addition work? The simplest addition you can do is to add together two 1-bit binary numbers. Suppose the numbers are called A and B. Each of these numbers can take the values 0 or 1. There are four possible additions:



As you can see, in binary addition, 0+0=0, 0+1=1, 1+0=1 and 1+1=10. For each of these additions, you can identify a SUM digit and a CARRY digit.

In a modern computer, the adder circuitry will include the means of negating one of the input numbers directly, so the circuit can perform either addition or subtraction on demand. Other functions are commonly included in modern implementations of the adder circuit, especially in modern microprocessors.

5.1.1 Half-Adder

Half adder is a circuit which has only 2 binary inputs (A and B) and two outputs, which are the sum (S) and carry (C) bits

The truth table and logic diagram for a half-adder is given below

input A	*input B*	*SUM (S)*	*CARRY (C)*
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

From the above truth table we have:-

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

$C = AB$

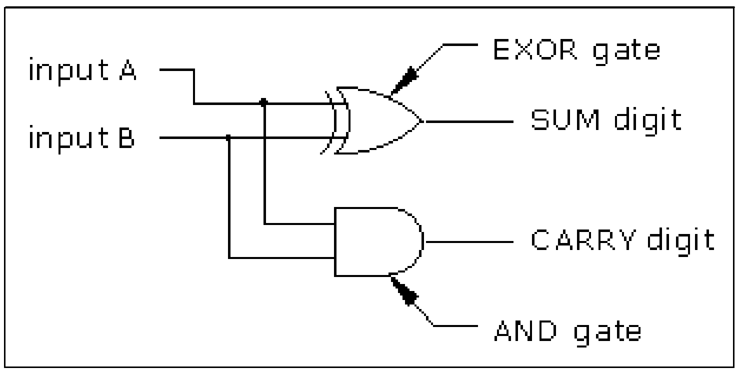
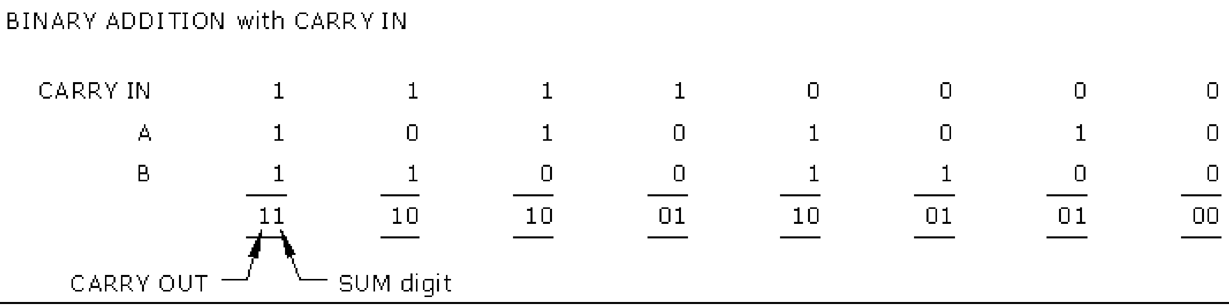


Fig. 5.1.1Logic Diagram for Half-Adder

5.1.2 Full-Adder

Full adder is a circuit which has 3 binary inputs - binary input A, binary input B and a Carry-in (C_{in}) from the previous column and produces two outputs, which are the sum (S) and carry-out (C_{out}) bits. You can make a full adder by linking together two half adder circuits:



```
BINARY ADDITION with CARRY IN

CARRY IN     1    1    1    1    0    0    0    0
       A     1    0    1    0    1    0    1    0
       B     1    1    0    0    1    1    0    0
            --   --   --   --   --   --   --   --
            11   10   10   01   10   01   01   00
```

input A	*input B*	*CARRY IN* (C_{in})	*SUM (S)*	*CARRYOUT* (C_{out})
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Truth table for a full-adder

From the above truth table we have:-

$S = \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + \bar{A}\bar{B}C_{in} + ABC_{in} = A \oplus B \oplus C_{in}$

$C = (A \oplus B)\, C_{in} + AB$

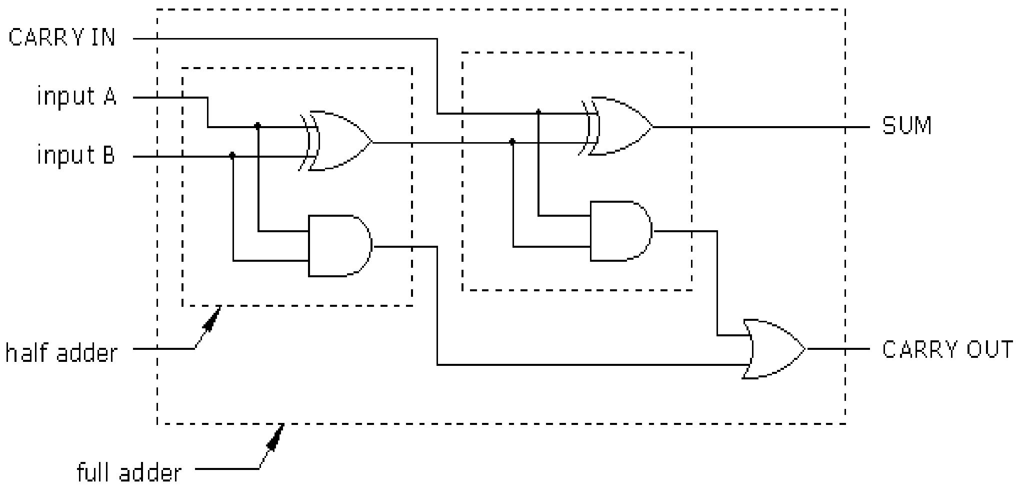


Fig. 5.1.2Logic Diagram for Full-Adder

5.2 SUBTRACTORS

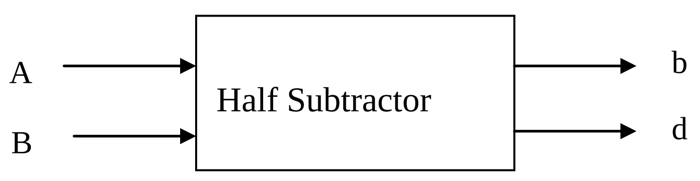
The subtraction of two binary numbers may be accomplished by taking the complement of the subtrahend and adding it to the minuend. By this method, the subtraction operation becomes an addition operation and instead of having a separate circuit for subtraction, the adder itself can be used to perform subtraction. In subtraction each subtrahend bit of the number is subtracted from its corresponding significant minuend bit to form a difference bit. If the minuend bit is smaller than the subtrahend bit, a 1 is borrowed from the next significant position.

5.2.1 The Half-Subtractor

The half-subtractor is a combinational circuit that subtracts one bit from the other and produces a difference. It also has an output to specify if a 1 has been borrowed. It is used to subtract the LSB of the subtrahend from the LSB of the minuend when one binary number is subtracted from the other.
A half-subtractor is shown below with two inputs A and B and two outputs d and b. d indicates the difference and b is the output signal which indicates that whether a 1 has been borrowed.

A	**B**	**d**	**b**
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(a) Truth Table



(b) Block Diagram

From the above truth table we have:-

$D = \overline{A}\,B + A\,\overline{B} = A \oplus B$ and $b = \overline{A}\,B$

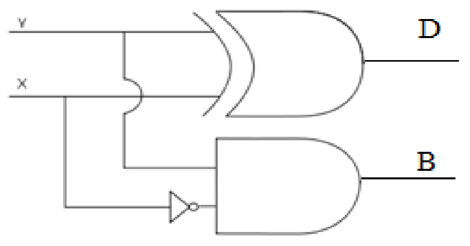
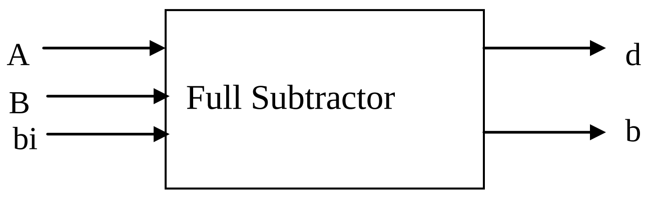


Fig. 5.2.1Logic Diagram for Half-Subtractor

5.2.2 The Full-Subtractor

The half-subtractor can be used only for LSB subtraction. If there is a borrow during the subtraction in the LSBs, it affects the subtraction in the next higher column; the subtrahend bit is subtracted from the minuend bit, considering the borrow from that column used for the subtraction in the preceding column. Such a subtraction is performed by a full-subtractor.
It subtracts one bit (B) from another bit (A), when already there is a borrow bi from this column for the subtraction in the preceding column, and outputs the difference bit (d) and the borrow bit (b) required from the next column. So a full-subtractor is a combinational circuit with three inputs (A, B, b) and two outputs d and b.

X	**Y**	**Z**	**d**	**b**
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



(a) Truth Table (b) Block Diagram

From the above truth table we have :-

$d = \bar{A}\,\bar{B}\,bi + \bar{A}\,B\,\overline{bi} + A\,\bar{B}\,\overline{bi} + A\,B\,bi$

$\quad = bi\,(AB + \bar{A}\,\bar{B}) + \overline{bi}\,(A\,\bar{B} + \bar{A}\,B)$

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

$d = \bar{A}\,\bar{B}\,bi + \bar{A}\,B\,\overline{bi} + \bar{A}\,B\,bi + A\,B\,bi$

$\quad = \bar{A}\,B\,(bi + \overline{bi}) + (AB + \bar{A}\bar{B})\,bi$

$\quad = \bar{A}B + (\overline{A \oplus B})\,bi$

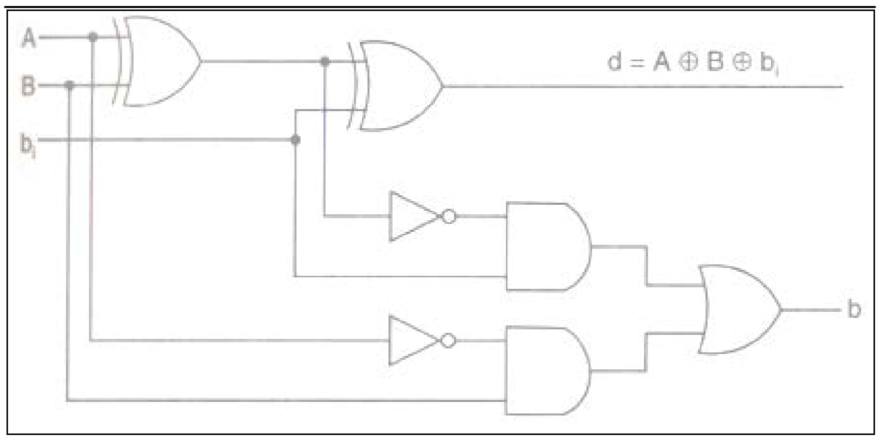


Fig. 5.2.2Logic Diagram for Full-Subtractor

5.3 BINARY PARALLEL ADDER

A binary parallel adder is a digital circuit that adds two binary numbers in parallel form and produces the arithmetic sum of those numbers in parallel form. It consists of full adders connected in a chain, with the output carry from each full-adder connected to the input carry of the next full adder in the chain.
The Figure below shows the interconnection of four full-adder (FA) circuits to providea4-bitparalleladder.The augend bits of A and addend bits of B are designated by subscript numbers from right to left, with subscript 1 denoting the lower-order bit. The carries are connected in a chain through the full-adders. The input carry to the adder is Cin and the output carry is C4. The S outputs generate the required sum bits.

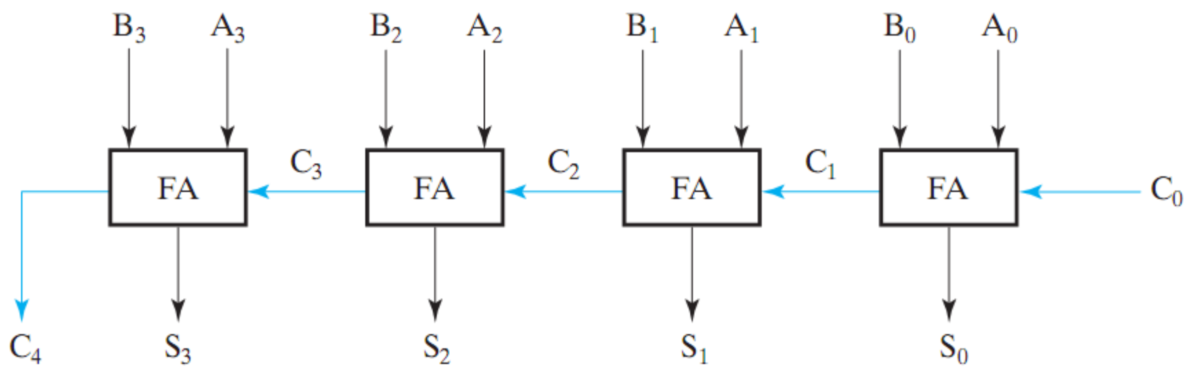


Fig. 5.3Logic Diagram for a 4-bitBinary Parallel Adder

5.4 4-BIT PARALLEL SUBTRACTOR

The subtraction of 2 binary numbers can be carried out most conveniently by means of complement. The subtraction A - B can be done by taking the 2'scomplement of B and adding it to A. The 2's complement can be obtained by taking the l's complement and then adding 1 to the least significant pair of bits. The l's complement can be implemented with inverters as shown in the figure 5.4

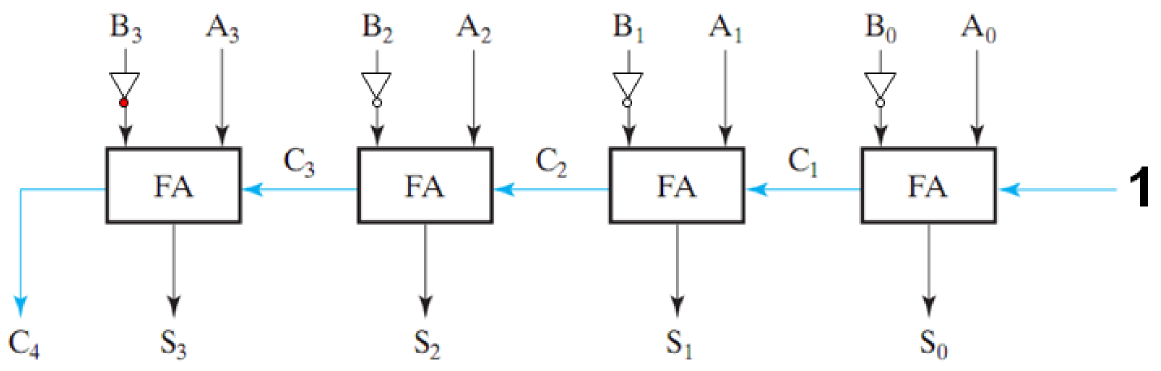


Fig. 5.4Logic Diagram for a 4-bitBinary Parallel Subtractor

5.6 BCD ADDER

The BCD addition process has the following steps:
1. Add the 4-bit BCD code groups for each decimal digit position using ordinary binary addition.
2. For those positions where the sum is 9 or less, the sum is in proper BCD form and no correction is needed.
3. When the sum of two digits is greater than 9, a correction of 0110 should be added to that sum, to produce the proper BCD result. This will produce a carry to be added to the next decimal position.

If the two BCD code groups A_3 A_2 A_1 A_0 and B_3 B_2 B_1 B_0 are applied to a 4-bitparallel adder, the adder will output S_4, S_3, S_2, S_1, S_0, where S_4, is actually C_4, the carry-out of the MSB bits.

The sum output S_4 S_3 S_2 S_1 S_0 can range anywhere from 00000 to 10010.The circuitry for a BCD adder must include the logic needed to detect whenever the sum is greater than 01001, so that the correction can be added in. Those cases, where the sum is greater than 1001 are listed in the table below

S4	S3	S2	S1	S0	Decimal Number
0	1	0	1	0	10
0	1	0	1	1	11
0	1	1	0	0	12
0	1	1	0	1	13
0	1	1	1	0	14
0	1	1	1	1	15
1	0	0	0	0	16
1	0	0	0	1	17
1	0	0	1	0	18

If we examine these cases, we see that X will be HIGH for either of the following conditions.

1) Whenever $S_4 = 1$ (sum greater than 15)
2) Whenever $S_3 = 1$ and either S2 or S1 or both are 1 (sums 10 to 15)

This condition can be expressed as
$X = S_4 + S_3 (S_2 + S_1)$

Whenever X =1, it is necessary to add the correction factor 0110 to the sum bits, and to generate a carry.

The circuit consists of three basic parts. The two BCD code groups $A_3 A_2 A_1 A_0$ and $B_3 B_2 B_1 B_0$ are added together in the upper 4-bit binary adder, to produce the sum $S_4 S_3 S_2 S_1 S_0$. The logic gates shown implement the expression for X. The lower 4-bit adder will add the correction 0110 to the sum bits, only when X =1, producing the final BCD sum output represented by $\Sigma_3 \Sigma_2 \Sigma_1 \Sigma_0$. When X =0, there is no carry and no addition of 0110. In such cases, $\Sigma_3 \Sigma_2 \Sigma_1 \Sigma_0 = S_3 S_2 S_1 S_0$

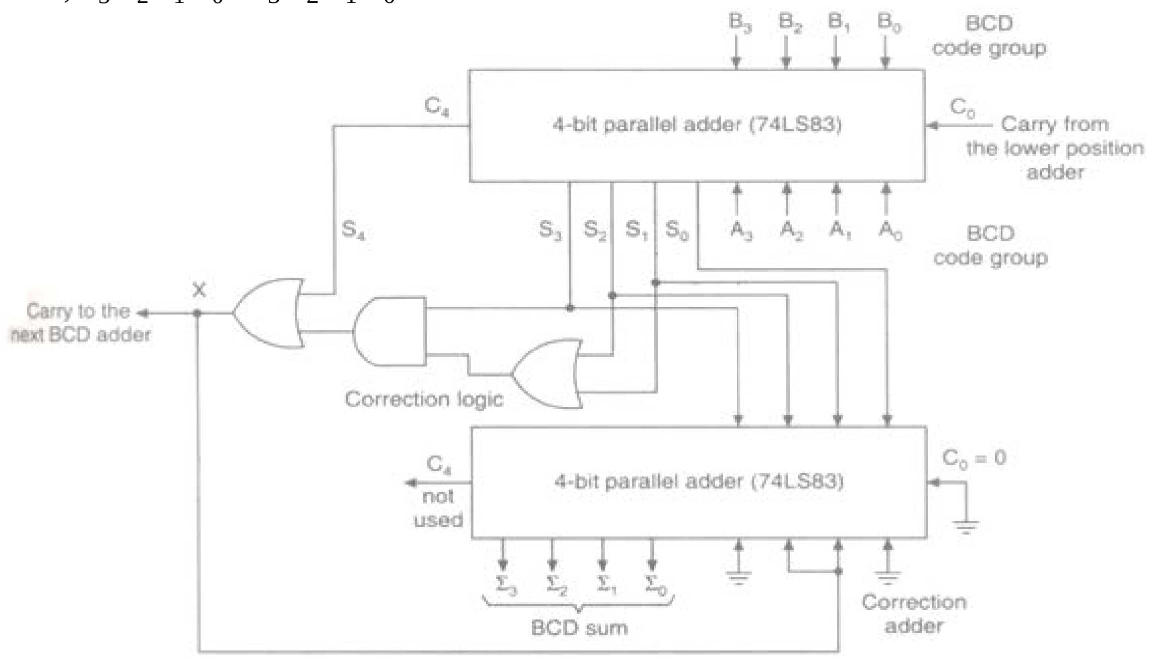


Fig. 5.6 Logic diagram for a BCD adder using two 4-bit adders and a correction-detector circuit

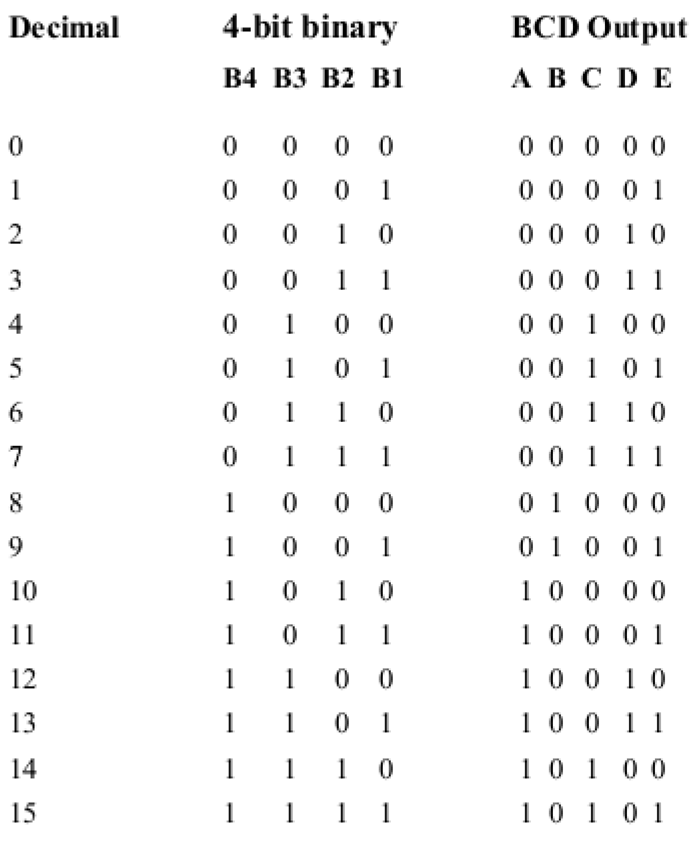
5.7 CODE CONVERTERS

A code converter is a logic circuit whose inputs are bit patterns representing numbers (or characters) in one code and whose outputs are the corresponding representations in a different code.

For example to convert from binary code A to binary code B, the input lines must supply the bit combination of elements as specified by code A and the output lines must generate the corresponding bit combination of code B. A combinational circuit performs this transformation by means of logic gates.
The logic expressions derived for the code converter can be simplified using K-Map and other usual techniques, including 'don't cares' if present.

Design of a 4-bit Binary-to-BCD Code Converter

The input is a 4-bit binary. There are 16 possible combinations of 4-bit binary inputs (representing0-15) and all are valid. Hence there are no don't cares. Since the input is of 4 bits (i.e. a maximum of 2 decimal digits), the output has to be an 8-bit one; but since the first three bits will all be a0for all combinations of inputs, the output can be treated as a 5-bit one. The conversion is shown in the conversion table below.



Decimal	4-bit binary				BCD Output				
	B4	B3	B2	B1	A	B	C	D	E
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	1
2	0	0	1	0	0	0	0	1	0
3	0	0	1	1	0	0	0	1	1
4	0	1	0	0	0	0	1	0	0
5	0	1	0	1	0	0	1	0	1
6	0	1	1	0	0	0	1	1	0
7	0	1	1	1	0	0	1	1	1
8	1	0	0	0	0	1	0	0	0
9	1	0	0	1	0	1	0	0	1
10	1	0	1	0	1	0	0	0	0
11	1	0	1	1	1	0	0	0	1
12	1	1	0	0	1	0	0	1	0
13	1	1	0	1	1	0	0	1	1
14	1	1	1	0	1	0	1	0	0
15	1	1	1	1	1	0	1	0	1

Conversion Table

From the conversion table, we observe that the expressions for BCD outputs are as follows:

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

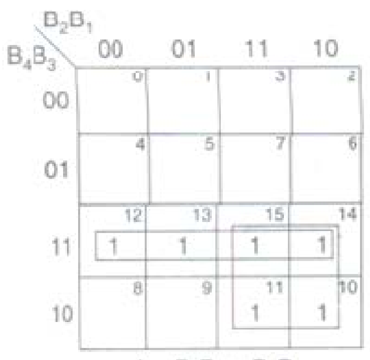
$B = \sum m\,(8, 9)$

$C= \sum m\,(4, 5, 6, 7, 14, 15)$

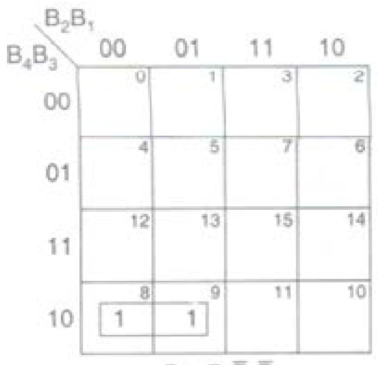
$D= \sum m\,(2, 3, 6, 7, 12, 13)$

$E = \sum m\,(1, 3, 5, 7, 9, 11, 13, 15)$

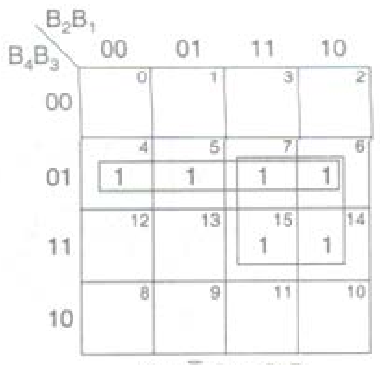
The K-maps for A, B, C, D and E and their minimizations in terms of the 4-bit binary inputs B_4, B_3, B_2, and B_1 are as follows:



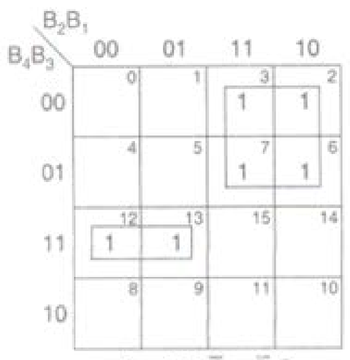
$A = B_4B_3 + B_4B_2$
K-map for A



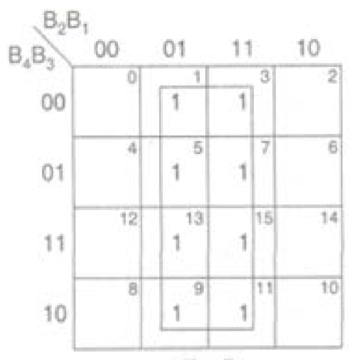
$B = B_4\overline{B}_3\overline{B}_2$
K-map for B



$C = \overline{B}_4B_3 + B_3B_2$
K-map for C



$D = B_4B_3\overline{B}_2 + \overline{B}_4B_2$
K-map for D



$E = B_1$
K-map for E

Thus the minimal expressions for the BCD outputs obtained from the K-map are:

$A = B4\ B3 + B4\ B2$
$B = B4\ \overline{B3}\ \overline{B2}$
$C = \overline{B4}\ B3 + B3\ B2$
$D = B4\ B3\ \overline{B2} + \overline{B4}\ B2$
$E = B1$

Design of a 4-bit BCD-to-XS-3 Code Converter

BCD means 8421 BCD. The 4-bit input BCD code (B_4 B_3 B_2 B_1) and the corresponding outputXS-3 code (X_4 X_3 X_2 X_1) numbers are shown in the conversion table below. The inputcombinations1010, 1011, 1100, 1101, 1110, and 1111 are invalid in BCD. So they are treated as don't cares.

BCD Code				X2-3 Code			
B4	B3	B2	B1	X4	X3	X2	X1
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

Conversion Table

From the conversion table, we observe that the expressions for the outputs X_4, X_3, X_2 and X_1 are as follows:

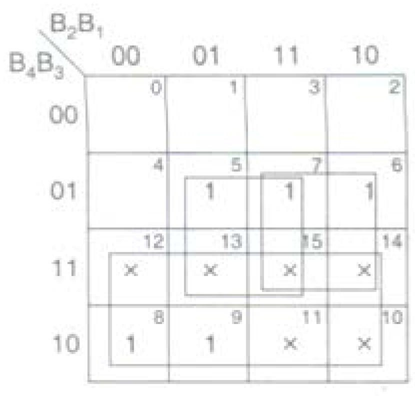
$X_4 = \sum m\,(5, 6, 7, 8, 9) + d\,(10, 11, 12, 13, 14, 15)$

$X_4 = \sum m\,(1, 2, 3, 4, 9) + d\,(10, 11, 12, 13, 14, 15)$

$X_4 = \sum m\,(0, 3, 4, 7, 8) + d\,(10, 11, 12, 13, 14, 15)$

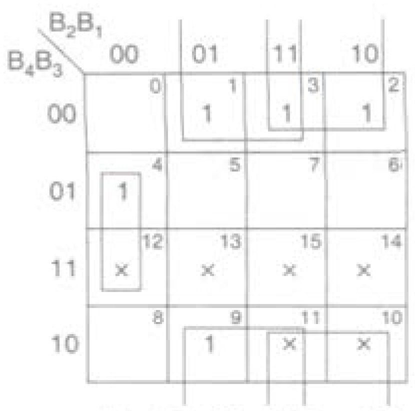
$X_4 = \sum m\,(0, 2, 4, 6, 8) + d\,(10, 11, 12, 13, 14, 15)$

The K-maps for X_4, X_3, X_2 and X_1 and their minimizations are shown below



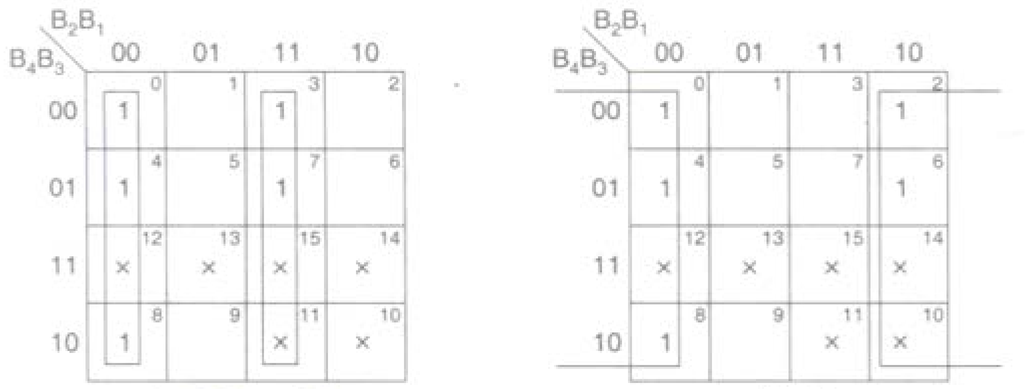
$X_4 = B_4 + B_3B_2 + B_3B_1$

K-map for X_4



$X_3 = B_3\bar{B}_2\bar{B}_1 + \bar{B}_3B_1 + \bar{B}_3B_2$

K-map for X_3



$X_2 = \overline{B}_2\overline{B}_1 + B_2B_1$
K-map for X_2

$X_1 = \overline{B}_1$
K-map for X_1

(c) K-maps

Thus the minimal expressions for the XS-3outputs obtained from the K-map are:

$X_4 = B_4 + B_3\,B_2 + B_3\,B_1$

$X_3 = B_3\,\overline{B}_2\,\overline{B}_1 + \overline{B}_3\,B_1 + \overline{B}_3\,B_2$

$X_2 = \overline{B}_2\,\overline{B}_1 + B_2\,B_1$

$X_1 = \overline{B}_1$

5.8 Magnitude Comparators

Comparator is a logic circuit used to compare the magnitudes of two binary numbers. Depending on the design, it may either simply provide an output that is active (goes HIGH for example) when the two numbers are equal, or additionally provide outputs that signify which of the numbers is greater or smaller when equality does not hold.
The X-NOR gate (coincidence gate) is a basic comparator, because its output is a 1 only if its two input bits are equal, i.e. the output is a 1 if and only if the input bits coincide.

5.8.1.1-bit Magnitude Comparator

The logic for a 1-bit magnitude comparator: Let the 1-bit numbers be A = Ao and B = Bo

If Ao = 1 and Bo = 0, then A> B.
Therefore A > B: G=Ao $\overline{\text{Bo}}$

If Ao = 0 and Bo = 1, then A < B.
Therefore, A < B: G=$\overline{\text{A}}$o Bo

If A0 and B0 coincide, i.e. Ao = Bo = 0 or if Ao = Bo = 1, then A = B.

Therefore, A = B: E =Ao ⊙ Bo

A_0	B_0	L	E	G
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

Truth table

L – Lesser than E – Equal G – Greater than

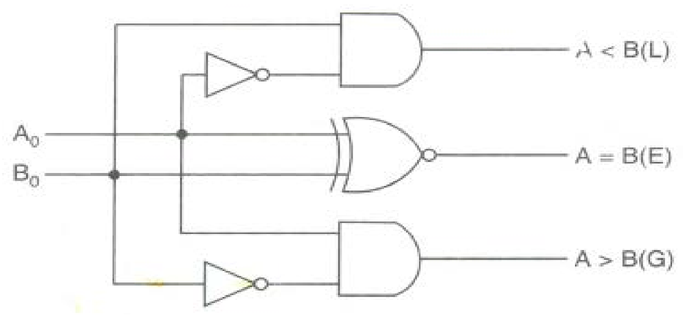


Fig 5.8 Logic diagram

5.8.2 2-bit Magnitude Comparator

The logic for a 2-bit magnitude comparator: Let the two 2-bit numbers be $A = A_1 A_0$ and $B = B_1 B_0$

1. If $A_1 = 1$ and $B_1 = 0$, then $A > B$ or
2. IfA_1 and $B_1$are equal and $A_0 = 1$ and $B_0 = 0$, then $A > B$.

So the logic expression for $A > B$ can be written as

$A>B: G=A_1 \overline{B}_1 + (A_1 \odot B_1) A_0 \overline{B}_0$

1. If $A_1 = 0$ and $B_1 = 1$, then $A < B$ or
2. If A_1 and B_1 coincide and Ao = 0 and Bo =1, then A< B.

So the expression for $A < B$ is

$A<B: G=\overline{A}_1 B_1 + (A_1 \odot B_1) \overline{A}_0 B_0$

If A_1 and B_1 coincide and if Ao and Bo coincide then A = B. So the expression for A = B is

$A = B: (A_1 \odot B_1) (A_0 \odot B_0)$

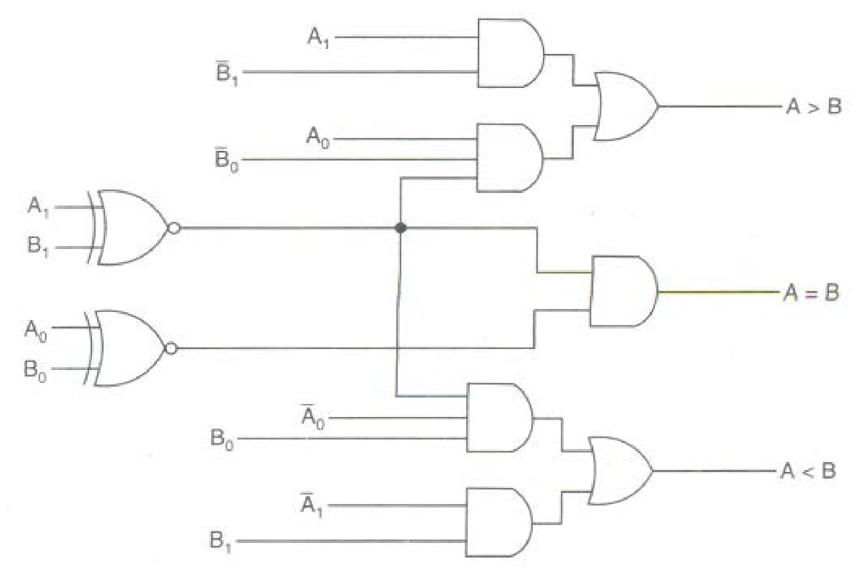


Fig 5.8.2 Logic diagram for a 2-bit comparator

5.8.3 4-bit Magnitude Comparator

The logic for a 4-bit magnitude comparator: Let the two 4-bit numbers be A = A_3 A_2 A_1 A_0 and B= B_3 B_2 B_1 B_0 1. If A_3 = 1 and B_3 = 0, then A > B. Or

2. If A_3 and B_3 .coincide, and if A_2 = 1 and B_2= 0, then A > B. Or
3. If A_3 and B_3 coincide, and if A_2 and B_2 coincide, and if A_1 = 1 and B_1 = 0, then A> B. Or
4. If A_3 and B_3 coincide, and if A_2 and B_2 coincide, and if A_1 and B_1 coincide, and if A_0 = 1 and B_0 = 0, then A > B.

From these statements, we see that the logic expression for A > B can be written as

$A>B: G= A_3 \overline{B}_3 + (A_3 \odot B3) A_2 \overline{B}_2 + (A_3 \odot B_3)(A_2 \odot B_2) A_1 \overline{B}_1 + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1) A_0 \overline{B}_0$

Similarly, the logical expression for A < B can be written as

$A<B: G= \overline{A}_3 B_3 + (A_3 \odot B3) \overline{A}_2 B_2 + (A_3 \odot B_3)(A_2 \odot B_2) \overline{A}_1 B_1 + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1) \overline{A}_0 B_0$

If A_3 and B_3 coincide and if A_2 and B_2 coincide and if A_l and B_1 coincide and if A_0 and B_0 coincide, then A = B.
So the expression for A = B can be written as
$A = B: (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)$

The logic diagram for a 4-bit magnitude comparator is as shown in fig 5.8.3

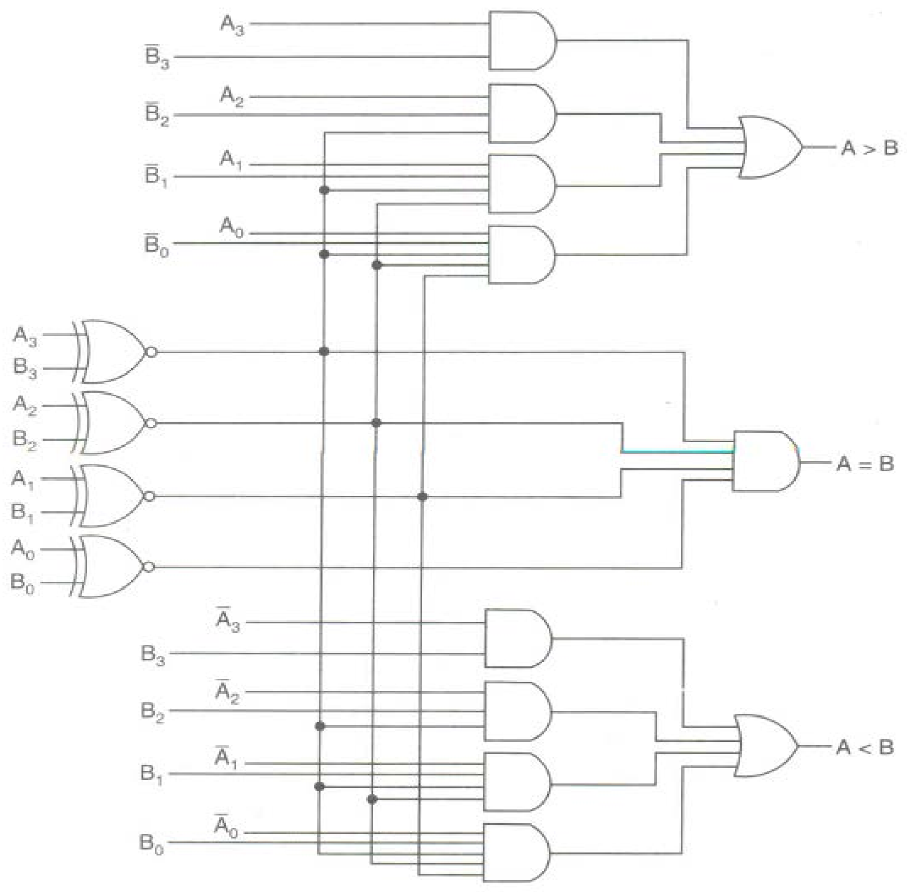


Fig 5.8.3logic diagram for a 4-bit comparator

5.9 Encoders

An encoder is device whose inputs are decimal digits or any other characters and whose outputs are the coded representations of those inputs. That is encoder is a device which converts familiar numbers or symbols into coded format.

5.9.1 Octal-to-Binary Encoder

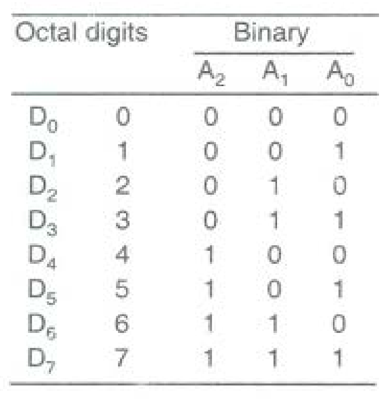
An octal-to-binary encoder (8-line to 3-line encoder) accepts 8 input lines and produces a 3-bit output code corresponding to the activated input. The truth table and the logic circuit for an octal-to-binary encoder with active HIGH inputs is given below

From the truth table, we see that A_2 is a 1 if any of the digits D_4 or D_5 or D_6 or D_7 is a 1. Therefore we have,

$A_2 = D_4 + D_5 + D_6 + D_7$

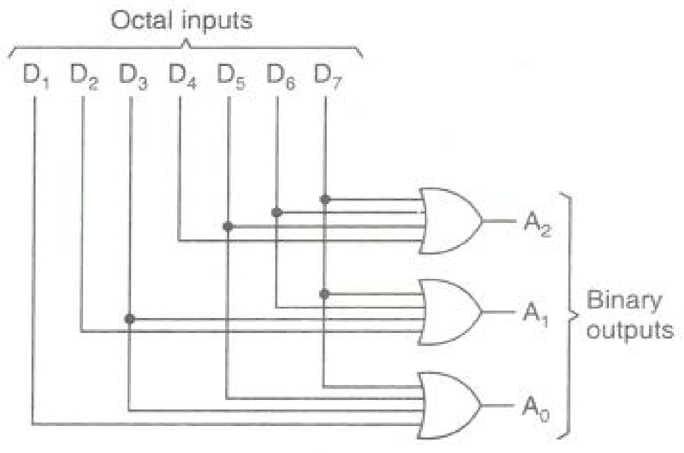
$A_1 = D_2 + D_3 + D_6 + D_7$

$A_0 = D_1 + D_3 + D_5 + D_7$



Octal digits		Binary		
		A_2	A_1	A_0
D_0	0	0	0	0
D_1	1	0	0	1
D_2	2	0	1	0
D_3	3	0	1	1
D_4	4	1	0	0
D_5	5	1	0	1
D_6	6	1	1	0
D_7	7	1	1	1

(a) Truth table



(b) Logic diagram

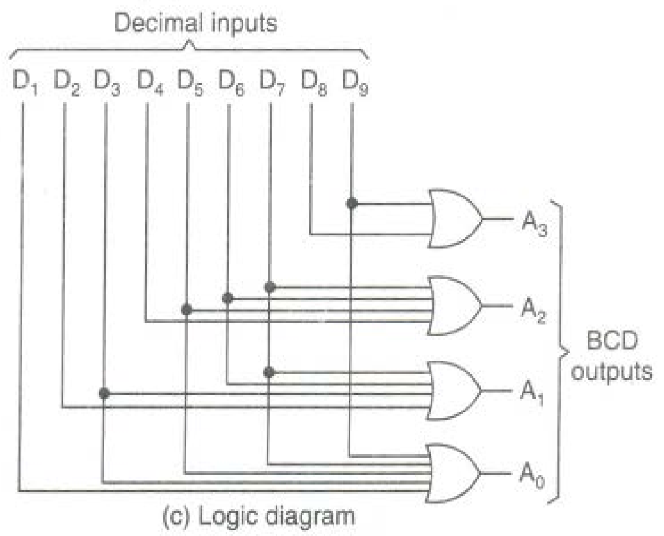
5.9.2 Decimal-to-BCD Encoder

This type of encoder has 10 inputs-one for each decimal digit, and 4 outputs corresponding to the BCD code as shown in the figure below. This is a basic 10-line to 4-line encoder. The BCD code is listed in the truth table given below and from this we can determine the relationship between each BCD bit and the decimal digits.

The truth table and the logic diagram for a decimal-to-BCD encoder is shown below

Decimal inputs		Binary			
		A_3	A_2	A_1	A_0
D_0	0	0	0	0	0
D_1	1	0	0	0	1
D_2	2	0	0	1	0
D_3	3	0	0	1	1
D_4	4	0	1	0	0
D_5	5	0	1	0	1
D_6	6	0	1	1	0
D_7	7	0	1	1	1
D_8	8	1	0	0	0
D_9	9	1	0	0	1

(b) Truth table



(c) Logic diagram

From the truth table, we see that the bit values for the BCD outputs are as follows

$A_3 = D_8 + D_9$

$A_2 = D_4 + D_5 + D_6 + D_7$

$A_1 = D_2 + D_3 + D_6 + D_7$

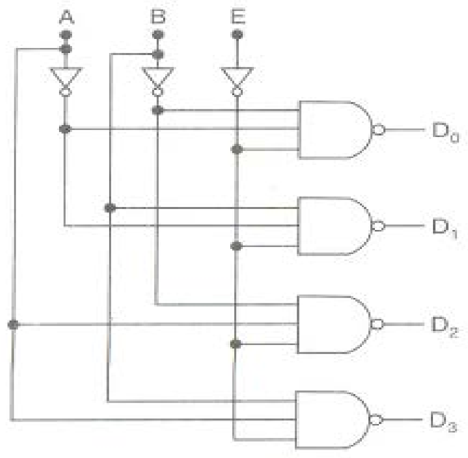
$A_0 = D_1 + D_3 + D_5 + D_7 + D_9$

5.10 Decoders

A decoder is a logic circuit which converts a N-bit binary input into M output lines such that only one output line is activated for each one of the possible combination of inputs.

5.10.1 2-Line-to-4-line Decoder

A 2-to-4 line decoder with an enable input is constructed with NAND gates. The decoder is enabled when E = 0. The inputs are active high and outputs active low as indicated by the truth table, only one output can be equal to 0 at any given time, all other outputs are equal to 1. The circuit is disabled when E = l, regardless of the values of the other two inputs.



(a) Logic diagram

E	A	B	D_0	D_1	D_2	D_3
1	×	×	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

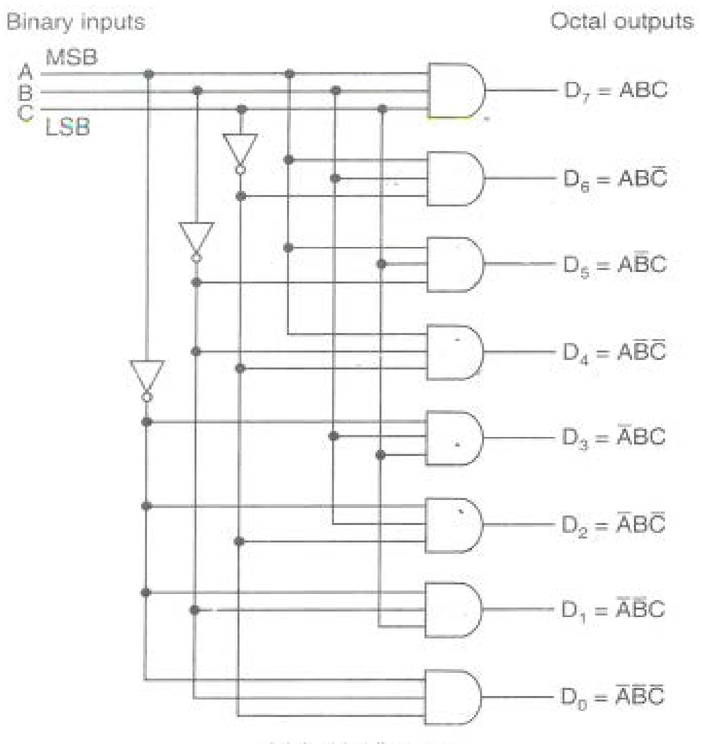
(b) Truth table

5.10.2 3-Line-to-8-Line Decoder

It can be called a 3-line to 8-line decoder because it has three input lines and eight output lines. It is also called a binary-to-octal decoder because it takes a 3-bit binary input code and activates one of the eight (octal) outputs corresponding to that code.

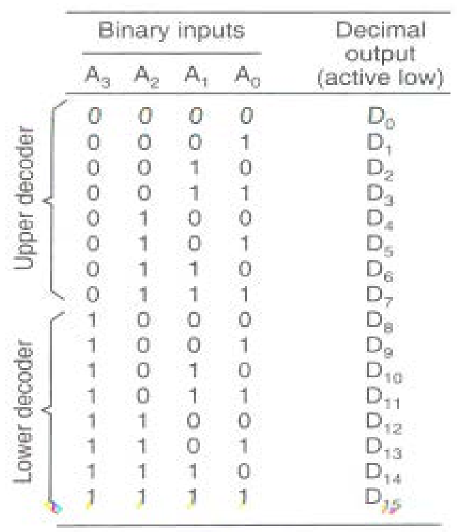
Inputs			Outputs							
A	B	C	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

(b) Truth table


Binary inputs
Octal outputs
A MSB
B
C LSB
$D_7 = ABC$
$D_6 = AB\overline{C}$
$D_5 = A\overline{B}C$
$D_4 = A\overline{B}\overline{C}$
$D_3 = \overline{A}BC$
$D_2 = \overline{A}B\overline{C}$
$D_1 = \overline{A}\overline{B}C$
$D_0 = \overline{A}\overline{B}\overline{C}$
(a) Logic diagram

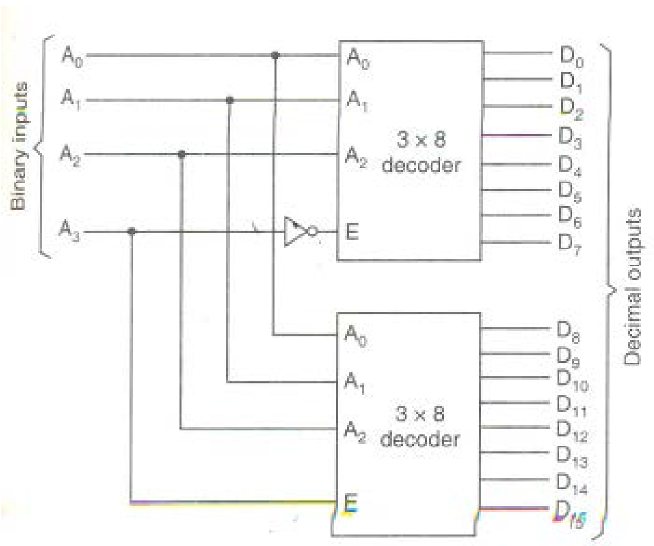
5.10.3 4-to-16 Decoder from Two 3-to-8 Decoders

Decoders with enable inputs can be connected together to form a larger decoder circuit. The most significant input bit A3 is connected through an inverter to E on the upper decoder (for D_0 to D_7) and directly to E on the lower decoder (for D_8 to D_{15}), Thus, when A3 is LOW, the upper decoder is enabled and the lower decoder is disabled. When A3 is HIGH, the lower decoder is enabled and the upper decoder is disabled. The bottom decoder outputs generate minterms 1000 to 1111, while the upper decoder outputs generate minterms 0000 to 0111.



	Binary inputs				Decimal output (active low)
	A_3	A_2	A_1	A_0	
Upper decoder	0	0	0	0	D_0
	0	0	0	1	D_1
	0	0	1	0	D_2
	0	0	1	1	D_3
	0	1	0	0	D_4
	0	1	0	1	D_5
	0	1	1	0	D_6
	0	1	1	1	D_7
Lower decoder	1	0	0	0	D_8
	1	0	0	1	D_9
	1	0	1	0	D_{10}
	1	0	1	1	D_{11}
	1	1	0	0	D_{12}
	1	1	0	1	D_{13}
	1	1	1	0	D_{14}
	1	1	1	1	D_{15}

(b) Function table


Binary inputs
A_0
A_1
A_2
A_3
3 × 8 decoder
E
D_0 D_1 D_2 D_3 D_4 D_5 D_6 D_7
D_8 D_9 D_{10} D_{11} D_{12} D_{13} D_{14} D_{15}
Decimal outputs
(a) Logic diagram

5.11 MULTIPLEXERS (DATA SELECTORS)

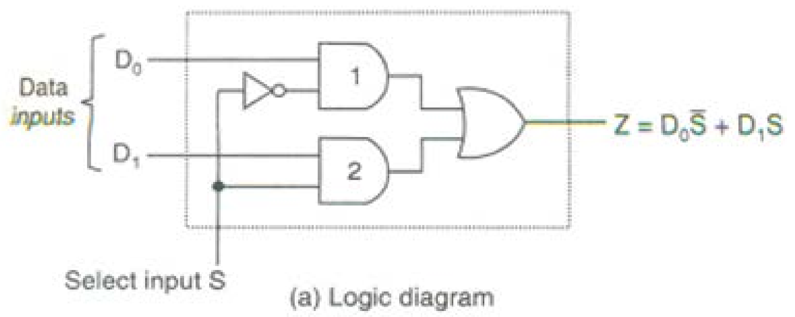
A multiplexer (MUX) or data selector is a logic circuit that accepts several data inputs and allows only one of them at a time to get through to the output. The routing of the desired data input to the output is controlled by SELECT inputs.

5.11.1 Basic 2-lnput Multiplexer

A 2-input multiplexer has two data inputs Do and D_1, and one data select input S. It connects two 1-bit sources to a common destination. It has two input lines, one data select line and one output line. The logic level applied to the S input determines which AND gate is enabled, so that its data input passes through the OR gate to the output.

S	Output (Z)
0	D_0
1	D_1

(b) Function table



(a) Logic diagram

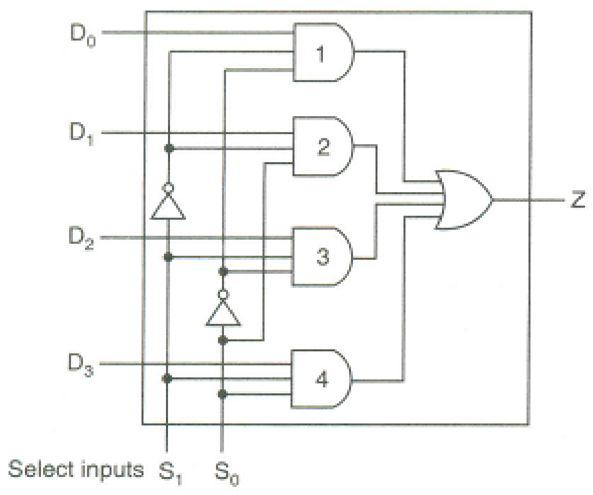
When S = 0, AND gate 1 is enabled and AND gate 2 is disabled. So, Z = D_0
When S = 1, AND gate 1 is disabled and AND gate 2 is enabled. So, Z = D_1
From the above truth table we see that the output, Z = $D_0 \bar{S} + D_1S$.

5.11.2 The 4-lnput Multiplexer

The 4-input multiplexer has 4 data inputs D_0, D_1, D_2 and D_3 and 2 data select inputs S_0 and S_1. The logic levels applied to the S0 and S1, inputs determine which AND gate is enabled, so that its data input passes through the OR gate to the output.

Select inputs		Output
S_1	S_0	Z
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

(b) Function table



(a) Logic diagram

From the above truth table we see that the output:

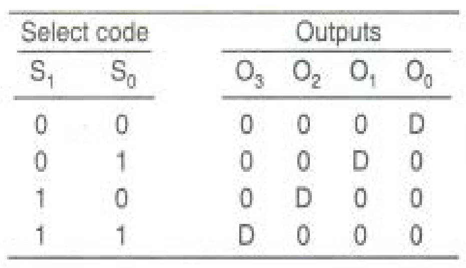
$$Z = \bar{S}_1\bar{S}_0D_0 + \bar{S}_1S_0D_1 + S_1\bar{S}_0D_2 + S_1S_0D_3$$

5.12 DEMULTIPLEXERS (DATA DISTRIBUTORS)

A multiplexer is a circuit which takes several inputs and transmits one of them to the output. Whereas a demultiplexer (DEMUX) performs the reverse operation; it takes a single input and distributes it over several outputs. So a demultiplexer can be thought of as a 'distributor', since it transmits the same data to different destinations.

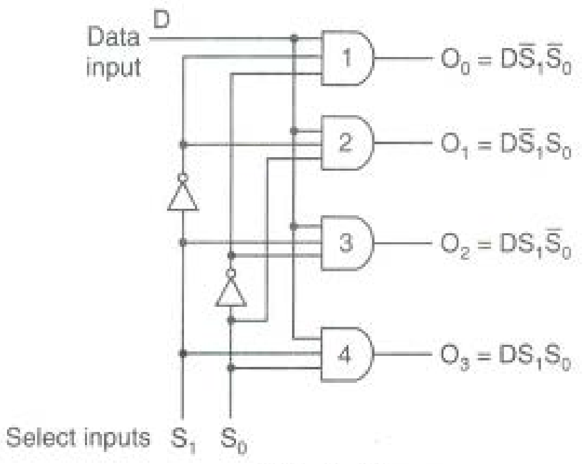
5.12.1 1-Line to 4-Line Demultiplexer

In a 1-line to 4-line demultiplexer circuit, the 4 input data lines goes to all of the AND gates. The two select lines S_0 and S_1, enable only one gate at a time, and the data appearing on the input line will pass through the selected gate to the associated output line.



Select code		Outputs			
S_1	S_0	O_3	O_2	O_1	O_0
0	0	0	0	0	D
0	1	0	0	D	0
1	0	0	D	0	0
1	1	D	0	0	0

(b) Truth table



(a) Logic diagram

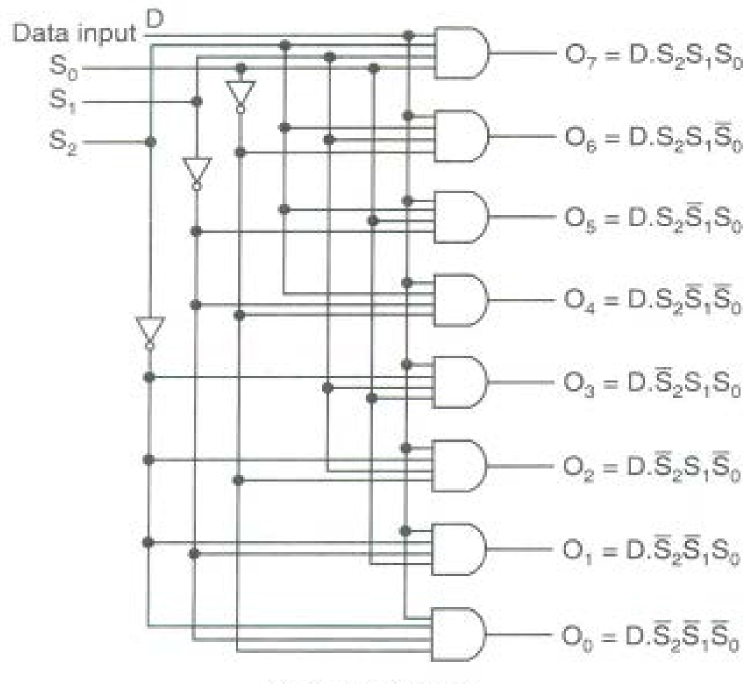
5.12.2 1-Line to 8-Line Demultiplexer

This demultiplexer distributes one input line to eight output lines. The single data input line D is connected to all eight AND gates, but only one of these gates will be enabled by the select input lines. For example, with S_2 S_1 S_0 = 000, only the AND gate O_0 will be enabled, and the data input D will appear at output $O_{0.}$ Other select codes cause input D to reach the other outputs.

The logic diagram and truth table for a 1-line to 8-line demultiplexer is as follows

Select code			Outputs							
S_2	S_1	S_0	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0	D
0	0	1	0	0	0	0	0	0	D	0
0	1	0	0	0	0	0	0	D	0	0
0	1	1	0	0	0	0	D	0	0	0
1	0	0	0	0	0	D	0	0	0	0
1	0	1	0	0	D	0	0	0	0	0
1	1	0	0	D	0	0	0	0	0	0
1	1	1	D	0	0	0	0	0	0	0

(b) Truth table


Data input D
S_0
S_1
S_2
$O_7 = D.S_2S_1S_0$
$O_6 = D.S_2S_1\overline{S}_0$
$O_5 = D.S_2\overline{S}_1S_0$
$O_4 = D.S_2\overline{S}_1\overline{S}_0$
$O_3 = D.\overline{S}_2S_1S_0$
$O_2 = D.\overline{S}_2S_1\overline{S}_0$
$O_1 = D.\overline{S}_2\overline{S}_1S_0$
$O_0 = D.\overline{S}_2\overline{S}_1\overline{S}_0$

(a) Logic diagram

5.13 Assignment Questions

Short answer questions

1) Write the truth table and logic diagram of half adder. (2013,2009,2007)
2) What is full adder? Write the truth table of full adder. (April/May 2012)
3) What is a magnitude comparator? Write the logic expression of 1-bit magnitude comparator. (April/May 2013, 2012)
4) Write the truth table and logic diagram of half subtractor.
5) What is full adder? Write the truth table of full subtractor. (April/May 2013,2009)
6) What is a code converter? Give an example
7) What is an encoder? Give an example (April/May 2013,May/June 2010)
8) Write the truth table for 2 to 4 lines decoder (April/May 2013)
9) What is a Multiplexer? Give the truth table and logic diagram for a 2 input multiplexer.
10) What is a Demultiplexer? Give the truth table for a 1-Line to 4-Line Demultiplexer

Long answer questions

1) With the help of a suitable diagram explain the working of a Half-Adder circuit along with the truth table (April/May 2012)
2) Explain the Full-Adder circuit using truth table and represent the Sum(S) and Carry(C) bits using logic gates (April/May 2013, May/June 2010)
3) Explain the Half-Subtractor circuit using truth table and a suitable diagram
4) Explain the working of a Full-Subtractor using truth table and a logic diagram
5) Explain the 4-bit binary parallel adder with logic diagram. (2013,2011,2010,2009,2008,2007)
6) Explain the 4-bit binary parallel subtractor with logic diagram
7) With a circuit diagram explain the working of a BCD adder (2013,2012,2011,2010,2009,2008,2007)
8) Design of a 4-bit Binary-to-BCD Converter
9) Design of a 4-bit BCD to Excess-3 Code Converter (April/May 2013)
10) With the help of a truth table and logic diagram explain the 2-bit magnitude comparator (2011,2010,2008,2007)
11) Explain the 4-bit magnitude comparator with truth table and suitable diagram
12) What are encoders? Explain the Octal-to-Binary with truth table and logic diagram
13) With the help of truth table and logic diagram explain the Decimal-to-BCD encoder
14) What are decoders? Explain the 2-to-4 line decoder with truth table and logic diagram
15) With the help of truth table and logic diagram explain the 3 line to 8 line decoder
16) What is multiplexer? Explain the 2-input multiplexer with truth table and logic diagram (April/May 2013)
17) With the help of truth table & logic diagram explain the 4-input multiplexer (April/May 2013)
18) What is a Demultiplexer? Explain the 1-line to 4-line demultiplexer with truth table and a logic diagram
19) With the help of truth table and logic diagram explain 1-line to 8-line demultiplexer

UNIT – IV

Chapter 6. SEQUENTIAL LOGIC DESIGN

6.1 INTRODUCTION :

Switching circuits are classified as combinational switching circuits or sequential switching circuits. Combinational switching circuits are those whose output levels at any instant of time are dependent only on the levels present at the inputs at that time. Any prior input level conditions have no effect on the present outputs because these have no memory.
On the other hand, sequential circuits are those whose output levels at any instant of time dependent not only on the levels present at the input at that time, but also on the prior input conditions. i.e., they have memory.

Sequential Circuits = Combinational Circuits + Memory

Both combinational circuits and memory circuits are made up of logic gates. Flip-flops are used as memory elements in all computers.

6.1.1 Flip-Flops :-

Flip-Flop is a basic electronic circuit used to store binary information in all digital computers. The two essential characteristics of all Flip-flops are :
(a)A flip-flop is a bistable device (has two stable states) & can store information either zero or one.
(b) A flip-flop can have two outputs and they are always complimentary to each other.

The block diagram of a particular type of flip-flop is shown in Figure 6.1.

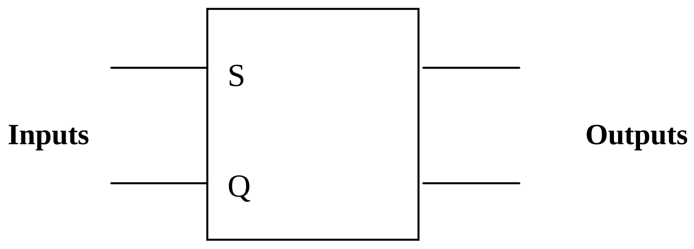


Figure 6.1: Block diagram of SR flip-flop.

The SR flip-flop contains two inputs designated as SET input (S) and RESET input (R) and two outputs marked as Q and Q'. These outputs are always complementary to each other (i.e., if Q = 1 state, then Q' = 0 state or visa-versa). The state of the flip-flop is considered as the state of Q output.

The state of the flip-flop is controlled by the state of input line S and R. There are 4 possibilities:
(a)When both inputs S and R are at 0 states, the flip-flops output state does not change. i.e., it will continue in its previous state.

(b) When the state of S input is 1 and the state of R input is 0, then, the flip-flop sets to 1 state. i.e., output Q becomes 1. This condition is called **Setting** the flip-flop.
(c)When the state of S input is 0 and the state of R input is 1, then, the flip-flop resets to 0 state. i.e., output Q becomes 0. This condition is called **Resetting** the flip-flop.
(d) When the state of S input is 1 and the state of R input is also 1, then, the flip-flop output Q can go to either state (ambiguous). Hence this condition of input is prohibited.
The following table gives the possible combinations of inputs and resultant output.

S	R	Q_n	Q_{n+1}	
0	0	0	0	Last state
0	0	1	1	
1	0	0	1	Set
1	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	1			Not allowed

6.2 SR Flip-flop using NOR gate:

The basic SR flip-flop circuit can be constructed using two NOR gates as shown below.

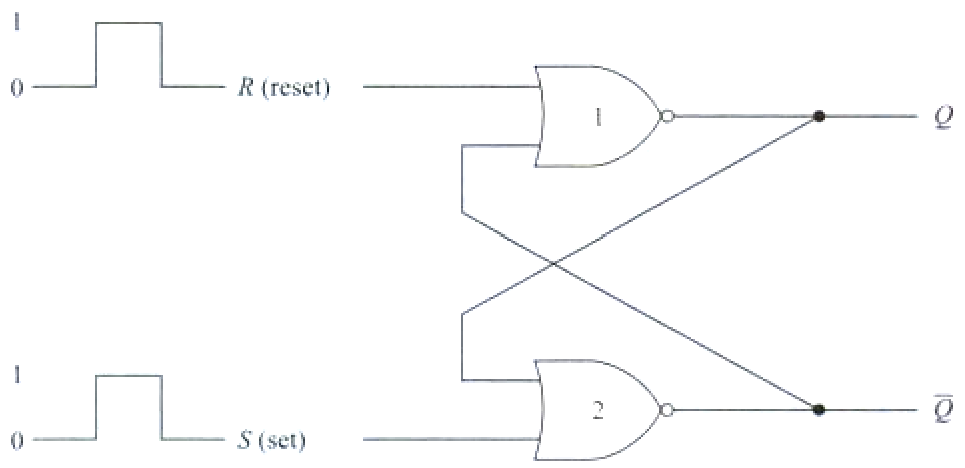
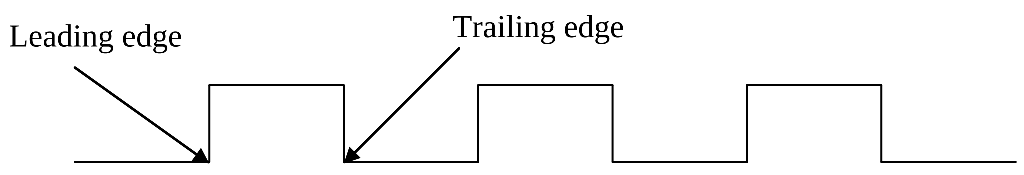


Figure 6.2 SR Flip-flop using NOR gate

6.3 Clocks :-

A flip-flops response can be controlled by connecting a clock pulse input with it. The operation of such a circuit with clock pulse input is called synchronous operation. In this case the flip-flop gives the proper output only when the clock pulse is present.

A clock pulse is a square or rectangular waveform voltage used to control the action of flip-flop. A typical clock waveform is shown in figure 6.3.



The leading edge of clock pulse is also called positive going or rising edge of the signal. The trailing edge of clock pulse is also called negative going or falling edge.

In some flip-flops, the positive going edge of clock pulse initiates the flip-flop action and the symbol for such flip-flop is shown in Figure 4 (a). In some other flip-flops, negative edge of clock pulse initiates the flip-flop action and the symbol for such flip-flop is shown in Figure 4 (b).

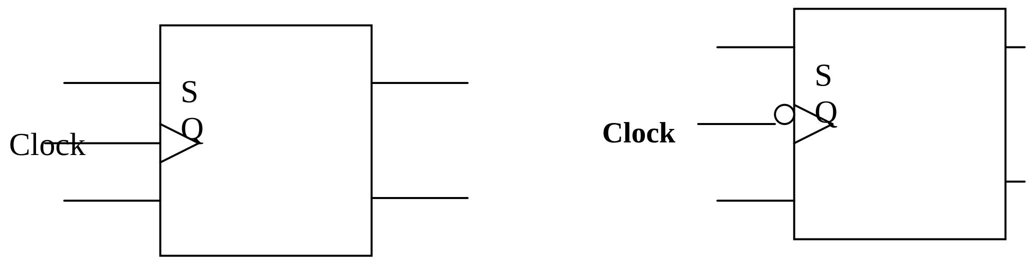


Figure 6.4 (a) SR flip-flop with active positive clock pulse. (b) Active negative clock pulse.

It is important to note that most of the Flip-flops respond to **change in clock level** rather than clock level itself.
Figure 4(c) shows the changes in flip-flop output during positive edge of the clock pulse and such flip-flops are called **positive edge triggered** flip-flops. Figure 4(d) shows the changes in flip-flop output during negative edge of the clock pulse and such flip-flops are called **negative edge triggered** flip-flops
The basic rules for the operation of a flip-flop are:

1. If the S and R inputs are 0s when the clock edge occurs, the flip-flop does not change states but remains in its present state (last state condition).
2. If the S input is a 1 and the R input is 0, when the clock edge occurs, the flip-flop goes to 1 state (Set condition).
3. If the S input is a 0 and the R input a 1, when the clock edge occurs, the flip-flop is cleared to 0 state (Reset condition).
4. Both S and R inputs should not be 1s when the clock edge occurs (prohibited condition).

6.4 Flip-Flop Designs:-

6.4.1 Basic RS flip-flop: The basic RS flip-flop without clock input is also called SR latch. The internal circuit of such flip-flop can be constructed using two NOR gates or two NAND gates as shown in Figure 5 (a) and 5 (b) respectively

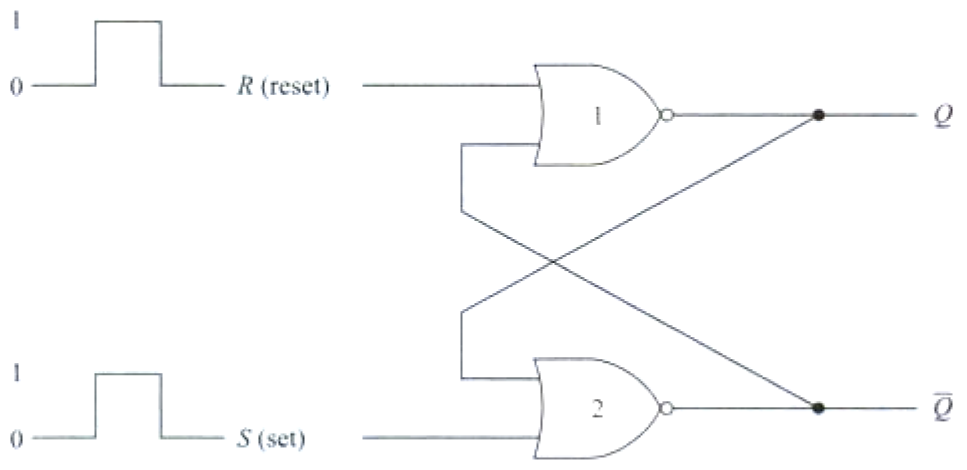


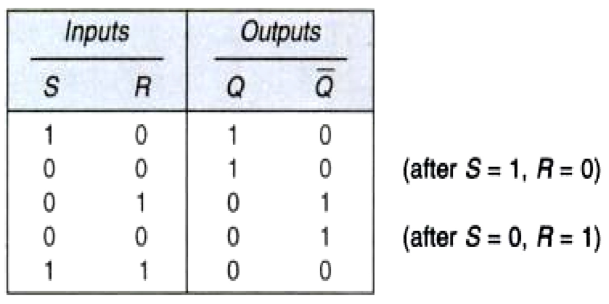
Fig. 6.5 (a) Basic SR flip-flop using NOR gates.

Circuit Operation

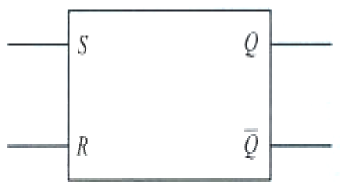
We know that output of a NOR gate is 0 if any input is 1, and that the output is 1 only when all the inputs are 0. To begin with let us assume that the set input is 1and the reset input is 0. Since gate 2 has an input of 1, its output $\overline{Q}$ must be 0, which places both inputs of gate 1 at 0, making the output Q equal to 1. When the set input is returned to 0,the outputs remain the same , because the output Q remains a 1, leaving one input of gate 2 at 0, so that output $\overline{Q}$ is a 0. In a similar fashion it is possible to show that a 1 in the reset input changes the output Q to 0 and $\overline{Q}$ to 1. When the reset input returns to 0, the output do not change.

When a 1 is applied to both the set and the reset inputs, both $\overline{Q}$ and Q output go to 0. This condition violates the fact the outputs $\overline{Q}$ and Q are the complements of each other. In normal operation, this condition is avoided by ensuring that 1s are not applied to both the inputs simultaneously.

Table 5.1 Truth table of basic flip-flop circuit with two NOR gates



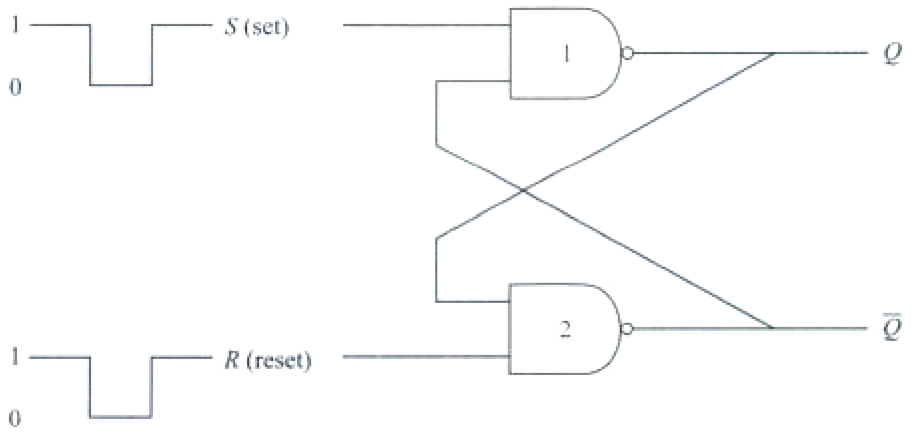
Inputs		Outputs		
S	R	Q	$\overline{Q}$	
1	0	1	0	
0	0	1	0	(after S = 1, R = 0)
0	1	0	1	
0	0	0	1	(after S = 0, R = 1)
1	1	0	0	



Logical Symbol of RS flip flop

We have thus seen that a flip flop has two useful states. When Q=1 and $\overline{Q}$ = 0, the flip-flop is in the set state(or 1-state). When Q =0 and $\overline{Q}$ = 1, the flip-flop is in the clear or reset state(or 0-state). The outputs Q and $\overline{Q}$ are the complements of each other and are referred to as the normal and complement outputs, respectively. The binary state of the flip-flop is taken to be the value of the normal output.

RS flip-flop using NAND gates



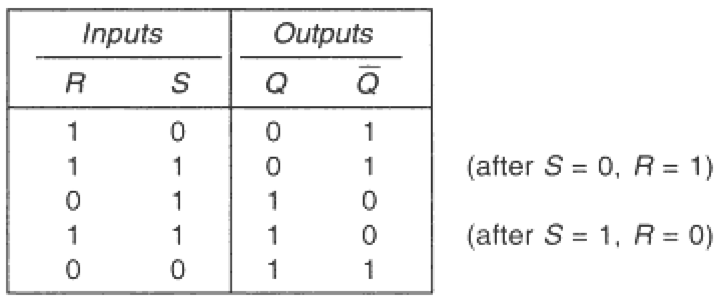
Logic Diagram of basic flip-flop circuit with two NAND gates

Circuit Operation

The basic flip-flop circuit with two NAND gates is shown above. It operates with both inputs normally at 1 unless the state of the flip-flop has to be changed. The application of a momentary 0 to the set input causes the output Q to go to 1 and $\overline{Q}$ to go to 0, thus putting the flip-flop into the set state. After the set input returns to 1, a momentary 0 to the reset input causes a transition to the clear state. When both inputs go to 0, both outputs go to 1—a condition required to be avoided in normal flip-flop operation

The circuit operation of the basic flip-flop circuit described above with two NAND gates is summarized in tabular form in a truth table

Truth table of basic flip-flop circuit with two NAND gates



Inputs		Outputs		
R	S	Q	$\overline{Q}$	
1	0	0	1	
1	1	0	1	(after S = 0, R = 1)
0	1	1	0	
1	1	1	0	(after S = 1, R = 0)
0	0	1	1	

6.4.2 Clocked or Gated Flip-Flops:-

6.4.2.1 Clocked RS flip-flop:-

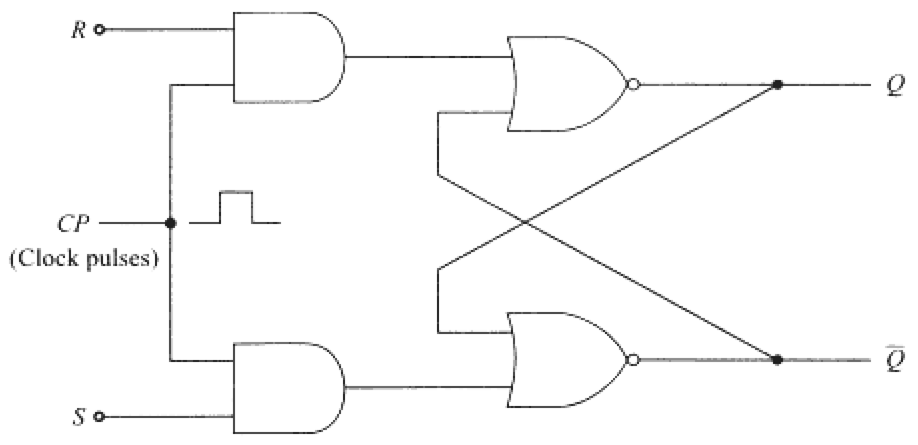
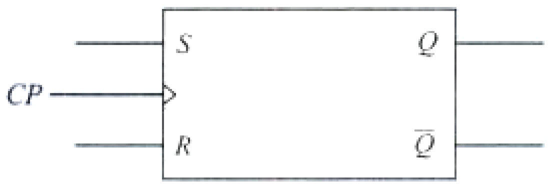


Fig. 6.6 Clocked SR flip-flop using NAND gates.

The basic flip-flop as we have seen is an asynchronous sequential circuit. It can, however, be redesigned to respond to R and S inputs during the occurrence of a clock pulse as well, by merely adding gates to the R and S inputs of the basic circuit. Such a clocked RS flip-flop is shown Fig 6.6. It consists of a basic NOR flip-flop and two AND gates. When the clock pulse (CP) is zero, the outputs of the two AND gates remain at 0 regardless of what the input values S and R are. On the other hand, when the clock pulse goes to a 1, the input values of S and R are allowed to reach the basic flip-flop. For example, when S=1, R=0, and CP=1, the set value (I.e. S=1) is allowed to reach the basic flip-flop. Similarly, the reset value(i.e, R=1) will reach the basic flip-flop when S=0, R=1 and CP=1. However, when both S=1 and R=1, the occurrence of a clock pulse makes both the outputs of flip-flop to momentarily go to 0. When the clock pulse is removed, the output of the flip-flop may go to either a 1 or a 0, i.e, it cannot be predicted and this state of the flip-flop is called indeterminate. Which output state would result, depends upon whether it is set or the reset input of the basic flip-flop remains a 1 longer before the transition of the clock pulse to 0.



Graphic Symbol of Clocked RS Flip-Flop

The graphic symbol for the clocked RS flip-flop is shown in above. It has three inputs S,R and CP. Note, CP input marked by a small triangle. This triangle is a symbolic representation for a **dynamic indicator.** It is meant to denote that the flip-flop responds to an input **clock transition** from low-level(binary 0) to a high-level (binary 1) signal.

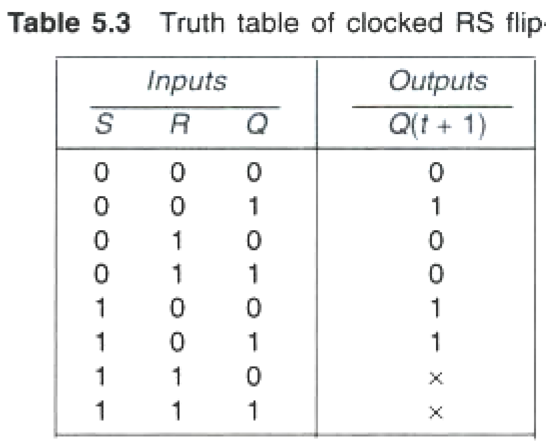
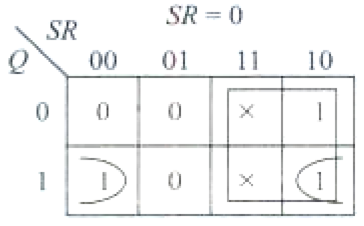


Table 5.3 Truth table of clocked RS flip-flop

Inputs			*Outputs*
S	*R*	*Q*	*Q*(*t* + 1)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	×
1	1	1	×

$$Q(t+1) = S + \bar{R}Q$$
$$SR = 0$$



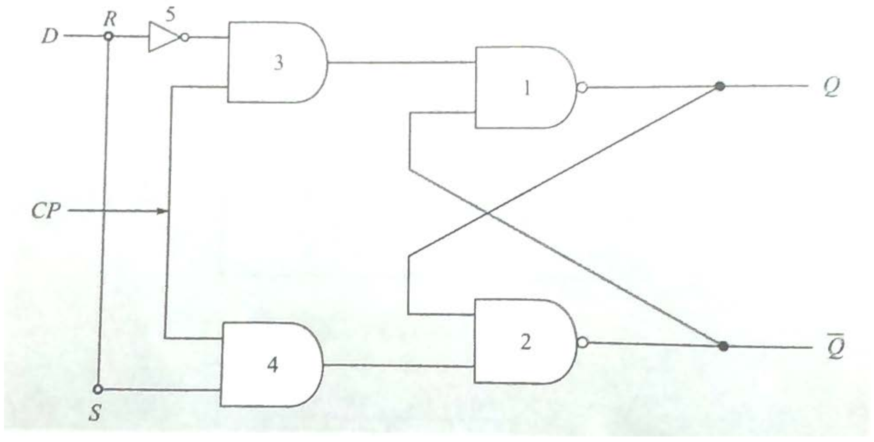
$$\begin{array}{c|cccc} Q \backslash SR & 00 & 01 & 11 & 10 \\ \hline 0 & 0 & 0 & \times & 1 \\ 1 & 1 & 0 & \times & 1 \end{array}$$

Characteristic equation of Clocked RS flip-flop

6.4.2.2 D flip-flop:-

The letter D stands for ‘delay’ and such flip-flops are used for storing information. In case of RS flip-flop, there are two inputs S and R, and for storing a low bit, the value of R is high and for storing a high bit, the value of S is high. The generation of two signals i.e, R and S, to drive a flip-flop is disadvantages in some applications. There are some forbidden conditions too. These are taken care in the D flip-flop. A D flip-flop has only one data input called D.

The D flip-flop allows the value of D to reach the output only when a clock pulse occurs and prevents the value of D to reach the output when there is no clock pulse. Therefore, in a D flip-flop the output takes the input as long as the clock is high and this type of D flip-flop is also called a D-latch or gated D-latch

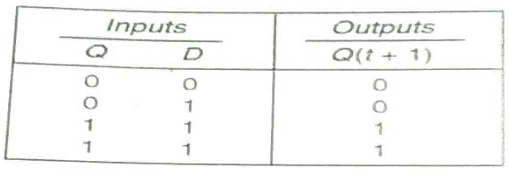


Logic Diagram of D flip-flop with NAND gates

The NAND gates 1 and 2 form a basic flip-flop and gates 3 and 4, as we have seen earlier, modify it into a clocked RS flip-flop. The D input goes directly to the S input, and its complement, through gate 5, is applied to the R input. As long as the clock pulse is at 0, gates 3 and 4 have a 1 in their outputs, regardless of the value of the other inputs. This conforms to the requirement that the two inputs of a basic NAND flip-flop, remain initially at the 1 level, The D input is sampled during the occurrence of a clock pulse. If it is a 1, the output of gate 4 goes to 0, switching the flip-flop to the set state. If it is a 0, the output of gate 3 goes to 0, switching the flip-flop to the clear state.

The D flip-flop receives the designation from its ability to transfer data into a flip-flop. It is basically an RS flip-flop with an inverter in the R input. The inverter so added reduces the no of inputs from two to one. The CP input is sometimes given the designation G to indicate that this input enables the gated latch to make possible the data entry into the flip-flop

Table 5.4 Truth table of clocked D flip-flop



Inputs		Outputs
Q	D	Q(t + 1)
0	0	0
0	1	0
1	1	1
1	1	1

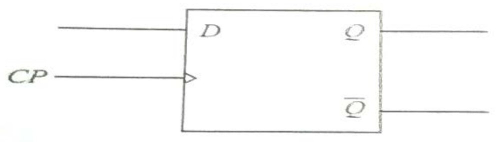
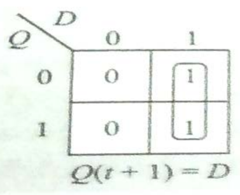


Figure 5.9 Graphic symbol of clocked D flip-flop.



$Q \backslash D$	0	1
0	0	1
1	0	1

$Q(t+1) = D$

6.4.2.3 J K Flip-Flop:-

We know that in RS flip-flop, the input condition S = 1 and R = 1 is prohibited. To overcome this problem some modification is made and the resultant flip-flop is named as JK flip-flop. In JK flip-flop, the input condition J = 1 and K = 1 is allowed. The specialty of JK flip-flop is when both inputs are 1 (J = 1, & K = 1), the flip-flop **toggles**. I.e., the output changes its state. For all other input combinations JK flip-flop behaves exactly like RS flip-flop.

Block diagram of JK flip-flop:

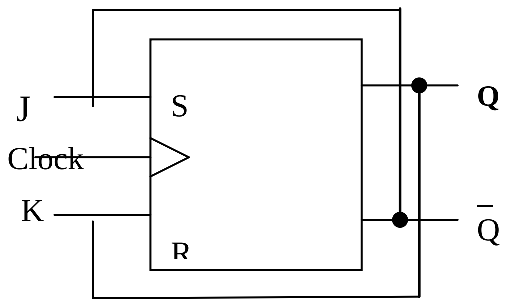


Fig. 6.8 (a) Block diagram of JK flip-flop

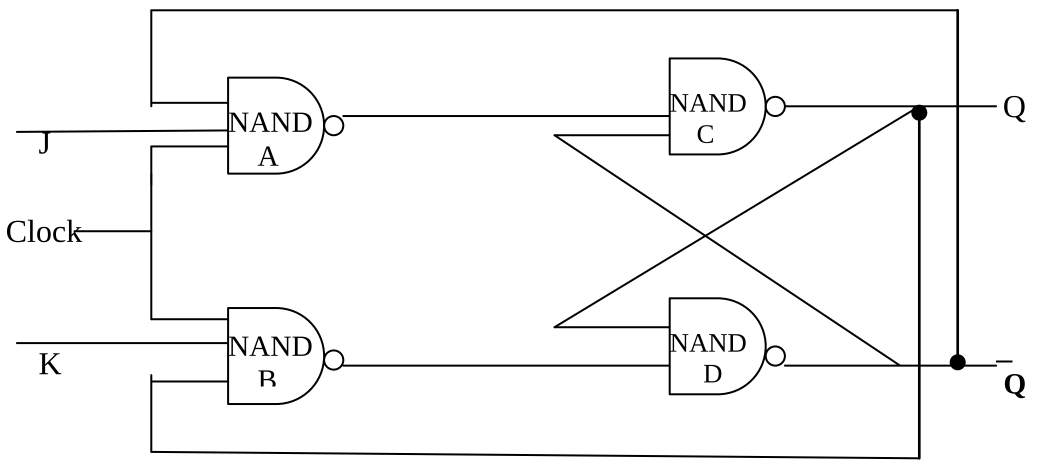


Fig. 6.8(b) Clocked JK flip-flop using NAND gates.

Working:

1. When J = 0, K = 0 :

If Q_n = 0, then during the presence of clock pulse, one input of NAND gate A becomes 0. As a result, its output becomes 1. This makes both inputs of the NAND gate C to 1 and output of it to 0. i.e., Qn_{+1} = 0 (initial condition).

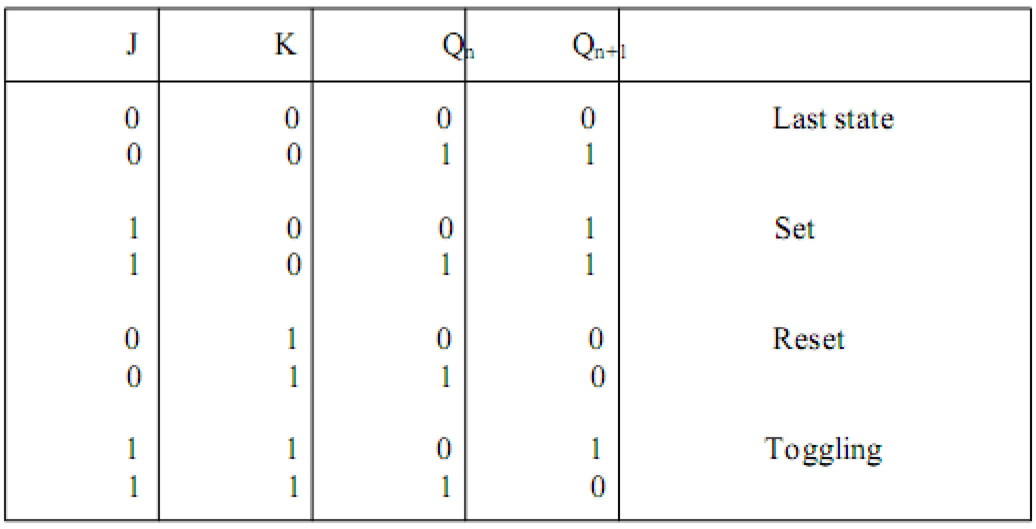
If Q_n = 1, then during the presence of clock pulse, two inputs of NAND gate A becomes 0. As a result, its output becomes 1. This makes one input of the NAND gate C to 0 and the output of NAND gate C to 1. i.e., Q_{n+1} = 1 (initial condition).

2. When J = 1, K = 0 :
If Q_n = 0, then during the presence of clock pulse, all three inputs of NAND gate A becomes 1. As a result, its output becomes 0. This makes one input of the NAND gate C to 0 and output of it to 1. i.e., Qn_{+1} = 1 (Set condition).
If Q_n = 1, then during the presence of clock pulse, one input of NAND gate A becomes 0. As a result, its output becomes 1. This makes one input of the NAND gate C to 0 and the output of NAND gate C to 1. i.e., Q_{n+1} = 1 (Set condition).

3. When J = 0, K = 1:
If Q_n = 0, then during the presence of clock pulse, one input of NAND gate A becomes 0. As a result, its output becomes 1. This makes both inputs of the NAND gate C to 1 and output of it to 0. i.e., Q_{n+1} = 0 (Reset condition).
If Q_n = 1, then during the presence of clock pulse, all the three inputs of NAND gate B becomes 1. As a result, its output becomes 0. This makes one input of the NAND gate D to 0 and the output of NAND gate D to 1. This in turn makes both inputs of NAND gate C to 1 and output of it becomes 0. i.e., Q_{n+1} = 0 (Reset condition).

4. When J = 1, K = 1:
If Q_n = 0, then during the presence of clock pulse, all 3 inputs of NAND gate A becomes 1. As a result, its output becomes 0. This makes one input of the NAND gate C to 0 and output of it becomes 1. i.e., Qn_{+1} = 1 (Toggling condition).
If Q_n = 1, then during the presence of clock pulse, one input of NAND gate A becomes 0. As a result, its output becomes 1. Also, all the inputs of the NAND gate B becomes 1 and the output of NAND gate A becomes 0. This in turn makes one input of NAND gate D to 0 and output of it becomes 1. Hence both inputs of NAND gate C becomes 1 and output of it becomes 0. i.e., Qn_{+1} = 0 (Toggling condition).

Truth-Table of JK flip-flop:



J	K	Q_n	Q_{n+1}	
0	0	0	0	Last state
0	0	1	1	
1	0	0	1	Set
1	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	1	0	1	Toggling
1	1	1	0	

Race around Condition:

This is the problem associated with JK flip-flop, when both the inputs are equal to 1. In JK flip-flop, due to feedback between input and output, any changes in output results in changes in input. Due to this, during positive half of the clock pulse if J and K are both high then output toggles continuously from 0 to 1, 1 to 0, 0 to 1 and so on, after each propagation delay of the circuit. This condition (problem) known as **race around condition** must be avoided.

6.4.2.4 T flip-flop:-

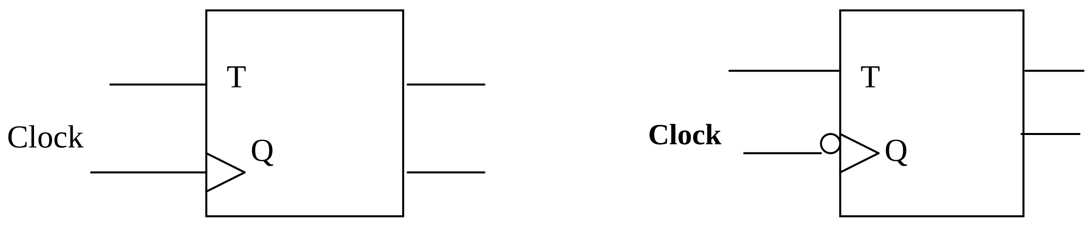
In T Flip-flop (Toggle flip-flop), the R input is connected to S input directly, unlike the D flip flop, where it is done by means of a NOT gate. Hence the conditions of both the inputs become same.

In T flip-flop, when T input is 1, the S input becomes 1 and R input is also a 1. As a result, the flip-flop RESETs and the output Q becomes 0.

When the T input is 0, the S input becomes 0 and R input is also a 0. As a result, the flip-flop SETs and the output Q becomes 1.

Hence, in T flip-flop, there is one clock pulse delay between input and output and output follows input. Figure 6.9(b) shows the basic circuit of clocked T flip-flop using NAND gates.

Fig. 6.9(a) Block Diagram of T Flip flop



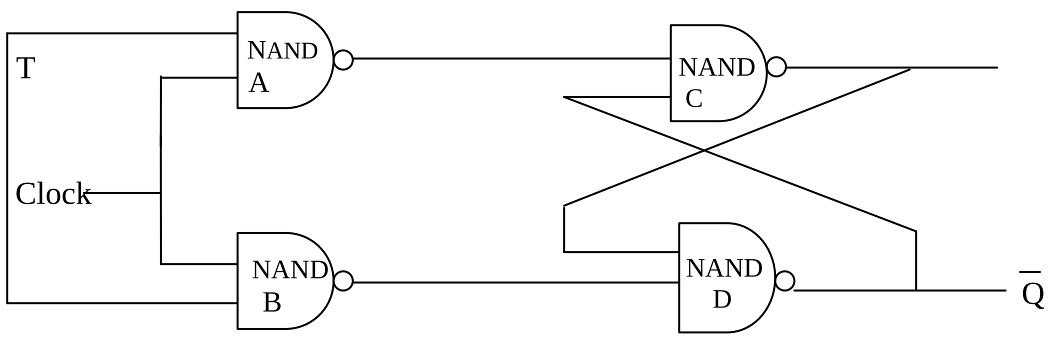


Fig. 6.9 (b) Clocked D flip-flop using NAND gates.

Truth table for T flip-flop:

CP	T	Q_{n+1}
0	Qn	X
1	0	1
1	1	0

Thus it is called as a Toggle(T) flip flop, as you can see in the truth table above, that when the T input is 0, the output Q is 1 and when the T input is 1, the output Q is 0. So the output Q here toggles to the opposite state based on the input T.

6.5 Master- Slave Flip-Flop:-

Master-Slave flip-flop is the modified version of clocked JK Flip-flop in which the problem of race around condition is eliminated. A master-slave flip-flop consists of two RS flip-flops and an inverter. One circuit serves as a master and the other as a slave. The block diagram of MS flip-flop is shown in Fig. below. Both the flip-flops are positive edge triggered, but the inverter connected at the clock input of the slave flip-flop forces it to trigger at the negative edge. Here, the master flip-flop triggers during positive edge of clock pulse and the slave flip-flop triggers during the negative edge of clock pulse. As a result output Q changes only during the end of the clock pulse.

Master slave SR Flip-Flop

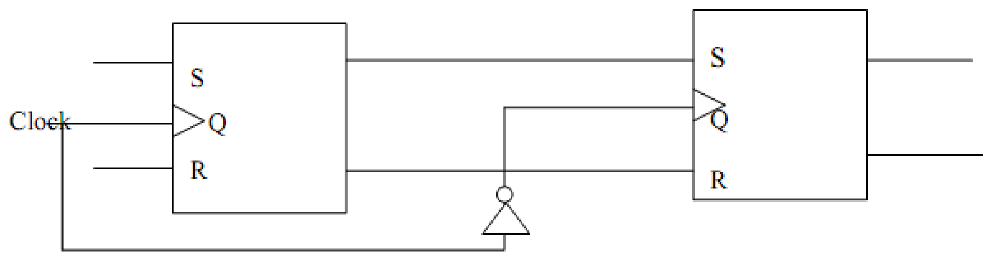
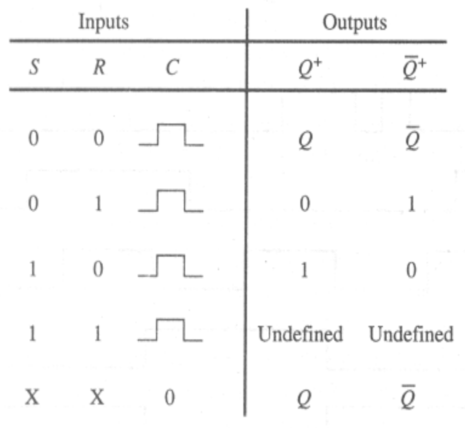
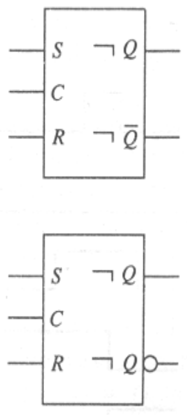


Fig. 6.10 (a) Block diagram of Master-Slave SR flip-flop.



Inputs			Outputs	
S	R	C	Q^+	$\bar{Q}^+$
0	0	⎍	Q	$\bar{Q}$
0	1	⎍	0	1
1	0	⎍	1	0
1	1	⎍	Undefined	Undefined
X	X	0	Q	$\bar{Q}$

(c)



(d)

Truth Table for Master slave SR Flip-Flop

Master slave JK Flip-Flop

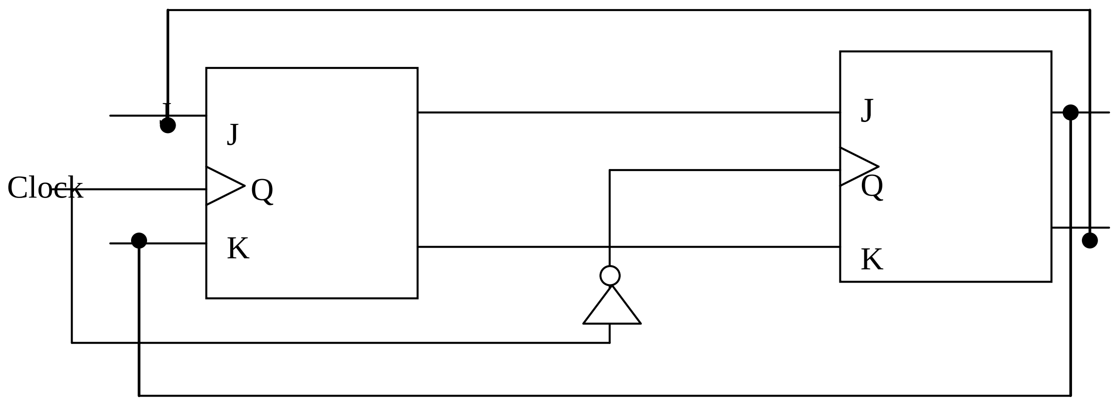
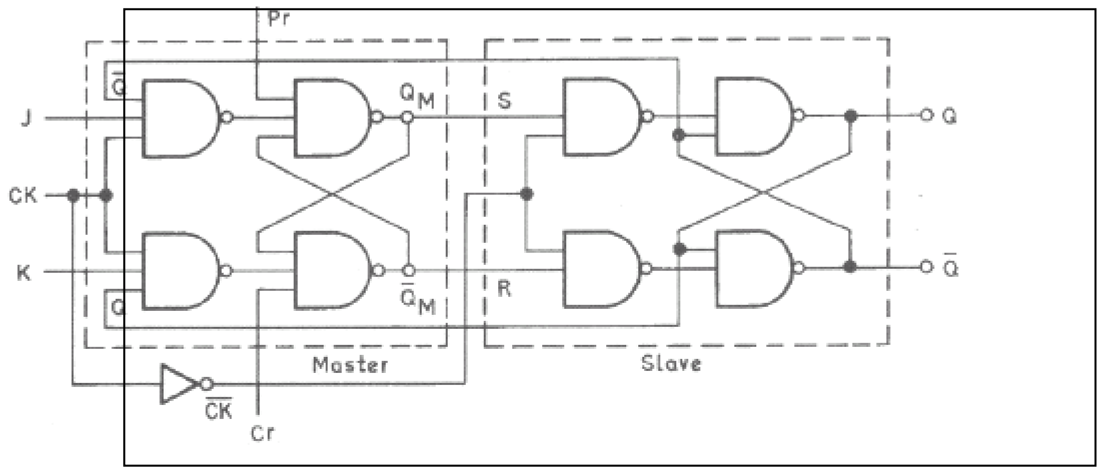


Fig. 6.10 (b) Block diagram of Master-Slave JK flip-flop.

The following figure Fig. 6.10 (c) shows the Master-slave JK flip-flop constructed using NAND gates.



Working :

When J = 1 and K = 0, the master sets on positive edge go the clock pulse. The high output (1 state) of master drives the S input of slave, so at negative edge of same clock pulse, slave sets, copying the action of the master.
When J = 0, K = 1, the master resets on the positive clock edge. The high output (1 state) of the master goes to the R input of slave. Therefore, at negative edge of clock pulse, slave resets, again copying the action of the master.
When J = 1 and K = 1, master toggles on positive edge of clock pulse and slave then copies the action of master during the negative edge of the clock pulse. At this instant, feedback inputs to the master flip-flop are complemented but as it is negative half of clock pulse master flip-flop is inactive. This prevents the race around condition.

Truth-Table of JK flip-flop:

J	K	Q_n	Q_{n+1}	
0	0	0	0	Last state
0	0	1	1	
1	0	0	1	Set
1	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	1	0	1	Toggling
1	1	1	0	

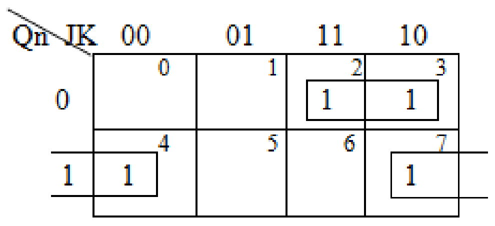
6.6 Triggering and Characteristic equations (State equations) for flip flops

The characteristic equation of a flip flop is the equation expressing the next state of a flip flop in terms of its present state and present excitations. To obtain the characteristic equation of a flip flop write the excitation requirements of the flip flop, draw a K-map for the next state of the flip flop in terms of its present state and inputs and simplify it as shown in the tables 6.6(a), 6.6(b), 6.6(c) and 6.6(d) for JK, SR, T and D flip flops respectively.

(a)

Present State	Inputs		Next State
Qn	J	K	Qn+1
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Excitation requirements of JK flip flop



K-Map for Qn+1

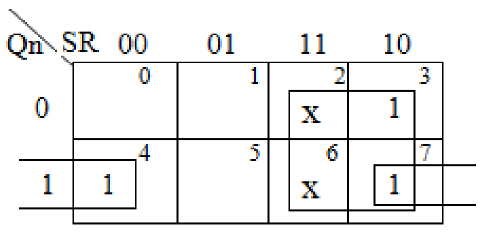
From the K-Map above we have the characteristic equation for JK flip flop as

$Qn+1 = \overline{Q}n\ J + Qn\ \overline{K}$

(b)

Present State	Inputs		Next State
Qn	S	R	Qn+1
0	0	0	0
0	0	1	0
0	1	0	1
1	0	0	1
1	0	1	0
1	1	0	1

Excitation requirements of JK flip flop



K-Map for Qn+1 of SR FF

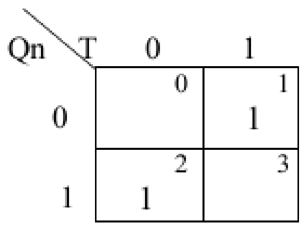
From the K-Map above we have the characteristic equation for SR flip flop as

$Qn+1 = S + Qn\ \overline{R}$

(c)

Present State	Inputs	Next State
Qn	T	Qn+1
0	0	0
0	1	1
1	0	1
1	1	0

Excitation requirements of T flip flop



K-Map for Qn+1 of T FF

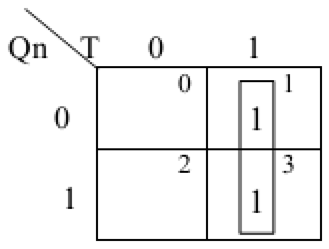
From the K-Map above we have the characteristic equation for T flip flop as

$Q_{n+1} = \overline{Q_n}\,T + Q_n\,\overline{T}$

(d)

Present State	Inputs	Next State
Qn	D	Qn+1
0	0	0
0	1	1
1	0	1
1	1	0

Excitation requirements of D flip flop



K-Map for Qn+1 of SR FF

From the K-Map above we have the characteristic equation for D flip flop as

$Q_{n+1} = D$

The Characteristic Equations of flip-flops are listed in the Table below

Flip-flop	Characteristic equation
J-K	$Q_{n+1} = \overline{Q_n}\,J + Q_n\,\overline{K}$
S-R	$Q_{n+1} = S + Q_n\,\overline{R}$
T	$Q_{n+1} = \overline{Q_n}\,T + Q_n\,\overline{T}$
D	$Q_{n+1} = D$

6.7 Shift Registers

A register is a group of memory elements that work together as a unit to store a word by shifting its bits left or right.

A shift register moves the stored bits of binary information either left or right. This bit shifting is essential for certain arithmetic and logic operations used in microcomputers.

Shift Left:

The following figure shows the block diagram of Shift-left register. Din sets up the right flip-flop, Q0 sets up the second flip-flop, Q1 the third, and so on. When the next positive clock edge strikes, therefore, the stored bits move one position to the left.

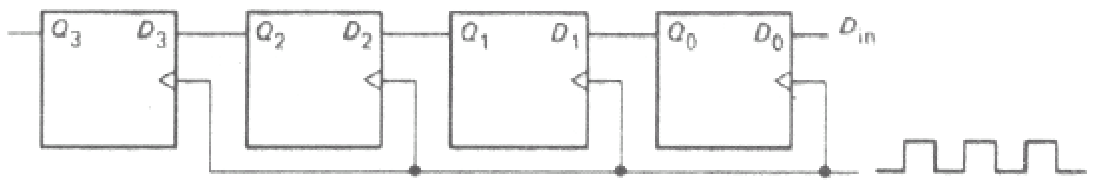


Fig. 6.11(a): Shift Left Register

Shift Right:

The following figure shows the block diagram of Shift-right register. Here, each Q output sets up the D input of the preceding flip-flop. When the positive clock pulse arrives, the stored bits move one position to the right.

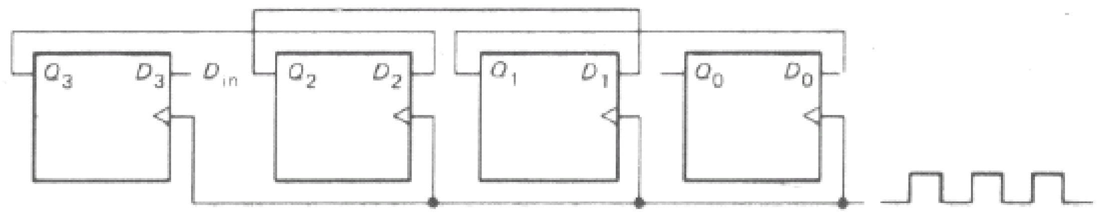


Fig. 6.11 (b): Shift Left Register

The main type of shift register is the Serial-In, Serial-Out shift register, which is the basic shift register that we use inside a computer to perform arithmetic and logic calculations and other shift operations. The logical configuration of a shift register consists of a chain of flip-flops connected in cascade, with the output of one flip-flop connected to the input of the next flip-flop. All flip-flops receive a common clock pulse which causes the shift from one stage to the next.

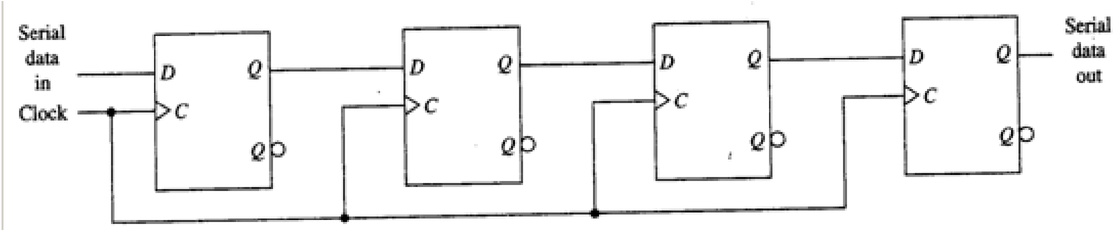


Fig. 6.11 (c) Serial-In Serial-Out Shift Register

The simplest possible shift register is one that uses only flip-flops, as shown in Fig. above. The Q output of a given flip-flop is connected to the D input of the flip-flop at its right. Each clock pulse shifts the contents of the register one bit position to the right. The serial input determines what goes into the leftmost flip-flop during the shift. The serial output is taken from the output of the rightmost flip-flop prior to the application of a pulse. Although this register shifts its contents to the right, if we turn the page upside down, we find that the register shifts its contents to the left. Thus a unidirectional shift register can function either as a shift-right or as a shift-left register.

The register in the Fig. above shifts its contents with every clock pulse during the negative edge of the pulse transition. (This is indicated by the small circle associated with the clock input in all flip-flops.) If we want to control the shift so that it occurs only with certain pulses but not with others, we must control the CP input of the register. It will be shown later that the shift operations can be controlled through the D inputs of the flip-flops rather than through the CP input. If, however, the shift register in the Fig. is used, the shift can easily be controlled by means of an external AND gate as shown below.

Similar to the above basic shift register that is the Serial-in Serial-out shift register we have:

- Serial-in Parallel Out shift register
- Parallel-in Serial-Out Shift Register and
- Parallel-in Parallel Out Shift Register

6.8 Bidirectional Shift Register

A bidirectional shift register is one in which the data bits can be shifted from left to right or from right to left.

Figure 6.8 shows the logic diagram of a 4 bit serial-in, serial out, bidirectional shift register. Right/$\overline{\text{Left}}$ is the mode signal. When Right/$\overline{\text{Left}}$ is a 1, the logic circuit works as a shift-right shift register. When Right/$\overline{\text{Left}}$ is a 0, the logic circuit works as a shift-left shift register. The bidirectional operation is achieved by using the mode signal and two AND gates and one OR gate for each stage as shown in fig. 6.8

A HIGH on the Right/$\overline{\text{Left}}$ control input enables the AND gates G_1, G_2, G_3 and G_4 and disables the AND gates G_5, G_6, G_7 and G_8, and the state of Q output of each flip flop is passed through the gate to the D input of the following FF. When a clock pulse occurs, the data bits are then effectively shifted one place to the right.

A LOW on the Right/$\overline{\text{Left}}$ control input enables the AND gates G_5, G_6, G_7 and G_8 and disables the AND gates G_1, G_2, G_3 and G_4, and the state of Q output of each flip flop is passed through the gate to the D input of the preceding FF. When a clock pulse occurs, the data bits are then effectively shifted one place to the left. Hence the circuit works as a bidirectional shift register.

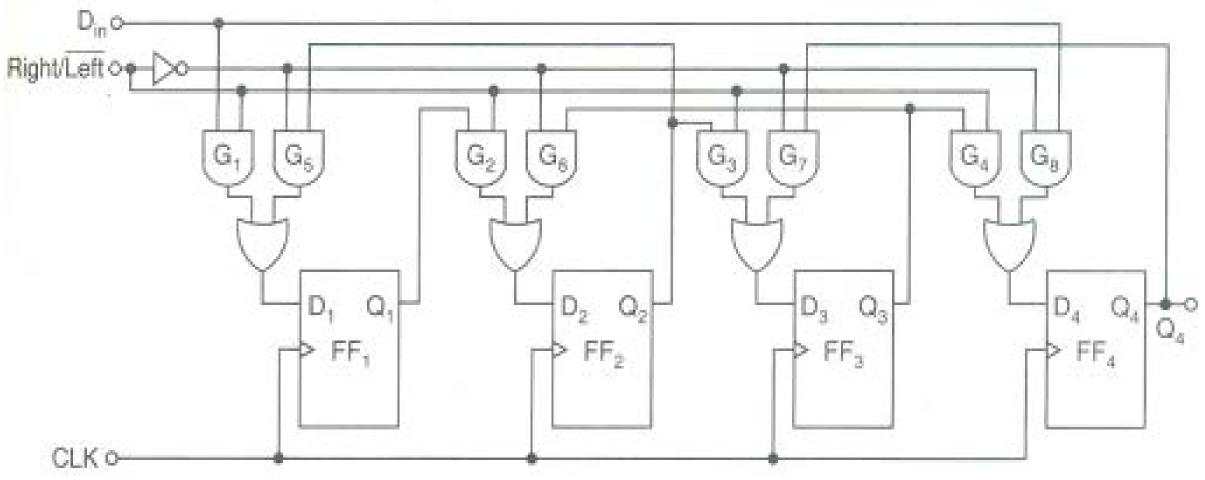


Fig. 6.12 Logic Diagram of a 4-bit Bidirectional Shift Register

6.9Binary Counters

Binary counter is an electronic circuit designed using flip-flops used to count the number of clock pulses.
There are different types of counters. The most commonly used are binary counter and BCD counters. The binary counter counts the incoming signal (clock pulse) continuously. A BCD counter counts the input clock pulses in the form of BCD numbers. Normally it counts up to 9 pulses.

- Counter is a register which counts the sequence in binary form.
- The state of counter changes with application of clock pulse.
- The counter is binary or non-binary.
- The total no. of states in counter is called as modulus.
- If counter is modulus-n, then it has n different states.
- State diagram of counter is a pictorial representation of counter states directed by arrows in graph.

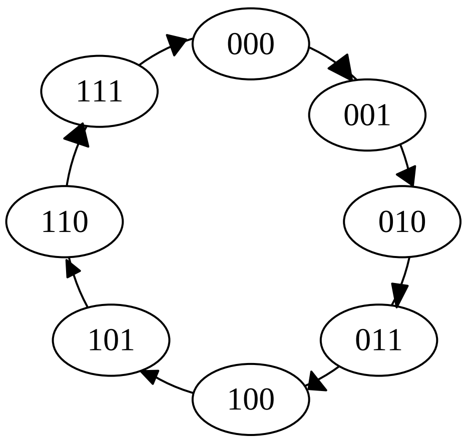


Fig 6.13. State diagram of mod-8 counter

6.10. Ripple Counter or Gated–clocked binary counter:

Ripple counters also called as asynchronous counters are simple to build and hence very popular counters in computers. In ripple counter, the clock is applied to least significant stage (i.e., JK flip –flop X1) and output transitions are applied to successive stages to work as clocks. Thus the clock ripples through the counter stage by stage. Because of these stage by stage changes, the ripple counters are slow. A four stage ripple counter is shown in following figure 6.10.

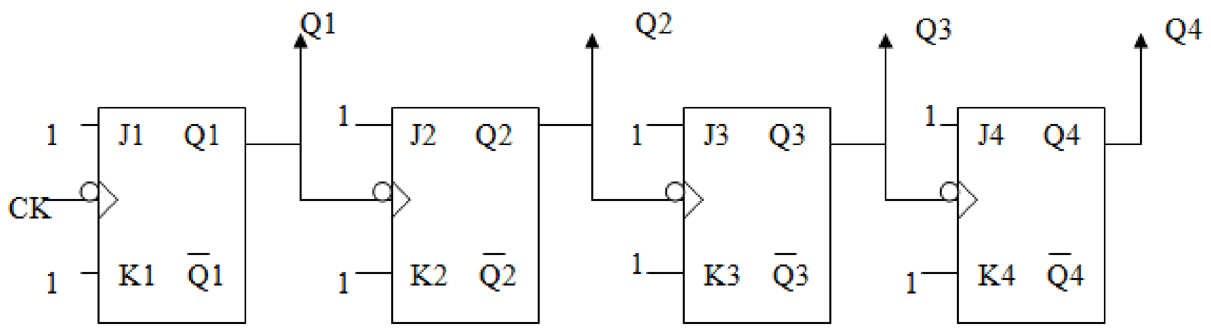


Fig 6.14. 4-bit binary ripple counter using JK flip flop

Working:

Here all the JK flip-flops are in toggling mode. The output of Q1 is connected to the clock input of Q2. The output of Q2 is connected to clock input of Q3. The output of Q3 is connected to clock input of Q4. Each and every positive clock applied to clock input toggles the corresponding flip-flop. As a result, the counter follows the sequence of counting as shown in counter table.

All Flip-Flops are in toggle mode.

- The clock input is applied.
- Count enable = 1.
- Counter counts from 0000 to 1111.

Since each flip-flop produces some delay between input to output, the total delay will be sum of 4 flip-flop delays. If clock frequency is high, then the total flip-flop delay will be greater than clock pulse width and hence the counter cannot function satisfactorily.

6.11 Integrated Circuits

All modern computing devices like calculators, microcomputers and large high speed computers are made up of mainly flip-flops and gates in the form of integrated circuits. These circuits are constructed and packed in the form of a single integrated circuit container using what is called integrated circuit technology. The integrated circuit (IC) containers provide input and output pins or connections which are then interconnected by means of wires or circuit boards to form complete computing device.

Two types of IC packages are available in the market. One is **dual inline** package and the other is **flat type** package. Dual in-line packages are available with 8 to 100 pins but 16 to

40 pins are popular. A particular circuit called transistor-transistor logic (TTL) and /or MOS circuits are used in the most integrated circuits.

6.12. State Tables

The time sequence of inputs, outputs, and flip-flop states may be enumerated in a state table. The state table for a circuit is shown in table below. It consists of three sections labeled present state, next state, and output. The present state designates the states of flip-flops before the occurrence of a clock pulse. The next state shows the states of flip-flops after the application of a clock pulse, and the output section lists the values of the output variables during the present state. Both the next state and output sections have two columns, one for x = 0 and the other for x = 1.

The derivation of the state table starts from an assumed initial state. The initial state of most practical sequential circuits is defined to be the state with 0's in all flip-flops. Some sequential circuits have a different initial state and some have none at all. In either case, the analysis can always start from any arbitrary state. In this example, we start deriving the state table from the initial state 00.

Consider the following state table of a circuit

	Next State		Output	
Present State	x=0	x=1	x=0	x=1
AB	AB	AB	y	y
00	00	01	0	0
01	11	01	0	0
10	10	00	0	1
11	10	11	0	0

The derivation of the state table starts from an assumed initial state. The initial state of most practical sequential circuits is defined to be the state with 0's in all flip-flops. Some sequential circuits have a different initial state and some have none at all. In either case, the analysis can always start from any arbitrary state. In this example, we start deriving the state table from the initial state 00.When the present state is 00, A = 0 and B = 0. From the logic diagram, we see that with both flip-flops cleared and x = 0, none of the AND gates produce a logic-1 signal. Therefore, the next state remains unchanged. With AB = 00 and x = 1, gate 2 produces a logic-I signal at the S input of flip-flop B and gate 3produces a logic-1 signal at the R input of flip-flop A. When a clock pulse triggers the flip-flops, A is cleared and B is set, making the next state 01. This information is listed in the first row of the state table.

In a similar manner, we can derive the next state starting from the other three possible present states. In general, the next state is a function of the inputs, the present state, and the type of flip-flop used. With RS flip-flops, for example, we must remember that a 1 in input S sets the flip-flop and a 1 in input R clears the flip-flop, regardless of its previous state. A 0 in both the Sand R inputs leaves the flip-flop unchanged, whereas a 1 in both the Sand R inputs shows a bad design and an indeterminate state table.

The entries for the output section are easier to derive. In this example, output y is equal to 1 only when x = 1, A = 1, and B = 0. Therefore, the output columns are marked with 0's, except when the present state is 10 and input x = 1, for which y is marked with a 1

The state table of any sequential circuit is obtained by the same procedure used in the example. In general, a sequential circuit with m flip-flops and n input variables will have 2^mrows, one for each state. The next state and output sections each will have 2^ncolumns, one for each input combination.

The external outputs of a sequential circuit may come from logic gates or from memory elements. The output section in the state table is necessary only if there are outputs from logic gates. Any external output taken directly from a flip-flop is already listed in the present state column of the state table. Therefore, the output section of the state table can be excluded if there are no external outputs from logic gates.

6.13. State Diagrams

The information available in a state table may be represented graphically in a state diagram.

In this diagram, a state is represented by a circle, and the transition between states is indicated by directed lines connecting the circles. The state diagram of the sequential circuit given in the table above is shown in the Fig. 6.15below.

The binary number inside each circle identifies the state the circle represents.

The directed lines are labeled with two binary numbers separated by a l.

The input value that causes the state transition is labeled first; the number after the symbol/gives the value of the output during the present state.

For example, the directed line from state 00 to 01 is labeled 1/0, meaning that the sequential circuit is in a present state00 while x = 1 and y = 0, and that on the termination of the next clock pulse, the circuit goes to next state 01.

A directed line connecting a circle with itself indicates that no change of state occurs. The state diagram provides the same information as the state table and is obtained directly from the state table.

The state diagram for the state table given above can be implemented as follows in fig. 6.15

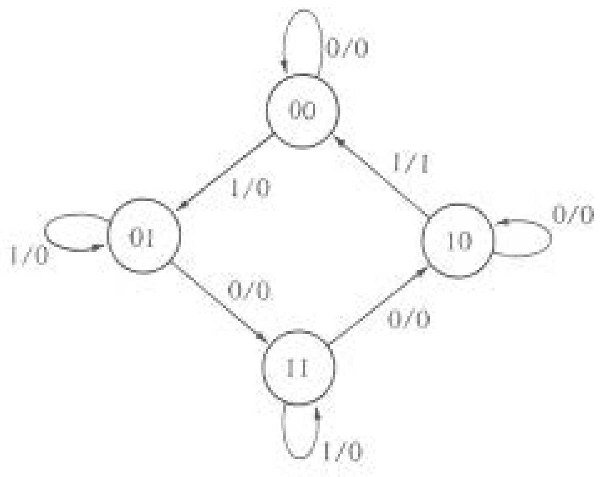
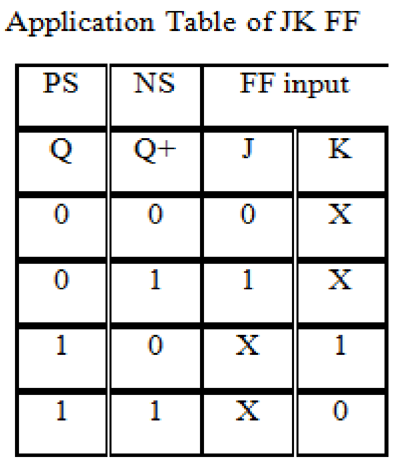


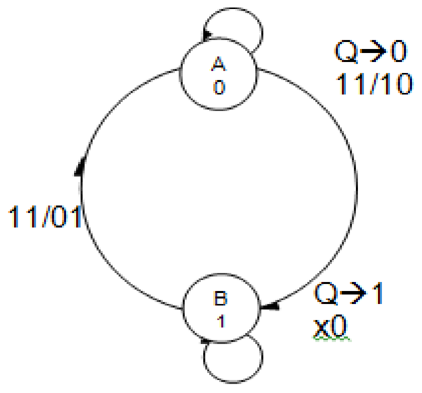
Fig. 6.15

State machine notations:

- Input Variables: External input variables to sequential machine as inputs.
- Output Variables: All variables that exit from the sequential machine are output variables.
- State: State of sequential machine is defined by the content of memory, when memory is realized by using FFs.
- Present State: The status of all state variable i.e. content of FF for given instant of time t is called as present state.
- Next State: The state of memory at t+1 is called as Next state.
- State Diagram: State diagram is graphical representation of state variables represented by circle. The connection between two states represented by lives with arrows and also indicates the excitation input and related outputs.
- Output Variables: All variables that exit from the sequential machine are output variables.


Application Table of JK FF

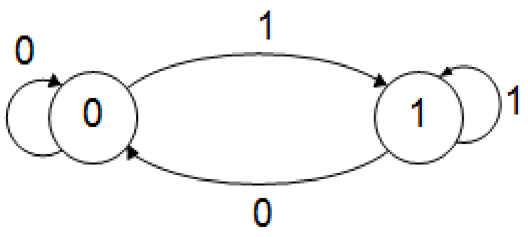
PS	NS	FF input	
Q	Q+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0



State diagram of J-K FF

Application Table of D FF

PS	NS	FF i/p
Q	Q+	D i/p
0	0	0
0	1	1
1	0	0
1	1	1



State diagram of D FF

6.14. Excitation Tables

The characteristic tables of flip-flops provide the next state when inputs and the present state are known. These tables are useful for analysis of sequential circuits. But, during the design process, we know the required transition from present state to next state and wish to find the required flip-flop inputs. Thus comes the need of a table that lists the required input values for given change of state. Such a table is called excitation Table. Fig below shows excitation tables for all flip-flops.

Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

(a) J-K Flip flop

Q((t)	Q(t+1)	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

(b) S-R Flip flop

Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

(c) D Flip flop

Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	1

(d) T Flip flop

Let us discuss more deeply, how these excitation tables are formed. For this, we take an example of
J-K Flip flop.
1) The state transition from present state 0 to next state 0 can be achieved when
(a) J=0, K=0, then no change in the state of flip flop
(b) J=0, K=1, then flip flop resets i.e. 0
(remember J-K Characteristic table from figure 4.6)
Thus in either case J=0 but K can be 0 or 1 that is represented by don’t care condition X.

2) The state transition from present state 0 to next state 1 can be achieved when
(a) J=1, K=0, then flip flop is set i.e. 1
(b) J=1, K=1, then flip flop is complemented i.e. change from 0 to 1
Here, also in either case J=1 but K can be 0 or1 that means again K is represented as a don't care case.

3) Similarly, state transition from present state 1 to next state 0 can be achieved when
(a) J=0, K=1, flip flop is reset i.e.0
(b) J=1, K=1, flip flop is complemented i.e. changes from 1 to 0

This indicates that in either case K=1 but J can be either 0 or 1 thus don't care case.
4) For state transition from present state 1 to next state 1 can be achieved when
(a) J=0, K=0, no change in flip flop
(b) J=1, K=0, flip flop is set i.e. 1
Thus J is don't care case but K=0.

6.15. Synchronous Counters

6.15.1 Design of Synchronous Counters

For a systematic design of synchronous counters, the following procedure is used.

Step 1.Number of flip-flops: Based on the description of the problem, determine the required number 'n' of the FFs-the smallest value of n is such that the number of states $N \leq 2^n$ and the desired counting sequence.

Step 2. State diagram: Draw the state diagram showing all the possible states. A state diagram, which can also be called the transition diagram, is a graphical means of depicting the sequence of states through which the counter progresses. In case the counter goes to a particular state from the invalid states on the next clock pulse, the same can also be included in the state diagram.

Step 3. Choice of flip-flops and excitation table: Select the type of flip-flops to be used and write the excitation table. An excitation table is a table that lists the present state (PS), the next state (NS) and the required excitations.

Step 4. Minimal expressions for excitations: Obtain the minimal expressions for the excitations of the FFs using the K-maps drawn for the excitations of the flip-flops in terms of the present states and inputs.

Step 5. Logic diagram: Draw a logic diagram based on the minimal expressions.

PS	NS	Required Inputs	
Qn	Qn+1	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

(a) S-R FF excitation table

PS	NS	Required Inputs	
Qn	Qn+1	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

(b) J-K FF excitation table

PS	NS	Required Input
Qn	Qn+1	D
0	0	0
0	1	1
1	0	0
1	1	1

(c) D – FF excitation table

PS	NS	Required Input
Qn	Qn+1	T
0	0	0
0	1	1
1	0	1
1	1	0

(d) T – FF excitation table

Design of a Synchronous 3-bit up-down counter using JK FFs

Step 1. Determine the number of flip-flops required: A 3-bit counter requires three FFs. It has 8 states (000, 001, 010, all, 100, 101, 110, 111) and all the states are valid. Hence no don't cares. For selecting up and down modes, a control or mode signal M is required. Let us say it counts up when the mode signal M = 1 and counts down when M = 0. The clock signal is applied to all the FFs simultaneously.

Step 2. Draw the state diagram: The state diagram of a 3 bit up-down counter is as shown in Figure 6.15(a), where when M=0 it counts down the order of numbers and when M=1 it counts up the order of numbers.

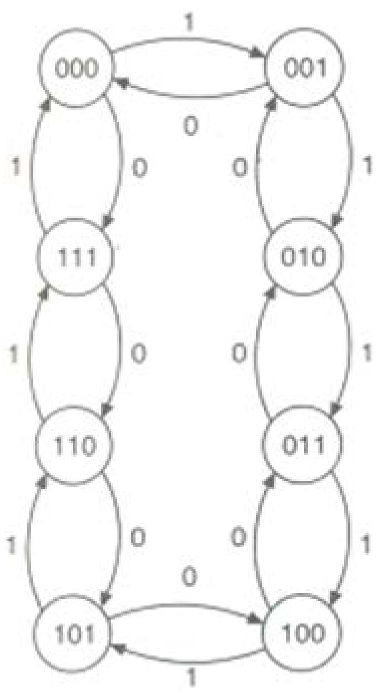


Fig. 6.15(a) State Diagram of a 3 bit up-down counter

Step 3. Select the type of flip-flops and draw the excitation table: JK flip-flops are selected and the excitation table of a 3-bit up-down counter using JK flip-flops is drawn as shown in Figure 6.15(b).

PS			Mode	NS			Required excitations					
Q_3	Q_2	Q_1	M	Q_3	Q_2	Q_1	J_3	K_3	J_2	K_2	J_1	K_1
0	0	0	0	1	1	1	1	×	1	×	1	×
0	0	0	1	0	0	1	0	×	0	×	1	×
0	0	1	0	0	0	0	0	×	0	×	×	1
0	0	1	1	0	1	0	0	×	1	×	×	1
0	1	0	0	0	0	1	0	×	×	1	1	×
0	1	0	1	0	1	1	0	×	×	0	1	×
0	1	1	0	0	1	0	0	×	×	0	×	1
0	1	1	1	1	0	0	1	×	×	1	×	1
1	0	0	0	0	1	1	×	1	1	×	1	×
1	0	0	1	1	0	1	×	0	0	×	1	×
1	0	1	0	1	0	0	×	0	0	×	×	1
1	0	1	1	1	1	0	×	0	1	×	×	1
1	1	0	0	1	0	1	×	0	×	1	1	×
1	1	0	1	1	1	1	×	0	×	0	1	×
1	1	1	0	1	1	0	×	0	×	0	×	1
1	1	1	1	0	0	0	×	1	×	1	×	1

Fig. 6.15(b) Excitation table of a 3-bit up-down counter

Step 4: Obtain the minimal expressions: From the excitation table we can conclude that $J_1 = 1$ and $K_1=1$, because all the entries for J_1 and K_1 are either X or 1. The K-maps for J_3, K_3, J_2 and K_2 based on the excitation table and the minimal expressions obtained from them are show in figure 6.15 (c).

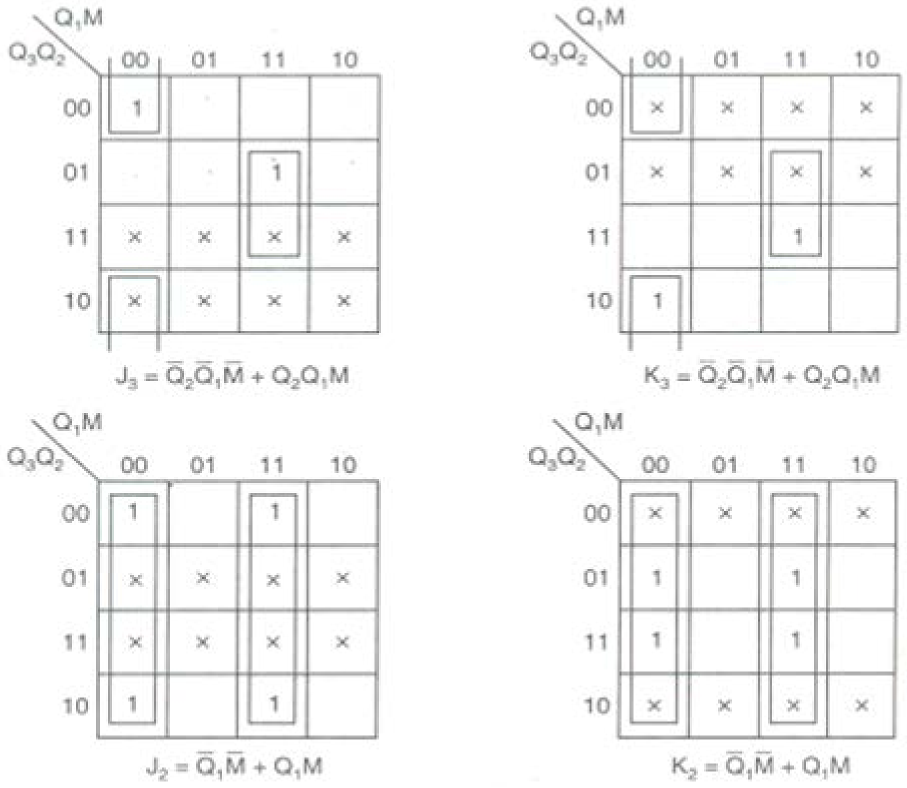

$J_3 = \bar{Q}_2\bar{Q}_1\bar{M} + Q_2Q_1M$
$K_3 = \bar{Q}_2\bar{Q}_1\bar{M} + Q_2Q_1M$
$J_2 = \bar{Q}_1\bar{M} + Q_1M$
$K_2 = \bar{Q}_1\bar{M} + Q_1M$

Fig 6.15(c) K-Maps for excitations of synchronous 3-bit up-down counter

Step 5: Draw the logic diagram. The logic diagram for a synchronous 3-bit up-down counter is as shown in figure 6.15 (d)

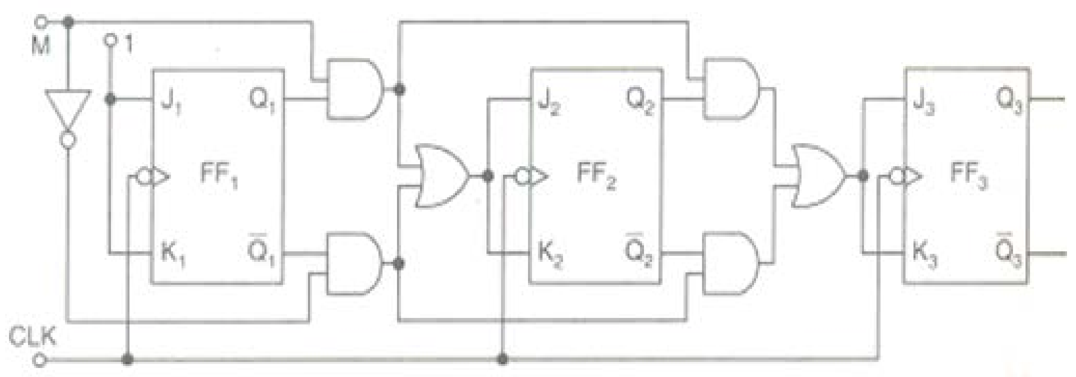


Fig. 6.15(d) Logic diagram for a synchronous 3-bit up-down counter

Design of a 3 bit binary counter (mod-8 counter) using T FFs

1) **No. of flip flops required** is 3 as the no. of bits is 3.

2) **State diagram**

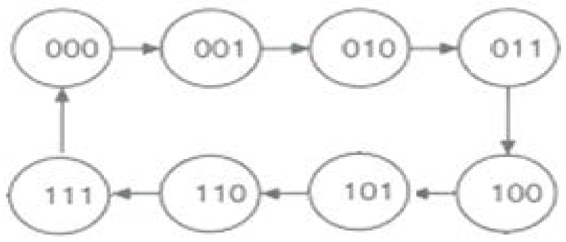


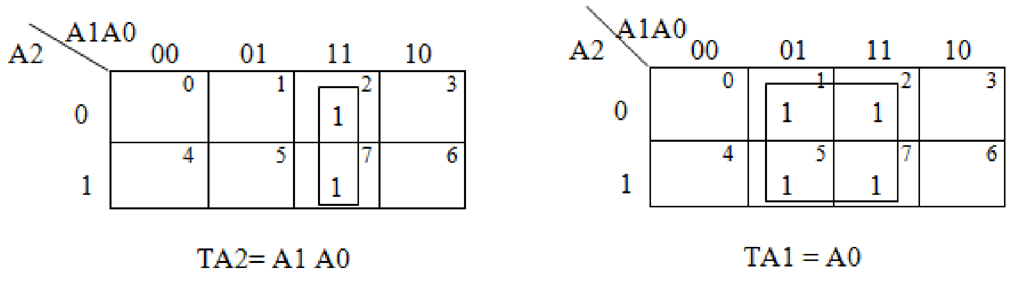
Fig 6.16(a) State diagram of 3 bit binary counter

The count sequence repeats after it reaches the last value, so that state 000 is the next state after 111. The count sequence gives all the information needed to design the circuit.

3) **Excitation Table**

Present State			Next State			Required Excitations		
A2	A1	A0	A2	A1	A0	TA2	TA1	TA0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

4) **Obtaining the minimal expression using K-Map:** From the excitation table, the three flip-flops are given variable designations A2, A1 and A0. The flip-flop excitation for the T inputs is derived from the excitation table of T flip-flop and from inspection of the state transition from a given count (present state) to the next below it (next state). As an illustration, consider the flip-flop input entries for row 001. The present state here is 001 and the next state is 010, which is the next count in the sequence. Comparing these two counts, we note that A2 goes from 0 to 0; so TA2 is marked with a 0 because 0 to 0 transition gives 0 in excitation table of T flip flop. Similarly A1 goes from 0 to 1; so TA1 is marked with a 1. Similarly, A0 goes from 1 to 0, so TA0 is marked with a 1 and so on.



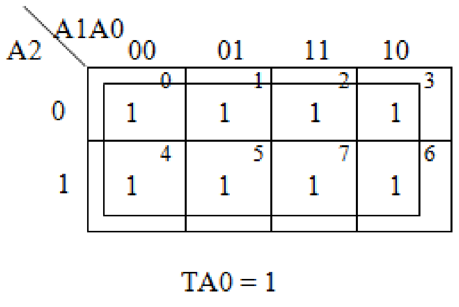


Fig 6.16(b) K-maps for a 3-bit binary counter

5) **Logic Diagram**

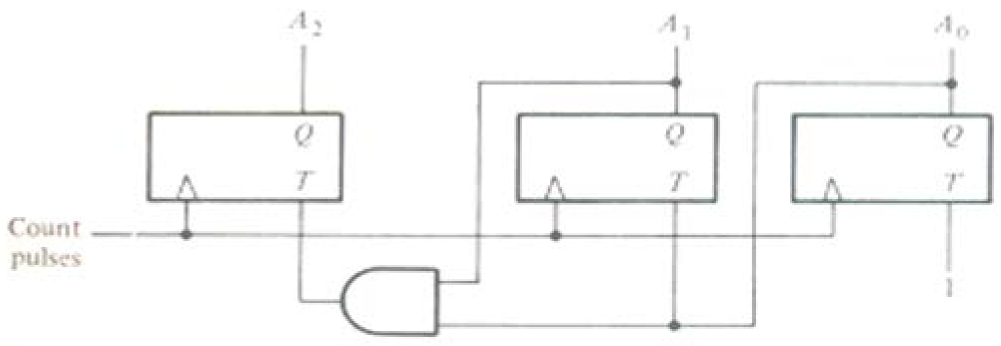


Fig. 6.16(c) Logic Diagram for a 3-bit binary counter

Design of a mod 10 (BCD) Counter Using J-K FFs

Step 1. The number of flip-flops: A BCD counter is nothing but a mod-l0 counter. It is a decade counter. It has 10 states (0000 through 1001). It requires n = 4 FFs ($N \leq 2^n$, i.e. $10 \leq 2^4$). Four FFs can have 16 states. After the tenth clock pulse, the counter resets. So, states 1010 through 1111 are invalid. The entries for excitations corresponding to invalid states are don't cares.

Step 2. The state diagram: The state diagram of the BCD counter is drawn as shown in Figure 6.17 a.

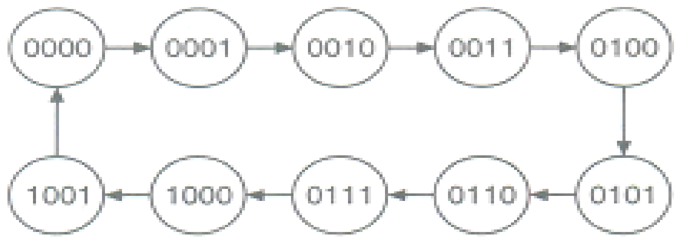


Fig. 6.17 (a) State Diagram of a BCD counter

Step 3. The type of flip-flops and the excitation table: JK flip-flops are selected and the excitation table of the BCD counter using J-K FFs is drawn as shown in Figure 6.17b.

Present State				Next State				Required Excitations							
Q4	Q3	Q2	Q1	Q4	Q3	Q2	Q1	J4	K4	J3	K3	J2	K2	J1	K1
0	0	0	0	0	0	0	1	0	x	0	x	0	x	1	x
0	0	0	1	0	0	1	0	0	x	0	x	1	x	x	1
0	0	1	0	0	0	1	1	0	x	0	x	x	0	1	x
0	0	1	1	0	1	0	0	0	x	1	x	x	1	x	1
0	1	0	0	0	1	0	1	0	x	x	0	0	x	1	x
0	1	0	1	0	1	1	0	0	x	x	0	1	x	x	1
0	1	1	0	0	1	1	1	0	x	x	0	x	0	1	x
0	1	1	1	1	0	0	0	1	x	x	1	x	1	x	1
1	0	0	0	1	0	0	1	x	0	0	x	0	x	1	x
1	0	0	1	0	0	0	0	x	1	0	x	0	x	x	1

Fig. 6.17(b) Excitation table of a BCD counter using JK FF

Step 4. The minimal expressions: From the excitation table we can conclude that J1 = 1 and K1=1, because all the entries for J1 and K1 are either X or 1.The K-maps for the excitations of FFs J4, K4, J3, K3, and J2, K2 in terms of the outputs of the FFs Q4, Q3, Q2 and Q1 based on the excitation table, their minimization and the minimal expressions obtained from them are shown in Figure 6.17 c.

Step 5. The logic diagram: The logic diagram based on those minimal expressions is drawn as shown in Figure 6.17 d.

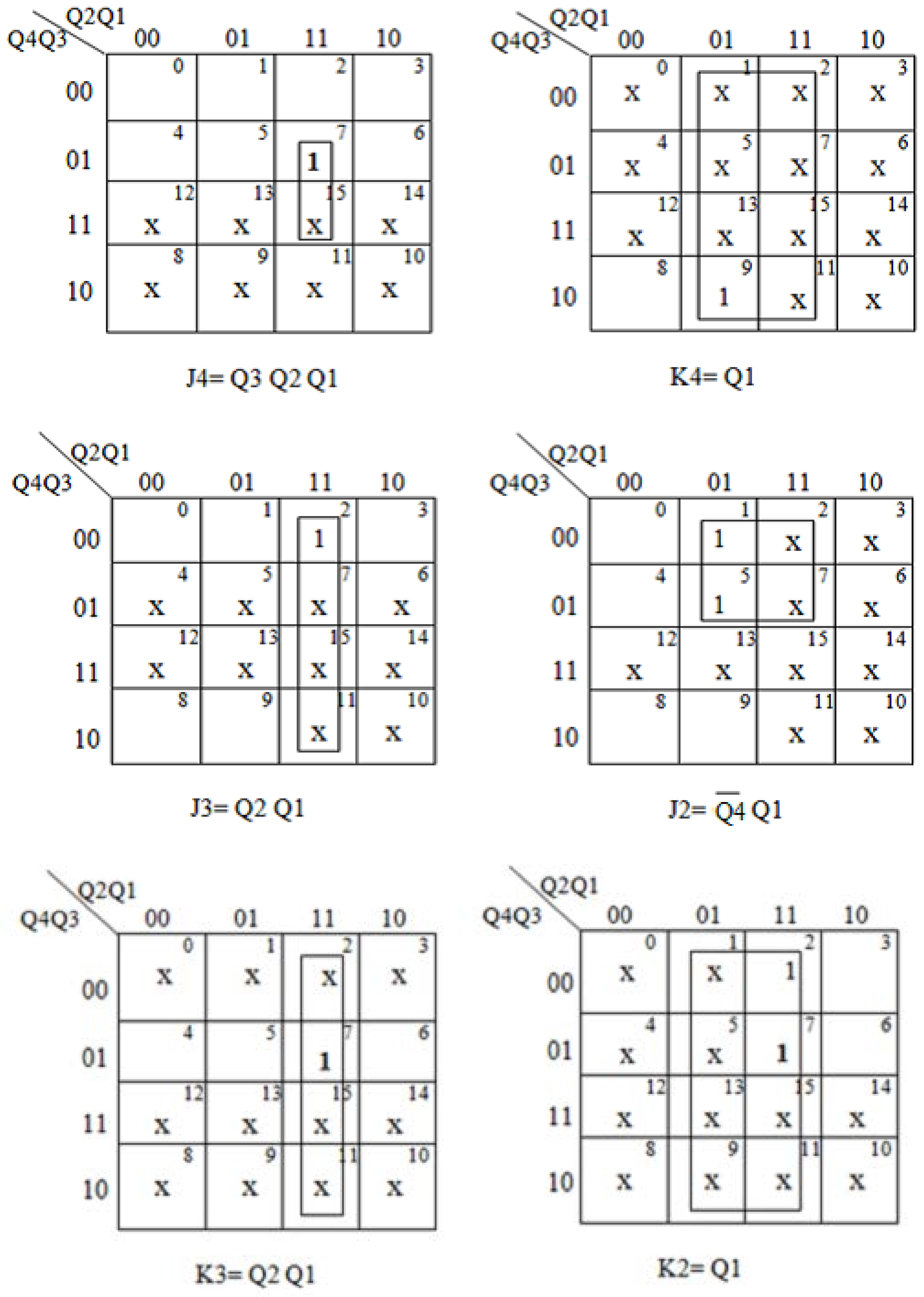


Fig. 6.17 (c) K-Maps for excitations of Synchronous BCD counter using JK flip-flops

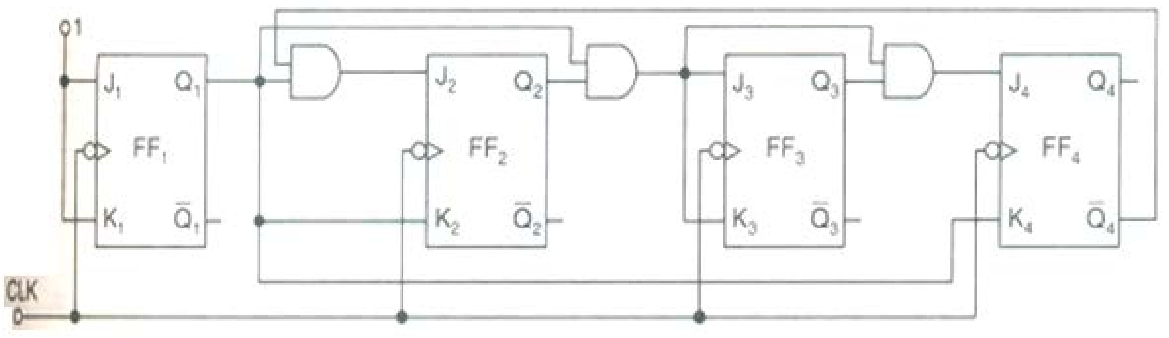


Fig 6.17(d) Logic diagram of Synchronous BCD counter using JK flip-flops

Design of a Mod·6 Counter Using J-K FFs

Step 1. The number of flip-flops: We know that the counting sequence for a mod-6 counter is 000,001, 010,011, 100, 101, and 000. It has six states. So it requires n = 3 FFs ($N \leq 2^n$, i.e. $6 \leq 2^3$). Three FFs can have eight states. So the remaining two states 110 and 111 are invalid. The entries for excitations corresponding to invalid states are don't cares.

Step 2. The state diagram: The state diagram for the mod-6 counter is drawn as shown in Figure 6.18 a.

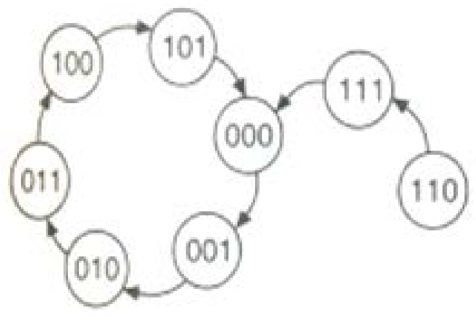


Figure 6.18(a) State diagram for the mod-6 counter

Step 3. The type of flip-flops and the excitation table: JK flip-flops are selected and the excitation table of a mod-6 counter using J-K FFs is drawn as shown in Figure 6.84b. (States 110 and 111 can be removed from the state diagram as it is not required).

PS			NS			Required Excitations					
Q3	Q2	Q1	Q3	Q2	Q1	J3	K3	J2	K2	J1	K1
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	0	0	0	X	1	0	X	X	1

Fig. 6.18(b) Excitation table of a mod-6 counter using JK FF

Step 4. The minimal expressions: The K-maps for excitations of FFs J3, K3, J2, K2, J1, and K1 in terms of the outputs of FFs Q3, Q2 and Q1, their minimization, and the minimal expressions for excitations obtained from them are shown in fig. 6.18 (c)

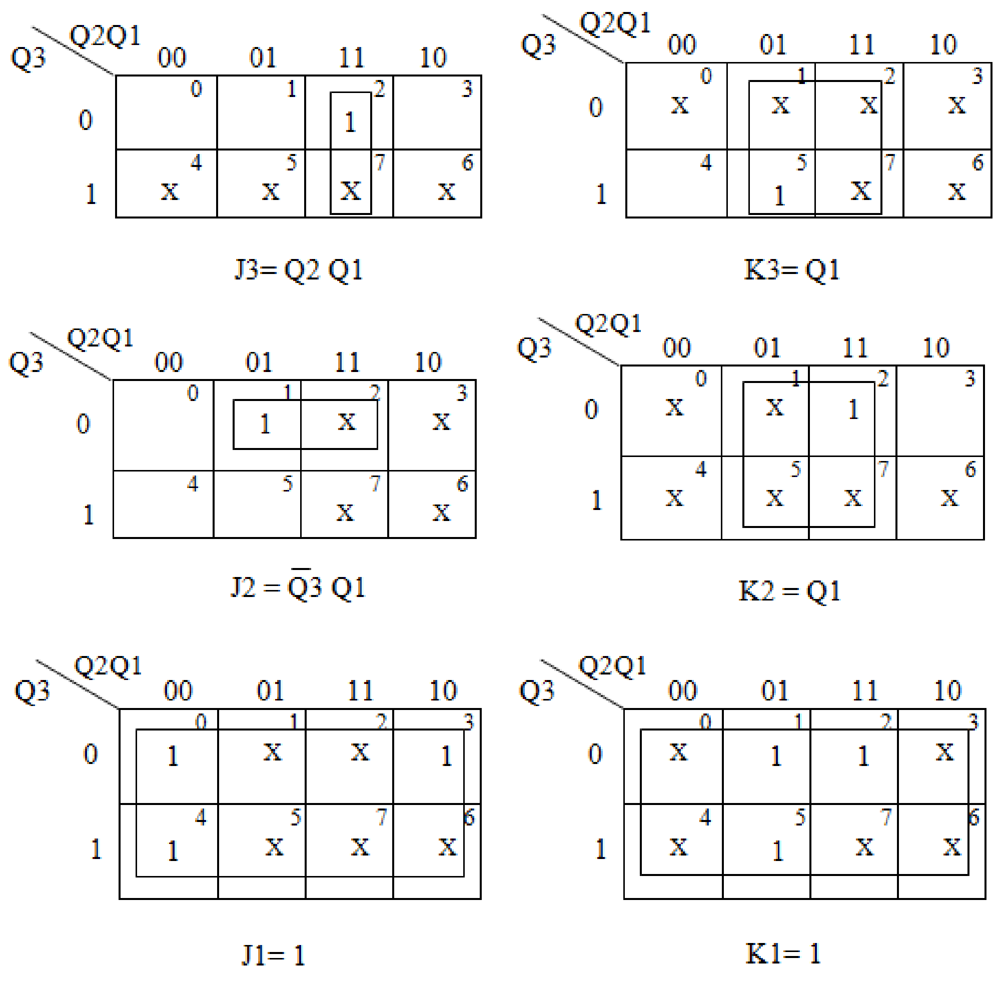


Fig. 6.18 (c) K-Maps for excitations of a mod-6 counter using JK flip-flops

Step 5. The logic diagram: The logic diagram based on these minimal expressions is drawn as shown in Figure 6.18 (d)

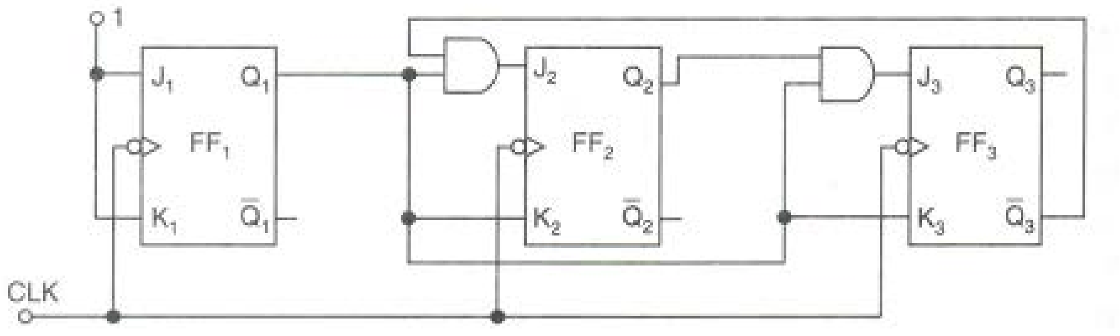


Fig 6.18(d) Logic diagram of Synchronous BCD counter using JK flip-flops

6.16 Assignment Questions

Short answer questions

1) Draw the block diagram of a sequential circuit (April/May 2012)
2) Draw the circuit diagram of an SR latch using NOR gate (April/May 2009)
3) What is flip flop? How is it different from combinational circuit? (April/May 2012)
4) Write the truth table for a clocked RS flip flop (April/May 2013)
5) Draw the logic diagram of RS flip flop using NAND or NOR gate (April/May 2008)
6) Write the truth table and logic diagram for a clocked D flip flop (2012,2011,2007)
7) Write the truth table for a clocked JK flip flop (April/May 2011)
8) Draw the logic diagram for JK flip flop
9) Write the characteristic equations for all the flip flops (April/May 2009)
10) What is shift register? Explain the shift operation with a neat diagram (April/May 2011)
11) What is a counter? How many flip flops are required to design a mod 12 counter? (April/May 2011)
12) Define a register and a counter. (2011,2010)
13) Draw a neat diagram of a 4- bit binary ripple counter (2013,2012,2009,2008,2007)
14) Define excitation table and characteristic table (April/May 2012,2008)
15) Write the excitation tables for RS and JK flip flops (April/May 2011)
16) What is an IC? Give the classification of IC based on no. of transistors(April/May 2011)
17) Define state table, state diagram and state equation (April/May 2013,2009)
18) What do you mean by race around condition in JK flip flop? How can it be solved? (April/May 2009)

Long answer questions

1) Explain the working of clocked- SR flip flop using truth table and logic diagram (2013,2012,2011,2009,2007)
2) Explain the working of D flip flop using NAND or NOR gate (2013,2012,2011)
3) With a neat diagram explain the working of a JK flip flop (April/May 2009)
4) Explain the working of T flip flop using NAND gate. Write the truth table and logical expression (May/June 2013)
5) Explain the working of Master-Slave flip-flop with truth table and logic diagram (April/May 2007)

6) Explain the triggering of flip flops & obtain the state equations for all the flip flops (April/May 2013)
7) What is Shift register? Explain the 4-bit shift register with neat diagram (2013,2012,2011,2010,2009,2007)
8) Write a note on Bidirectional Shift Register (April/May 2013)
9) Explain the working of a 4 bit binary ripple counter with a neat diagram (2013,2012,2009,2008,2007)
10) Explain state table and state diagram with the help of a suitable example (2013,2010,2009)
11) Define excitation table. Write the excitation tables for all the flip flops(2013,2011,2009)
12) Explain RS flip flop with logic diagram, characteristic table and characteristic equation (2013,2011,2009,2008)
13) Explain D FF with logic diagram, characteristic table & characteristic equation (April/May 2013)
14) Design of a 3-bit or mod-8 counter using T flip flop (April/May 2012,2011,2008)
15) Design a mod-7 synchronous counter using JK flip flop (April/May 2012,2009,2008)
16) Design a BCD or mod-10 counter using T flip flop
17) Design a mod-6 synchronous counter using D flip flop (April/May 2011,2013,2007)

Credit based Semester System Oct/Nov 2014-15

Time: 3 Hours **Marks:80**

1)(a) Convert $(153.513)_8$ to Binary 10 * 2 = 20

(b) Obtain the 1's and 2's complements of the following binary numbers:

(i) 1010101 (ii) 000001

(c) Explain the general Structure of 2 and 3 variable K-Map.

(d) What is minterm and maxterm?

(e) How to write complement of a Boolean function? Also write the complement of F(X,Y,Z) = X' YZ'+ X'Y'Z

(f) What is the difference between canonical forms and standard forms?

(g) What is Half Substractor? Write the truth table of Half Substractor

(h) Write the simplified Boolean functions for sum and carry of a half adder.

(i) What is a magnitude comparator?

(j) Write the SR latch circuit using NAND gate.

(k) Write the excitation table of SR flipflop

(l) Define state diagram and state equation

UNIT – I

2 (a) State the postulates of Boolean Algebra

(b) Perform the following substraction using 1's and 2's Complement

(i) (1001)2-$(1011)_2$ ii) (10011)2-$(1001)_2$

(c) State and Prove demorgan's theorem (5+5+5)

OR

3 (a) Convert $(225.225)_{10}$= ( ) $_2$=()$_8$=( )$_{16}$

(b) Using Boolean theorems and postulate, prove the following:

(i) x + x'y = x + y (ii) x'y'z + x'yz + xy' = x'z + xy'.

(c)Perform following Substraction using 9's and 10's Complements

(i) $(8052)_{10} – (3250)_{10}$ (ii) $(6320)_{10} – (8659)_{10}$ (5+5+5)

UNIT- II

4 (a) Implement Boolean function F=x’y’z + x’yz + yz’ with basic gates.

(b) Simplyfy the following Boolean functions using Karnaugh map method

F(x,y,z) = π 2,3,6,7

(c) What is gate ? Explain the working principles of AND, OR, and NOT gate (5+5+5)

OR

5(a) How do you get complement of a function? Find the complement of

F1 = x’y’z + z’y’z

F2 = x(y’z’ + yz)

(b) Prove that NAND is universal gate

(c) Using K-map simplify the following expression

F(A,B,C,D) = ∑ (0,1,2,4,5,6,8,9,12,13,14) (5+5+5)

UNIT –III

6 (a) What is full adder? Explain its working.

(b) Design 2-bit Magnitude Comparator

(c) What is binary parallel adder? Draw a block diagram of a 4-bit binary parallel adder and explain its working (5+5+5)

OR

7 (a) Explain the working of BCD adder with block diagram

(b) Design BCD to Excess-3 Code Converter

(c) What is Multiplexer? Describe a 4-line to 1-line multiplexer with logic diagram and function table. (6+5+4)

UNIT – IV

8 a) Explain the clocked RS flip-flop with logic diagram, characterstic table and characterstic equation

b) Design 4-bit ripple counter

c) What is register? Design 4-bit register . (5+6+4)

OR

9 a) Explain D Flip-flop with logic diagram, characteristic table and characteristic equation.

b) Write a note on Bidirectional Shift Register

c) Design a mod 6 synchronous counter (5+5+5