CHAPTER 5

COMBINATIONAL LOGIC DESIGN

5.1 INTRODUCTION

Logic operations and Boolean algebra have already been discussed in Chapter 1. Boolean algebraic theorems are used for the manipulations of logic expressions. It has also been demonstrated that a logic expression can be realised using the logic gates. The number of gates and the number of input terminals for the gates required for the realisation of a logic expression, in general, get reduced considerably if the expression can be simplified. Therefore, the simplification of logic expression is very important as it saves the hardware required to design a specific system. A large number of functions are available in IC form and therefore, we should be able to make optimum use of these ICs in the design of digital systems. That is, our aim should be to minimise the number of IC packages. Some of the logic gates available in ICs have been discussed in Section 1.7. We will be discussing in Chapter 6, some of the MSI digital circuits available in IC form and design of digital systems using these ICs. Programmable logic devices (PLDs), field-programmable gate arrays (FPGAs) and the design using these devices have been discussed in Chapter 12.

Basically, digital circuits are divided into two broad categories:

1. Combinational circuits, and
2. Sequential circuits.

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

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

1. A set of statements,
2. Boolean expression, and
3. Truth table.

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

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

1. Algebraic method,
2. Karnaugh-map technique,
3. Quine–McCluskey method, and
4. Variable entered mapping (VEM) technique.

The algebraic method, the Karnaugh map (K-map) technique, and the Quine-McCluskey method have been discussed here. The K-map is the simplest and most commonly used method. It is a manual method and depends to a great extent upon human intuition. This method can be used conveniently up to six variables beyond which it is very cumbersome.

The Quine–McCluskey method is suitable for computer mechanisation and is seldom used by logic designers manually. The variable entered mapping (VEM) has been discussed in Fletcher[2] and the interested reader can refer to it.

5.2 STANDARD REPRESENTATIONS FOR LOGIC FUNCTIONS

Logic functions are expressed in terms of logical variables. The values assumed by the logic functions as well as the logic variables are in the binary form. Any arbitrary logic function can be expressed in the following forms:

1. Sum-of-products form (SOP), and
2. Product-of-sums form (POS)

This does not mean that the logic function cannot be written in any other form. It can be written in various forms but the above two forms are conveniently suited in arriving at the standard methods for designing the circuits which will become clear from the following discussions.

Example 5.1

Given the logic equation

$$Y = (A + BC)(B + \overline{C}A) \tag{5.1}$$

(a) Design a circuit using gates to realise this function.

[2] Fletcher, W.L., *An Engineering Approach to Digital Design*, Prentice-Hall, Englewood Cliffs, New Jersey, 1980.

(b) Find out whether it is possible to design the circuit with only one type of gates (NAND or NOR). If yes, design the circuits.
(c) Find out whether it is possible to simplify this equation. If yes, simplify it.
(d) Now design the circuit using the simplified expression obtained in part (c).
(e) Compare the circuits obtained in parts (a), (b), and (d) from the point of view of number of gates, number of inputs for the gates, types of gates, and propagation delay.

Solution

In Eq. (5.1), there are three input logic variables A, B, and C, and Y is the output. The variable C appears as C in one term and as $\overline{C}$ in the other term. A variable in *uncomplemented* or *complemented* form is known as a *literal.*

(a) A circuit using gates can simply be designed by looking at the expression and finding out the basic gates which can be used to realise the various terms and then connect these gates appropriately. We assume that the signals corresponding to each literal are available, that is the variables are available in uncomplemented as well as the complemented form.
  (i) The first term (A) has only one literal A and the second term (BC) has two literals B and C. The second term is recognised as an AND operation and can be realised by using a 2-input AND gate. The combination of these two terms is realised by using a 2-input OR gate. The complete realisation of first two terms is shown in Fig. 5.1*a*.
  (ii) The third term (B) is again a single literal term and the fourth term ($\overline{C}A$) has two literals. The combination of these two terms is similar to the combination of the first two terms and is realised in a similar way. This realisation is shown in Fig. 5.1*b*.
  (iii) Now, the complete realisation is obtained by using a 2-input AND gate with Y_1 and Y_2 as the inputs and the output of this gate will be the required output Y. This realisation is given in Fig. 5.1*c*. The above design requires three 2-input AND gates and two 2-input OR gates.

(b) (i) Sum-of-Products Form
Equation (5.1) can be written as

$$\begin{aligned} Y &= A(B+\overline{C}A) + (BC)(B+\overline{C}A) && \text{(Theorem 1.9)} \\ &= AB + A\overline{C}A + BCB + BC\overline{C}A && \text{(Theorem 1.9)} \quad (5.2) \\ &= AB + A\overline{C} + BC && (5.3) \end{aligned}$$

Equation (5.3) is obtained from Eq. (5.2) in the following way:

$$\begin{aligned} A\overline{C}A &= (A \cdot A) \cdot \overline{C} = A\overline{C} && \text{(Theorem 1.6)} \\ BCB &= (B \cdot B) \cdot C = BC && \text{(Theorem 1.6)} \\ BC\overline{C}A &= B \cdot (C \cdot \overline{C}) \cdot A = B \cdot 0 \cdot A && \text{(Theorem 1.8)} \\ &= 0 && \text{(Theorem 1.4)} \end{aligned}$$

This term does not appear in Eq. (5.3), because of Theorem 1.1.

The representation of Eq. (5.3) is known as *sum-of-products* (SOP) form. This can be realised using AND–OR configuration as shown in Fig. 5.2*a*. This realisation is known as *two-level realisation*. The first level consists of AND gates and the second level consists of the OR gate. By making use of the De-Morgan's theorem (Theorem 1.22) we can write Eq. (5.3) as

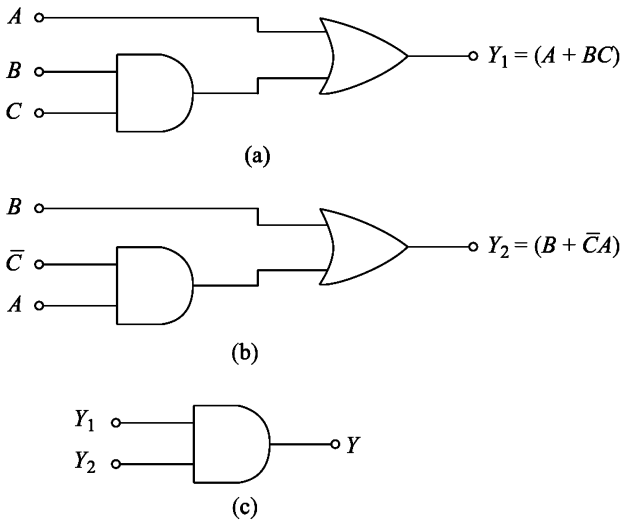


Fig. 5.1 ***Logic Circuits for the Realisation of Eq. (5.1)***

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

Or
$$Y = \overline{Y_1 \cdot Y_2 \cdot Y_3} \tag{5.4}$$

Where
$$Y_1 = \overline{AB}$$
$$Y_2 = \overline{A\overline{C}}$$
$$Y_3 = \overline{BC}$$

Equation (5.4) can be realised using NAND gates only. The realisation is given in Fig. 5.2*b*. This is also a two-level realisation and in this only NAND gates are used. Therefore, if we express the equation in the SOP form we can always design the circuit using only one type of gates (NAND).

(ii) Product-of-Sums Form

It is possible to represent Eq. (5.1) in another form.

$$Y = (A+B)(A+C)(B+\overline{C})(B+A) \quad \text{(Theorem 1.10)}$$
$$= (A+B)(A+C)(B+\overline{C}) \quad \text{(Theorem 1.6)} \tag{5.5}$$

The representation of Eq. (5.5) is known as *Product-of-sums* (POS) form. This can be realised using OR–AND gates as shown in Fig. 5.3*a*. This is also a two-level realisation

Using De-Morgan's theorem (Theorem 1.21), we can write Eq. (5.5) as

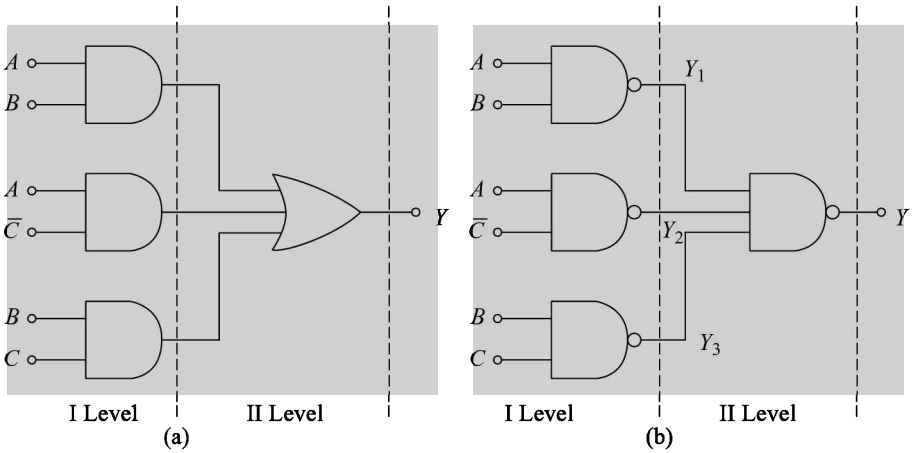


Fig. 5.2 ***Realisation of Eq. (5.3) Using (a)*** AND–OR ***(b)*** NAND–NAND

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

Or

$$Y = \overline{Y_A + Y_B + Y_C} \tag{5.6}$$

Where

$$Y_A = \overline{A+B}$$
$$Y_B = \overline{A+C}$$
$$Y_C = \overline{B+\overline{C}}$$

Equation (5.6) can be realised using NOR gates only. This realisation is given in Fig. 5.3*b* which is a two-level realisation. Hence, if we express the equation in POS form we can always design the circuit using only one type of gates (NOR).

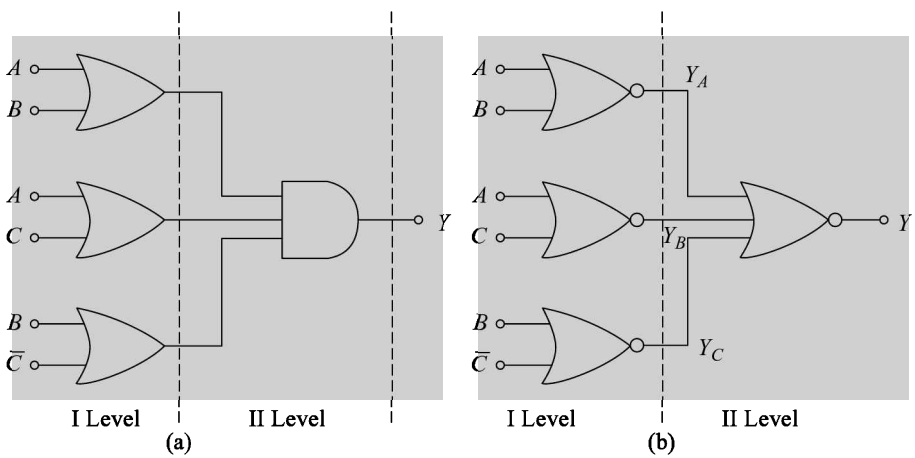


Fig. 5.3 ***Realisation of Eq. (5.5) Using (a)*** OR–AND ***(b)*** NOR–NOR

(c) (i) Simplification of Eq. (5.3),

$$\begin{aligned} Y &= AB + A\bar{C} + BC \\ &= BC + A\bar{C} \qquad \text{(Theorem 1.19)} \end{aligned} \tag{5.7}$$

(ii) Simplification of Eq. (5.5),

$$\begin{aligned} Y &= (A + B)(A + C)(B + \bar{C}) \\ &= (A + C)(B + \bar{C}) \qquad \text{(Theorem 1.20)} \end{aligned} \tag{5.8}$$

(d) Realisation of Eqs (5.7) and (5.8) are given in Figs 5.4*a* and 5.4*b*, respectively.

(e) The gate requirements corresponding to parts (a), (b), and (d) are given in Table 5.1. The realisation of part (a) needs maximum number of gates. Also, it is a three-level realisation which increases the propagation delay time which is same as decrease in speed of operation. Realisations corresponding to parts (b) and (d) are very useful since only one type of gates (NAND/NOR) are required which is very convenient to use when we use ICs because a number of similar gates are available in the same package. Realisation corresponding to part (d) requires minimum number of gates. Therefore, the simplification of logic expressions is very useful

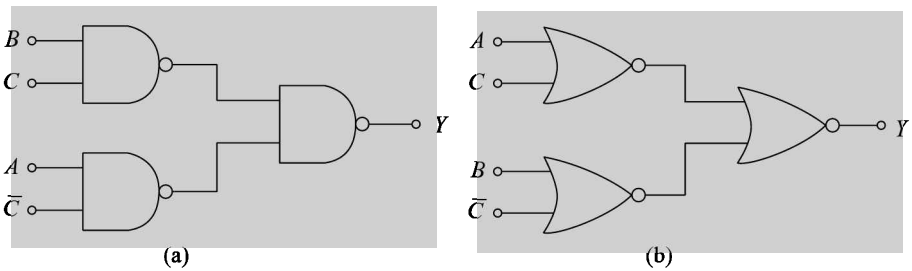


Fig. 5.4 ***Realisation of (a) Equation (5.7) (b) Equation (5.8)***

Table 5.1 ***Gate Requirements***

Part (a)	Part (b)		Part (d)
	AND–OR/NAND–NAND	OR–AND/NOR–NOR	
3, 2-input AND	3,2-input AND	3, 2-input OR	3, 2-input NAND or 3, 2-input NOR
2, 2-input OR	1, 3-input OR or 3, 2-input NAND 1, 3-input NAND	1, 3-input AND or 3, 2-input NOR 1, 3-input NOR	

We notice that in the SOP form of Eq. (5.3) and the POS form of Eq. (5.5), all the individual terms do not involve all the three literals. If each term in SOP and POS forms contains all the literals then these are known as *canonical* SOP and POS, respectively. Each individual term in canonical SOP form is called as *minterm* and in *canonical* POS form as *maxterm*. The usefulness of the minterm and maxterm representations will become clear from the discussion which follows. SOP form can be converted to canonical SOP by ANDing the terms in the expression with terms formed by ORing the variable and its complement which are not present in that term. For example for a three-variable expression with variables A, B, and C, if there is a term A, where B and C variables are missing, then we form two terms $(B + \overline{B})$ and $(C + \overline{C})$ and AND them with A. Therefore, we get $A \cdot (B + \overline{B}) \cdot (C + \overline{C}) = ABC + A\overline{B}C + AB\overline{C} + A\overline{B}\,\overline{C}$.

Example 5.2

Convert Eq. (5.3) into canonical SOP form.

Solution

$$\begin{aligned} Y &= AB(C + \overline{C}) + A\overline{C}(B + \overline{B}) + BC(A + \overline{A}) \\ &= ABC + AB\overline{C} + AB\overline{C} + A\overline{B}\,\overline{C} + ABC + \overline{A}BC \\ &= ABC + AB\overline{C} + A\overline{B}\,\overline{C} + \overline{A}BC \qquad \text{(Theorem 1.5)} \end{aligned} \tag{5.9}$$

Similarly, POS form can be converted to canonical POS by ORing the terms in the expression with terms formed by ANDing the variable and its complement which are not present in that term. For example for a three-variable expression with variables A, B, and C, if there is a term A where B and C variables are missing, then we form two terms $B \times \overline{B}$ and $C \times \overline{C}$ and OR A with these terms. Therefore, we get

$$\begin{aligned} A + B\overline{B} + C\overline{C} &= (A + B\overline{B} + C)(A + B\overline{B} + \overline{C}) \qquad \text{(Theorem 1.10)} \\ &= (A + B + C)(A + \overline{B} + C)(A + B + \overline{C})(A + \overline{B} + \overline{C}) \end{aligned} \tag{5.10}$$

Example 5.3

Convert Eq. (5.5) into canonical POS form.

Solution

$$\begin{aligned} Y &= (A + B + C\overline{C})(A + B\overline{B} + C)(A\overline{A} + B + \overline{C}) \\ &= (A + B + C)(A + B + \overline{C})(A + B + C)(A + \overline{B} + C) \\ &\quad (A + B + \overline{C})(\overline{A} + B + \overline{C}) \qquad \text{(Theorem 1.10)} \\ &= (A + B + C) \cdot (A + B + \overline{C}) \cdot (A + \overline{B} + C) \cdot (\overline{A} + B + \overline{C}) \qquad \text{(Theorem 1.6)} \end{aligned} \tag{5.11}$$

The concept of minterm and maxterm introduced above allows us to introduce a very convenient shorthand notation to express logical functions. Table 5.2 gives the minterms and maxterms for a four variable logical function where the number of minterms as well as maxterms is $2^4 = 16$. In general, for an n-variable logical function there are 2^n minterms and an equal number of maxterms. While writing a particular minterm or maxterm, we shall always write it with the variables in an orderly way as can be seen from Table 5.2. Each minterm is represented by m_i where the subscript i is the decimal equivalent of the natural binary number corresponding to the minterm with normal (uncomplemented) variables taken as 1's and the complemented variables taken as 0's.

Table 5.2 ***Minterms/Maxterms for Four Variables***

Variable				Miniterm	Maxterm
A	B	C	D	m_i	M_i
0	0	0	0	$\bar{A}\bar{B}\bar{C}\bar{D} = m_0$	$A + B + C + D = M_0$
0	0	0	1	$\bar{A}\bar{B}\bar{C}D = m_1$	$A + B + C + \bar{D} = M_1$
0	0	1	0	$\bar{A}\bar{B}C\bar{D} = m_2$	$A + B + \bar{C} + D = M_2$
0	0	1	1	$\bar{A}\bar{B}CD = m_3$	$A + B + \bar{C} + \bar{D} = M_3$
0	1	0	0	$\bar{A}B\bar{C}\bar{D} = m_4$	$A + \bar{B} + C + D = M_4$
0	1	0	1	$\bar{A}B\bar{C}D = m_5$	$A + \bar{B} + C + \bar{D} = M_5$
0	1	1	0	$\bar{A}BC\bar{D} = m_6$	$A + \bar{B} + \bar{C} + D = M_6$
0	1	1	1	$\bar{A}BCD = m_7$	$A + \bar{B} + \bar{C} + \bar{D} = M_7$
1	0	0	0	$A\bar{B}\bar{C}\bar{D} = m_8$	$\bar{A} + B + C + D = M_8$
1	0	0	1	$A\bar{B}\bar{C}D = m_9$	$\bar{A} + B + C + \bar{D} = M_9$
1	0	1	0	$A\bar{B}C\bar{D} = m_{10}$	$\bar{A} + B + \bar{C} + D = M_{10}$
1	0	1	1	$A\bar{B}CD = m_{11}$	$\bar{A} + B + \bar{C} + \bar{D} = M_{11}$
1	1	0	0	$AB\bar{C}\bar{D} = m_{12}$	$\bar{A} + \bar{B} + C + D = M_{12}$
1	1	0	1	$AB\bar{C}D = m_{13}$	$\bar{A} + \bar{B} + C + \bar{D} = M_{13}$
1	1	1	0	$ABC\bar{D} = m_{14}$	$\bar{A} + \bar{B} + \bar{C} + D = M_{14}$
1	1	1	1	$ABCD = m_{15}$	$\bar{A} + \bar{B} + \bar{C} + \bar{D} = M_{15}$

Similar to minterms, each maxterm is represented by M_i, where the subscript i is the decimal equivalent of the natural binary number corresponding to the maxterm with uncomplemented variables taken as 0's and complemented variables taken as 1's. Using these notations the canonical SOP (Eq. (5.9)) can be written as:

$$\begin{aligned} Y &= m_3 + m_4 + m_6 + m_7 \\ &= \sum m(3, 4, 6, 7) \end{aligned} \tag{5.12}$$

Where $\bar{A}BC = m_3$

$A\bar{B}\bar{C} = m_4$

$AB\bar{C} = m_6$

and $ABC = m_7$

Similarly, the canonical POS (Eq. (5.11)) can be written as

$$Y = M_0 \cdot M_1 \cdot M_2 \cdot M_5$$
$$= \Pi M(0, 1, 2, 5) \tag{5.13}$$

Where $A + B + C = M_0$
$A + B + \bar{C} = M_1$
$A + \bar{B} + C = M_2$
and $\bar{A} + B + \bar{C} = M_5$

Equations (5.12) and (5.13) are the shorthand forms of canonical SOP and POS respectively. Since these two equations represent the same logical function (Eq. (5.1)), therefore we notice that there is a complementary type of relationship between a function expressed in terms of minterms and in terms of maxterms. Here, we are dealing with a three-variable function where the number of minterms and maxterms are $2^3 = 8$ and the corresponding decimal numbers are 0 through 7. Out of this the terms corresponding to decimal numbers 3, 4, 6, and 7 are minterms, and the terms corresponding to decimal numbers 0, 1, 2, and 5 are maxterms. Hence, if a logic function is specified in terms of minterm/maxterm, its maxterm/minterm representation can be determined by using this complementary property. For example, for a four-variable case if

$$Y = \Sigma m(0, 3, 6, 7, 10, 12, 15)$$

then $Y = \Pi M(1, 2, 4, 5, 8, 9, 11, 13, 14)$.

5.3 KARNAUGH MAP REPRESENTATION OF LOGIC FUNCTIONS

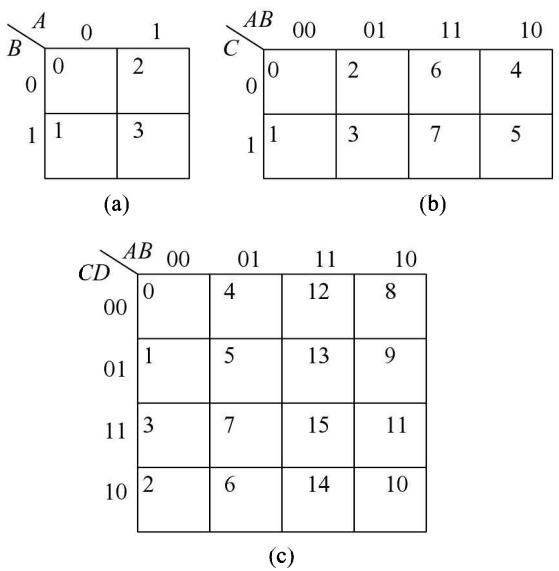


Fig. 5.5 ***Karnaugh-maps (a) Two-variable (b) Three-variable (c) Four-variable***

We have discussed the two standard forms of logic functions and their realisations using gates. We have also established the need for the simplification of the Boolean expressions and introduced the algebraic method of simplification using Boolean algebraic theorems. Sometimes it is difficult to be sure that a logic expression can be simplified. There is another technique, which is graphical, known as the Karnaugh map technique which provides a systematic method for simplifying and manipulating Boolean expressions. In this technique, the information contained in a truth table or available in POS or SOP form is represented on Karnaugh map (K-map). This is perhaps the most extensively used tool for simplification of Boolean functions. Although the technique may be used for any number of variables, it is generally used up to six variables beyond which it becomes very cumbersome.

Figure 5.5 shows the K-maps for two, three, and four variables. In an n-variable K-map there are 2^n cells. Each cell corresponds to one of the combinations of n variables, since there are 2^n combinations of n variables. Therefore, we see that for each

row of the truth table, for each minterm and for each maxterm there is one specific cell in the K-map. The variables have been designated as A, B, C, and D, and the binary numbers formed by them are taken as AB, ABC, and $ABCD$ for two, three, and four variables respectively. In each map the variables and all possible values of the variables are indicated (the first bit corresponds to the first variable and the second bit corresponds to the second variable) to identify the cells. Gray code has been used for the identification of cells. The reason for using Gray code will become clear when we discuss the application of K-map. You can verify the decimal number corresponding to each cell which is written in the top left corner of the cell as shown in Fig. 5.5.

Figure 5.6 shows the minterm/maxterm corresponding to each cell and the term is written inside the cell for clear understanding.

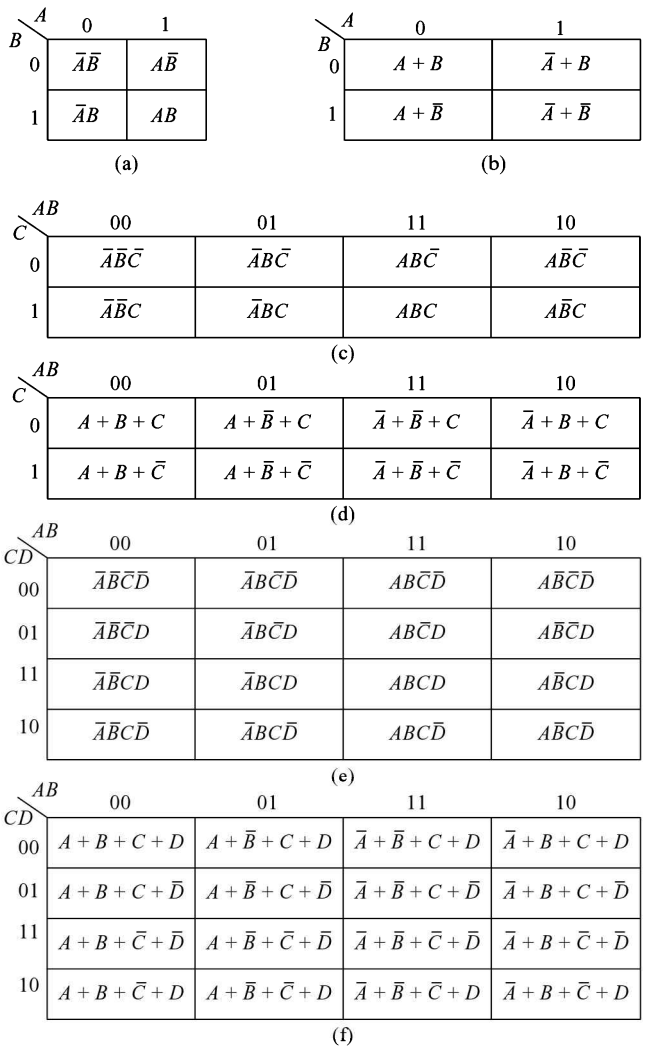


Fig. 5.6 ***Maxterm/Minterm Corresponding to Each Cell of K-maps***

5.3.1 Representation of Truth Table on K-Map

Consider the truth table of the 3-variable logic function given in Table 5.3. The output Y is logic 1 corresponding to the rows 1, 2, 4, and 7. Corresponding to this we can write the equation in terms of canonical SOP as given below:

$$Y = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC \tag{5.14}$$

Equation (5.14) represents the complete truth table in canonical SOP form. Similarly, we note that the output Y is logic 0 corresponding to the rows 0, 3, 5, and 6 and the output Y can be represented in terms of canonical POS form as given below:

$$Y = (A + B + C)(A + \bar{B} + \bar{C})(\bar{A} + B + \bar{C})(\bar{A} + \bar{B} + C) \tag{5.15}$$

Equation (5.15) also represents the complete truth table, and Eqs (5.14) and (5.15) are equivalent. We shall make use of the 3-variable K-map of Fig. 5.5*b* and enter the value of the output variable Y (0 or 1) in each cell corresponding to its decimal or minterm or maxterm identification. Figure 5.7 gives the complete K-map of the truth table given in Table 5.3.

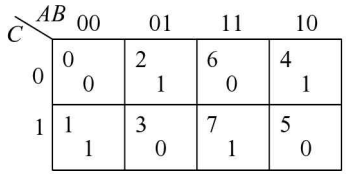


Fig. 5.7 ***K-map for Table 5.3***

The procedure used above is general and is used to represent a truth table on the K-map. On the other hand, if a K-map is given we can make the truth table corresponding to this by following the reverse process. That is, the output Y is logic 1 corresponding to the decimal numbers/minterms represented by cells with entries 1. In all other rows, the output Y is logic 0.

Table 5.3 ***Truth Table of a 3-variable Function***

Row No.	Inputs			Output
	A	*B*	*C*	*Y*
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1

Example 5.4

Prepare the truth table for K-map of Fig. 5.8.

Solution

The truth table is given in Table 5.4.

Note that the 0's are not written in the K-map of Fig. 5.8. Actually, we need enter either 0's or 1's only in the K-map. If only 1's are entered the empty cells are 0's and if only 0's are entered then the empty cells are 1's.

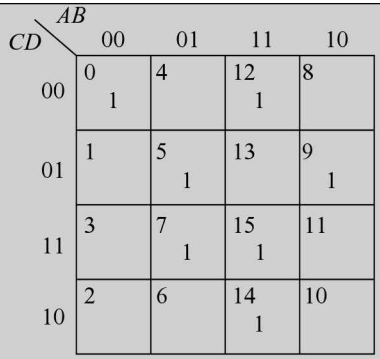


Fig. 5.8 ***K-map of Ex. 5.4***

Table 5.4 ***Truth Table for K-map of Fig. 5.8***

Row No.	Inputs				Output
	A	*B*	*C*	*D*	*Y*
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	1

5.3.2 Representation of Canonical SOP Form on K-Map

A logical equation in canonical SOP form can be represented on a K-map by simply entering 1's in the cells of the K-map corresponding to each minterm present in the equation.

Example 5.5

Represent Eq. (5.14) on K-map.

Solution

Corresponding to each minterm in the equation, there is a cell in the K-map and a 1 is entered in each one of these cells. The K-map will be as shown in Fig. 5.7.

Similarly, from the K-map, we can write the corresponding logic equation in canonical SOP form by ORing the minterms corresponding to each 1 entry in the K-map.

Example 5.6

Write the logic equation in the canonical SOP form for the K-map of Fig. 5.8.

Solution

$$\begin{aligned} Y &= \bar{A}\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C\bar{D} + A\bar{B}C\bar{D} + \bar{A}BCD + ABCD + ABC\bar{D} \\ &= \Sigma m(0, 5, 7, 9, 12, 14, 15) \end{aligned} \tag{5.16}$$

If the equation is in SOP form, it can be converted to canonical SOP form by using the method discussed earlier and then it can be represented on K-map. Another method of representing SOP form on K-map without converting it to canonical SOP form will become clear from the discussions of Sections 5.4 and 5.6.

5.3.3 Representation of Canonical POS Form on K-Map

Logic equation in canonical POS form can be represented on K-map by entering 0's in the cells of K-map corresponding to each maxterm present in the equation.

Example 5.7

Represent Eq. (5.15) on K-map.

Solution

Corresponding to each maxterm in the equation, there is a cell in the K-map and a 0 is entered in each one of these cells. The K-map will be as shown in Fig. 5.7.

From a given K-map, we can write the logic equation in the canonical POS form by ANDing the maxterms corresponding to each 0 entry in the K-map.

Example 5.8

Write the logic equation in the canonical POS form for the K-map of Fig. 5.8.

Solution

$$\begin{aligned} Y = {} & (A+B+C+\bar{D})(A+B+\bar{C}+D)\,(A+B+\bar{C}+\bar{D}) \\ & (A+\bar{B}+C+D)(A+\bar{B}+\bar{C}+D)\,(\bar{A}+B+C+D) \\ & (\bar{A}+B+\bar{C}+D)(\bar{A}+B+\bar{C}+\bar{D})(\bar{A}+\bar{B}+C+\bar{D}) \\ = {} & \Pi M(1, 2, 3, 4, 6, 8, 10, 11, 13) \end{aligned} \tag{5.17}$$

If the equation is in POS form, it can be converted into canonical POS form by the method discussed earlier and then it can be represented on K-map. Another method of representing POS form on K-map without converting it to canonical POS form will become clear from the discussion of Sections 5.4 and 5.6.

Equation (5.16) represents the K-map of Fig. 5.8 in canonical SOP form and Eq. (5.17) represents the same K-map in standard POS form. Therefore, these two equations are equivalent. Alternatively stated, an equation in SOP form can be converted into an equivalent POS form and vice-versa.

5.4 SIMPLIFICATION OF LOGIC FUNCTIONS USING K-MAP

Simplification of logic functions with K-map is based on the principle of combining terms in adjacent cells. Two cells are said to be adjacent if they differ in only one variable. For example, in the two-variable K-maps of Figs 5.6*a* and *b*, the top two cells are adjacent and the bottom two cells are adjacent. Also, the left two cells and the right two cells are adjacent. It can be verified that in adjacent cells one of the literals is same, whereas the other literal appears in uncomplemented form in one and in the complemented form in the other cell.

Similarly, we observe adjacent cells in the 3-variable and 4-variable K-maps. Table 5.5 gives the adjacent cells of each cell in 2-, 3-, and 4-variable K-maps. From this it becomes clear that if the Gray code is used for the identification of cells in K-map, physically adjacent (horizontal and vertical but not diagonal) cells differ in only one variable. Also, the left-most cells are adjacent to their corresponding right-most cells and similarly the top cells are adjacent to their corresponding bottom cells. The simplification of logical function is achieved by grouping adjacent 1's or 0's in groups of 2^i, where i = 1, 2,…n and n is the number of variables.

The process of simplification involves grouping of minterms and identifying *prime-implicants* (PI) and *essential prime-implicants* (EPI).

A *prime-implicant* is a group of minterms that cannot be combined with any other minterm or groups. An *essential prime-implicant* is a *prime-implicant* in which one or more minterms are unique; i.e. it contains at least one minterm which is not contained in any other prime-implicant.

5.4.1 Grouping Two Adjacent Ones

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

Table 5.5 ***Adjacent Cells in K-maps***

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

Example 5.9

Simplify the K-map of Fig. 5.9.

Solution

The canonical SOP form of equation can be written by inspection as

$$Y = \bar{A}\bar{B}\bar{C} + \bar{A}BC + ABC + A\bar{B}\bar{C} \tag{5.18}$$

If we combine the ones in adjacent cells (0, 4) and (3, 7), Eq. (5.18) can be written as

$$Y = (\bar{A} + A)\bar{B}\bar{C} + (\bar{A} + A)BC \tag{5.19}$$

$$= \bar{B}\bar{C} + BC \text{ (Theorems 1.7 and 1.2)} \tag{5.20}$$

Equation (5.20) can be directly obtained from the K-map by using the following procedure:

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

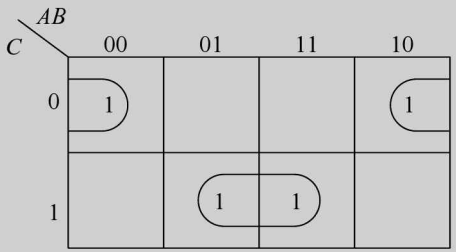


Fig. 5.9 ***K-map of Ex. 5.9***

Here, $\overline{B}\overline{C}$ and BC are the two prime-implicants, since grouping of the two minterms m_0 and m_4 cannot be combined with anyone of the remaining minterms m_3 and m_7, or the combination of the minterms m_3 and m_7. Similarly, the grouping of m_3 and m_7 is a prime-implicant. We also observe that both the groups are also essential prime-implicants, since each one of them contains unique minterms which are not contained in the other group.

5.4.2 Grouping Four Adjacent Ones

Four cells form a group of four adjacent ones if two of the literals associated with the minterms/maxterms are not same and the other literals are same. Table 5.6 gives all possible groups of four adjacent ones for each cell in a 3-variable map. In case of 2-variable map, there is only one possibility corresponding to entry 1 in all the four cells, and the simplified expression will be $Y = 1$. That is, Y always equals 1 (independent of the variables).

On the basis of groupings of 4 adjacent ones given in Table 5.6, we can find the groupings in K-maps of four or more variables. In the case of a four-variable K-map, there are six possible groupings of 4-variables involving any cell. It is left to the reader to verify this fact.

Table 5.6 ***Groups of Four Adjacent Ones in a 3-variable K-map***

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

Example 5.10

Simplify the K-map of Fig. 5.10.

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

Fig. 5.10 ***K-map of Ex. 5.10***

Solution

The canonical SOP form of equation can be written by inspection as

$$\begin{aligned} Y &= m_0 + m_1 + m_3 + m_7 + m_8 + m_9 + m_{11} + m_{15} \\ &= (m_0 + m_1 + m_8 + m_9) + (m_3 + m_7 + m_{15} + m_{11}) \end{aligned} \tag{5.21}$$

In the K-map of Fig. 5.10, there are two groups of four adjacent ones. One corresponding to cells 0, 1, 8, and 9, and the other one corresponding to 3, 7, 15, and 11. In Eq. (5.21), the minterms corresponding to each group are

combined. The first term can be written as

$$\begin{aligned} m_0 + m_1 + m_8 + m_9 &= \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + A\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}D \\ &= \bar{B}\bar{C}(\bar{A}\bar{D} + \bar{A}D + A\bar{D} + AD) \\ &= \bar{B}\bar{C}\left[\bar{A}(\bar{D} + D) + A(\bar{D} + D)\right] \\ &= \bar{B}\bar{C}\left[\bar{A} \cdot 1 + A \cdot 1\right] \\ &= \bar{B}\bar{C}(\bar{A} + A) \\ &= \bar{B}\bar{C} \cdot 1 = \bar{B}\bar{C} \end{aligned}$$

In the first term of Eq. (5.21) we observe the following:

1. In this group of four minterms, two of the variables appear as $\bar{B}$ and $\bar{C}$ in all the four terms.
2. The variable A appears as A in two and as $\bar{A}$ in the other two minterms.
3. The variable D appears as D in two and as $\bar{D}$ in the other two minterms.
4. The combination of these four minterms results in one term with two literals which are present in all the four terms. Similarly, the second term of Eq. (5.21) is simplified to CD. Therefore, the K-map is simplified to

$$Y = \bar{B}\bar{C} + CD \tag{5.22}$$

Example 5.11

In Fig. 5.10, show that the following groups of minterms are not prime-implicants:

(a) m_0, m_1 (d) m_7, m_{15} (g) m_9, m_{11} (i) m_1, m_9
(b) m_1, m_3 (e) m_{11}, m_{15} (h) m_0, m_8 (j) m_3, m_{11}
(c) m_3, m_7 (f) m_8, m_9

Solution

(a) m_0, m_1 group can be combined with m_8, m_9 group to form a group of four adjacent 1s. Hence, m_0, m_1 group is not a prime-implicant.
(b) m_1, m_3 group can be combined with m_9, m_{11} group, therefore, it is not a prime-implicant.
(c) m_3, m_7 group can be combined with m_{15}, m_{11} group.
(d) m_7, m_{15} group can be combined with m_3, m_{11} group.
(e) m_{11}, m_{15} group can be combined with m_3, m_7 group.
(f) See (a)
(g) See (b)
(h) m_0, m_8 group can be combined with m_1, m_9 group
(i) See (h)
(j) See (d)

Example 5.12

Show that the following groups of minterms in Fig. 5.10 are essential prime-implicants:

(a) m (0, 1, 8, 9) (b) m (3, 7, 11, 15)

Solution

(a) The group formed by the minterms m_0, m_1, m_8, m_9 is a prime-implicant, since it can not be combined with any other minterm or group. Also, it includes minterms which can not be included in the other group, therefore, it is an essential prime-implicant.

(b) The group formed by the minterms m_3, m_7, m_{11}, m_{15} have m_7 and m_{15} which can not be included in any other prime-implicant, hence this is an essential prime-implicant.

5.4.3 Grouping Eight Adjacent Ones

Eight cells form a group of eight adjacent ones if three of the literals associated with the minterms/maxterms are not same and the other literals are same. In case of 3-variable K-map, there is only one possibility of eight ones appearing in the K-map and this corresponds to output equal to 1, irrespective of the values of the input variables. Table 5.7 gives all possible groups of eight adjacent ones in a 4-variable K-map. From an understanding of this, we can easily find out such combinations for 5- and 6-variable K-maps. When eight adjacent ones are combined, the resulting equation will have only one term with the number of literals three less than the number of literals in the original minterms. Similar to the groupings of adjacent two and four ones, the literals which are common in all the eight minterms will be present and the literals which are not same get eliminated in the resulting term.

Table 5.7 ***Groups of Eight Adjacent Ones in 4-variable K-map***

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

The reader is advised to verify the simplification of eight adjacent ones into a single term with three variables eliminated. For example, let us take the first group of eight adjacent ones in Table 5.7. For all these eight cells, the variable C appears as $\overline{C}$ in the minterms and the other three variables are not same. Therefore, the grouping of these eight cells results in a term $\overline{C}$. Figure 5.11 shows the simplified expression for each of the groupings of eight ones for a 4 variable K-map.

5.4.4 Grouping 2, 4, and 8 Adjacent Zeros

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

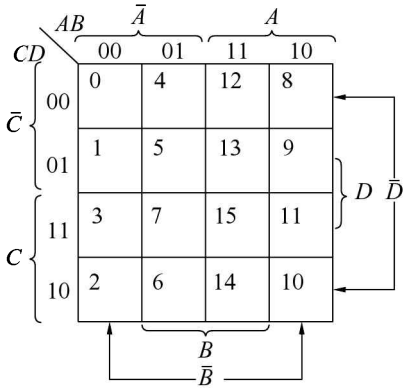


Fig. 5.11 ***Four-variable K-map Illustrating the Groupings of Eight Adjacent Ones***

1. Group of two adjacent zeros result in a term with one literal less than the number of variables. The literal which is not same in the two maxterms gets eliminated.
2. Group of four adjacent zeros result in a term with two literals less than the number of variables. The two literals which are not same in all the four maxterms get eliminated.
3. Group of eight adjacent zeros result in a term with three literals less than the number of variables. The three literals which are not same in all the eight maxterms get eliminated.

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

5.5 MINIMISATION OF LOGIC FUNCTIONS SPECIFIED IN MINTERMS/MAXTERMS OR TRUTH TABLE

5.5.1 Minimisation of SOP Form

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

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

1. Identify the ones which can not be combined with any other ones and encircle them. These are *essential prime-implicants.*
2. Identify the ones that can be combined in groups of two in only one way. Encircle such groups of ones.
3. Identify the ones that can be combined with three other ones, to make a group of four adjacent ones, in only one way. Encircle such groups of ones.
4. Identify the ones that can be combined with seven other ones, to make a group of eight adjacent ones, in only one way. Encircle such groups of ones.
5. After identifying the essential groups of 2, 4, and 8 ones, if there still remains some ones which have not been encircled then these are to be combined with each other or with other already encircled ones. Of course, however, we should combine the left-over ones in largest possible groups and in as few groupings as possible. In this, the groupings may not be unique and we should make the groupings in an optimum manner. You can verify that any one can be included any number of times without affecting the expression.

The logic function consisting of the essential prime-implicants obtained in steps 1 to 4 and the prime-implicants obtained in step 5 will be the minimised function. The above algorithm will be used to minimise the logic functions in the examples given.

Example 5.13

Minimise the four-variable logic function using K-map.

$$f(A, B, C, D) = \Sigma m(0, 1, 2, 3, 5, 7, 8, 9, 11, 14) \tag{5.23}$$

Solution

The K-map of Eq. (5.23) is shown in Fig. 5.12. The equation is minimised in the following steps:

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

Since all the ones have been encircled, therefore, the minimised equation is

$$f(A,B,C,D) = ABC\overline{D} + \overline{B}\overline{C} + \overline{B}D + \overline{A}D + \overline{A}\overline{B} \tag{5.24}$$

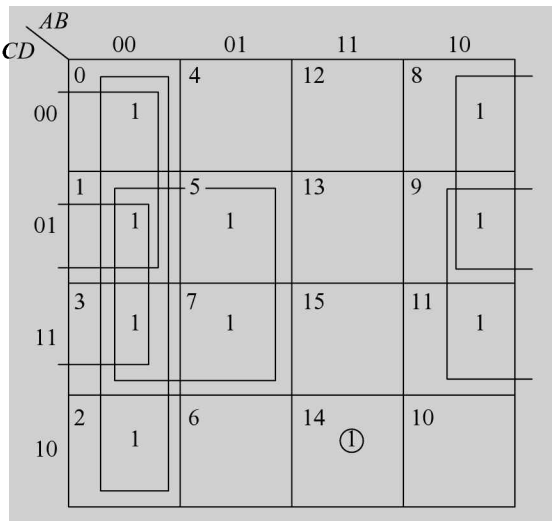


Fig. 5.12 ***K-map for Eq. (5.23)***

Example 5.14

Determine the minimised expression in SOP form for the truth table given in Table 5.8.

Solution

The K-map for the truth table of Table 5.8 is shown in Fig. 5.13. Using the minimisation steps, we obtain the minimised expression.

$$Y = \overline{B} + A\overline{C} + \overline{A}CD \tag{5.25}$$

Table 5.8

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

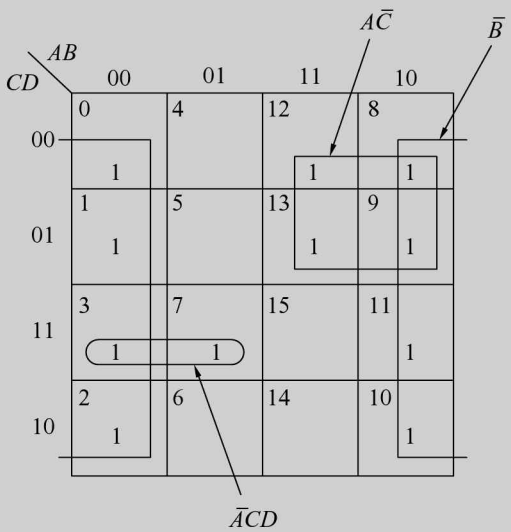


Fig. 5.13 ***K-map of Table 5.8***

5.5.2 Minimisation of POS Form

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

Example 5.15

Minimise the logic function of Eq. (5.23) in POS form.

Solution

Equation (5.23) can be expressed in canonical POS form as

$$f(A, B, C, D) = \Pi M(4, 6, 10, 12, 13, 15) \tag{5.26}$$

The K-map corresponding to Eq. (5.26) is shown in Fig. 5.14. Note that the K-map can also be obtained directly from Eq. (5.23).

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

$$f = (\bar{A} + B + \bar{C} + D) \cdot (\bar{A} + \bar{B} + C) \cdot (\bar{A} + \bar{B} + \bar{D}) \cdot (A + \bar{B} + D) \tag{5.27}$$

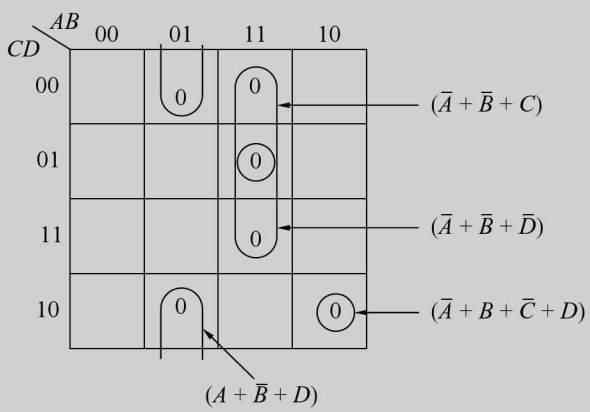


Fig. 5.14 ***K-map of Eq. (5.26)***

If we compare Eqs (5.24) and (5.27), we observe that the number of terms are not same in the two minimisations. In fact, in general the two minimisations will not have the same number of terms and will require different quantities of hardware. Therefore, one can obtain both minimisations and select the one which requires minimum hardware. In some situations there may not be any choice to the designer because of non-availability of certain ICs.

Example 5.16

Minimise the truth table given in Table 5.8 using maxterms.

Solution

The K-map is given in Fig. 5.15.
The simplified expression is

$$Y = (A + \bar{B} + C)(\bar{A} + \bar{B} + \bar{C})(\bar{B} + \bar{C} + D) \tag{5.28}$$

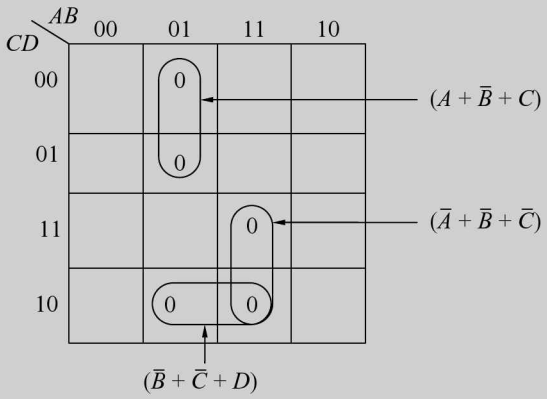


Fig. 5.15 ***K-map of Table 5.8***

Comparison of Eqs (5.25) and (5.28) confirms our generalizations made in Ex. 5.15 regarding the hardware requirements in the two methods.

5.6 MINIMISATION OF LOGIC FUNCTIONS NOT SPECIFIED IN MINTERMS/MAXTERMS

If the function is specified in one of the two canonical forms, its K-map can be prepared and the function can be minimised. Now we consider the cases where the functions are not specified in canonical forms. In such cases, the equations can be converted into canonical forms using the techniques given in Section 5.2, the K-maps obtained and minimised. Alternately, we can directly prepare K-map using the following algorithm:

1. Enter ones for minterms and zeros for maxterms.
2. Enter a pair of ones/zeros for each of the terms with one variable less than the total number of variables.
3. Enter four adjacent ones/zeros for terms with two variables less than the total number of variables.
4. Repeat for other terms in the similar way.

Once the K-map is prepared the minimisation procedure is same as discussed earlier. The following examples will help in understanding the above procedure:

Example 5.17

Minimise the four variable logic function

$$f(A, B, C, D) = AB\bar{C}D + \bar{A}BCD + \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}\bar{D} + A\bar{C} + A\bar{B}C + \bar{B} \tag{5.29}$$

Solution

The method for obtaining K-map is

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

Fig. 5.16 ***K-map of Eq. (5.29)***

1. Enter 1 in the cell with $A = 1, B = 1, C = 0, D = 1$ corresponding to the minterm $AB\bar{C}D$.
2. Enter 1 in the cell with $A = 0, B = 1, C = 1, D = 1$ corresponding to the minterm $\bar{A}BCD$.
3. Enter 1's in the two cells with $A = 0, B = 0, C = 0$ corresponding to the term $\bar{A}\bar{B}\bar{C}$.
4. Enter 1's in the two cells with $A = 0, B = 0, D = 0$ (one of these is already entered) corresponding to the term $\bar{A}\ \bar{B}\ \bar{D}$
5. Enter 1's in the two cells with $A = 1, B = 0, C = 1$ corresponding to the term $A\bar{B}C$.
6. Enter 1's in the four cells with $A = 1, C = 0$ (one of them is already entered) corresponding to the term $A\bar{C}$.
7. Enter 1's in the eight cells with $B = 0$ (all of them except one have already been entered) corresponding to the term $\bar{B}$.

The K-map is given in Fig. 5.16 which is same as the K-map of Fig. 5.13 and the minimised expression is given by Eq. (5.25).

Example 5.18

Minimise the four variable logic function

$$f(A, B, C, D) = (A + B + \bar{C} + \bar{D}) \cdot (\bar{A} + C + \bar{D}) \cdot (\bar{A} + B + \bar{C} + \bar{D}) \cdot (\bar{B} + C) \cdot (\bar{B} + \bar{C}) \cdot (A + \bar{B}) \cdot (\bar{B} + \bar{D}) \tag{5.30}$$

Solution

The K-map cells in which 0's are to be entered corresponding to each term are given in Table 5.9. Even if a cell is involved in more than one terms, a 0 is to be entered only once.
The K-map is given in Fig. 5.17. The minimised expression is

Table 5.9 ***K-map Cells with 0 Entries***

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

(*Continued*)

Table 5.9 (*Continued*)

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

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

$$f(A, B, C, D) = \overline{B} \cdot (\overline{A} + \overline{D})(\overline{C} + \overline{D}) \tag{5.31}$$

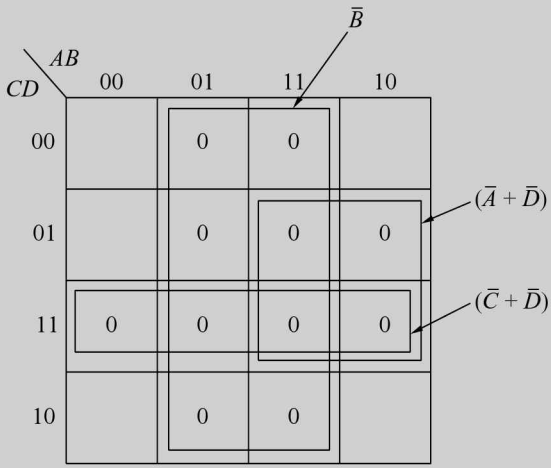


Fig. 5.17 ***K-map of Eq. (5.30)***

5.7 DON'T-CARE CONDITIONS

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

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

$$f(A, B, C, D) = \Sigma m(1, 3, 7, 11, 15) + d(0, 2, 5) \tag{5.32}$$

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

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

$$f(A, B, C, D) = \Pi M(4, 5, 6, 7, 8, 12) \cdot d(1, 2, 3, 9, 11, 14) \tag{5.33}$$

Its K-map and the minimised expression are given in Fig. 5.18*b*.

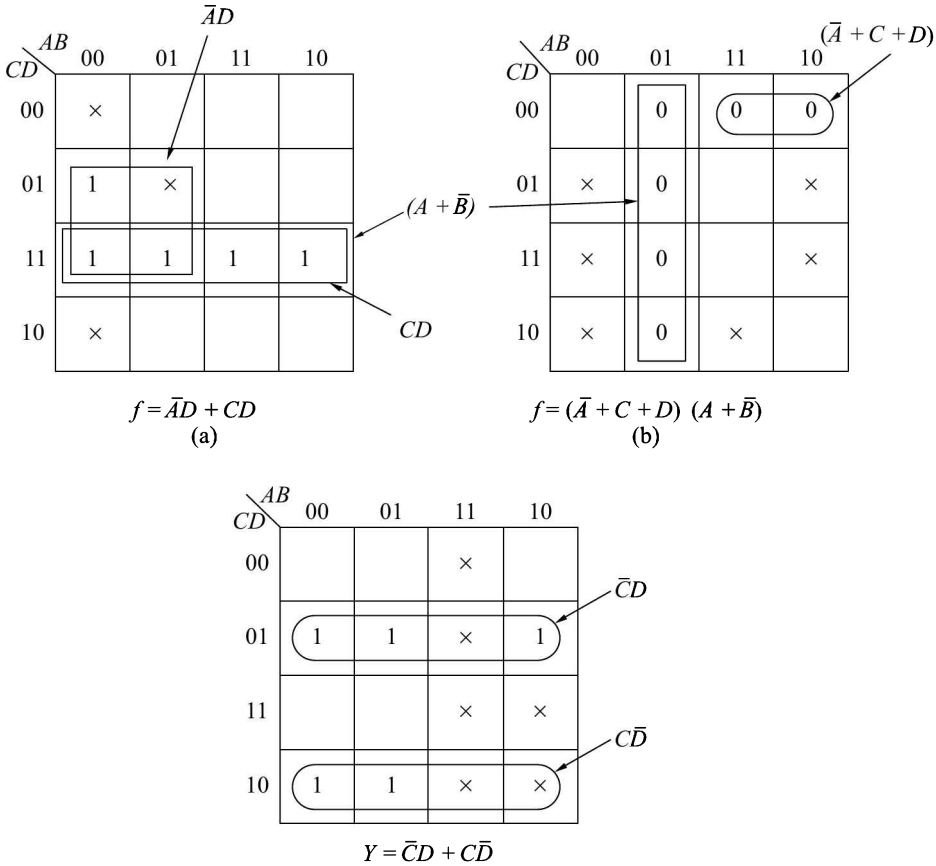


Fig. 5.18 ***K-maps with Don't-care Conditions***

3. In terms of truth table. For example, consider the truth table of Table 5.10.

Its K-map and the minimised expression in SOP form are given in Fig. 5.18*c*.

Table 5.10

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

5.8 DESIGN EXAMPLES

5.8.1 Arithmetic Circuits

1. Half-adder A logic circuit for the addition of two one-bit numbers is referred to as an *half-adder*. The addition process is illustrated in Section 2.5 and is reproduced in truth table form in Table 5.11. Here, A and B are the two inputs and S (SUM) and C (CARRY) are the two outputs.

Table 5.11 ***Truth Table of an Half-adder***

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

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

$$= \bar{A}B + A\bar{B} = A \oplus B \tag{5.34a}$$

$$C = AB \tag{5.34b}$$

The realisation of an half-adder using gates is shown in Fig. 5.19.

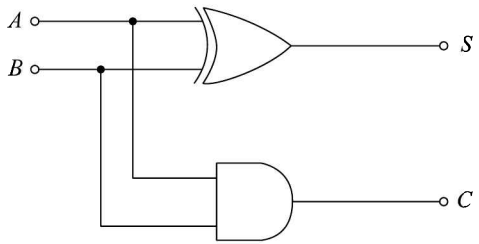


Fig. 5.19 ***Realisation of an Half-adder***

2. Full-adder An half-adder has only two inputs and there is no provision to add a carry coming from the lower order bits when multibit addition is performed.

For this purpose, a third input terminal is added and this circuit is used to add A_n, B_n, and C_{n-1}, where A_n and B_n are the nth order bits of the numbers A and B respectively and C_{n-1} is the carry generated from the addition of $(n-1)$th order bits. This circuit is referred to as *full-adder* and its truth table is given in Table 5.12.

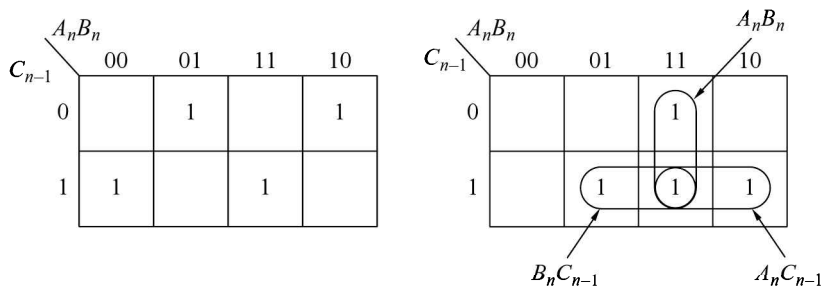
The K-maps for the outputs S_n and C_n are given in Fig. 5.20 and the minimised expressions are given by Eq. (5.35).

$$S_n = \bar{A}_n\, B_n\, \bar{C}_{n-1} + \bar{A}_n\, \bar{B}_n\, C_{n-1} + A_n\, \bar{B}_n\, \bar{C}_{n-1} + A_n\, B_n\, C_{n-1} \tag{5.35a}$$

$$C_n = A_n\, B_n + B_n\, C_{n-1} + A_n\, C_{n-1} \tag{5.35b}$$

Table 5.12 ***Truth Table of a Full-adder***

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



(a) (b)

Fig. 5.20 ***K-maps for (a) S_n (b) C_n***

The NAND–NAND realisations are given in Fig. 5.21.

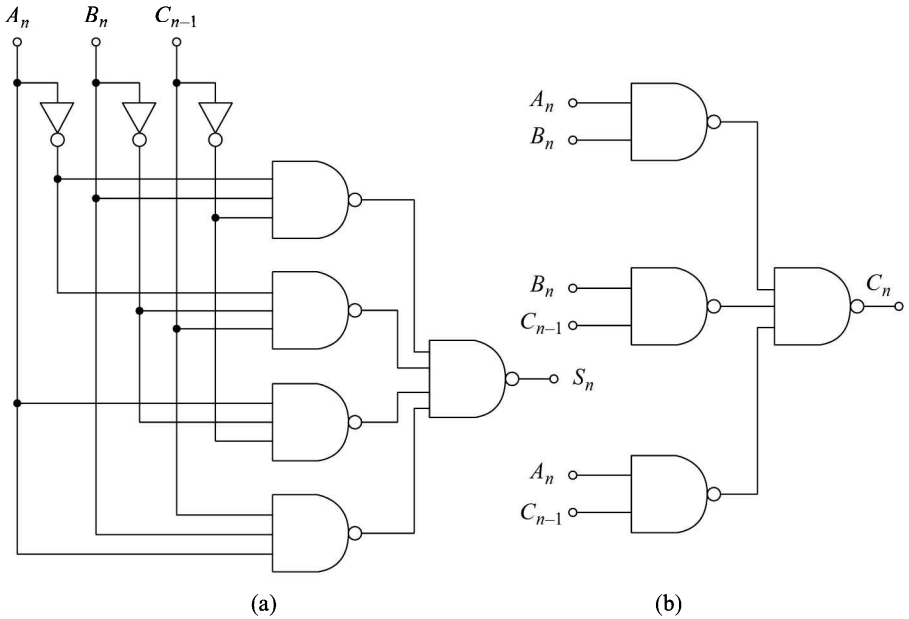


Fig. 5.21 **NAND–NAND *Realisation of (a) S_n (b) C_n***

3. Half-subtractor A logic circuit for the subtraction of B (subtrahend) from A (minuend) where A and B are 1-bit numbers is referred to as a *half-subtractor*. The subtraction process is illustrated in Section 2.5 and is reproduced in truth table form in Table 5.13. Here, A and B are the two inputs and D (difference) and C (borrow) are the two outputs.

Table 5.13 ***Truth Table of a Half-subtractor***

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

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

$$D = \bar{A}B + A\bar{B} = A \oplus B \tag{5.36a}$$

$$C = \bar{A}B \tag{5.36b}$$

The realisation of a half-subtractor using gates is shown in Fig. 5.22.

4. Full-subtractor Just like a full-adder, we require a *full-subtractor* circuit for performing multibit subtraction wherein a borrow from the previous bit position may also be there. A full-subtractor will have three inputs, A_n (minuend), B_n (subtrahend) and C_{n-1} (borrow from the previous stage) and two outputs, D_n (difference) and C_n (borrow). Its truth table is given in Table 5.14. The K-map for the output D_n is exactly same as the K-map for S_n of the adder circuit and therefore, its realisation is same as given in Fig. 5.21*a*.

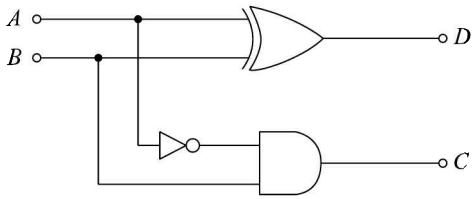


Fig. 5.22 ***Realisation of a Half-subtractor***

Table 5.14 ***Truth Table of a Full-subtractor***

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

The K-map for C_n is given in Fig. 5.23*a* and its realisation is given in Fig. 5.23*b*. The simplified expression for C_n is

$$C_n = \bar{A}_n B_n + \bar{A}_n C_{n-1} + B_n C_{n-1} \tag{5.37}$$

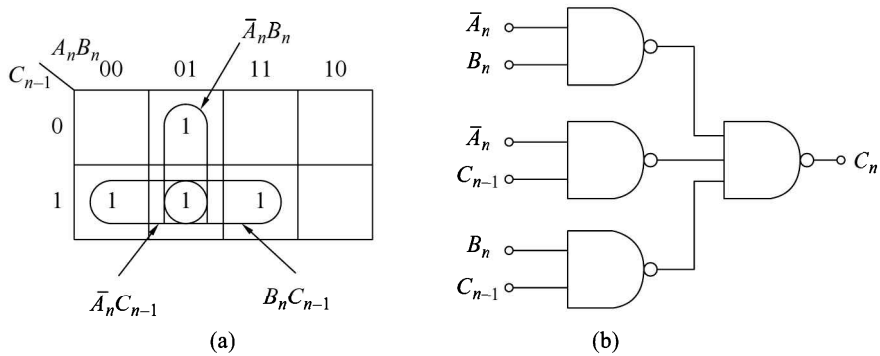


Fig. 5.23 ***(a) K-map for C_n (b) Realisation of C_n***

5.8.2 BCD-to-7-Segment Decoder

A digital display that consists of seven LED segments is commonly used to display decimal numerals in digital systems. Most familiar examples are electronic calculators and watches where one 7-segment display device is used for displaying one numeral 0 through 9. For using this display device, the data has to be converted from some binary code to the code required for the display. Usually, the binary code used is natural BCD. Figure 5.24*a* shows the display device, Fig. 5.24*b* shows the segments which must be illuminated for each of the numerals and Fig. 5.24*c* gives the display system.

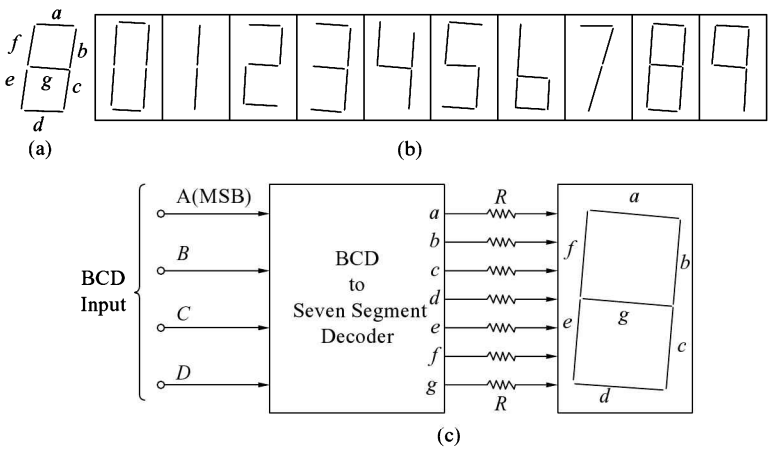


Fig. 5.24 ***(a) 7-segment Display (b) Display of Numerals (c) Display System***

Table 5.15 gives the truth table of BCD-to-7-segment decoder. Here *ABCD* is the natural BCD code for numerals 0 through 9. The K-maps for each of the outputs *a* through *g* are given in Fig. 5.25. The entries

Table 5.15 ***Truth Table of BCD-to-7 Segment Decoder***

Decimal digit displayed	Inputs				Outputs						
	A	*B*	*C*	*D*	*a*	*b*	*c*	*d*	*e*	*f*	*g*
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1

in the K-map corresponding to six binary combinations not used in the truth table are × – don't-care. The K-maps are simplified and the minimum expressions are given by:

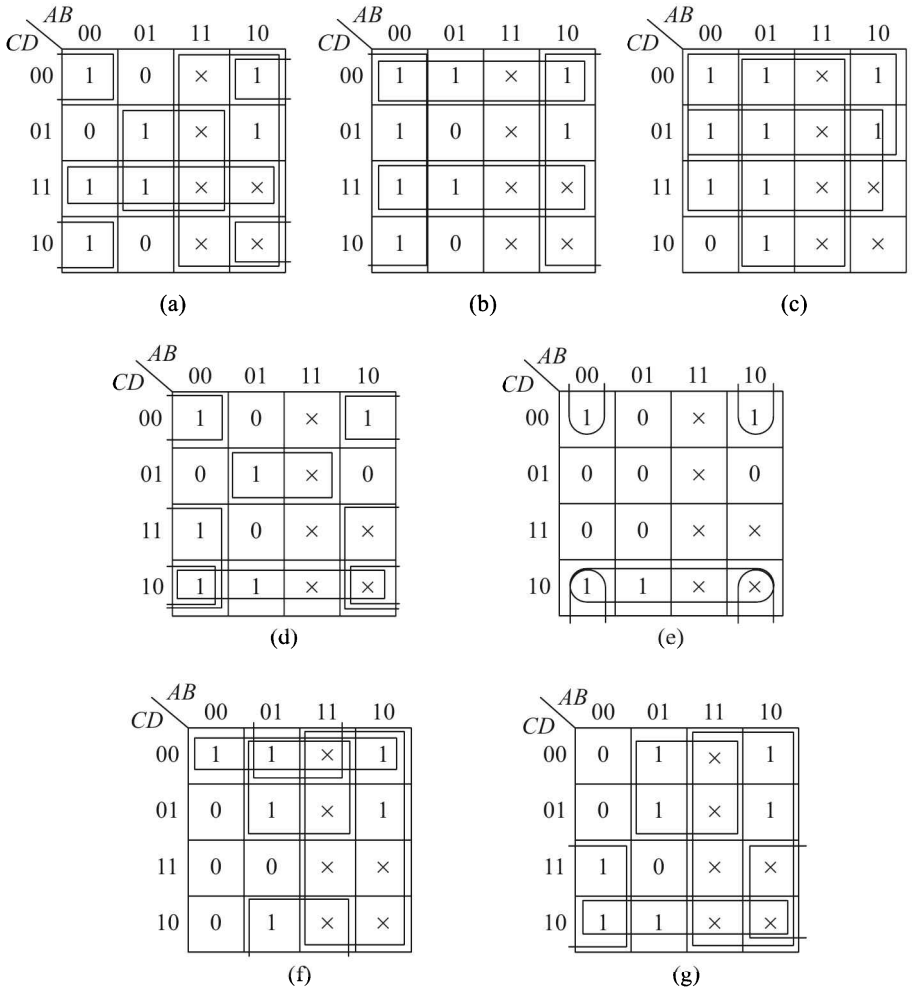


Fig. 5.25 ***K-maps of Table 5.15***

$$a = \bar{B}\bar{D} + BD + CD + A \tag{5.38}$$

$$b = \bar{B} + \bar{C}\bar{D} + CD \tag{5.39}$$

$$c = B + \bar{C} + D = \overline{\bar{B}C\bar{D}} \tag{5.40}$$

$$d = \bar{B}\bar{D} + C\bar{D} + \bar{B}C + B\bar{C}D \tag{5.41}$$

$$e = \bar{B}\bar{D} + C\bar{D} \tag{5.42}$$

$$f = A + \bar{C}\bar{D} + B\bar{C} + B\bar{D} \tag{5.43}$$

$$g = A + B\bar{C} + \bar{B}C + C\bar{D} \tag{5.44}$$

The NAND gate realisations are shown in Fig. 5.26.

We see from the realisations of Fig. 5. 26 that a term with single literal must be inverted and then applied to the second level NAND gate.

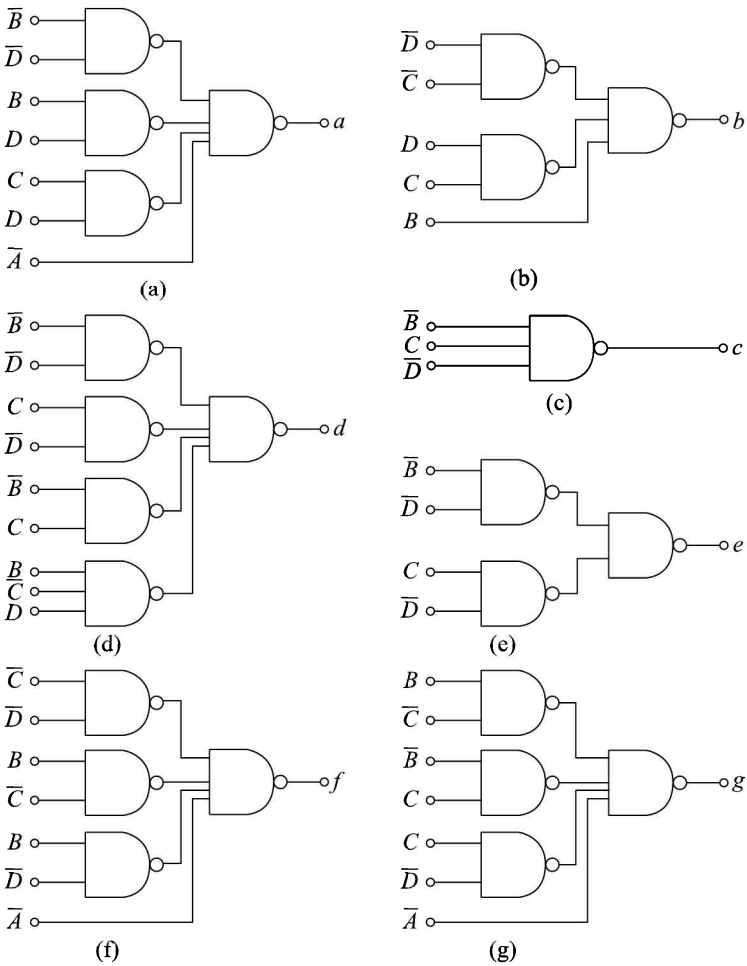


Fig. 5.26 NAND ***Gate Realisations of Eqs (5.38) to (5.44)***

5.8.3 Encoder

An encoder is a logic circuit which converts digits and/or some special symbols to a binary coded format. This process is known as *encoding*. It has n inputs and m outputs. For example, a decimal-to-BCD encoder has

ten inputs—one for each decimal digit, and four outputs corresponding to the natural BCD code. Figure 5.27 shows its block diagram and Table 5.16 gives its truth table.

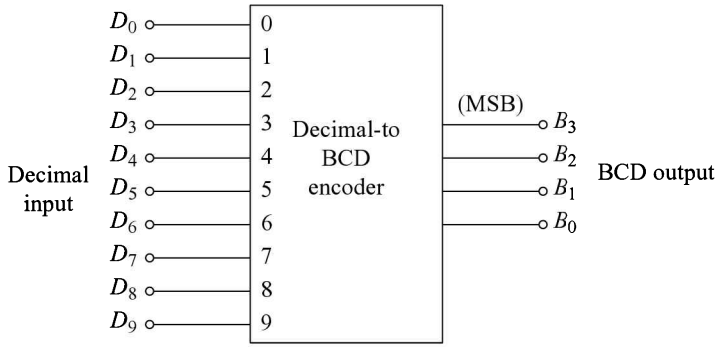


Fig. 5.27 ***Block Diagram of Decimal-to-BCD Encoder***

Table 5.16 ***Truth Table of Decimal-to-BCD Encoder***

Input	**Output**			
Decimal digit	B_3	B_2	B_1	B_0
D_0	0	0	0	0
D_1	0	0	0	1
D_2	0	0	1	0
D_3	0	0	1	1
D_4	0	1	0	0
D_5	0	1	0	1
D_6	0	1	1	0
D_7	0	1	1	1
D_8	1	0	0	0
D_9	1	0	0	1

From the truth table, we obtain the logic expressions for the outputs.

$$B_3 = D_8 + D_9 \tag{5.45}$$

$$B_2 = D_4 + D_5 + D_6 + D_7 \tag{5.46}$$

$$B_1 = D_2 + D_3 + D_6 + D_7 \tag{5.47}$$

$$B_0 = D_1 + D_3 + D_5 + D_7 + D_9 \tag{5.48}$$

From the expressions Eqs. (5.45) to (5.48), the logic circuit obtained is given in Fig. 5.28.

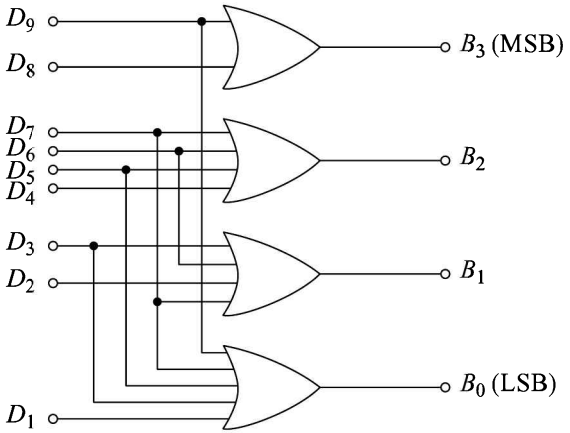


Fig. 5.28 ***Implementation of Decimal-to-BCD Encoder***

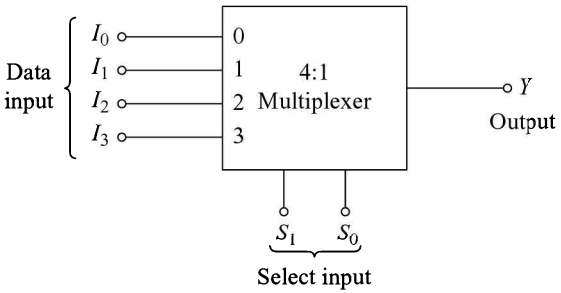


Fig. 5.29 ***Block Diagram of a 4:1 Multiplexer***

5.8.4 Multiplexer

The multiplexer (or data selector) is a logic circuit that allows one of the n data inputs at the output. Figure 5.29 shows a 4:1 multiplexer. It has four data input lines (I_0–I_3), two select lines (S_1, S_0) and Y is the data output line. One of the four data inputs will appear at the output depending on the value of S_1 S_0. Its truth table is given in Table 5.17.

From Table 5.17 , the logic expression for the output is

Table 5.17 ***Truth Table of a 4:1 Multiplexer***

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

$$Y = \overline{S}_0 \cdot \overline{S}_1 \cdot I_0 + S_0 \cdot \overline{S}_1 \cdot I_1 + \overline{S}_0 \cdot S_1 \cdot I_2 + S_0 \cdot S_1 \cdot I_3 \quad (5.49)$$

Its realisation using NAND gates and inverters is shown in Fig.5.30.

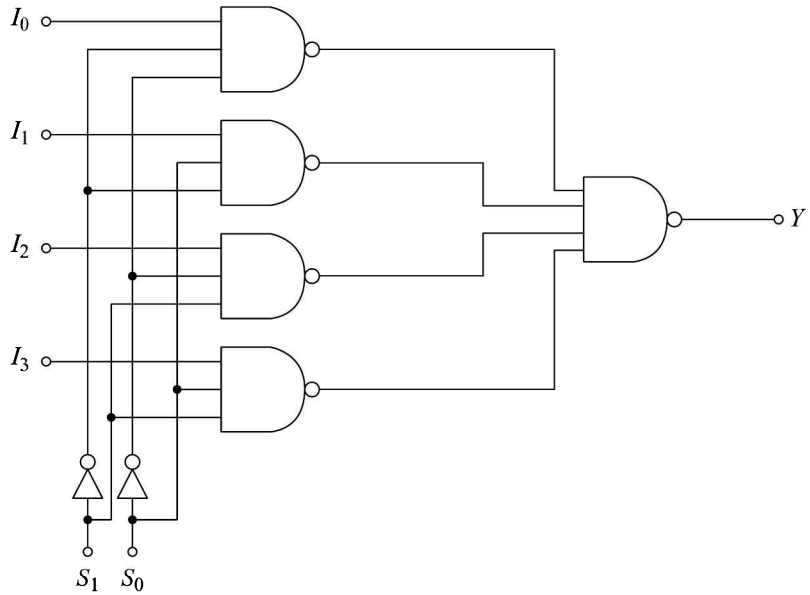


Fig. 5.30 ***Realisation of a 4:1 Multiplexer***

5.9 EX–OR AND EX–NOR SIMPLIFICATION OF K-MAPS

There are many situations in logic design in which simplification of logic expression is possible in terms of EX–OR and EX–NOR operations. These functions are widely used in digital design and therefore are available in IC form. This section deals with recognising K-map patterns indicating EX–OR and EX–NOR functions. Ring map has been proposed by Fronek, Donald K in his paper entitled 'Ring Map Minimises Logic Circuit', *Electronic Design*17 (1972) for simplifying EX–OR functions and their complements. This is an unnecessary modification of K-map and hence not adopted in practice. The interested reader may refer to the original paper. In AND–OR or OR–AND simplification, the adjacent ones or zeros are grouped. The adjacency used is horizontal or vertical. In the case of EX–OR/EX–NOR simplification we have to look for:

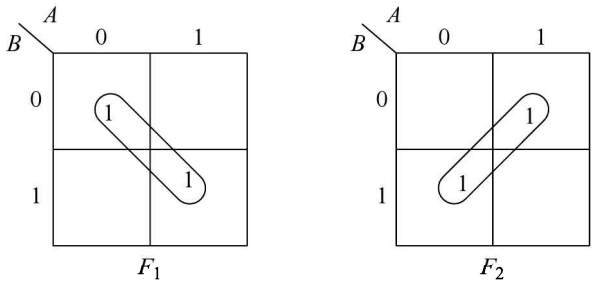
1. Diagonal adjacencies, and
2. Offset adjacencies.

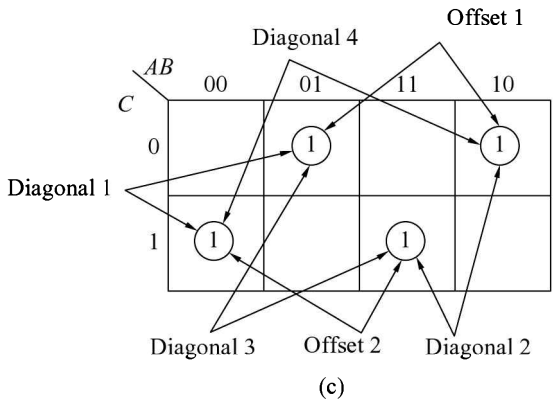
Examples of diagonal and offset adjacencies for single ones are given in Fig. 5.31. All the possible diagonal and offset adjacencies are marked in the figures and the terms corresponding to each group of such adjacency involving EX–OR or EX–NOR operation are given below.

Two-variable K-maps

$$F_1 = \bar{A}\bar{B} + AB = \overline{A \oplus B} = A \odot B$$

$$F_2 = A\bar{B} + \bar{A}B = A \oplus B$$





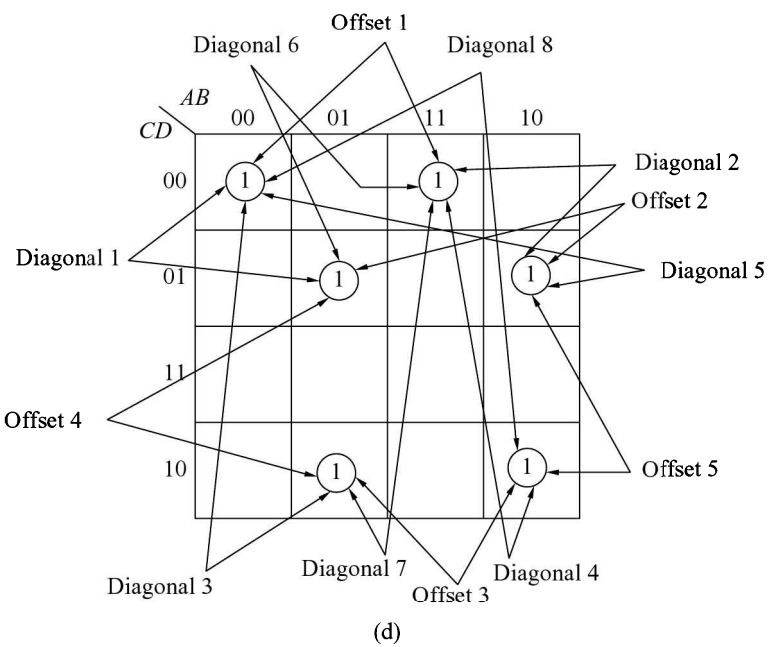


Fig. 5.31 ***K-maps Illustrating Diagonal and Offset Adjacencies Groups of Two Ones***

Three-variable K-map

Offset 1: $F_3 = (\overline{A}B + A\overline{B})\overline{C}$
$= (A \oplus B)\overline{C}$

Offset 2: $F_4 = (\overline{A}\overline{B} + AB)C$
$= (\overline{A \oplus B})C$
$= (A \odot B)C$

Diagonal 1: $F_5 = \overline{A}B\overline{C} + \overline{A}\overline{B}C$
$= \overline{A}(B \oplus C)$

Diagonal 2: $F_6 = A\overline{B}\overline{C} + ABC$
$= A(B \odot C)$

Diagonal 3: $F_7 = B(A \odot C)$

Diagonal 4: $F_8 = \overline{B}(A \oplus C)$

Four-variable K-map

Offset 1: $F_9 = \overline{A}\overline{B}\overline{C}\overline{D} + AB\overline{C}\overline{D} = \overline{C}\overline{D}(A \odot B)$

Offset 2: $F_{10} = \overline{C}D(A \oplus B)$

Offset 3: $F_{11} = C\overline{D}(A \oplus B)$

Offset 4: $F_{12} = \overline{A}B(C \oplus D)$

Offset 5: $F_{13} = A\overline{B}(C \oplus D)$

Diagonal 1: $F_{14} = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}B\overline{C}D = \overline{A}\overline{C}(B \odot D)$

Diagonal 2: $F_{15} = A\overline{C}(B \oplus D)$

Diagonal 3: $F_{16} = \overline{A}\overline{D}(B \odot C)$

Diagonal 4: $F_{17} = A\overline{D}(B \oplus C)$

Diagonal 5: $F_{18} = \overline{B}\overline{C}(A \odot D)$

Diagonal 6: $F_{19} = B\overline{C}(A \oplus D)$

Diagonal 7: $F_{20} = B\overline{D}(A \oplus C)$

Diagonal 8: $F_{21} = \overline{B}\overline{D}(A \odot C)$

From this we can find the way to identify these adjacencies and the method of obtaining the term corresponding to each grouping.

5.9.1 Diagonal and Offset Adjacencies of Groups of Ones

Figure 5.32 illustrates the diagonal and offset adjacencies of standard groups of two ones. Their simplified terms are also given in the figures.

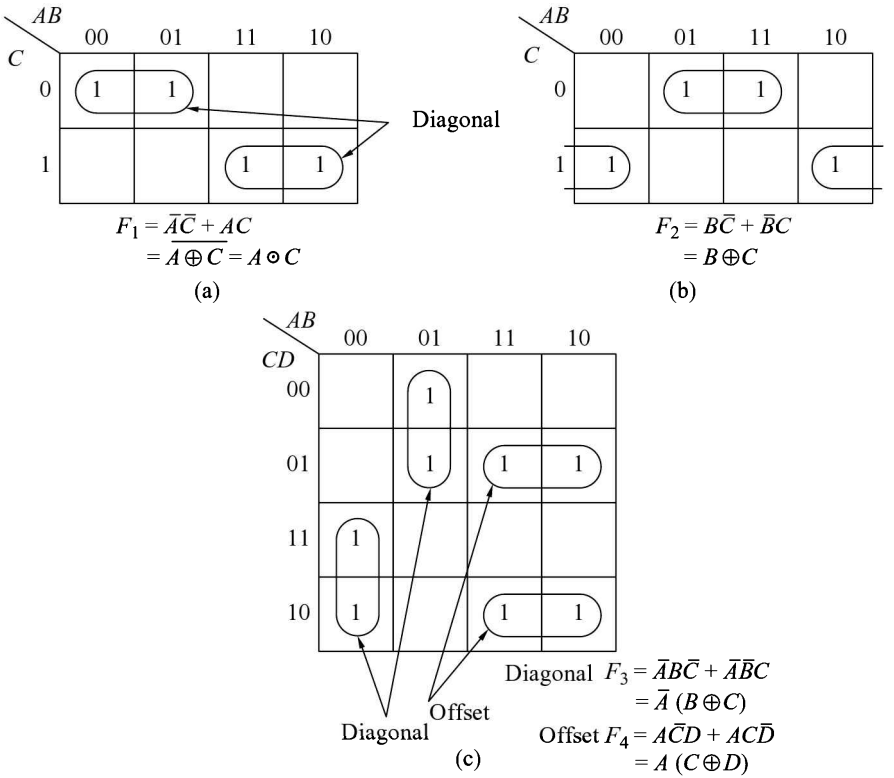


Fig. 5.32 ***K-maps Illustrating Diagonal and Offset Adjacencies of Groups of Two Ones***

From this, we observe that if standard K-map groupings of two ones occur in a diagonal or offset adjacent pattern, these can be recognised as EX–OR or EX–NOR functions and the function can be simplified in terms of EX–OR/EX–NOR operations.

Figure 5.33 illustrates the diagonal and offset adjacencies of standard groups of four ones. Their simplified terms are also given in the figures.

From this we observe the EX–OR/EX–NOR function patterns for standard K-map groupings of four ones.

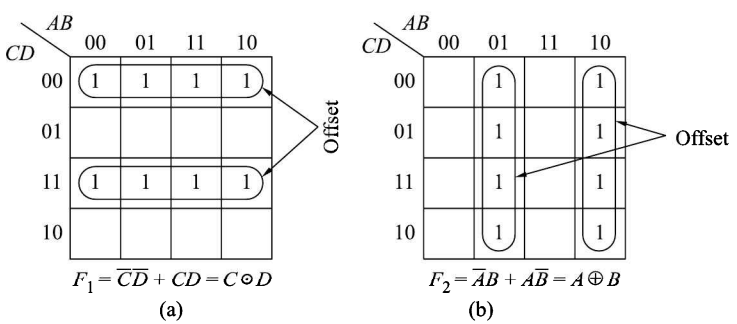


Fig. 5.33 ***K-maps Illustrating Diagonal and Offset Adjacencies of Groups of Four Ones***

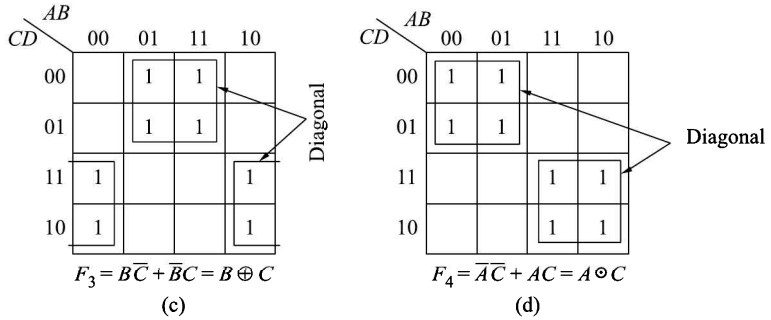


Fig. 5.33 ***(Continued)***

Although it is possible to write a set of rules for K-map simplification in terms of EX-OR/OR-NOR operations, it will unnecessarily complicate the matter. The K-map should be simplified first using standard methods and then the diagonal and offset adjacencies must be identified and the expression simplified partially by using Boolean theorems and EX-OR/EX-NOR operations. The following examples will clarify the procedure further.

Example 5.19

Design a Binary-to-Gray code converter.

Solution

The truth table of Binary-to-Gray code converter is given in Table 2.8. For each of the four outputs, K-maps are prepared and simplified. The K-maps are given in Fig. 5.34 and the simplified expressions are given by Eqs (5.50). The circuit is given in Fig. 5.35.

$$_3 = B_3 \tag{5.50a}$$

$$_2 = B_2 \oplus B_3 \tag{5.50b}$$

$$_1 = B_1 \oplus B_2 \tag{5.50c}$$

$$_0 = B_0 \oplus B_1 \tag{5.50d}$$

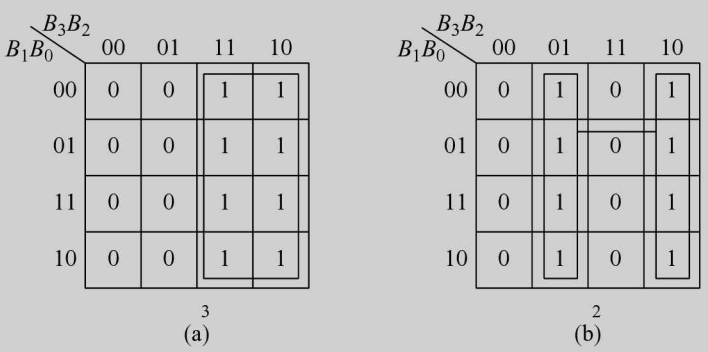


Fig. 5.34 ***K-maps of Ex. 5.19***

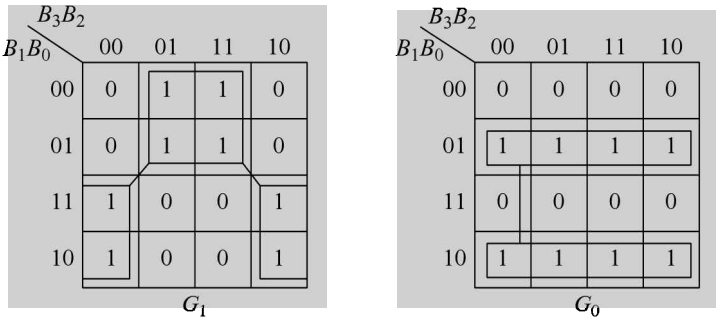


Fig. 5.34 ***(Continued)***

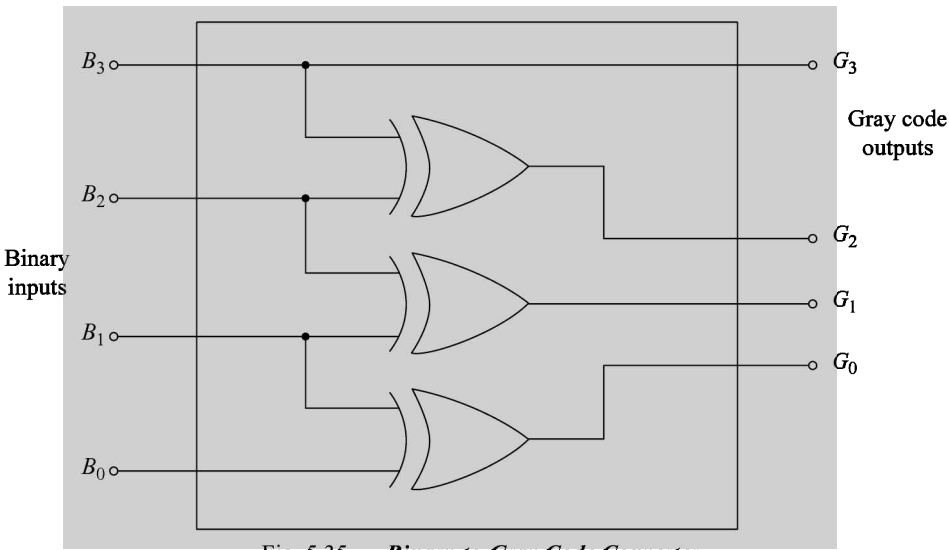


Fig. 5.35 ***Binary-to-Gray Code Converter***

Example 5.20

Design a Gray-to-Binary code converter.

Solution

The truth table for this is given in Table 2.8. The K-maps are given in Fig. 5.36 and the simplified expressions are given by Eqs (5.51).

The converter circuit is given in Fig. 5.37.

$$B_3 = G_3 \tag{5.51a}$$

$$B_2 = G_2 \oplus G_3 \tag{5.51b}$$

$$B_1 = G_1 \oplus G_2 \oplus G_3 \tag{5.51c}$$

$$B_0 = G_0 \oplus G_1 \oplus G_2 \oplus G_3 \tag{5.51d}$$

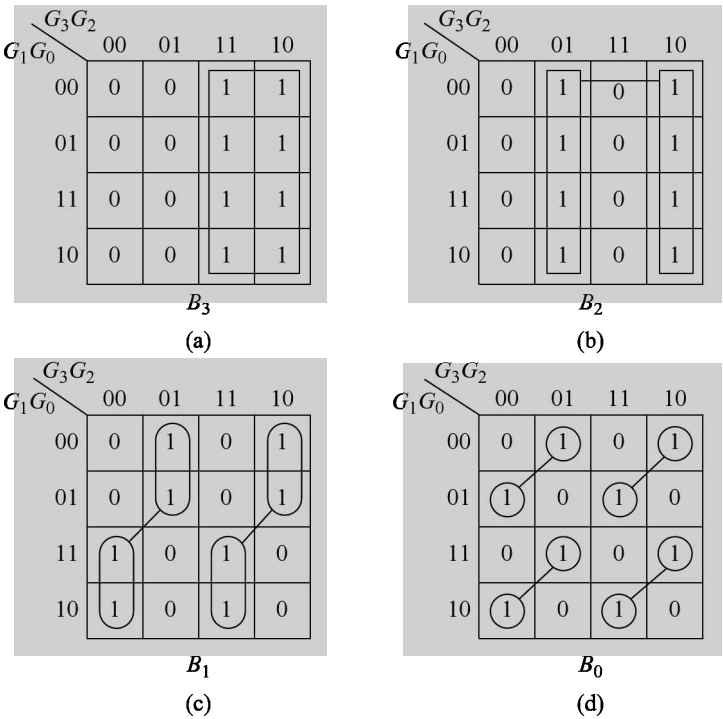


Fig. 5.36 ***K-maps of Ex. 5.20***

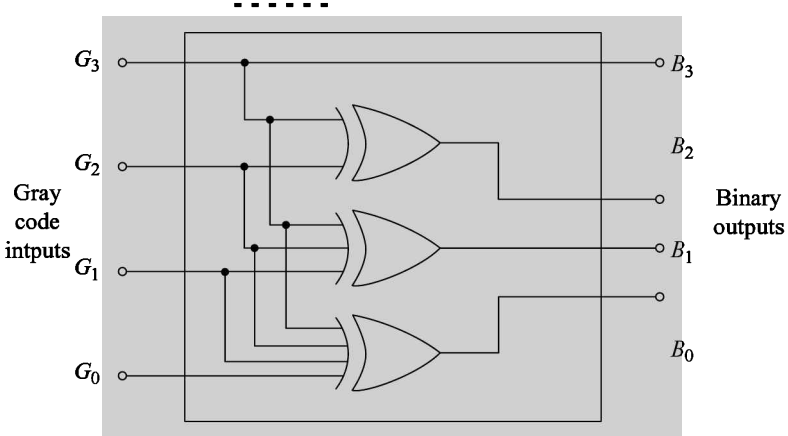


Fig. 5.37 ***Gray-to-Binary Code Converter***

5.10 FIVE- AND SIX-VARIABLE K-MAPS

A five and a six variable K-maps are shown in Figs 5.38 and 5.39, respectively. In a five variable map, we have to consider the two four-variable maps superimposed on one another; not 'hinged' or 'mirror imaged'. A six-variable map has four four-variable maps. The adjacencies between entries in each four-variable map are visualized in the normal way. However, the adjacencies between the four-variable maps are visualized as

$A = 0$

D \ BC	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

$A = 1$

D \ BC	00	01	11	10
00	16	20	28	24
01	17	21	29	25
11	19	23	31	27
10	18	22	30	26

Fig. 5.38 ***A Five-variable K-map***

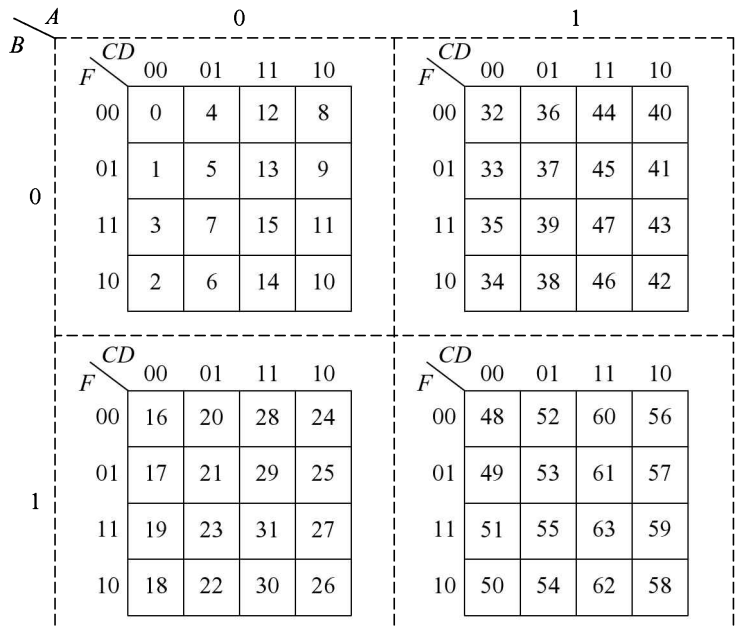


Fig. 5.39 ***A Six-variable K-map***

groupings in a two variable map. The use of five and six variable K-maps is illustrated with the help of the following examples.

Example 5.21

Simplify the logic expression

$$F(A,\ B, C, D, E) = \Sigma m(0, 5, 6, 8, 9, 10, 11, 16, 20, 24, 25, 26, 27, 29, 31) \tag{5.52}$$

Solution

The K-map is shown in Fig. 5.40 and the simplified expression is,

$$F = \bar{A}\bar{B}C\bar{D}E + \bar{A}\bar{B}CD\bar{E} + A\bar{B}\bar{D}\bar{E} + \bar{C}\bar{D}\bar{E} + ABE + B\bar{C} \tag{5.53}$$

Equation (5.53) can be realised using NAND–NAND configuration. Try this.

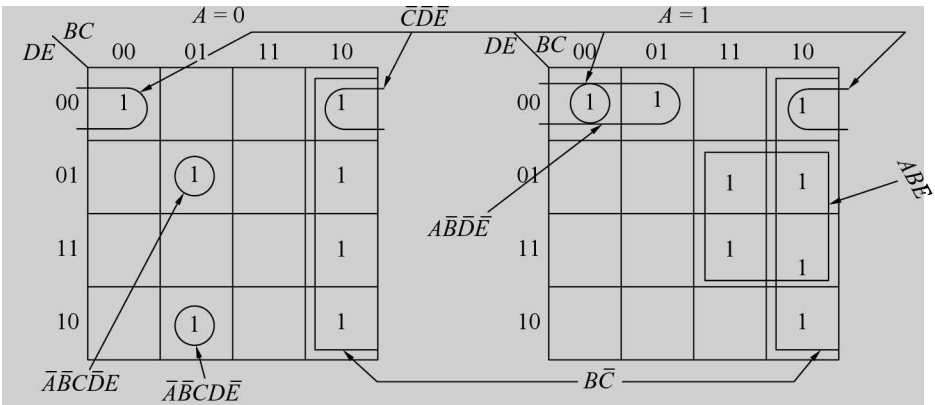


Fig. 5.40 ***K-map of Eq. (5.52)***

Example 5.22

Simplify the six-variable logic expression

$$f(A,\ B, C, D, E, F) = \Sigma m(0, 5, 7, 8, 9, 12, 13, 23,\ 24, 25, 28, 29,\ 37, 40, 42, 44, 46, 55, 56, 57, 60, 61) \tag{5.54}$$

Solution

The K-map of Eq. (5.54)is shown in Fig 5.41 and the similified expression is

$$F = \bar{A}\bar{B}\bar{D}\bar{E}\bar{F} + \bar{A}\bar{B}\bar{C}DF + \bar{B}\bar{C}D\bar{E}F + B\bar{C}DEF + A\bar{B}C\bar{F} + \bar{A}C\bar{E} + BC\bar{E} \tag{5.55}$$

Equation (5.55) can be realised using NAND–NAND configuration. Try this.

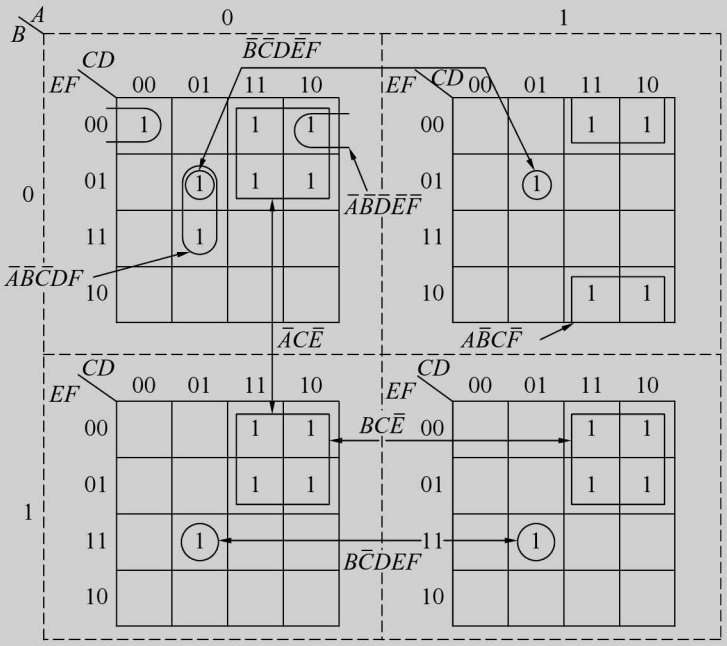


Fig. 5.41 ***K-map of Eq. (5.54)***

5.11 QUINE-McCLUSKEY MINIMISATION TECHNIQUE

Modern digital systems are designed using complex programmable logic devices (CPLDs), field-programmable gate arrays (FPGAs), and other very large scale integrated circuits that can be configured by the end user. These devices are highly complex and therefore, the techniques required for designing digital systems using these devices have to be computer driven rather than manual. A logic minimisation technique which has the following characteristics is therefore, required:

1. It should have the capability of handling large number of variables.
2. It should not depend on the ability of a human user for recognising prime-implicants.
3. It should ensure minimised expression.
4. It should be suitable for computer solution.

The Quine-McCluskey minimisation technique satisfies the above requirements and hence can be effectively used for the design of logic circuits. The K-map technique is not suitable for handling the design of complex digital systems because of the following disadvantages:

1. Minimisation of logic functions involving more than six variables is unwieldy.
2. Recognition of prime-implicants that may form part of the simplified function relies on the ability of the human user making it difficult to be sure whether the best selection has been made.

The Quine-McCluskey method consists of two parts:

1. To find by an exhaustive search all the prime-implicants that may form part of the simplified function.
2. To identify essential prime-implicants obtained from part 1 and choose among the remaining prime-implicants those that give an expression with the least number of literals.

The method can be best understood with the help of examples. This method is also known as Tabular method.

Example 5.23

Simplify the logic function of Eq.(5.21) using the Quine-McCluskey minimisation technique.

Solution

The logic function of Eq. (5.21) can be written as

$$Y(A, B, C, D) = \Sigma m(0, 1, 3, 7, 8, 9, 11, 15) \tag{5.56}$$

Step 1. The logic function to be minimised here is in the minterm form, therefore, we go to Step 2. In case a function to be minimised is not specified in the minterm form, it is required to be converted to this form using the method of Section 5.2.

Step 2. Arrange all the minterms of the function in binary representation form in a Table according to the number of ones contained and form the groups containing no ones, one 1, two 1s, three 1s and so on. The groups are separated by horizontal lines.

Table 5.18 shows the arrangement of groups, minterms, and the variables. Here group 0 contains no 1s {0}; group 1 contains minterms having a single 1 {1,8}; group 2 contains minterms with two 1s {3,9}; group 3 contains minterms with three 1s {7,11}; and group 4 contains minterms with four 1s {15}.

Table 5.18 ***Grouping Minterms According to the Number of 1s***

Group	Minterm	Variables				See Step 3
		A	*B*	*C*	*D*	
0	0	0	0	0	0	✓
1	1	0	0	0	1	✓
	8	1	0	0	0	✓
2	3	0	0	1	1	✓
	9	1	0	0	1	✓
3	7	0	1	1	1	✓
	11	1	0	1	1	✓
4	15	1	1	1	1	✓

Step 3. The Boolean algebraic theorem $A + \overline{A} = 1$ (Theorem 1.7) is applied to pairs of minterms in which only one variable is different and all the other variables are same. This kind of relationship will be applicable only to the minterms belonging to adjacent groups of minterms. For this search, compare each minterm in group

n with each minterm in group $(n + 1)$ and identify the matched pairs. Put a check (✓) on each matched pair as shown in Table 5.18.

The detailed procedure for comparison and identification of matched pairs for Table 5.18 is given below and another Table 5.19 is prepared.

Table 5.19 ***Combination of Minterm Groups of Two***

Group	Minterms	Variables				See Step 4
		A	*B*	*C*	*D*	
0	0, 1	0	0	0	–	✓
	0, 8	–	0	0	0	✓
1	1, 3	0	0	–	1	✓
	1, 9	–	0	0	1	✓
	8, 9	1	0	0	–	✓
2	3, 7	0	–	1	1	✓
	3, 11	–	0	1	1	✓
	9, 11	1	0	–	1	✓
3	7, 15	–	1	1	1	✓
	11, 15	1	–	1	1	✓

(i) The minterm 0 from group 0 is compared with the minterm 1 in the adjacent group 1. The three variables A, B, C are same in both with value 0 and the variable D is 0 in minterm 0 and 1 in minterm 1. Check (✓) marks are placed on both the terms in Table 5.18 and a new term is generated as a result of the matching of these two terms which will contain $A = 0$, $B = 0$, $C = 0$ and a dash (–) mark is placed in D. Since the variable D differs and hence it gets eliminated and the resulting combination of these two terms is $\overline{A}\,\overline{B}\,\overline{C}$.

(ii) Next the minterm 0 is compared with the minterm 8. The minterms match and their combination results in a term $\overline{B}\,\overline{C}\,\overline{D}$. Check mark is placed on minterm 8 in Table 5.18, check mark on minterm 0 has already been placed. A—is placed under variable A.

(iii) Similarly, comparison of minterm 1 with 3 results in
$A = 0$, $B = 0$, $C = -$, and $D = 1$
The minterm 3 is checked in Table 5.18.

(iv) Comparison of minterm 1 with minterm 9 yields
$A = -$, $B = 0$, $C = 0$, and $D = 1$
and the minterm 9 is checked.

(v) Now compare 8 with 3 and 9. The minterms 8 and 3 do not match. The comparison of minterms 8 and 9 results in
$A = 1$, $B = 0$, $C = 0$, and $D = -$

(vi) Next compare the minterms 3 and 7, it results in
$A = 0$, $B = -$, $C = 1$, and $D = 1$ and the minterm 7 is checked in Table 5.18.

(vii) Comparison of the minterms 3 and 11 results in
$A = -$, $B = 0$, $C = 1$, and $D = 1$
and the minterm 11 is checked in Table 5.18

(viii) The minterms 9 and 7 do not match.

(ix) Comparison of the minterms 9 and 11 results in
$A = 1, B = 0, C = -$, and $D = 1$

(x) Next compare the minterms 7 and 15, the result is
$A = -, B = 1, C = 1$, and $D = 1$
and the minterm 15 is checked in Table 5.18

(xi) Comparison of the minterms 11 and 15 results in
$A = 1, B = -, C = 1$, and $D = 1$

Table 5.19 lists the results of all the above matchings and all of the minterms in each group have been compared to those in the next higher group.

Step 4. Next all the minterms in the adjacent groups in Table 5.19 are compared to see if groups of four can be made by matching. For this the dashes must be in the same bit position in the groups of two and only one variable must differ (0 in one group and 1 in the other). The matched pairs of minterms are checked in Table 5.19 and a new Table 5.20 is created.

Table 5.20 ***Combination of Minterm Groups of Four***

Group	Minterms	Variables			
		A	*B*	*C*	*D*
0	0, 1, 8, 9	–	0	0	–
	0, 8, 1, 9	–	0	0	–
1	1, 3, 9, 11	–	0	–	1
	1, 9, 3, 11	–	0	–	1
2	3, 7, 11, 15	–	–	1	1
	3, 11, 7, 15	–	–	1	1

In Table 5.20 we observe that in each group the two terms are same, therefore, only one is to be taken in each group.

Step 5. Repeat the process of grouping of eight minterms. In this case both dashes (–) must be in the same bit position and only one other variable must be different for matching. Since, there is no matching possible here, therefore, the process is complete. In general, this same process is repeated until no further combinations of minterm groups is possible.

Step 6. All nonchecked minterm groups in Tables 5.18, 5.19, and 5.20 are the prime-implicants of the function. The function can now be written as

$$Y(A,B,C,D) = \overline{B}\,\overline{C} + \overline{B}D + CD \tag{5.57}$$

Step 7. Next a prime-implicant table is prepared listing each of the minterms contained in the original function, PI terms, and the decimal numbers of minterms that make up the PI. Put cross (×) marks in the table in each row under the minterms contained in that PI. Table 5.21 is the PI table for the logic equation Eq. (5.21). Find the minterms that contain only one × in its column. These ×'s are encircled and the corresponding PI terms are checked (✓).

Table 5.21 ***PI table***

PI terms	Decimal Numbers	Minterms							
		0	1	3	7	8	9	11	15
$\overline{B}\,\overline{C}$ ✓	0, 1, 8, 9	⊗	×			⊗	×		
$\overline{B}D$	1, 3, 9, 11		×	×			×	×	
CD ✓	3, 7, 11, 15			×	⊗			×	⊗

The minterms 0 and 8 are contained in only one PI $\overline{B}\ \overline{C}$, the minterms 7 and 15 are contained in only PI CD. Therefore, the prime-implicants $\overline{B}\ \overline{C}$ and CD are essential prime-implicants. Now observe the other minterms and see whether these are contained in EPIs or not. Here, the minterms 1 and 9 are contained in $\overline{B}\ \overline{C}$ and 3 and 11 are contained in CD. Therefore, all the minterms of the original function are included in the two EPIs and the minimised expression will be

$$Y(A,B,C,D) = \overline{B}\overline{C} + CD \tag{5.58}$$

This is same as the minimised function obtained using K-map technique Eq. (5.22).

Example 5.24

Simplify the logic function of Eq. (5.23) using the Quine-McCluskey method.

Solution

Table 5.22 is prepared grouping the minterms according to the number of 1s contained.

Table 5.22

Group	Minterm	Variables				
		A	*B*	*C*	*D*	
0	0	0	0	0	0	✓
1	1	0	0	0	1	✓
	2	0	0	1	0	✓
	8	1	0	0	0	✓
2	3	0	0	1	1	✓
	5	0	1	0	1	✓
	9	1	0	0	1	✓
3	7	0	1	1	1	✓
	11	1	0	1	1	✓
	14	1	1	1	0	

Table 5.23 is created by comparing and matching the minterms in adjacent groups. In Table 5.22 all the minterms except m_{14} are checked (✓).

Table 5.23

Group	Minterms	Variables				
		A	*B*	*C*	*D*	
0	0, 1	0	0	0	–	✓
	0, 2	0	0	–	0	✓
	0, 8	–	0	0	0	✓
1	1, 3	0	0	–	1	✓
	1, 5	0	–	0	1	✓
	1, 9	–	0	0	1	✓
	2, 3	0	0	1	–	✓
	8, 9	1	0	0	–	✓
2	3, 7	0	–	1	1	✓
	3, 11	–	0	1	1	✓
	5, 7	0	1	–	1	✓
	9, 11	1	0	–	1	✓

Next Table 5.24 is created by comparing and matching minterms in Table 5.23. All the minterms in Table 5.23 are checked (✓).

Table 5.24

Group	Minterms	Variables			
		A	*B*	*C*	*D*
0	0, 1, 2, 3	0	0	–	–
	0, 1, 8, 9	–	0	0	–
	0, 2, 1, 3	0	0	–	–
	0, 8, 1, 9	–	0	0	–
1	1, 3, 5, 7	0	–	–	1
	1, 3, 9, 11	–	0	–	1
	1, 5, 3, 7	0	–	–	1
	1, 9, 3, 11	–	0	–	1

There is no matching possible in Table 5.24. The function f consists of the prime-implicant m_{14} (unchecked minterm in Table 5.22) and the terms corresponding to the prime-implicants of Table 5.24. It is given as

$$f(A, B, C, D) = ABC\overline{D} + \overline{A}\overline{B} + \overline{B}\overline{C} + \overline{A}D + \overline{B}D \tag{5.59}$$

The PI table corresponding to Eq. (5.59) is prepared as Table 5.25.

Table 5.25

PI terms		Decimal numbers	Minterms									
			0	*1*	*2*	*3*	*5*	*7*	*8*	*9*	*11*	*14*
$ABC\bar{D}$	✓	14										⊗
$\bar{A}\bar{B}$	✓	0, 1, 2, 3	×	×	⊗	×						
$\bar{B}\bar{C}$	✓	0, 8, 1, 9	×	×					⊗	×		
$\bar{A}D$	✓	1, 3, 5, 7		×		×	⊗	⊗				
$\bar{B}D$	✓	1, 9, 3, 11		×		×				×	⊗	
					✓		✓	✓	✓		✓	✓

Observe the PI table and find the columns containing only a single cross (×). There are six minterms whose columns have a single cross; 2, 5, 7, 8, 11, and 14. Prime-implicants that cover minterms with a single cross in their column are essential prime-implicants. The single crosses are encircled. Check marks are placed below these columns. A check mark is also placed in the table on the essential prime-implicants.

Since all the prime-implicants are essential prime-implicants, therefore, Eq. (5.59) is the minimised function. This is same as Eq.(5.24) obtained by K-map method.

In case of incompletely specified function, i.e., for functions containing don't-care terms, the don't-care terms are considered as if they are the required minterms. The prime implicants are determined using these minterms alongwith the other specified minterms. To differentiate the minterms corresponding to the don't-care conditions, an asterisk (*) mark is put on the decimal numbers corresponding to the don't-care terms. The PIs are thus determined. When forming the PI chart, the don't-care terms are not listed at the top. The PI chart is solved and the resulting expression will contain all the required don't-care minterms. The following example illustrates the complete procedure.

Example 5.25

Simplify the logic function of Eq. (5.32) using the Quine-McCluskey method

Solution

Using the procedure discussed above, Tables 5.26(a), (b), and (c) are prepared.

Table 5.26(a)

Group	Minterm/ Don't care term*	Variables *A*	*B*	*C*	*D*	
0	0*	0	0	0	0	✓
1	1	0	0	0	1	✓
	2*	0	0	1	0	✓
2	3	0	0	1	1	✓
	5*	0	1	0	1	✓
3	7	0	1	1	1	✓
	11	1	0	1	1	✓
4	15	1	1	1	1	✓

Table 5.26(b)

Group	Minterm/ Don't care term*	Variables *A*	*B*	*C*	*D*	
0	0*, 1	0	0	0	–	✓
	0*, 2*	0	0	–	0	✓
1	1, 3	0	0	–	1	✓
	1, 5*	0	–	0	1	✓
	2*, 3	0	0	1	–	✓
2	3, 7	0	–	1	1	✓
	3, 11	–	0	1	1	✓
	5*, 7	0	1	–	1	✓
3	7, 15	–	1	1	1	✓
	11, 15	1	–	1	1	✓

Table 5.26(c)

Group	Minterms/ Don't care term*	Variables A	B	C	D
0	0*, 1, 2*, 3	0	0	–	–
	0*, 2*, 1, 3	0	0	–	–
1	1, 3, 5*, 7	0	–	–	1
	1, 5*, 3, 7	0	–	–	1
2	3, 7, 11, 15	–	–	1	1
	3, 11, 7, 15	–	–	1	1

Table 5.27 gives the PI table.

Table 5.27

PI terms	Decimal numbers (Minterms)	Minterms 1	3	7	11	15
$\bar{A}\bar{B}$	0*, 1, 2*, 3	×	×			
$\bar{A}D$	1, 3, 5*, 7	×	×	×		
CD ✓	3, 7, 11, 15		×	×	⊗	⊗
					✓	✓

From the PI table, we observe only one × mark in columns 11 and 15, therefore, the PI term CD is an essential prime-implicant. Check marks are placed at the EPI and in columns 11 and 15. The remaining three minterms 1,3, and 7 are covered by the PI term $\bar{A}D$. Therefore, the minimal function is given by

$$Y = \bar{A}D + CD$$

This is same as obtained by K-map method given in Fig. 5.18(a).

From the above result, we see that the minterm $5(m_5)$ has been included in the simplification.

5.12 HAZARDS IN COMBINATIONAL CIRCUITS

In the preceding discussions, we have assumed that the logic circuits which are designed using elcctronic devices respond instantaneously to input signals i.e. the signal propagation time from input to output of a circuit has been assumed to be zero. This is infact the ideal behaviour and the actual circuits never have ideal behaviour. In practice, the electronic components used have associated propagation delays because of which the changes in response of a circuit do not occur instantaneously with the changes in the input. This may result in an unwanted phenomena known as *hazard.* Therefore, it is necessary, to have methods for detection of hazards, and to design hazard free switching circuits.

5.12.1 Detection of Hazard

The hazard in a digital circuit is detected by using K-map of the circuit. For example, consider a 3-variable logic function $Y(A, B, C) = \Sigma m(2, 3, 5, 7)$. Its K-map is given in Fig. 5.42*a* and logic circuit using AND–OR configuration in Fig. 5.42*b*.

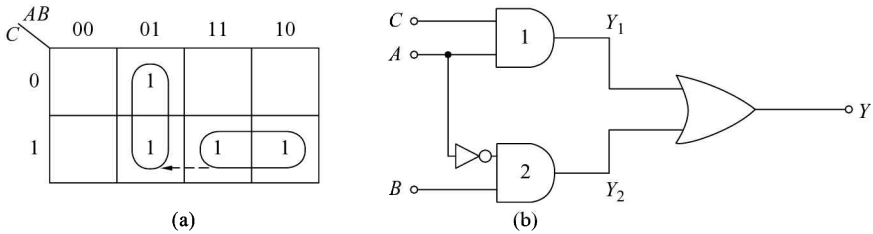


Fig. 5.42 ***(a) K-map (b) Logic Circuit***

This circuit uses minimum number of components. In the ideal case, when the propagation delay of all the gates is zero, the operation of the circuit is instantaneous i.e. Y responds to the inputs instantaneously. However because of different propagation delays of the two paths in the circuit from input to output, its transient behaviour gives rise to hazard, although its steady-state behaviour is unaffected.

Let us assume A= 1, B= 1, and C= 1. Corresponding to this input condition, the output of AND gate 1 Y_1 is 1 and the output of AND gate 2 Y_2 is 0. Now, if A changes from 1 to 0, B and C remaining 1, the output of the AND gate 1 Y_1 becomes 0 AND gate 2 Y_2 becomes 1.The value of Y continues to remain 1. If we assume that the propagation delay of AND gate 1 is less than the propagation delay of AND gate 2, then for the above input conditions, the output Y_1 becomes 0 before the output Y_2 becomes 1 and for some time the outputs of both the AND gates will be 0. This makes $Y = 0$ during that period. When the output Y_2 becomes 1, Y becomes 1. This is illustrated in Fig. 5.43

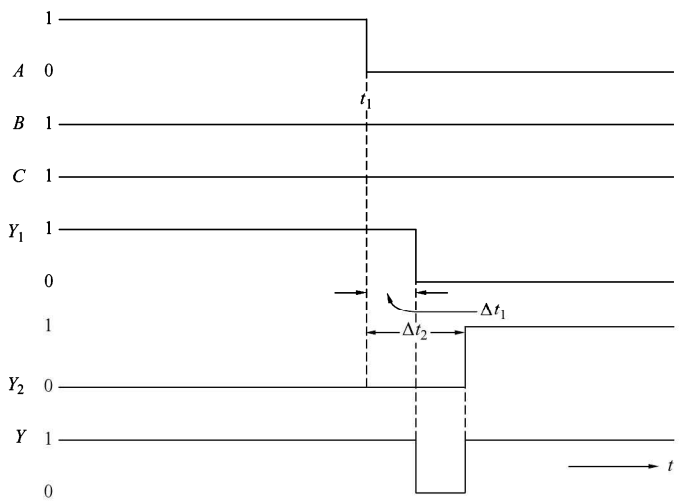


Fig. 5.43 ***Timing Diagrams of Fig. 5.42b***

Here Δt_1 and Δt_2 are the propagation delays of the paths through AND gate 1 and 2 respectively. From the timing diagram, we observe that for the time $\Delta t_2 - \Delta t_1$, the output Y becomes 0 which should have been 1 for the proper operation of the circuit. This type of transient behaviour is known as hazard and it produces a short pulse of duration $\Delta t_2 - \Delta t_1$ which is referred to as *glitch*. If the output Y of the circuit is observed or sampled during the glitch an erroneous value appears that can result in the failure of the system. A glitch can also cause noise which is undesirable.

Now let us study the occurrence of hazard from the K-map of Fig. 5.42*a*. $A = 1$, $B = 1$, and $C = 1$ corresponds to the minterm m_7 which is included in the combination of two 1's corresponding to m_5 and m_7. AND gate 1 corresponds to this condition, which means $Y_1 = 1$. When A changes from 1 to 0, B and C remaining 1, the operation is shifted to another combination of two 1's consisting of minterms m_2 and m_3. This corresponds to AND gate 2, which means $Y_2 = 1$. This change of operation from one combination to another combination is shown by dotted arrow in Fig. 5.42*a*. This shows that the hazard occurs because of two adjacent 1's in the K-map which do not form a combination of 1's together. In such a situation, whenever the circuit must move from one product term to another, there is a hazard condition resulting in a glitch. This type of circuit implementation may cause the output to go to 0 when it should remain 1, due to such adjacent 1's.

Example 5.26

(a) Minimise the logic function $Y = \Pi M\,(0, 1, 4, 6)$ and realise using NOR gates.
(b) Determine whether hazard occurs in this circuit. If yes, find the condition under which it occurs and give the timing diagrams. Assume propagation delay of NOR gate 1 to be lower than that of 2.

Solution

$$Y = \Pi M\,(0, 1, 4, 6) \tag{5.60}$$

(a) Its K-map and circuit diagram are given in Figs. 5.44*a* and *b* respectively.

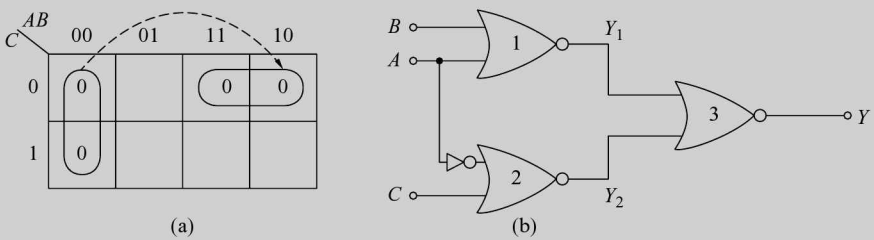


Fig. 5.44 ***(a) K-map***
(b) NOR-NOR ***Realisation For Eq. (5.60)***

(b) When $ABC = 000$ then $Y_1 = 1$, $Y_2 = 0$, and $Y = 0$

Now let ABC change to 100. Y_1 responds to this change after time Δt_1 and Y_2 responds after Δt_2. Since $\Delta t_1 < \Delta t_2$ therefore, during the interval $\Delta t_2 - \Delta t_1$ both Y_1 and Y_2 become 0 which make $Y = 1$. Under steady-state condition $Y = 0$. Therefore, a positive pulse of short duration $\Delta t_2 - \Delta t_1$ occurs which shows the occurrence of hazard. The reason of occurrence can be seen from the K-map, shown by dotted line. It is because of the circuit moving from maxterm M_0 to M_4 or in other words from one sum term to another sum term. Here in this case, it is because of two adjacent 0's not covered in a common combination of 0's. Its timing diagrams are given in Fig. 5.45.

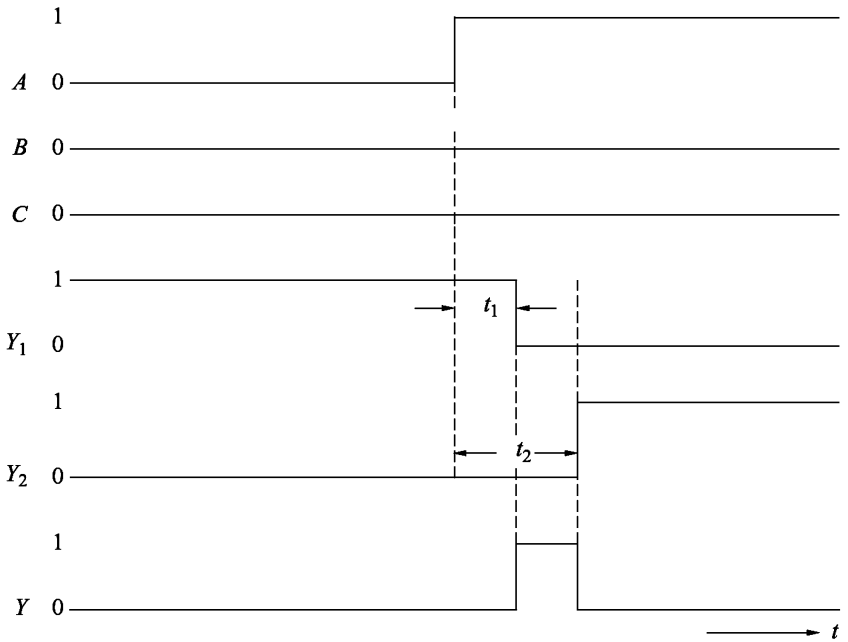


Fig. 5.45 ***Timing Diagrams of Fig. 5.44(b)***

Hazards occurring because of change of only one variable can be detected easily. In case of more than one variable changing simultaneously, it is not easy to detect the occurrence of hazard. Therefore, we shall be dealing with cases in which only one variable changes.

5.12.2 Types of Hazards

From the above discussion, we see that the occurrence of hazard in SOP and POS realisations can be observed from their K-maps. In the case of hazard occurring in SOP form of realisation the output goes to 0 momentarily when it should have remained 1; and in POS form of realisation the output goes to 1 when it should have remained 0. The first type refers to *static-1 hazard* and the second type refers to *static-0 hazard*. A third type of hazard, known as *dynamic hazard*, causes the output to change number of times when it should change from 1 to 0 or from 0 to 1. All the above three types of hazards are illustrated in Fig. 5.46

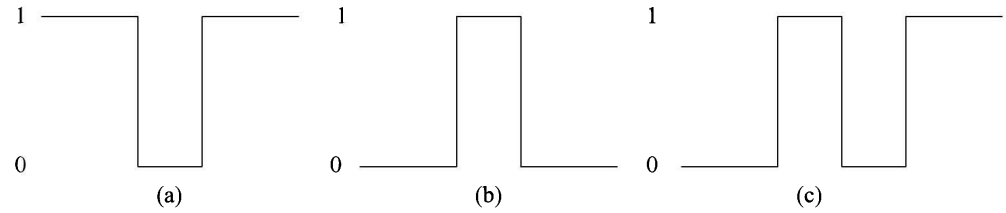


Fig. 5.46 ***Types of Hazard (a) Static-1 Hazard (b) Static-0 Hazard (c) Dynamic Hazard***

5.12.3 Design of Hazard Free Circuits

The occurrence of hazard can be easily observed from the K-map. Let us consider the K-map of Fig. 5.42*a*. The cause of hazard is due to the movement of circuit operation from one PI to another PI in which each PI has a cell with entry 1 which are not combined to form a PI. If we form another combination of two minterms m_3 and m_7 as shown in Fig. 5.47*a*, the resultant expression for Y will be

$$Y = \bar{A}B + AC + BC \tag{5.61}$$

and its realisation will be as shown in Fig. 5.47*b*. By this process hazard is eliminated and we obtain a hazard free circuit

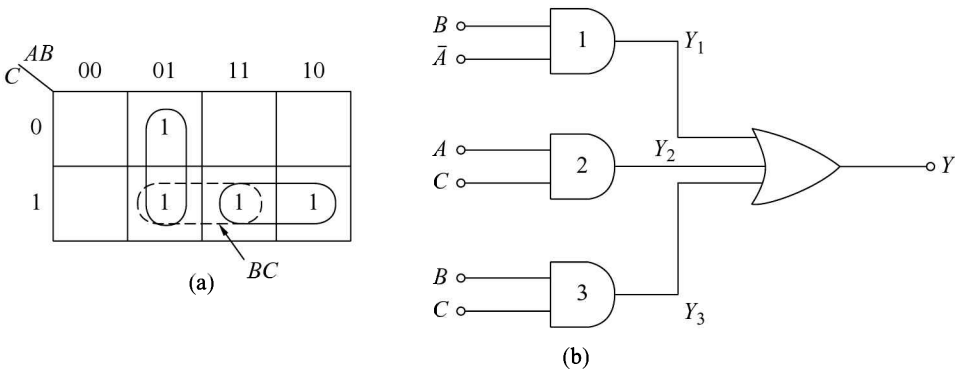


Fig. 5.47 ***(a) K-map (b) Hazard Free Logic Circuit***

In this circuit because of AND gate 3, Y_3 will continue to remain 1 even when A changes from 0 to 1 or from 1 to 0 while $B = 1$ and $C = 1$. Therefore, the hazard gets eliminated. Thus we can generalise the principle of obtaining hazard free circuit as: If every pair of adjacent cells with entry 1 is covered by atleast one combination of 1's the circuit will be hazard free. The principle of duality can be used while dealing with realisation in POS form.

From the above discussion we observe that there are two conflicting considerations for the design of switching circuits:

- *Simplicity of circuit* By using minimisation (or simplification) we obtain a circuit with minimum number of components but the minimal circuits are most vulnerable to hazards.
- *Reliability of operation* Redundancy i.e. inclusion of additional gates is required to be added in order to avoid hazards and increase reliability.

Example 5.27

Design hazard free circuit for Eq.(5.60)

Solution

The K-map shown in Fig. 5.44*a* is redrawn as shown in Fig. 5.48*a* with an additional combination of 0's combining the maxterms M_0 and M_4. The resulting NOR–NOR circuit realisation is shown in Fig. 5.48*b*

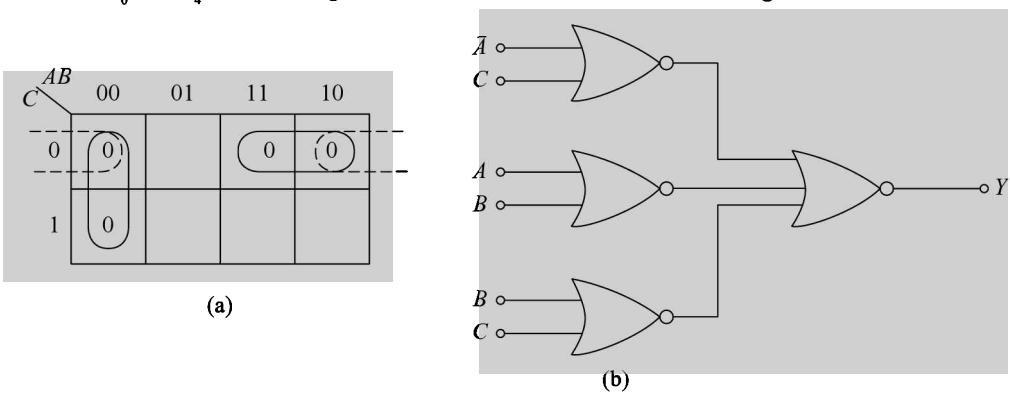


Fig. 5.48 ***(a) K-map***
(b) Hazard Free NOR–NOR ***Realisation***

When a circuit is implemented in SOP form using AND–OR or NAND–NAND configuration, the elimination of static-1 hazard guarantees that the static-0 hazard or dynamic hazard will not occur. However, it is not necessary that a hazard free AND–OR (or NAND–NAND) realisation will be hazard free equivalent OR–AND (or NOR–NOR) realisation.

Example 5.28

Show that the hazard free AND–OR realisation (Fig. 5.47*b*) does not guarantee hazard free equivalent OR–AND realisation (Fig. 5.44*b*)

Solution

The logic circuits of Figs. 5.42*b* and 5.44*b* are equivalent. Figure 5.47*b* is the hazard free circuit corresponding to Fig. 5.42*b*. The equivalent circuit of Fig. 5.44*b* is not hazard free, which illustrates that for each type of realisation, hazard free circuits are to be designed exclusively.

Example 5.29

Design a hazard free circuit in AND–OR configuration for the logic function

$$Y(A, B, C, D) = \Sigma m\ (1, 3, 5, 7, 12, 13) \tag{5.62}$$

Solution

The K-map of Eq. (5.62) is given in Fig. 5.49.

Its minimal expression is

$$Y = (A, B, C, D) = \bar{A}D + AB\bar{C} \tag{5.63}$$

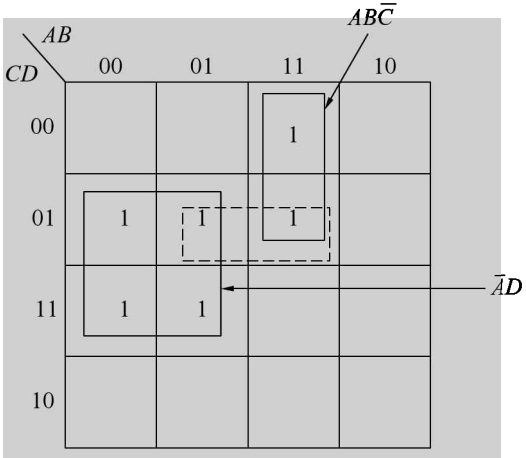


Fig. 5.49 ***K-map of Eq. (5.62)***

The circuit realised from Eq. (5.63) will have hazard because of the minterms m_5 and m_{13}. Therefore, an additional combination of these two minterms is formed and the logic expression for the hazard free realisation will be

$$Y = (A, B, C, D) = \bar{A}D + AB\bar{C} + B\bar{C}D \tag{5.64}$$

Its AND–OR realisation is given in Fig. 5.50

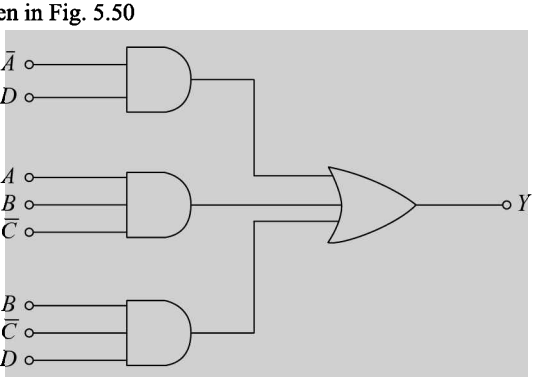


Fig. 5.50 ***Hazard Free*** AND–OR ***Realisation For Eq. (5.62)***

SUMMARY

For any logic design, it is always essential to design a product which meets the following requirements:

- Minimum cost.
- Minimum space requirement.
- Maximum speed of operation.
- Easy availability of components.
- Ease of interconnecting the components.
- Easy to design.

To achieve most of the above requirements, it is necessary to have some tools for simplification of logic expressions. Some of the traditional and widely used methods of logic design, such as, Boolean algebraic method, Karnaugh map technique, and Quine-McCluskey methods have been discussed. The logic expressions can be expressed in one of the standard forms: SOP or POS and then simplified using, K-maps or Quine-McCluskey method. This results in saving in required hardware in terms of number of gates required and the number of input terminals (fan-in) of the gates.

As discussed in Chapter 1, a large number of gates are available in IC form. The number of gates which can be packaged in an IC chip decreases with the number of input terminals of the gates. Therefore, the simplification results in lesser number of IC chips required, thereby resulting in reduced cost and space requirement.

Since the simplified boolean expressions are in SOP or POS form, therefore, only one type (NAND or NOR) of gates are required which again is useful in reducing the number of IC chips required. The propagation delay time is also minimum because of two-level realisation. Because of the availability of a number of logic functions in a logic family which are compatible, it is very convenient to interconnect them to obtain the complete circuit.

The simplification methods discussed are very convenient to use. However, K-maps can conveniently be used upto six variables beyond which it becomes very cumbersome also it can not be mechanised. The Quine-McCluskey method can be used for larger number of variables and can be mechanised.

In addition to minimisation of logical functions for their realisation using NAND/NOR gates, simplification techniques have also been discussed which enables us to design logic circuits using EX-OR and EX-NOR gates, since EX-OR is a very useful and commonly used gate.

Many complex digital functions have been designed which are directly available in IC form. Such circuits have been discussed in Chapter 6 and their use reduces our dependence on simplification procedures. However, discrete gates are employed for interfacing MSI and LSI devices.

A large number of design examples have been included to illustrate the design procedures discussed.

In some combinational circuits, hazard may occur causing glitch in the output. This may result in the malfunctioning of the circuit. The method of detection and elimination of hazard have been discussed. The design of hazard free combinational circuit have been discussed which have higher reliability than the design using minimal components.

GLOSSARY

Adder A logic circuit that can add two numbers.

AND-OR realisation A two-level realisation of logic functions that has AND gates in the first level and an OR gate in the second level.

Canonical POS A POS form of logic expression consisting of only maxterms.

Canonical SOP A SOP form of logic expression consisting of only minterms.

Code converter A logic circuit that converts data from one binary code to another binary code.

Combinational logic The logic in which the outputs at any instant of time are dependent only on the inputs present at that time.

Decoder A logic circuit used to decode a coded binary word.

Don't care A minterm/maxterm in a logic function which may or may not be included.

Dynamic hazard Hazard that causes output to change from 0 to 1 and 1 to 0 number of times.

Encoder A logic circuit that produces coded binary outputs from unencoded inputs.

Essential prime-implicant A term in SOP form in a logic function that must be present in the minimal expression.

Full-adder A logic circuit that accepts two one-bit signals and a carry-in as inputs and produces their sum and carry bits as outputs.

Full-subtactor A digital circuit that accepts two one-bit signals and a borrow-in as inputs and produces their difference and borrow as outputs.

Glitch A momentary (or short duration) pulse.

Half-adder A logic circuit that accepts two single bit inputs and produces their sum bit and carry bit as outputs.

Half-subtractor A logic circuit that accepts two bits as inputs and produces their difference bit and borrow bit as outputs.

Hazard A condition in digital circuits caused by unequal propagation delay of two paths resulting in glitch.

Hazard free circuit A logic circuit free of hazard.

Karnaugh-map (K-map) A mapping technique used to minimise logic expressions.

LED (Light-emitting diode) A diode that emits visible radiant energy when conducting.

Literal A logic variable in either inverted or non-inverted form.

Maxterm A logic term consisting of all the literals in the ORed form in logic function.

Minimisation The process of reducing a logic expression to a minimum form to make it realisable using minimum number of gates with minimum fan-in.

Minterm A logic term consisting of all the literals in the ANDed form.

Multiplexer A logic circuit that selects one of several input lines and connects it to a single output line. Same as data selector.

NAND-NAND realisation A two-level realisation of logic functions that have only NAND gates.

NOR-NOR realisation A two-level realisation of logic functions that have only NOR gates.

OR-AND realisation A two-level realisation of logic functions that has OR gates in the first level and an AND gate in the second level.

Prime-implicant A product term of logic function that does not include any other product term with fewer literals of the same function.

Product-of-sums (POS) A logic expression in the form of ORed terms ANDed together.

Sequential logic Logic circuits whose outputs are determined by the sequence in which input signals are applied.

Static-1 hazard Hazard in which output goes to 0 momentarily, when it sould have remained 1.

Static-0 hazard Hazard in which output goes to 1, momentarily, when it should have remained 0.

Sum-of-products (SOP) A logic expression in the form of ANDed terms ORed together.

Two-level realisation Realisation of logic functions in which signals pass through two gates, such as AND-OR, OR-AND, NAND-NAND, NOR-NOR realisations.

REVIEW QUESTIONS

5.1 A logic variable in complemented or uncomplemented form is known as a _____.

5.2 A combinational circuit does not have _____.

5.3 A canonical SOP logic expression of 4 variables contains each term with _____ literals.

5.4 A logic expression form most suitable for realisation using only NAND gates is _____.

5.5 AND-OR realisation is equivalent to _____ realisation.

5.6 NOR-NOR realisation is preferred over OR-AND realisation because of reduced _____ count.

5.7 A logic term containing literals corresponding to all the variables in ANDed form is known as a _____.

5.8 A logic term containing literals corresponding to all the variables in ORed form is known as a _____.

5.9 A K-map of n-variables contains _____ cells.

5.10 _____ code is used for labelling the cells of K-map.

5.11 A minterm corresponding to don't care condition may have a value _____.

5.12 Number of inputs for a full adder is _____.

5.13 A NOR-NOR realisation is a _____ level realisation.

5.14 A 1 in a cell of K-map can be combined with three other 1's in only one combination. The resulting term of these four 1's is _____.

5.15 In a PI chart of Quine-McCluskey simplification a column containing only one × contributes a term _____.

5.16 A full-adder circuit has both the inputs 1 and carry-in is also 1.Its sum and carry outputs will be _____ and _____ respectively.

5.17 An SOP expression has one term $\bar{A}$, its corresponding input to the second level NAND gate will be _____.

5.18 Hazard in logic circuit can be detected by observing _____.

5.19 _____ type of hazard is possible only in AND-OR realisation.

5.20 _____ type of hazard is possible only in OR-AND realisation.

5.21 Output timing diagram of a logic circuit having static-1 type of hazard will be _____.

5.22 Output timing diagram of a logic circuit having static-0 type of hazard will be _____.

5.23 In an AND-OR logic realisation having hazard, static-0 type of hazard is _____.

5.24 In an OR-AND logic realisation having hazard, static-1 type of hazard is _____.

5.25 Adjacent 1's not included in a common combination of 1's causes _____ in the circuit

5.26 Adjacent 0's not included in a common combination of 0's causes _____ in the circuit

5.27 In general the design of hazardless circuit requires _____ gates than its corresponding circuit having hazard.

PROBLEMS

5.1 A staircase light is controlled by two switches, one at the top of the stairs and another at the bottom of stairs.
(a) Make a truth table for this system.
(b) Write the logic equation in SOP form.
(c) Realise the circuit using AND–OR gates.
(d) Realise the circuit using NAND gates only.

5.2 Given the logic equation

$$f = ABC + B\bar{C}D + \bar{A}BC$$

(a) Make a truth table.
(b) Simplify using K-map.
(c) Realise f using NAND gates only.

5.3 For the truth table given in Table 5.28
(a) Write logic equations for f_1 and f_2 in POS form.
(b) Use K-map for minimisation and obtain the minimised expressions.
(c) Realise these using OR–AND gates.
(d) Realise these using only NOR gates.
(e) Compare the IC packages required in parts (c) and (d).

Table 5.28 ***Truth Table for Problem 5.3***

Inputs				Outputs	
A	*B*	*C*	*D*	f_1	f_2
0	0	0	0	1	0
0	0	0	1	0	0
1	0	0	0	0	1
1	1	0	0	0	1
1	1	1	0	1	1
1	1	1	1	1	1
0	1	1	1	0	0
0	0	1	1	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	0	1	1	1
0	1	1	0	0	0
1	0	0	1	1	0
1	0	1	0	0	1
1	0	1	1	1	0
1	1	0	1	0	0

5.4 (a) Realise Eq. (5.24) using only NAND gates.

(b) Realise Eq. (5.27) using only NOR gates.
(c) Compare the IC packages required in parts (a) and (b). See Table 1.11 for IC packages.

5.5 (a) Realise Eq. (5.25) using minimum number of available NAND gate ICs.
(b) Realise Eq. (5.28) using minimum number of available NOR gate ICs.
(c) Compare the IC packages required in parts (a) and (b).

5.6 Realise Eq. (5.31) using minimum number of available NOR gate chips.

5.7 (a) Make a K-map for the function

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

(b) Express f in canonical SOP form.
(c) Minimise it and realise the minimised expression using NAND gates only.

5.8 Minimise the following functions and realise using minimum number of gates.
(a) $f_1 = \Sigma m\ (0, 3, 5, 6, 9, 10, 12, 15)$
(b) $f_2 = \Sigma m\ (0, 1, 2, 3, 11, 12, 14, 15)$

5.9 Design a BCD-to-Excess-3-code converter using minimum number of NAND gates.

5.10 Design a Excess-3-to-BCD-code converter using minimum number of NAND gates.

5.11 Minimise the following expressions using K-maps and realise using NOR gates only.
(a) $f_1\ (A, B, C, D) = \Pi\ M\ (1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15)$
(b) $f_2\ (A, B, C, D) = \Pi\ M\ (1, 4, 6, 9, 10, 11, 14, 15)$
(c) $f_3\ (A, B, C, D) = \Pi\ M\ (2, 7, 8, 9, 10, 12)$.

5.12 Minimise the following expressions using K-maps and realise with NAND gates.
$f_1\ (A, B, C, D, E) = \Sigma m\ (8, 9, 10, 11, 13, 15, 16, 18, 21, 24, 25, 26, 27, 30, 31)$
$f_2\ (A, B, C, D, E) = \Pi M\ (6, 9, 11, 13, 14, 17, 20, 25, 28, 29, 30)$

5.13 Realise the logic function of Truth Table 5.10 using minimum number of NAND gates
(a) assuming 0's for all the don't-care conditions.
(b) assuming a 0 or 1 for the don't-care conditions leading to a simpler expression.
(c) comment on the effect of don't-care conditions on the hardware requirement.

5.14 Minimise the following logic functions and realise using NAND/NOR gates.
(a) $f_1\ (A, B, C, D) = \Sigma\ m(1, 3, 5, 8, 9, 11, 15) + d(2, 13)$
(b) $f_2\ (A, B, C, D) = \Pi\ M(1, 2, 3, 8, 9, 10, 11, 14) \cdot d(7, 15)$

5.15 Realise the following expressions using EX–OR and EX–NOR gates.
(a) $f_1 = \bar{A}B\bar{C}\bar{D} + A\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + AB\bar{C}D$
(b) $f_2 = \bar{A}B\bar{C} + \bar{A}\bar{B}C + A\bar{C}D + AC\bar{D}$
(c) $f_3 = AB\bar{C}\bar{D} + A\bar{B}\bar{C}D + \bar{A}\bar{B}CD + \bar{A}BC\bar{D}$

5.16 Design a parity generator to generate an odd parity bit for a 4-bit word. Use EX–OR and $\overline{\text{EX–OR}}$ gates.

5.17 Repeat Prob. 5.16 for an even parity bit.

5.18 Simplify the following logic expressions and realise using NAND/NOR gates.
(a) $f_1\ (A, B, C, D, E, F) = \Sigma m(6, 9, 13, 18, 19, 25, 27, 29, 41, 45, 57, 61)$
(b) $f_2\ (A, B, C, D, E, F) = \Pi M(4, 5, 6, 7, 8, 12, 13, 16, 17, 18, 19, 21, 22, 25, 28, 32, 35, 37, 38, 39, 40)$.

5.19 Design a full-adder circuit using two half-adders. Use
(a) Block diagram of half-adder.
(b) Circuit of Fig. 5.19 for half-adder.

5.20 For the full-adder circuit of Prob. 5.19(b) determine the propagation delay time for Sum output (S_n) and carry output (C_n), assuming the propagation delay time of gates as

EX-OR	20 ns
AND	10 ns
OR	10 ns

and presence of data inputs A_n, B_n and carry-in (C_{n-1}) simultaneously.

5.21 Realise the logic function in SOP form using Quine-McCluskey method

$$f(A, B, C, D) = \Pi M\,(2, 7, 8, 9, 10, 12)$$

5.22 Minimise the logic function using Quine-McCluskey method

$$f(A, B, C, D) = \Sigma m(1, 3, 5, 8, 9, 11, 15) + d(2, 13)$$

5.23 Minimise the logic function using Quine-McCluskey method

$$f(A, B, C, D, E) = \Sigma m\,(8, 9,10, 11, 13, 15, 16, 18, 21, 24, 25, 26, 27, 30, 31)$$

5.24 Will there be any hazard in the K-maps of Fig. 5.20?.

5.25 In each of the K-maps of Fig. 5.25, determine whether hazard occurs or not. In case hazard occurs, design the hazardless circuit.

5.26 For the logic function

$$f(A, B, C, D) = \Pi M(2, 3, 4, 5, 6, 7, 8, 11, 12)$$

(a) Design logic circuit with minimum number of NOR gates only.

(b) Examine whether hazard is present in the circuit obtained in (a). If yes, what is the type of hazard present?

(c) Design its hazardless circuit.

5.27 Repeat Prob. 5.26(a), (b), and (c) for NAND only realisation.