

---

## UNIT 3 PRINCIPLES OF LOGIC CIRCUITS I

---

Structure	Page Nos.
3.0 Introduction	60
3.1 Objectives	60
3.2 Logic Gates	60
3.3 Logic Circuits	62
3.4 Combinational Circuits	63
3.4.1 Canonical and Standard Forms	
3.4.2 Minimization of Gates	
3.5 Design of Combinational Circuits	72
3.6 Examples of Logic Combinational Circuits	73
3.6.1 Adders	
3.6.2 Decoders	
3.6.3 Multiplexer	
3.6.4 Encoder	
3.6.5 Programmable Logic Array	
3.6.6 Read Only Memory ROM	
3.7 Summary	82
3.8 Solutions/ Answers	82

---

### 3.0 INTRODUCTION

---

In the previous units, we have discussed the basic configuration of computer system von Neumann architecture, data representation and simple instruction execution paradigm. But 'How does a computer actually perform computations?'. Now, we will attempt to find answer of this basic query. In this unit, you will be exposed to some of the basic components that form the most essential parts of a computer. You will come across terms like logic gates, binary adders, logic circuits and combinational circuits etc. These circuits are the backbone of any computer system and knowing them is quite essential. The characteristics of integrated digital circuits are also discussed in this unit.

### 3.1 OBJECTIVES

---

After going through this unit you will be able to :

- define logic gates;
  - describe the significance of Boolean algebra in digital circuit design;
  - describe the necessity of minimizing the number of gates in design;
  - describe how basic mathematical operations, viz. addition and subtraction, are performed by computer; and
  - define and describe some of the useful circuits of a computer system such as multiplexer, decoders, ROM etc.
- 

### 3.2 LOGIC GATES

---

A logic gate is an electronic circuit which produces a typical output signal depending on its input signal. The output signal of a gate is a simple Boolean operation of its input signal. Gates are the basic logic elements that produce signals of binary 1 or 0

We can represent any Boolean function in the form of gates.

In general we can represent each gate through a distinct graphic symbol and its operation can be given by means of algebraic expression. To represent the input-

output relationship of binary variables in each gate, truth tables are used. The notations and truth -tables for different logic gates are given in Figure 3.1.






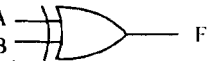
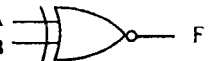
Name	Graphic Symbol	Algebraic function	Truth Table															
NOT		$F = \bar{A}$ or $F = A$	<table><tr><th>A</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	F	0	1	1	0									
A	F																	
0	1																	
1	0																	
AND		$F = A.B$ or $F = AB$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	F	0	0	0	0	1	0	1	0	0	1	1	1
A	B	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = A + B$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	1
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NAND		$F = \overline{A.B}$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	1	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = \overline{A + B}$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	1	0	1	0	1	0	0	1	1	0
A	B	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
Exclusive-OR (XOR)		$F = A\bar{B} + \bar{A}B$ $F = A \oplus B$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
Exclusive-NOR (XNOR)		$F = \overline{A \oplus B}$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	F	0	0	1	0	1	0	1	0	0	1	1	1
A	B	F																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

Figure 3.1: Logic Gates

The truth table of NAND and NOR can be made from NOT (A AND B) and NOT (A OR B) respectively. Exclusive OR (XOR) is a special gate whose output is one only if the two inputs are not equal. The inverse of exclusive OR, called as XNOR gate, can be a comparator which will produce a 1 output if two inputs are equal.

The digital circuits use only one or two types of gates for simplicity in fabrication purposes. Therefore, one must think in terms of functionally complete set of gates. What does functionally complete set imply? A set of gates by which *any* Boolean function can be implemented is called a functionally complete set. The functionally complete sets are: [AND, NOT], [NOR], [NAND], [OR, NOT].

### 3.3 LOGIC CIRCUITS

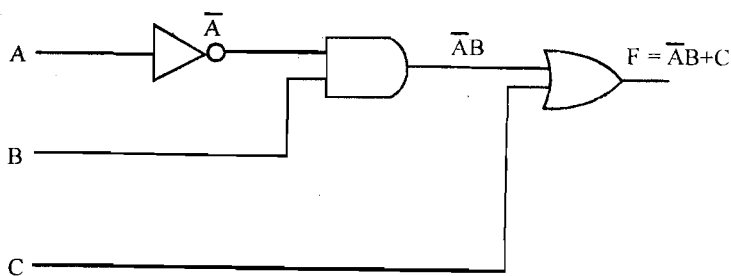
A Boolean function can be implemented into a logic circuit using the basic gates:- AND , OR & NOT. Consider, for example, the Boolean function: -

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

The relationship between this function and its binary variables A, B, C can be represented in a truth table as shown in figure 3.2(a) and figure 3.2(b) shows the corresponding logic circuit.

Inputs			Output
A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

(a) Truth Table



(b) Logic Circuit

Figure 3.2 : Truth table & logic diagram for  $F = \bar{A}B + C$

Thus, in a logic circuit, the variables coming on the left hand side of boolean expression are inputs to circuit and the variable function coming on the right hand side of expression is taken as output.

Here, there is one important point to note i.e. there is only one way to represent the boolean expression in a truth table but can be expressed in variety of logic circuits. How? [try to find the answer]

#### Check Your Progress 1

- 1) What are the logic gates and which gates are called as Universal gates.

.....

.....

- 2) Simplify the Boolean function:  $F = \overline{\left\{ \left( \overline{\overline{A} + \overline{B}} \right) + \left( \overline{A + \overline{B}} \right) \right\}}$

.....

.....

.....

- 3) Draw the logic diagram of the above function.

.....

.....

.....

.....

- 4) Draw the logic diagram of the simplified function.

.....  
 .....

- 5) Show implementation of AND, NOT and OR Operations using NAND gates.

.....  
 .....

### 3.4 COMBINATIONAL CIRCUIT

Combinational circuits are interconnected circuits of gates according to certain rules to produce an output depending on its input value. A well-formed combinational circuit should not have feedback loops. A combinational circuit can be represented as a network of gates and, therefore, can be expressed by a truth table or a Boolean expression.

The output of the combinational circuit is related to its input by a combinational function, which is independent of time. Therefore, for an ideal combinational circuit the output should change instantaneously according to changes in input. But in actual case there is a slight delay. The delay is normally proportional to *depth* or number of levels i.e. the maximum numbers of gates on any path from input to output. For example, the depth of the combinational circuit in figure 3.3 is 2.

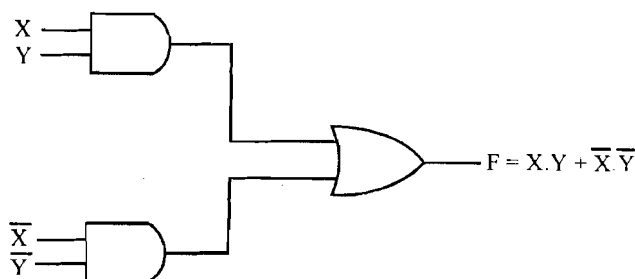


Figure 3.3 : A two level AND-OR combinational circuit

The basic design issue related to combinational circuits is: the *Minimization of number of gates*. The normal circuit constraints for combinational circuit design are :

- The depth of the circuit should not exceed a specific level,
- Number of input lines to a gate (fan in) and to how many gates its output can be fed (fan out) are constraint by the circuit power constraints.

#### 3.4.1 Canonical and Standard Forms

An algebraic expression can exist in two forms :

- Sum of Products (SOP) e.g.  $(A \cdot \bar{B}) + (\bar{A} \cdot \bar{B})$
- Product of Sums (POS) e.g.  $(\bar{A} + B) \cdot (A + B)$

If a product term of SOP expression contains every variable of that function either in true or complement form then it is defined as a **Minterm or Standard Product**. This minterm will be true only for one combination of input values of the variables. For example, in the SOP expression

$$F(A, B, C) = (A \cdot B \cdot C) + (\bar{A} \cdot \bar{B} \cdot C) + (A \cdot B)$$

We have three product terms namely  $A \cdot B \cdot C$ ,  $\bar{A} \cdot \bar{B} \cdot C$  and  $A \cdot B$ . But only first two of them qualifies to be a minterm, as the third one does not contain variable C or its

complement. In addition, the term  $A.B.C$  will be one only if  $A = 1$ ,  $B = 1$  and  $C = 1$ , for any other combination of values of  $A$ ,  $B$ ,  $C$  the minterm  $A.B.C$  will have 0 value. Similarly, the minterm  $\bar{A} \bar{B} . C$  will have value 1 only if  $\bar{A} = 1$ ,  $\bar{B} = 1$  and  $C = 1$  (It implies  $A=0$ ,  $B=0$  and  $C=1$ ) for any other combination of values the minterm will have a zero value. \*

Similar type of term used in POS form is called **Maxterm or Standard Sum**. Maxterm is a term of POS expression, which contains all the variables of the function in true or complemented form. For example,  $F(A, B, C) = (A + B + C) . (\bar{A} + \bar{B} + C)$  has two maxterms. A maxterm has a value 0, for only one combination of input values.

The maxterm  $A + B + C$  will have 0 value only for  $A = 0$ ,  $B = 0$  and  $C = 0$  for all other combination of values of  $A$ ,  $B$ ,  $C$  it will have a value 1.

Figure 3.4 indicates the  $2^n$  different minterms and maxterms where  $n$  is number of variables.

Variable's Value			Minterm		Maxterm	
a	b	c	Term	Representation	Term	Representation
0	0	0	$\bar{a} \bar{b} \bar{c}$	$m_0$	$a + b + c$	$M_0$
0	0	1	$\bar{a} \bar{b} c$	$m_1$	$a + b + \bar{c}$	$M_1$
0	1	0	$\bar{a} b \bar{c}$	$m_2$	$a + \bar{b} + c$	$M_2$
0	1	1	$\bar{a} b c$	$m_3$	$a + \bar{b} + \bar{c}$	$M_3$
1	0	0	$a \bar{b} \bar{c}$	$m_4$	$\bar{a} + b + c$	$M_4$
1	0	1	$a \bar{b} c$	$m_5$	$\bar{a} + b + \bar{c}$	$M_5$
1	1	0	$a b \bar{c}$	$m_6$	$\bar{a} + \bar{b} + c$	$M_6$
1	1	1	$a b c$	$m_7$	$\bar{a} + \bar{b} + \bar{c}$	$M_7$

Figure 3.4: Maxterms and Minterms for 3 variables

We can represent any Boolean function algebraically directly in minterm and maxterm form from the truth table. For *minterms*, consider each combination of variables that produces a 1 output in function and then taking OR of all those terms. For example, the function  $F$  in figure 3.5 is represented in minterm form by ORing the terms where the output  $F$  is 1 i.e.  $\bar{a} \bar{b} c$ ,  $\bar{a} b \bar{c}$ ,  $\bar{a} b c$ ,  $a \bar{b} \bar{c}$  &  $a b c$ .

a	b	c	F	
0	0	0	0	$m_0$
0	0	1	1	$m_1$
0	1	0	1	$m_2$
0	1	1	1	$m_3$
1	0	0	0	$m_4$
1	0	1	0	$m_5$
1	1	0	1	$m_6$
1	1	1	1	$m_7$

Figure 3.5: Function of three variables

$$\begin{aligned}
 \text{Thus, } F(a,b,c) &= \bar{a} \bar{b} c + \bar{a} b \bar{c} + \bar{a} b c + a \bar{b} \bar{c} + a b c \\
 &= m_1 + m_2 + m_3 + m_6 + m_7 \\
 &= \sum (1,2,3,6,7)
 \end{aligned}$$

The complement of function  $F$  can be obtained by ORing of the minterms corresponding to the combinations that produce a 0 output in function. Thus,

$$\bar{F}(a, b, c) = \bar{a}\bar{b}\bar{c} + a\bar{b}\bar{c} + a\bar{b}c$$

If we take the complement of  $\bar{F}$ , we get the function  $F$  in maxterm form.

$$\begin{aligned} F(a, b, c) &= (\bar{F}) = \overline{(\bar{a}\bar{b}\bar{c} + a\bar{b}\bar{c} + a\bar{b}c)} = \overline{(\bar{a}\bar{b}\bar{c})} \cdot \overline{(a\bar{b}\bar{c})} \cdot \overline{(a\bar{b}c)} \\ &= (a + b + c)(\bar{a} + b + c)(\bar{a} + b + \bar{c}) \quad [\text{De Morgan's law}] \\ &= M_0 \cdot M_4 \cdot M_5 \\ &= \Pi(0, 4, 5) \end{aligned}$$

The product symbol  $\Pi$  stands for ANDing the maxterms.

Here, you will appreciate the fact that the terms which were missing in minterm form are present in maxterm form. Thus if any form is known then the other form can be directly formed.

The Boolean function expressed as a sum of minterms or product of maxterms has the property that each and every literal of the function should be present in each and every term in either normal or complemented form.

### 3.4.2 Minimization of Gates

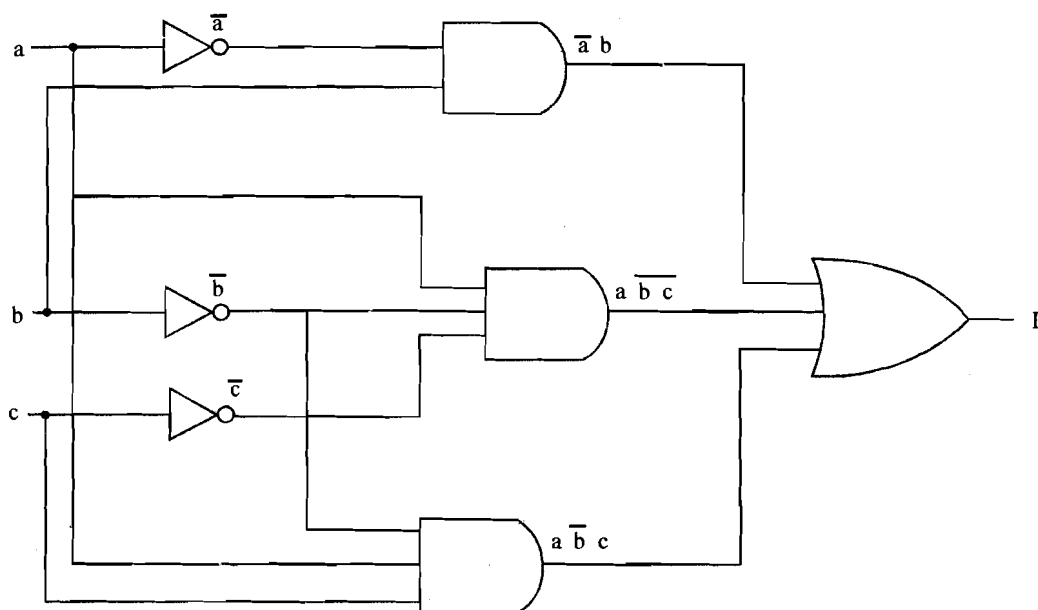
The simplification of Boolean expression is very useful for combinational circuit design. The following three methods are used for this:

- Algebraic Simplification
- Karnaugh Maps
- Quine McCluskey Method

#### Algebraic Simplification

We have already discussed algebraic simplification of logic circuit. An algebraic expression can exist in POS or SOP forms. Let us examine the following example to understand how it helps in implementing any logic circuit.

Example : Consider the function  $F(a, b, c) = a\bar{b}\bar{c} + a\bar{b}c + \bar{a}b$ . The logic circuit implementation of this function is shown in fig 3.6(a).



(a)  $F = a\bar{b}\bar{c} + a\bar{b}c + \bar{a}b$

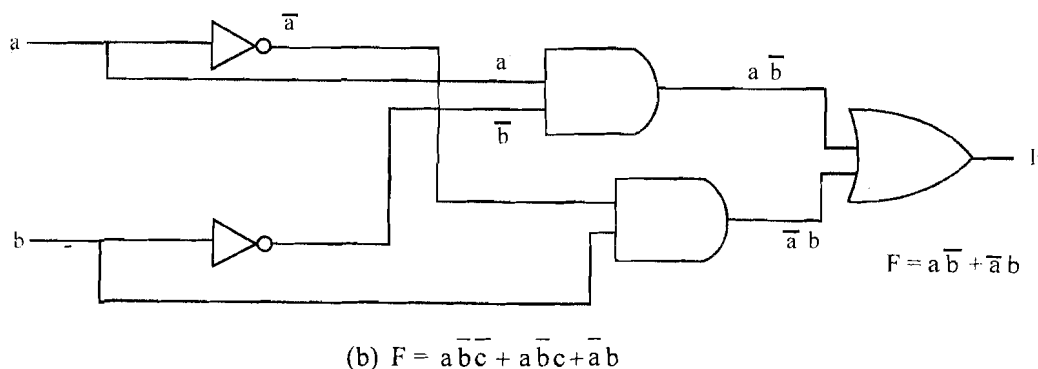


Figure 3.6 : Two logic diagrams for same boolean expression

The expression  $F$  can be simplified using boolean algebra.

$$\begin{aligned}
 F(a,b,c) &= a\bar{b}\bar{c} + a\bar{b}c + \bar{a}b \\
 &= a\bar{b}(\bar{c} + c) + \bar{a}b \quad [ \text{as } c + \bar{c} = 1 ] \\
 &= a\bar{b} + \bar{a}b \\
 &= a \oplus b
 \end{aligned}$$

The logic diagram of the simplified expression is drawn in fig 3.6 (b) using NOT, OR and AND gates (the same operation can be performed by using a single XOR gate). Thus the number of gates are reduced to 5 gates (2 inverters, 2 AND gates & 1 OR) instead of 7 gates. (3 inverters, 3 AND & 1 OR gate).

The algebraic function can appear in many different forms although a process of simplification exists yet it is cumbersome because of absence of routes which tell what rule to apply next. The Karnaugh map is a simple direct approach of simplification of logic expressions.

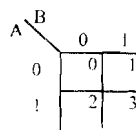
### Karnaugh Maps

Karnaugh maps are a convenient way of representing and simplifying Boolean function of 2 to 6 variables. The stepwise procedure for Karnaugh map is.

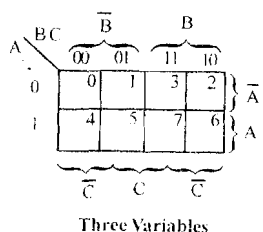
**Step 1:** Create a simple map depending on the number of variables in the function. Figure 3.7(a) shows the map of two, three and four variables. A map of 2 variables contains 4 value position or elements, while for 3 variables it has  $2^3 = 8$  elements. Similarly for 4 variables it is  $2^4 = 16$  elements and so on. Special care is taken to represent variables in the map. The value of only one variable changes in two adjacent columns or rows. The advantage of having change in one variable is that two adjacent columns or rows represent a true or complement form of a single variable.

For example, in figure 3.7(a) the columns which have positive  $A$  are adjacent and so are the column for  $\bar{A}$ . Please note the adjacency of the corners. The right most column can be considered to be adjacent to the first column since they differ only by one variable and are adjacent. Similarly the top most and bottom most rows are adjacent.

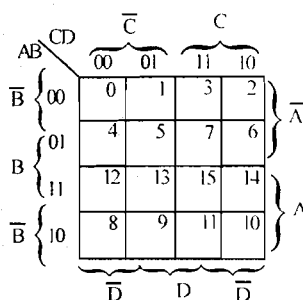
Decimal	A	B
0	0	0
1	0	1
2	1	0
3	1	1



Decimal	A	B	C
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

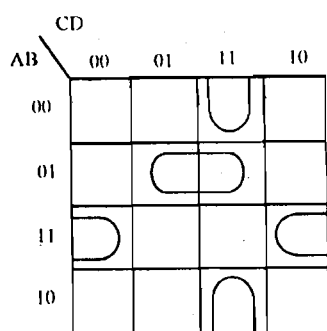


Decimal	A	B	C	D
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

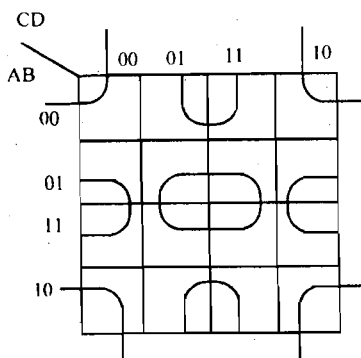


Four Variables

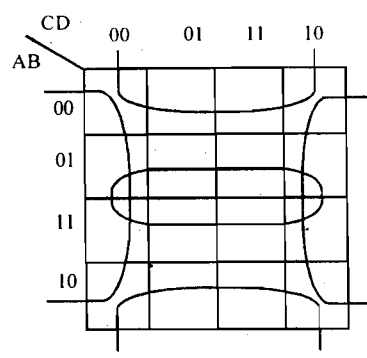
(a) Maps for 2, 3 and 4 variables



Type of adjacencies for two squares



Type of adjacencies for 4 squares



Type of adjacencies for 8 squares

(b) Possible adjacencies

Figure 3.7: Maps and their adjacencies

**Please note:**

- 1) Decimal equivalents of column are given for help in understanding where the position of the respective set lies. It is *not* the value filled in the square. A square can contain one or nothing.
- 2) The 00, 01, 11 etc written on the top implies the value of the respective variables.
- 3) Wherever the value of a variable is 0 it is said to represent its complement form.
- 4) The value of only one variable changes when we move from one row to the next row or one column to the next column.

**Step 2:** The next step in Karnaugh map is to map the truth table into the map. The mapping is done by putting a 1 in the respective square belonging to the 1 value in the truth table. This mapped map is used to arrive at simplified Boolean expression which then can be used for drawing up the optimal logical circuit. Step 2 will be more clear in the example.

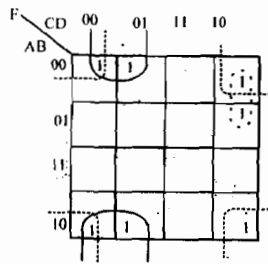
**Step 3:** Now, create simple algebraic expression from the K-Map. These expressions are created by using adjacency if we have two adjacent 1's then the expression for those can be simplified together since they differ only in 1 variable. Similarly, we search for the adjacent pairs of 4, 8 and so on. A 1 can appear in more than one adjacent pairs. We should search for octets first then quadrets and then for doublets. The following example will clarify the step 3.

**Example:** Now, let us see how to use K map simplification for finding the Boolean function for the cases whose truth table is given in figure 3.8(a) and 3.8(B) shows the K-Map for this.

Decimal	A	B	C	D	Output F
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

$$\text{Or } F = \sum (0, 1, 2, 6, 8, 9, 10)$$

(a) Truth table



(b) Karnaugh's map

Figure 3.8 : Truth table & K-Map of Function  $F = \sum (0, 1, 2, 6, 8, 9, 10)$

Let us see what the pairs which can be considered as adjacent in the Karnaugh's here.

The pairs are:

- 1) The four corners
- 2) The four 1's as in top and bottom in column 00 & 01
- 3) The two 1's in the top two rows of last column.

The corners can be represented by the expressions :

- 1) Four corners

$$\begin{aligned}
 &= (\bar{A} \bar{B} \bar{C} \bar{D} + \bar{A} \bar{B} C \bar{D}) + (\bar{A} \bar{B} \bar{C} D + \bar{A} \bar{B} C D) \\
 &= \bar{A} \bar{B} \bar{D} (\bar{C} + C) + \bar{A} \bar{B} D (\bar{C} + C) \quad [\text{as } C + \bar{C} = 1] \\
 &= \bar{A} \bar{B} \bar{D} + \bar{A} \bar{B} D \\
 0 &= \bar{B} \bar{D} (\bar{A} + A) \\
 &= \bar{B} \bar{D}
 \end{aligned}$$

- 2) The four 1's in column 00 and 01 gives the following terms

$$\begin{aligned}
 &= (\bar{A} \bar{B} \bar{C} \bar{D} + \bar{A} \bar{B} \bar{C} D) + (A \bar{B} \bar{C} \bar{D} + A \bar{B} \bar{C} D) \\
 &= \bar{A} \bar{B} \bar{C} (\bar{D} + D) + A \bar{B} \bar{C} (\bar{D} + D) \\
 &= \bar{A} \bar{B} \bar{C} + A \bar{B} \bar{C} \\
 &= \bar{B} \bar{C}
 \end{aligned}$$

- 3) The two 1's in the last columns

$$\begin{aligned}
 &= \bar{A} \bar{B} C \bar{D} + \bar{A} B C \bar{D} \\
 &= \bar{A} C \bar{D} (\bar{B} + B) \\
 &= \bar{A} C \bar{D}
 \end{aligned}$$

Thus, the Boolean expression derived from this K-Map is

$$F = \bar{B} \bar{D} + \bar{B} \bar{C} + \bar{A} C \bar{D}$$

[Note : This expression can be directly obtained from the K-Map after making quadrets and doublets. Try to find how ?]

The expressions so obtained through K-Maps are in the forms of the sum of the product form i.e. it is expressed as the sum of the products of the variables. This expression can be expressed in product of sum form, but for this special method are required to be used [already discussed in last section].

Let us see how we can modify K-Map simplification to obtain POS form. Suppose in the previous example instead of using 1 we combined the adjacent 0 squares then we will obtain the inverse function and on taking transform of this function we will get the POS form.

Another important aspect about this simple method of digital circuit design is DONOT care conditions. These conditions further simplify the algebraic function. These conditions imply that it does not matter whether the output produced is 0 or 1 for the specific input. These conditions can occur when the combination of the number of inputs are more than needed. For example, calculation through BCD where 4 bits are used to represent a decimal digit implies we can represent  $2^4 = 16$  digits but since we have only 10 decimal digits therefore 6 of those input combination values do not matter and are a candidate for DONOT care condition.

For the purpose of exercises you can do the exercise from the reference [1], [2], [3] given in Block introduction.

What will happen if we have more than 4– 6 variables? As the numbers of variables increases K-Maps become more and more cumbersome as the numbers of possible combinations of inputs keep on increasing.

### Quine McKluskey Method

A tabular method was suggested to deal with the increasing number of variables known as Quine McKluskey Method. This method is suitable for programming and hence provides a tool for automating design in the form of minimizing Boolean expression.

The basic principle behind the Quine McKluskey Method is to remove the terms, which are redundant and can be obtained by other terms.

To understand Quine - Mc Kluskey method, let's see following example:-

$$\begin{aligned}
 \text{Given, } F(A,B,C,D,E) &= ABCDE + ABC\bar{D}E + A\bar{B}\bar{C}DE + \bar{A}BCD\bar{E} + \\
 &\quad \bar{A}\bar{B}CDE + \bar{A}\bar{B}\bar{C}DE + A\bar{B}\bar{C}\bar{D}E + \bar{A}\bar{B}\bar{C}\bar{D}E
 \end{aligned}$$

**Step I:** The terms of the function are placed in table as follows:

Term/var	A	B	C	D	E	Checked/Unchecked
ABCDE	1	1	1	1	1	✓
ABC $\bar{D}$ E	1	1	1	0	1	✓
A $\bar{B}$ $\bar{C}$ DE	1	0	0	1	1	✓
$\bar{A}$ BCD $\bar{E}$	0	1	1	1	0	✓
$\bar{A}$ $\bar{B}$ CD $\bar{E}$	1	0	1	1	0	✓
$\bar{A}$ $\bar{B}$ $\bar{C}$ DE	0	0	0	1	1	✓
A $\bar{B}$ $\bar{C}$ $\bar{D}$ E	1	0	0	0	1	✓
$\bar{A}$ $\bar{B}$ $\bar{C}$ $\bar{D}$ $\bar{E}$	0	0	0	0	0	✓

**Step II:** Forming the pairs which differ in only one variable, also put check (✓) against the terms selected and finding resultant terms as follows :-

$$\left. \begin{array}{l} ABCDE \\ ABC\bar{D}E \end{array} \right\} \longrightarrow \boxed{ABCE}$$

$$\left. \begin{array}{l} \bar{A}\bar{B}\bar{C}DE \\ A\bar{B}\bar{C}DE \end{array} \right\} \longrightarrow \boxed{\bar{B}\bar{C}DE} \quad \checkmark$$

$$\left. \begin{array}{l} \bar{A}BCD\bar{E} \\ A\bar{B}CD\bar{E} \end{array} \right\} \longrightarrow \boxed{\bar{A}CD\bar{E}}$$

$$\left. \begin{array}{l} A\bar{B}\bar{C}\bar{D}E \\ A\bar{B}\bar{C}\bar{D}\bar{E} \end{array} \right\} \longrightarrow \boxed{\bar{B}\bar{C}\bar{D}E} \quad \checkmark$$

In the new terms, again find all the terms which differ only in one variable and put a check (✗) across those terms i.e.

$$\left. \begin{array}{l} \bar{B}\bar{C}DE \\ \bar{B}\bar{C}\bar{D}E \end{array} \right\} \longrightarrow \boxed{\bar{B}\bar{C}E}$$

**Step III:** Now, constructing final table as :

	ABCDE	ABC $\bar{D}$ E	$\bar{A}\bar{B}\bar{C}$ DE	A $\bar{B}\bar{C}$ D $\bar{E}$	$\bar{A}BC\bar{D}$	$\bar{A}\bar{B}CD\bar{E}$	$\bar{A}\bar{B}\bar{C}\bar{D}E$	$\bar{A}\bar{B}\bar{C}\bar{D}\bar{E}$
ABCE	✗	✗						
$\bar{A}CD\bar{E}$					✗	✗		
$\bar{B}\bar{C}E$			✗	✗			✗	✗

Thus all columns have mark 'X'. Thus the **final expression** is:

$$F(A,B,C,D,E) = ABCE + \bar{A}CD\bar{E} + \bar{B}\bar{C}E$$

The process can be summarised as follows:-

**Step I :** Build a table in which each term of the expression is represented in row (Expression should be in SOP form). The terms can be represented in the 0 (Complemented) or 1 (normal) form.

**Step II :** Check all the terms that differ in only one variable and then combine the pairs by removing the variable that differs in those terms. Thus a new table is formed.

This process is repeated, if necessary, in the new table also until all uncommon terms are left i.e. no matches left in table.

**Step III :**

- a) Finally, a two dimensional table is formed all terms which are not eliminated in the table form rows and all original terms form the column.
- b) At each intersection of row and column where row term is subset of column term, a 'X' is placed.

**Step IV :**

- a) Put a square around each 'X' which is alone in column
- b) Put a circle around each 'X' in any row which contains a squared 'X'
- c) If every column has a squared or circled 'X' then the process is complete and the corresponding minimal expression is formed by all row terms which have marked Xs.

## Check Your Progress 2

1) Prepare the truth table for the following boolean expressions:

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

(ii)  $(A+B) \cdot (\bar{A} + \bar{B})$

2 Simplify the following functions using algebraic simplification procedures and draw the logic diagram for the simplified function.

(i)  $F = ((A \cdot B) + B)$

(ii)  $F = ((A \cdot B) \cdot (\bar{A} \bar{B}))$

.....

.....

.....

.....

.....

.....

3) Simplify the following boolean functions in SOP and POS forms by means of K-Maps.

Also draw the logic diagram.

$F(A,B,C,D) = \Sigma (0,2,8,9,10,11,14,15)$

.....

.....

.....

.....

### 3.5 DESIGN OF COMBINATIONAL CIRCUITS

The digital circuits, which we use now-a-days, are constructed with NAND or NOR gates instead of AND-OR-NOT gates. NAND & NOR gates are called *Universal Gates* as we can implement any digital system with these gates. To prove this point we need to only show that the basic gates : AND , OR & NOT, can be implemented with either only NAND or with only NOR gate. This is shown in figure 3.9 below:

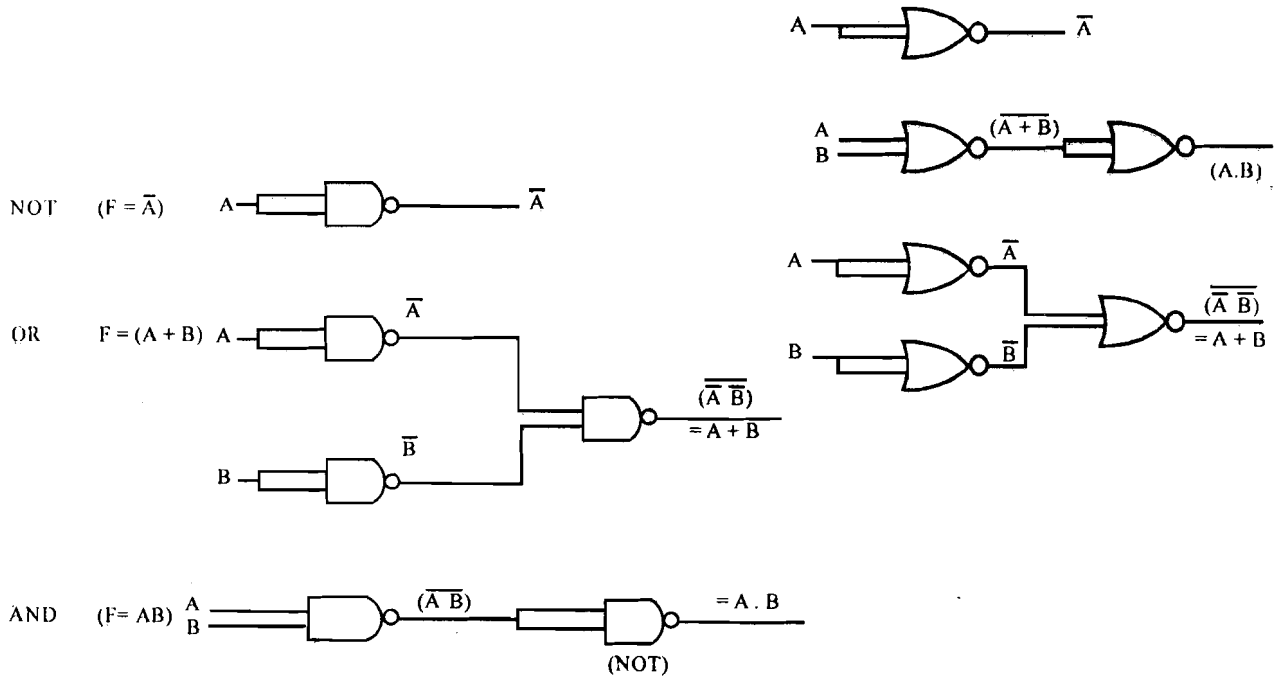


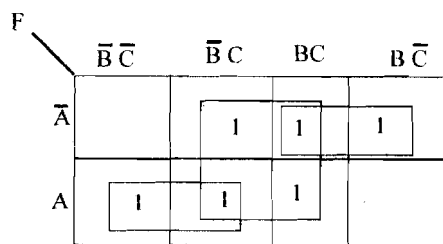
Figure 3.9 : Basic Logic Operations with NAND and NOR gates

Any Boolean expression can be implemented with NAND gates, by expressing the function in sum of product form.

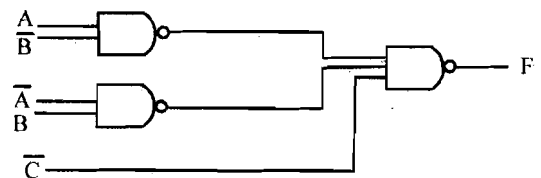
*Example:* Consider the function  $F(A, B, C) = \Sigma(1, 2, 3, 4, 5, 7)$ . Firstly bring it in SOP form. Thus, from the K-Map shown in figure 3.10(a), we find

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

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



(a) K-Map



(b) Logic circuit using NAND only

Figure 3.10: K-Map & Logic circuit for function  $F(A, B, C) = \Sigma(1, 2, 3, 4, 5, 7)$ .

Similarly, any Boolean expression can be implemented with only NOR gate by expressing in POS form. Let us take same example,  $F(A, B, C) = \Sigma(1, 2, 3, 4, 5, 7)$ .

As discussed in section 3.4.1, the above function  $F$  can be represented in POS form as

$$F(A, B, C) = \prod(0, 6)$$

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

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

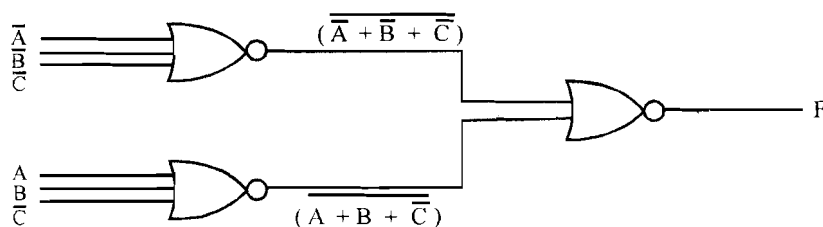


Figure 3.11: Logic circuit for function  $F(A, B, C) = \Sigma(1, 2, 3, 4, 5, 7)$  using NOR gates

After discussing so much about the design let us discuss some important combinational circuits. We will not go into the details of their design in this unit.

## 3.6 EXAMPLES OF COMBINATIONAL CIRCUITS

The design of combinational circuits can be demonstrated with some basic combinational circuits like adders, decoders, multiplexers etc. Let us discuss each of these examples briefly.

### 3.6.1 Adders

Adders play one of the most important roles in binary arithmetic. In fact fixed point addition is often used as a simple measure to express processor's speed. Addition and subtraction circuit can be used as the basis for implementation of multiplication and division. ( we are not giving details of these, you can find it in Suggested Reading).

Thus, considerable efforts have been put in designing of high speed addition and subtraction circuits. It is considered to be an important task since the time of Babbage. Number codes are also responsible for adding to the complexity of arithmetic circuit. The 2's complement notation is one of the most widely used codes for fixed-point binary numbers because of ease of performing addition and subtraction through it.

A combinational circuit which performs addition of two bits is called a *half adder*, while the combinational circuit which performs arithmetic addition of three bits (the third bit is the previous carry bit) is called a *full adder*.

In half adder the inputs are:

The augend lets say 'x' and addend 'y' bits.

The outputs are sum 'S' and carry 'C' bits.

The logical relationship between these are given by the truth table as shown in figure 3.12 (a). Carry 'C' can be obtained on applying AND gate on 'x' & 'y' inputs, therefore,  $C = x.y$ , while S can be found from the Karnaugh Map as shown in figure 3.12(b). The corresponding logic diagram is shown in figure 3.12(c).

Thus, the sum and carry equations of half- adder are:

$$S = x.\bar{y} + \bar{x}.y$$

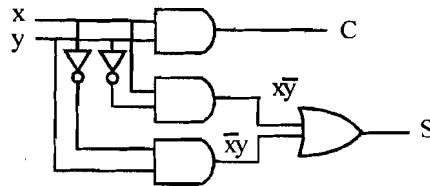
$$C = x.y$$

Inputs		Carry	Sum
x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(a) Truth table

S	y		
		0	1
x	0		1
	1	1	

(b) K- Map for 'S'



(c) Logic Diagram

Figure 3.12: Half – Adder implementation

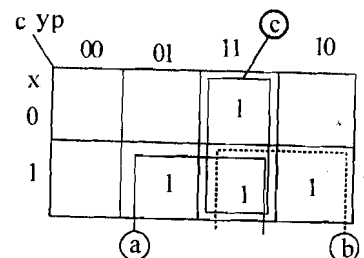
Let us take the full adder. For this another variable carry from previous bit addition is added let us call it 'p'. The truth table and K-Map for this is shown in figure 3.13.

Inputs			Carry	Sum
x	y	p	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

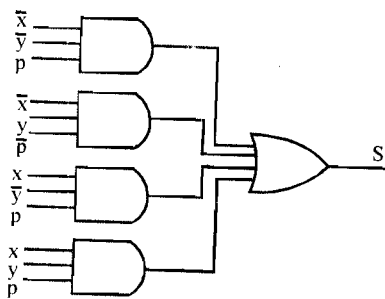
(a) Truth table

S	yp				
		00	01	11	10
x	0		1		1
	1	1		1	

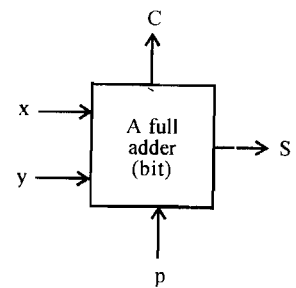
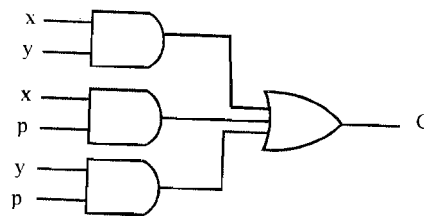
(b) K-Map for 'S'



(c) K – Maps for 'C'



(d) Logic diagram



(e) Block Diagram

Figure 3.13 : Full-adder implementation

Three adjacencies marked a,b,c in K-Map of 'C' are

$$\begin{aligned} \text{a) } & x \bar{y} p + x y p \\ &= x p (y + \bar{y}) \\ &= x p \end{aligned}$$

$$\begin{aligned} \text{b) } & x y p + x y \bar{p} \\ &= x y \end{aligned}$$

$$\begin{aligned} \text{c) } & \bar{x} y p + x y \bar{p} \\ &= y p \end{aligned}$$

Thus,  $C = x p + x y + y p$

In case of K-Map for 'S', there are no adjacencies. Therefore,

$$S = \bar{x} \bar{y} p + \bar{x} y \bar{p} + x \bar{y} \bar{p} + x y p$$

Till now we have discussed about addition of bit only but what will happen if we are actually adding two numbers. A number in computer can be 4 byte i.e. 32 bit long or even more. Even for these cases the basic unit is the full adder. Let us see (for example) how can we construct an adder which adds two 4 bit numbers. Let us assume that the numbers are:  $x_3 x_2 x_1 x_0$  and  $y_3 y_2 y_1 y_0$ ; here  $x_i$  and  $y_i$  ( $i = 0$  to  $3$ ) represent a bit. The 4-bit adder is shown in figure 3.14.

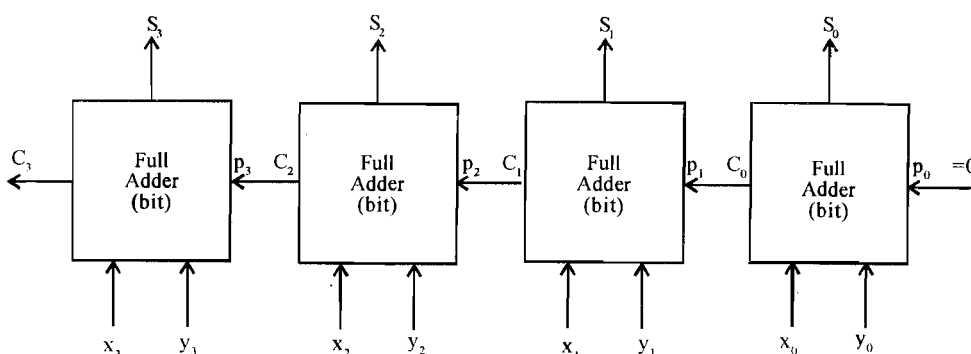


Figure 3.14 : 4-bit Adder

The overall sum is represented by  $S_3 S_2 S_1 S_0$  and over all carry is  $C_3$  from the 4th bit adder. The main feature of this adder is that carry of each lower bit is fed to the next higher bit addition stage, it implies that addition of the next higher bit has to wait for the previous stage addition. This is called ripple carry adder. The ripple carry becomes time consuming when we are going for addition of say 32 bit. Here the most significant bit i.e. the 32<sup>nd</sup> bits has to wait till the addition of first 31 bits is complete.

Therefore, a high-speed adder, which generates input carry bit of any stage directly from the input to previous stages was developed. These are called carry lookahead adders. In this adder the carry for various stages can be generated directly by the logic expressions such as:

$$C_0 = x_0 y_0$$

$$C_1 = x_1 y_1 + (x_1 + y_1) C_0$$

The complexity of the look ahead carry bit increases with higher bits. But in turn it produces the addition in a very small time. The carry look ahead becomes increasingly complicated with increasing numbers of bits. Therefore, carry look ahead adders are normally implemented for adding chunks of 4 to 8 bits and the carry is rippled to next chunk of 4 to 8 bits carry look ahead circuit.

### Adder- subtractor

The subtraction operation on binary numbers can be achieved by sequence of addition operations only i.e. to perform subtraction,  $A-B$ , we can find 2's complement of  $B$ . This can be calculated using 1's complement & then adding 1 to it. Thus, a common circuit can perform the addition and subtraction operation. A 4-bit adder- subtraction circuit is shown in figure 3.15, which is formed by using XOR gate with every full adder. The XOR gate with output 0 is for detecting overflow.

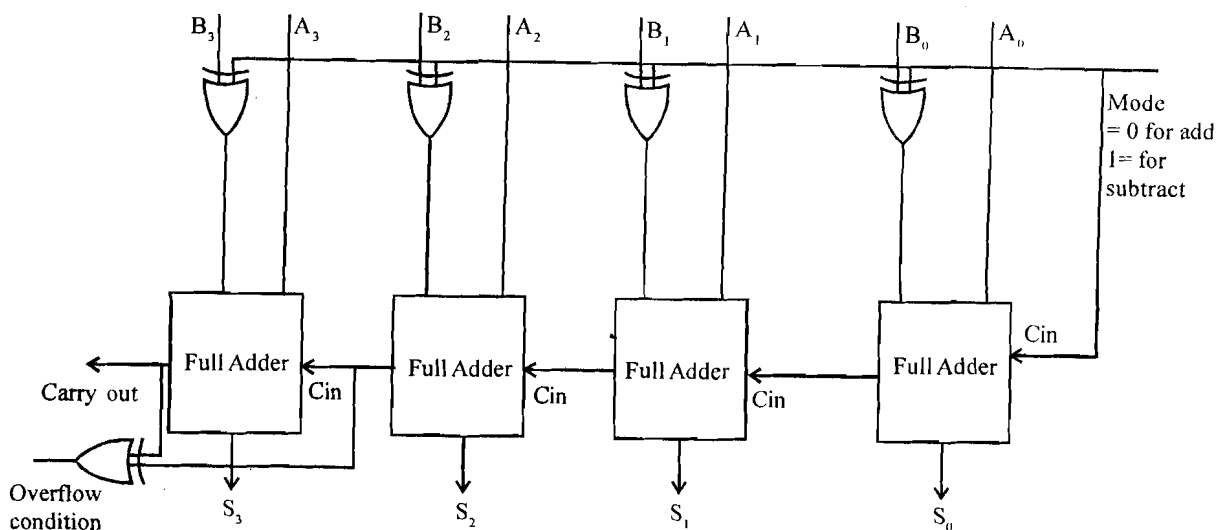


Figure 3.15: 4-bit adder-subtractor circuit

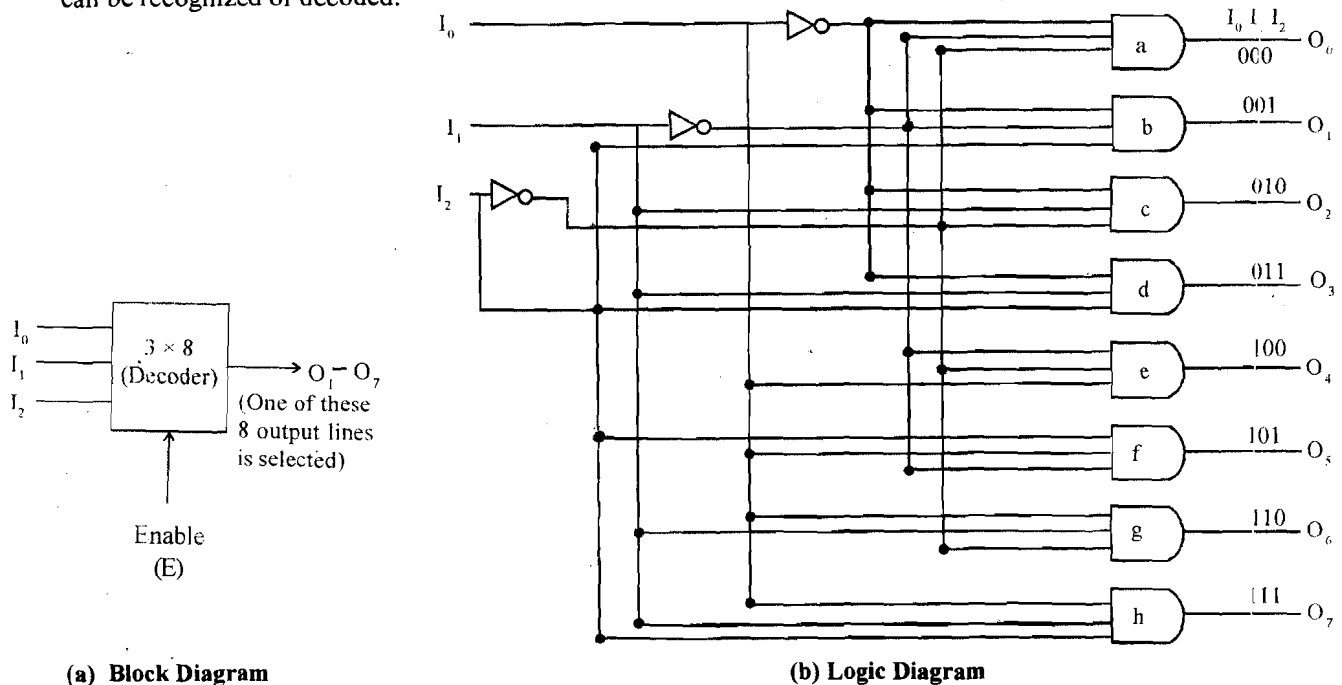
The control input 'x' controls the operations i.e. if  $x = 0$  then the circuit behaves like an adder and if  $x = 1$  then circuit behaves like a subtractor. The operation is summarized as :

- When  $x = 0$ ,  $c = 0$ , the output of all XOR gates will be the same as the corresponding input  $B_i$  where  $i = 0$  to 3. Thus,  $A_i$  &  $B_i$  are added through full adders giving Sum,  $S_i$  & carry  $C_i$

- b) When  $x = 1$ , the output of all XOR gates will be complement of input  $B_i$  where  $i = 0$  to 3, to which carry  $C_0 = 1$  is added. Thus, the circuit finds  $A$  plus 2's complement of  $B$ , that is equal to  $A - B$ .

### 3.6.2 Decoders

Decoder converts one type of coded information to another form. A decoder has ' $n$ ' inputs and an enable line (a sort of selection line) and  $2^n$  output lines. Let us see an example of  $3 \times 8$  decoder which decodes a 3 bit information and there is only one output line which gets the value 1 or in other words, out of  $2^3 = 8$  lines only 1 output line is selected. Thus, depending on selected output line the information of the 3 bits can be recognized or decoded.



Input			Output							
$I_0$	$I_1$	$I_2$	$O_0$	$O_1$	$O_2$	$O_3$	$O_4$	$O_5$	$O_6$	$O_7$
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

(c) Truth Table

Figure 3.16 :  $3 \times 8$  decoder

Please make sure while constructing the logic diagram wherever the values in the truth table are appearing as zero in input and one in output the input should be fed in complemented form e.g. the first 4 entries of truth table contains 0 in  $I_0$  position and hence  $I_0$  value 0 is passed through a NOT gate and fed to AND gates 'a', 'b', 'c' and 'd' which implies that these gates will be activated/selected only if  $I_0$  is 0. If  $I_0$  value is 1 then none of the top 4 AND gates can be activated. Similar type of logic is valid for  $I_1$ . Please note the output line selected is named 000 or 010 or 111 etc. The output value of only one of the lines will be 1. These 000, 010 indicates the label and suggest that if you have these  $I_0 I_1 I_2$  input values the labeled line will be selected for the output. The enable line is a good resource for combining two  $3 \times 8$  decoders to make one  $4 \times 16$  decoder.

### 3.6.3 Multiplexer

Multiplexer is one of the basic building units of a computer system which in principle allows sharing of a common line by more than one input lines. It connects multiple input lines to a single output line. At a specific time one of the input lines is selected and the selected input is passed on to the output line. The diagram  $4 \times 1$  multiplexer (MUX) is given in figure 3.16.

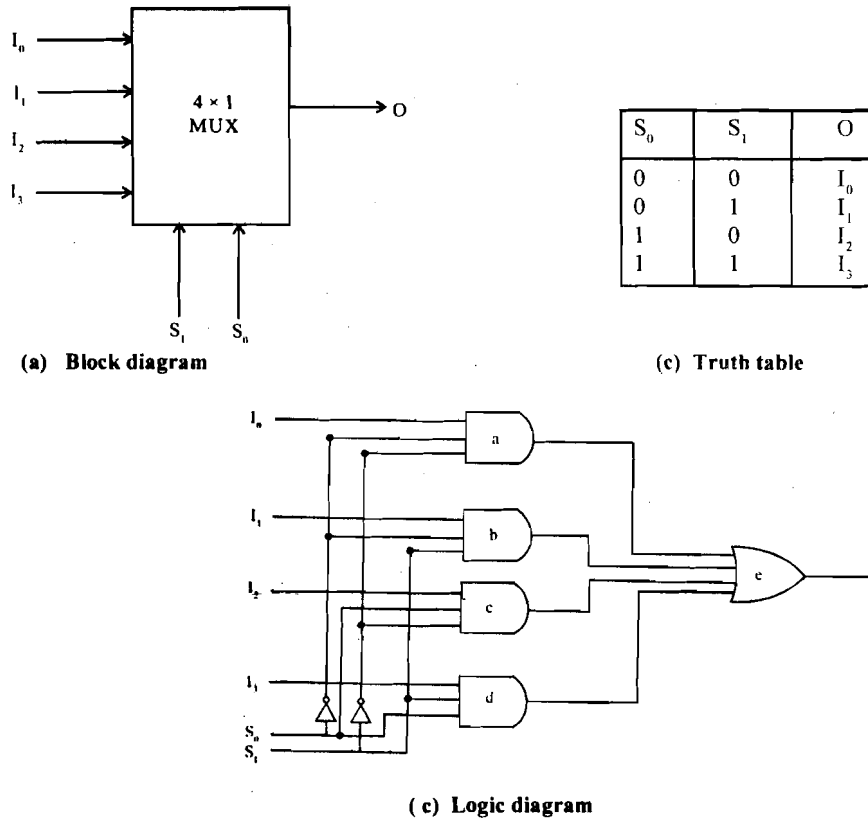


Figure 3.17:  $4 \times 1$  Multiplexer

But how does the multiplexer know which line to select? This is controlled by the select lines. The select lines provide the communication among the various components of a computer. Now let us see how the multiplexer also known as MUX works, here for simplicity we will take the example of  $4 \times 1$  MUX i.e. there are 4 input lines connected to 1 output line. For the sake of consistency we will call input line as  $I$ , and output line as  $O$  and control line a selection line  $S$  or enable as  $E$ .

Please notice the way in which  $S_0$  and  $S_1$  are connected in the circuit. To the 'a' AND gate  $S_0$  and  $S_1$  are inputted in complement form that means 'a' gate will output  $I_0$  when both the selection lines have a value 0 which implies  $\overline{S_0} = 1$  and  $\overline{S_1} = 1$ , i.e.  $S_0 = 0$  and  $S_1 = 0$  and hence the first entry in the truth table. Please note that at  $S_0 = 0$  and  $S_1 = 0$ , AND gate 'b', 'c', 'd' will yield 0 output and when all these outputs will pass OR gate 'e' they will yield  $I_0$  as the output for this case. That is for  $S_0 = 0$  and  $S_1 = 0$  the output becomes  $I_0$ , which in other words can be said as "For  $S_0 = 0$  and  $S_1 = 0$ ,  $I_0$  input line is selected by MUX". Similarly other entries in the truth table are corresponding to the logical nature of the diagram. Therefore, by having two control lines we could have a  $4 \times 1$  MUX. To have  $8 \times 1$  MUX we must have 3 control lines or with 3 control lines we could make  $2^3 = 8$  i.e.  $8 \times 1$  MUX. Similarly, with 'n' control lines we can have

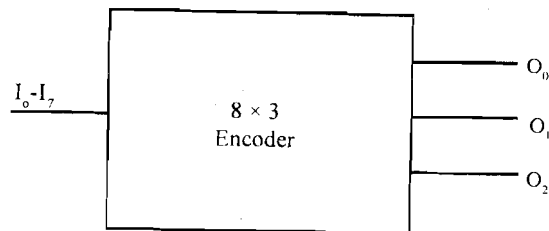
$2^n \times 1$  MUX. Another parameter which is predominant in MUX design is a number of inputs to AND gate. These inputs are determined by the voltage of the gate, which normally support a maximum of 8 inputs to a gate.

Where can these devices be used in the computer? The multiplexers are used in digital circuits for data and controlled signal routing.

We have seen a concept where out of 'n' input lines, 1 can be selected, can we have a reverse concept i.e. if we have one input line and data is transmitted to one of the possible  $2^n$  lines where 'n' represents the number of selection lines. This operation is called *Demultiplexing*.

### 3.6.4 Encoders

An Encoder performs the reverse function of the decoder. An encoder has  $2^n$  input lines and 'n' output line. Let us see the  $8 \times 3$  encoder which encodes 8 bit information and produces 3 outputs corresponding to binary numbers. This type of encoder is also called octal-to-binary encoder. The truth table of encoder is shown in figure 3.17.



(a) Block diagram

$I_0$	$I_1$	$I_2$	$I_3$	$I_4$	$I_5$	$I_6$	$I_7$		$O_2$	$O_1$	$O_0$
1	0	0	0	0	0	0	0	$D_0$	0	0	0
0	1	0	0	0	0	0	0	$D_1$	0	0	1
0	0	1	0	0	0	0	0	$D_2$	0	1	0
0	0	0	1	0	0	0	0	$D_3$	0	1	1
0	0	0	0	1	0	0	0	$D_4$	1	0	0
0	0	0	0	0	1	0	0	$D_5$	1	0	1
0	0	0	0	0	0	1	0	$D_6$	1	1	0
0	0	0	0	0	0	0	1	$D_7$	1	1	1

(b) Truth Table

Figure 3.18: Encoder

From the encoder table, it is evident that at any given time only one input is assumed to have 1 value. This is a major limitation of encoder. What will happen when two inputs are together active? The obvious answer is that since the output is not defined the ambiguity exists. To avoid this ambiguity the encoder circuit has input priority so that only one input is encoded. The input with high subscript can be given higher priority. For example, if both  $D_2$  and  $D_6$  are 1 at the same time, then the output will be 110 because  $D_6$  has higher priority than  $D_2$ .

The encoder can be implemented with 3 OR gates whose inputs can be determined from the truth table. The output can be expressed as:

$$O_0 = I_1 + I_3 + I_5 + I_7$$

$$O_1 = I_2 + I_3 + I_6 + I_7$$

$$O_2 = I_4 + I_5 + I_6 + I_7$$

You can draw the K-Maps to determine above functions and draw the related combinational circuit

### 3.6.5 Programmable Logic Array

Till now the individual gates are treated as basic building blocks from which various logic functions can be derived. We have also learned about the strategies of minimization of number of gates. But with the advancement of technology the integration provided by integrated circuit technology has increased resulting into production of one to ten gates on a single chip (in small scale integration). The gate level designs are constructed at the gate level only but if the design is to be done using these SSI chips the design consideration needs to be changed as a number of such SSI chips may be used for developing a logic circuit. With MSI and VLSI we can put even more gates on a chip and can also make gate interconnections on a chip. This integration and connection brings the advantages of decreased cost, size and increased speed. But the basic drawback faced in such VLSI & MSI chip is that for each logic function the layout of gate and interconnection needs to be designed. The cost involved in making such custom designed is quite high. Thus, came the concept of Programmable Logic Array, a general purpose chip which can be readily adopted for any specific purpose.

The PLA are designed for SOP form of Boolean function and consist of regular arrangements of NOT, AND & OR gate on a chip. Each input to the chip is passed through a NOT gate, thus the input and its complement are available to each AND gate. The output of each AND gate is made available for each OR gate and the output of each OR gate is treated as chip output. By making appropriate connections any logic function can be implemented in these Programmable Logic Array.

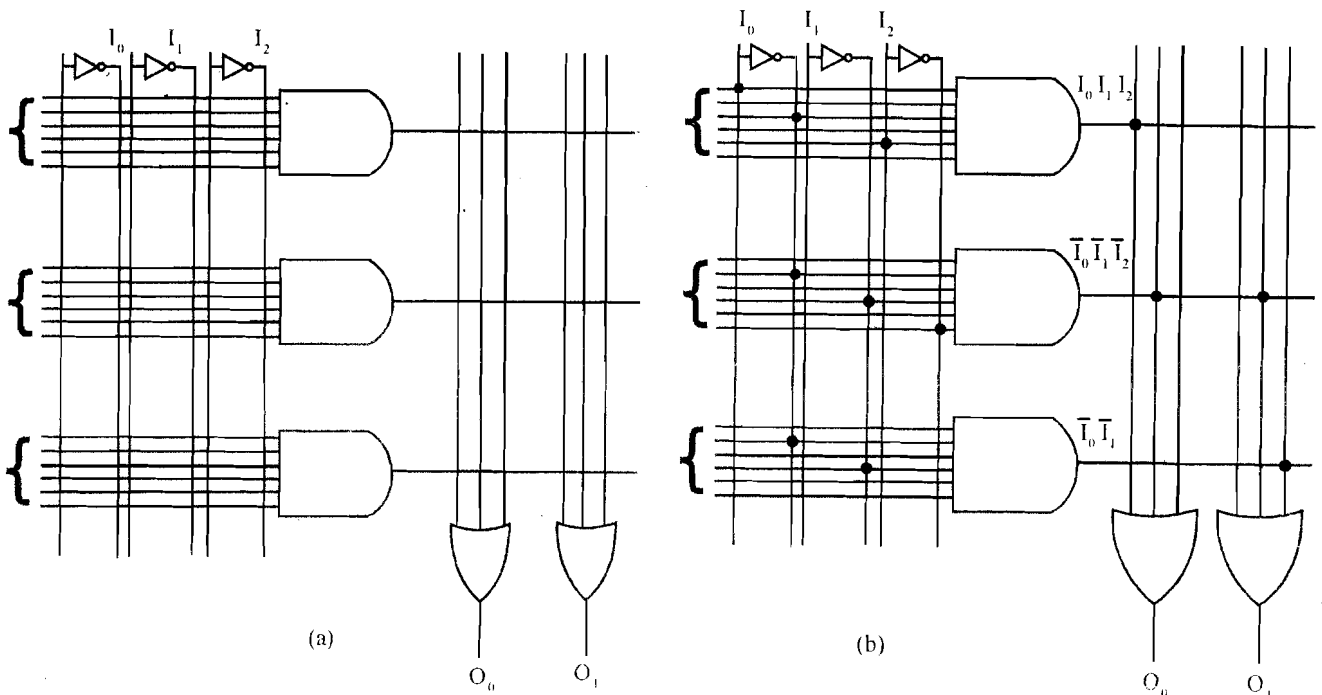


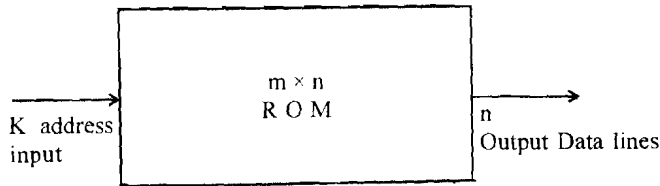
Figure 319: Programmable Logic Array

The figure 3.18(a) shows a PLA of 3 inputs and 2 outputs. Please note the connectivity points, all these points can be connected if desired. Figure 3.18(b) shows an implementation of logic function:

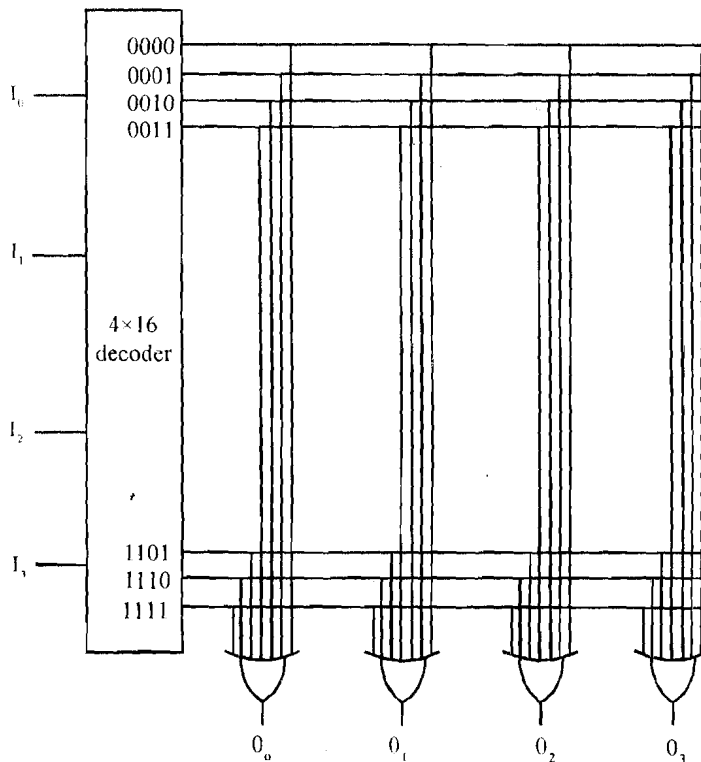
$O_0 = I_0, I_1, I_2 + \bar{I}_0, \bar{I}_1, \bar{I}_2$  and  $O_1 = \bar{I}_0, \bar{I}_1, \bar{I}_2 + I_0, I_1$  through the PLA.

### 3.6.6 Read-only-Memory (ROM)

The read-only-memory is an example of a Programmable Logic Device (PLD) i.e the binary information that is stored within a PLD is specified in some fashion and embedded within the hardware. Thus the information remains even when the power goes.



(a) Block Diagram



b) Logic Diagram of 64-bit ROM

Figure 3.20: ROM Design

Figure 3.19 shows the block diagram of ROM. It consists of 'k' input address lines and 'n' output data lines. An  $m \times n$  ROM is an array of binary cell organised into  $m$  ( $2^k = m$ ) words of 'n' bits each. The ROM does not have any data input because the write operation is not defined for ROM. ROM is classified as a combinational circuit and constructed internally with decoder and a set of OR gates.

In general, a  $m \times n$  ROM (where  $m = 2^k$ ,  $k = \text{no. of address lines}$ ) will have an internal  $k \times 2^k$  decoder and 'n' OR gate. Each OR gates has  $2^k$  inputs which are connected to each of the outputs of the decoder.

### Check Your Progress 3

- 1) Draw a Karnaugh Map for 5 variables.  
.....  
.....  
.....
- 2) Map the function having 4 variables in a K- Map and draw the truth table. The function is  
 $F(A, B, C, D) = (2, 6, 10, 14)$ .  
.....  
.....  
.....
- 3) Find the optimal logic expression for the above function. Draw the resultant logic diagram.  
.....  
.....  
.....
- 4) What are the advantages of PLA?  
.....  
.....  
.....
- 5) Can a full adder be constructed using 2 half adders?  
.....  
.....  
.....

---

## 3.7 SUMMARY

---

This unit provides you the information regarding a basis of a computer system. The key elements for the design of a combinational circuit like adders etc. are discussed in this unit. With the advent of PLA's the designing of circuit is changing and now the scenario is moving towards micro processors. With this developing scenario in the forefront and the expectation of Ultra- Large- Integration (ULSI) in view, time is not far of when design of logic circuits will be confined to single microchip components. You can refer to latest trends of design and development including VHDL (a hardware design language) in the further readings.

---

## 3.8 SOLUTIONS/ANSWERS

---

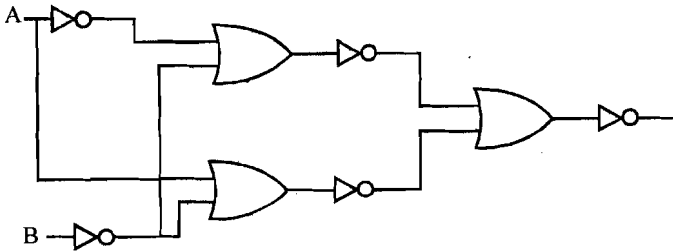
### Check Your Progress 1

1. Logic gates produce typical outputs based on input values NAND and NOR are universal gates as they can be used to construct any other logic gate.

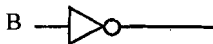
2.

$$\begin{aligned}
 F &= \overline{\left\{ \left( \overline{A+B} \right) + \left( \overline{A+B} \right) \right\}} \\
 &= \overline{\left( \overline{A+B} \right) \cdot \left( \overline{A+B} \right)} \\
 &= \left( \overline{A+B} \right) \cdot \left( A+B \right) \\
 &= \left( \overline{A+B} \right) \cdot A + \left( \overline{A+B} \right) \cdot \overline{B} \\
 &= \overline{A} \cdot A + \overline{A} \cdot \overline{B} + \overline{A} \cdot B + \overline{B} \cdot \overline{B} \\
 &= 0 + \overline{A} \cdot \overline{B} + \overline{A} \cdot B + \overline{B} \\
 &= 0 + \overline{B} (\overline{A} + A) + \overline{B} \\
 &= 0 + \overline{B} + \overline{B} = \overline{B}
 \end{aligned}$$

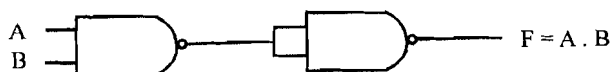
3.



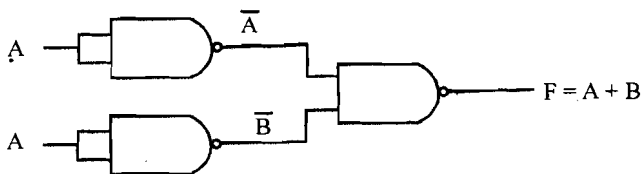
4.



5.



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



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

### Check Your Progress 2

1 (i):

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

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

(ii)

A	B	$F = (A+B) \cdot (\bar{A} + \bar{B})$
0	0	0
0	1	1
1	0	1
1	1	0

2 (i)

$$\begin{aligned}
 F &= ((\bar{A} \cdot B) + B) \\
 &= \bar{A} + B + B \\
 &= \bar{A} + 1 \quad (B + B \text{ is always } 1) \\
 &= 1
 \end{aligned}$$

(ii)

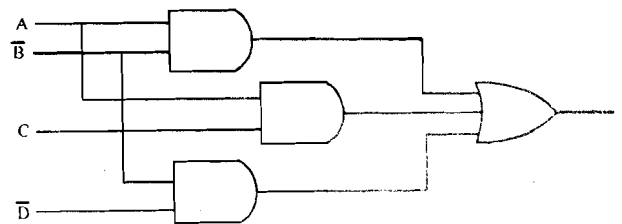
$$\begin{aligned}
 F &= (\bar{A} \cdot B) \cdot (\bar{A} \cdot \bar{B}) \\
 &= (\bar{A} + B) \cdot (\bar{A} \cdot \bar{B}) \\
 &= \bar{A} \bar{A} \bar{B} + \bar{A} \bar{B} \bar{B} \\
 &= \bar{A} \bar{B} + \bar{A} \bar{B} \\
 &= \bar{A} \bar{B}
 \end{aligned}$$

3

SOP Form:

CD	00	01	11	10
AB	1 0	1 3	1 2	
00				
01	4	5	7	6
11	12	13	13 14	
10	8	9	10	11

$$F = A\bar{B} + \bar{B}\bar{D} + AC$$



POS Form:

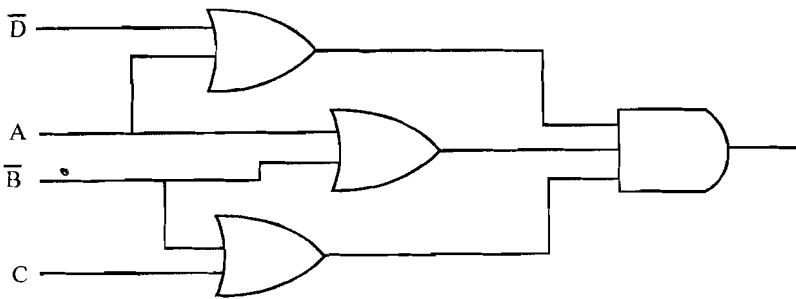
CD	00	01	11	10
AB		0 0		
00				
01	0 0	0 0	0 0	0 0
11	0 0			
10				

$$\bar{F} = \bar{A}B + B\bar{C} + \bar{A}D$$

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

$$F = (\bar{A}B) \cdot (B\bar{C}) \cdot (\bar{A}D)$$

$$F = (A + \bar{B}) \cdot (\bar{B} + C) \cdot (A + \bar{D})$$



### Check Your Progress 3 :

1

		C D E							
A B		000	001	011	010	110	111	101	100
	00	0	1	3	2	6	7	5	4
	01	8	9	11	10	14	15	13	12
	11	24	25	27	26	30	31	29	28
	10	16	17	19	18	22	23	21	20

### 2 K-Map

		C D			
A B		00	01	11	10
	00	0	1	3	2
	01	4	5	7	6
	11	12	13	15	14
	10	8	9	11	10

### Truth table

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0

**Introduction to Digital  
Circuits**

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

3. One adjacency of 4 variables, So  
 $F = C.\bar{D}$
4. PLA's are generic chips that can be used to implement a number of SOP logic function
- 5.

