CHAPTER 2

COMBINATIONAL CIRCUITS—SELECTED TOPICS

In Chapter 1, we discussed procedures for obtaining minimal 2-level realizations of combinational functions. In this chapter, we shall extend these procedures to multiple output circuits for realizing several combinational functions and also multiple level realizations of combinational functions. Several important classes of combinational functions are then considered: symmetric functions, unate functions, and threshold functions. Finally, we consider sets of logic functions that can be used to realize arbitrary combinational functions.

2.1 MULTIPLE OUTPUT COMBINATIONAL CIRCUITS

When several combinational functions are to be realized, it is often possible to obtain a multiple output circuit requiring fewer gates than separate circuits realizing the individual functions. This is accomplished by the sharing of logic among the parts of the circuit realizing the different functions. The sharing of logic implies that the output of a gate may be connected to the inputs of more than one gate. The number of gates connected to the output of a gate is called its fanout index. If the fanout index is restricted to 1, the problem of obtaining a minimal 2-level circuit for realizing a set of functions is the same as that of realizing each individual function with a minimal 2-level circuit. (Note that even for realizing a single function, a fanout index of greater than 1 may be necessary from the inputs.)

In multiple output 2-level AND-OR circuits, gate sharing is possible if and only if two or more functions have a common implicant. The concept of prime implicants must also be generalized to permit sharing.
of logic. This is demonstrated by the pair of combinational functions of Figure 2.1. The 2-level AND-OR circuit realization which minimizes the total number of gate inputs for this pair of functions, is shown in Figure 2.2. The term \( x_2 x_3 x_4 \) is not a prime implicant of either \( f_1 \) or \( f_2 \), as we can easily see from the Karnaugh maps of Figure 2.1. However, it is a prime implicant of the product \( f_1 \cdot f_2 \) of these two functions, as seen from the Karnaugh map of Figure 2.3. Thus it is necessary to generalize the concept of prime implicant to multiple output prime implicant [9].

Let \( F = \{ f_1, f_2, \ldots, f_r \} \) be a set of combinational functions and let \( F' \) be a subset of \( F \). A product term \( P \) is a multiple output prime implicant of \( F' \) if \( P \) is a prime implicant of \( G = \prod_{f_i \in F'} f_i \) (the product of all functions in \( F' \)). Since \( F' \) can contain a single function \( f_i \), all prime implicants of single functions are also multiple output prime implicants. The following theorem shows that minimal 2-level AND-OR realizations of sets of combinational functions can be obtained in which the output of each AND gate is a multiple output prime implicant.

**Theorem 2.1** Given a set of combinational functions \( F = \{ f_1, f_2, \ldots, f_r \} \) there exists a minimal 2-level AND-OR realization of \( F \) in which the output of every AND gate is a multiple output prime implicant.

**Proof:** Consider a minimal 2-level AND-OR realization of a set of functions \( F = \{ f_1, f_2, \ldots, f_r \} \) and suppose the output of some AND gate \( A \) is a product term \( P \) which is not a multiple output prime implicant.

Assume the output of \( A \) is an input to the OR gates realizing the subset of combinational functions \( F' \subseteq F \). Then since \( P \) is not a multiple output prime implicant, there must exist a multiple output prime implicant \( P' \) of the function \( G = \prod_{f_i \in F'} f_i \) such that \( P' \) covers every 1-point of each \( f_i \in F' \) covered by \( P \). Every point covered by \( P' \) is a 1-point or a don't care for each \( f_i \in F' \), and the number of literals in \( P' \) does not exceed the number of literals in \( P \). Hence, we can replace gate \( A \) by a gate \( A' \) which realizes the multiple output prime implicant \( P' \) and the circuit remains a minimal 2-level AND-OR realization of the set of functions.
F. This can be repeated until all such AND gates are replaced and the output of every AND gate is a multiple output prime implicant.

Thus, the problem of generating a minimal 2-level AND-OR realization for a set of combinational functions can be solved by the following general procedure.

Procedure 2.1
1. Generate the set of all multiple output prime implicants. (Recall that prime implicants of an individual function $f_i$ are also multiple output prime implicants.)
2. Select a minimal cost set of multiple output prime implicants to realize the set of combinational functions. If a multiple output prime implicant of $F' \subseteq F$ is selected, it is used for realizing each $f_i \in F'$.

2.1.1 Generation of Multiple Output Prime Implicants

The tabular method for generation of prime implicants of single combinational functions (Procedure 1.2) can be extended to generate multiple output prime implicants for a set of functions $F$ without explicitly generating the products of all subsets of $F$. A function identifier is associated with each minterm to indicate whether it is a 1-point or a 0-point for each of the functions to be realized. As terms are combined, this information is used to determine subsets of $F$ for which the resulting term is a multiple output prime implicant. For example, if $f_1(x_1, x_2, x_3) = \bar{x}_1 \bar{x}_2 x_3 + \bar{x}_1 x_2 x_3$ and $f_2 = \bar{x}_1 \bar{x}_2 x_3 + x_1 x_2 x_3$, then $\bar{x}_1 \bar{x}_2$ is not a multiple output prime implicant of $\{f_1, f_2\}$, although it is a prime implicant of $f_1$, since $\bar{x}_1 \bar{x}_2 \bar{x}_3$ is not a 1-point of $f_2$.

Procedure 2.2
1. Form a table listing all 1-points and don’t care points for the entire set of functions $\{f_1, f_2, ... f_r\}$. Partition this list into sets $S_0, S_1$,..., $S_v$, where set $S_i$ consists of all such points with $i$ variables equal to 1. Add a set of $r$ columns to the table headed by $f_1$, $f_2$, ...$f_r$. These columns will be used to indicate for each row in the table, functions for which it is a 1-point (or a don’t care point) and for which it is a 0-point. Associate with each point $(x_i)$ in the table a function identifier in these $r$ columns with a 1 in column $f_i$ if $f_i(x_i) \neq 0$ and a 0 in column $f_i$ if $f_i(x_i) = 0$.

*Using this labeling, product terms may be generated which do not cover any 1-points. To avoid this a 3-valued function identifier must be used.

2. Rows of this table will be merged to form sets $S_i'$ in the same manner as for single combinational functions (Procedure 1.2). When rows $x_i$ and $x_j$ are merged the function identifier of the resultant row has a 1 in column $f_k$, if and only if $f_k(x_i) \neq 0$ and $f_k(x_j) \neq 0$. Otherwise the entry in column $f_k$ is 0. If all $r$ entries of the function identifier are 0 the associated row merger does not represent a multiple output prime implicant and the row can be removed from the table. If the resultant function identifier is identical to the function identifier of $x_k$ (and/or $x_j$) then row $x_k$ (and/or $x_j$) is not a multiple output prime implicant. This can be denoted by flagging $(\checkmark)$ $x_k$ (and/or $x_j$).

3. Continue combining rows in this manner. When no more rows can be combined, those rows without checks are multiple output prime implicants and the associated set of functions is indicated by the function identifiers.

Procedure 2.2 may generate terms which do not cover any 1-points of any function, since the function identifier was made equal to 1 for don’t care points as well as 1-points of a function. These terms will, of course, be discarded while obtaining a minimal covering set using Procedure 2.1. Instead, 3-valued function identifiers can be used for 0-points, 1-points and unspecified entries respectively and the procedure generalized to enable identification of implicants which do not cover any 1-points.

Example 2.1 Consider the functions $f_1$ and $f_2$ of Figure 2.1. The table generated in Step (1) of Procedure 2.2 is shown in Figure 2.4(a). Figure 2.4(b) shows the results of combining the rows of Figure 2.4(a) and Figure 2.4(c) is obtained by combining rows of Figure 2.4(b). The multiple output prime implicants and the associated functions (shown in parentheses) are:

\[
\bar{x}_1 \bar{x}_3 x_4(f_1), \bar{x}_1 \bar{x}_2 x_3(f_2), x_1 x_2 \bar{x}_4(f_1).
\]

\[
x_1 \bar{x}_3 x_4(f_2), x_2 x_3 x_4(f_1, f_2), x_2 x_3(f_1), x_2 x_4(f_2).
\]

Procedure 2.2 will generate all multiple output prime implicants of a set of functions $F = \{f_1, f_2, ... f_r\}$ with the following exception. A multiple output prime implicant of a subset $F'$ of $F$ may also be a multiple output prime implicant of subset $F'' \subseteq F'$. Procedure 2.2 will only have one row corresponding to this prime implicant and the associated function identifier will correspond to $F'$. In general, if $P_i$ is a multiple
output prime implicant of \( F' \subseteq F \), it is also a multiple output prime implicant of \( F'' \subseteq F' \), unless there is another prime implicant \( P_i \) which covers all 1-points of functions \( f_i \in F'' \) covered by \( P_i \), and has fewer literals. Thus, we can determine whether a multiple output prime implicant of \( F' \) is also a multiple output prime implicant of a subset of \( F' \) by examining all multiple output prime implicants of all sets \( F^* \subseteq F' \).

**Example 2.2** For the pair of combinational functions shown in Figure 2.5(a), Procedure 2.2 would lead to the tables shown in Figure 2.5(b). The resultant prime implicants are \( x_2 \bar{x}_3 \) for \( f_2 \), \( x_1 x_2 \) for \( f_1 \cdot f_2 \) and \( x_1 \bar{x}_3 \) for \( f_2 \). However, \( x_1 x_2 \) is also a multiple output prime implicant of \( f_1 \) alone and \( f_2 \) alone.

The above shortcoming of the procedure results from the loss of information due to the method of assigning function identifiers. For example, if a row is a 1-point of two functions \( f_1 \) and \( f_2 \), we assign the function identifier to be 11. However, the fact that the row also

![Figure 2.4](image1.png)

**Figure 2.4** Tabular generation of multiple output prime implicants

![Figure 2.5](image2.png)

**Figure 2.5** (a) Two combinational functions (b) multiple output prime implicant generation table
represents a 1-point of $f_i$ and $f_j$ taken individually is not indicated by the function identifier. This difficulty can be overcome by repeating the row with all possible function identifiers. Thus, for the example just mentioned, three rows corresponding to the same input combination with function identifiers 01, 10, and 11 will be necessary. However, this modification is likely to become very unwieldy, due to the increase in the number of rows.

Multiple output prime implicants can also be determined from the Karnaugh maps of the individual functions and the products of all subsets of this set of functions. For the functions shown in Figure 2.1, the complete set of multiple output prime implicants are the prime implicants $x_2x_3$, $x_1x_3x_4$ and $x_1x_3x_4$ of $f_1$, the prime implicants $x_2x_3, x_1x_3x_4$ and $x_1x_3x_4$ of $f_2$, and the prime implicant $x_2x_3x_4$ of $f_1f_2$ (Figure 2.3).

In defining the product, $f_1f_2$, of two functions $f_1$ and $f_2$ that may not be completely specified, $f_1f_2$ is equal to 1 at points where both $f_1$ and $f_2$ are equal to 1, $f_1f_2$ is 0 at points where $f_1$ or $f_2$ is 0, and $f_1f_2$ is unspecified at points where neither $f_1$ nor $f_2$ is 0 and $f_1$ and/or $f_2$ is unspecified.

2.1.2 Selection of a Minimal Covering Set of Multiple Output Prime Implicants

The selection of a minimal covering set of multiple output prime implicants can be formulated as a prime implicant table covering problem. The table has a row for each multiple output prime implicant and a column for each 1-point of each function $f_j$. If column $j$ represents a 1-point of $f_j$, the entry in row $i$ column $j$ is 1 if the 1-point represented by column $j$ is covered by the multiple output prime implicant represented by row $i$. As in the case of single function minimization the cost associated with a prime implicant depends on the optimization goal. To minimize the total number of gates, the cost associated with prime implicant $P_i$ containing $q_i$ literals is 1 if $q_i > 1$ and 0 if $q_i = 1$. If the optimization goal is to minimize the total number of gate inputs required, then if an implicant $P_i$, with $q_i$ literals is a multiple output prime implicant of a set of $k$ functions, the cost of $P_i$ is $q_i + k$ if $q_i > 1$ and the cost is $k$ if $q_i = 1$.

As indicated previously, a multiple output prime implicant $P_i$ for the subset $F' \subseteq F$ may also be a multiple output prime implicant for $F'' \subseteq F'$. In this case, the prime implicant $P_i$ must be included in the prime implicant covering table more than once, once for each product of functions covered by $P_i$. For example, if $P_i$ is a prime implicant of $f_1f_2$ and also of $f_1$ alone, but not $f_1$ alone, there will be two rows corresponding to $P_i$ in the prime implicant table. Two different costs will be associated with the two rows. As a prime implicant of $f_1f_2$, the cost (for minimizing the number of gate inputs) will be $q_i + 2$, where $q_i$ is the number of literals in $P_i$, and it will be indicated as covering the appropriate 1-points of both $f_1$ and $f_2$. The cost associated with it as a prime implicant of $f_1$ alone will be $q_i + 1$, and it will cover only the appropriate 1-points of $f_1$. The table can frequently be simplified by the same type of dominance relations as for the single function case. In addition, we can sometimes take advantage of the fact that the same implicant appears several times in the table. Suppose $P_i$ appears as a prime implicant of $f_1$ and of $f_2f_3$, and in order to cover all 1-points of $f_1$ one of these rows must be selected. We can select $P_i$ as a prime implicant of $f_1$ and reduce the table. In reducing the table the cost of the row corresponding to $P_i$ as a multiple output prime implicant of $f_1f_2$ is reduced to 1 since sharing of a term which has already been selected to be generated requires only one additional gate input. If this row is eventually selected as part of the minimal cover, it replaces the use of $P_i$ as a prime implicant of $f_1$ alone. (In effect we select $P_i$ to be generated by an AND gate but defer the decision as to whether $P_i$ will be used only to generate $f_1$ or to generate both $f_1$ and $f_2$).

Example 2.3 For the set of functions $\{f_1, f_2, f_3\}$ shown in Figure 2.6(a) the set of multiple output prime implicants are as follows:

$$x_1x_4(f_1), x_2x_3x_4(f_2), x_2x_3x_4(f_1f_2), \bar{x}_2x_3x_4(f_3), \bar{x}_2x_3x_4(f_1f_2).$$

The prime implicant covering table is shown in Figure 2.6(b). Notice that there are two rows corresponding to $x_2x_3x_4$ and $\bar{x}_2x_3x_4$.

The implicant $x_2x_3x_4$ is essential to cover the 1-points of $f_2$. We select row 2 of the table and reduce the cost of row 3 to 1. Similarly row 4 is selected to cover 1-points of $f_3$ and the cost of row 5 is reduced to 1. In the reduced table of Figure 2.6(c), the minimal cover consists of prime implicants $x_2x_3x_4$ and $\bar{x}_2x_3x_4$. Hence, these terms are shared between $f_1$ and $f_2$, and $f_1$ and $f_3$, respectively, as shown in Figure 2.6(d).

As for the single function case, minimal covering sets can sometimes be obtained for a set of functions $F = \{f_1, f_2, \ldots, f_s\}$ from the Karnaugh maps of the individual functions and the maps of the products of all subsets of these functions. This is done by selecting essential multiple output prime implicants, (a multiple output prime implicant is essential with respect to a 1-point $m$, of a function $f \in F$, if it is the only multiple
output prime implicant which covers \( m_i \) of \( f_j \) or good multiple output prime implicants. A multiple output prime implicant \( P_i \) is good with respect to a 1-point \( m_i \) of a function \( f_k \in F \) if for any other multiple output prime implicant \( P_j \) which covers \( m_i \) of \( f_k \):

1. \( P_i \) covers all previously uncovered 1-points of all functions in \( F \) which are covered by \( P_j \), and
2. the cost of \( P_i \) is the cost of \( P_j \).

In applying this rule care must be exercised since a previously selected prime implicant which covers \( m_i \) of \( f_k \) has cost 1. Once a multiple output prime implicant is selected to cover 1-points of a set of functions \( F' \), all 1-points of all \( f_i \in F' \) which are covered by it are changed to don't care entries.

The determination of essential multiple output prime implicants is simplified by the observation that such terms must be essential for some individual function, \( f_i \). Hence, the essential prime implicants of the individual functions should be examined. Also, if \( f_i \) has a 1-point which is a 0-point of every other function in \( F \), then the term covering this 1-point cannot be shared and can therefore be determined from the map of \( f_i \) alone.

**Example 2.4** For the set of functions \( F = \{ f_1, f_2 \} \), the Karnaugh maps of the individual functions and their product are shown in Figure 2.7.
First, we observe that the prime implicant $x_2 x_4$ is essential for covering the 1-point $m_{10}$ of $f_1$. Similarly, the prime implicant $x_1 x_2 x_4$ is essential for covering the 1-point $m_{12}$ of $f_2$. Selecting these prime implicants and changing the 1-points covered by them to don't cares, we obtain the maps of Figure 2.8. The costs of these prime implicants are reduced to 1. The prime implicant $x_1 x_2 x_3$ is a good prime implicant for covering the 1-point $m_{12}$ of $f_2$ and also $f_1$ and is therefore selected and shared by both functions. The previously selected prime implicant $x_1 x_2 x_4$ is good with respect to $m_{12}$ and $m_{14}$ of $f_1$ and is used to cover these points thus completing the solution. The minimal 2-level realization of the two functions is given by:

\[ f_1 = x_2 x_4 + \bar{x}_1 x_2 x_3 + x_1 x_3 \bar{x}_4 \]
\[ f_2 = \bar{x}_1 x_2 x_3 + x_1 x_2 \bar{x}_4 \]

The underlined terms are shared by both functions and the total number of gate inputs is 13.

2.2 MULTIPLE LEVEL COMBINATIONAL CIRCUITS

So far we have restricted our designs to be 2-level circuits. There are many practical reasons why we might wish to design circuits with more than two levels. A striking example of the tradeoff between hardware cost and speed (i.e. the number of levels) is the parity check function $f(x_1, x_2, ..., x_n) = x_1 \oplus x_2 \oplus ... \oplus x_n$. This function has the value 1 if and only if an odd number of input variables are 1. The 2-level realization requires $2^{n-1}$ AND gates each with $n$ inputs and one OR gate with...
are usually restricted to be within certain maximum values. In order to satisfy these constraints, it is frequently necessary to realize circuits with more than two levels of logic. One procedure used in the generation of such circuits is called factorization. Consider the function \( F = x_1x_2x_3\bar{x}_4 + x_1x_2x_3x_4 \). The two terms have the common factor \( x_1x_2 \) and hence \( F \) can be rewritten as \( x_1x_2(x_3\bar{x}_4 + x_3x_4) \). The respective realizations are shown in Figure 2.10. The maximum gate fan-in has been reduced from four to two by factoring, while the number of levels of logic has increased from two to three (consequently increasing the circuit delay).

Any arbitrary combinational circuit can be converted to a logically equivalent circuit satisfying fan-in and fan-out constraints in a straightforward manner. For example, an \( n \)-input AND gate can be realized by a tree of 2-input AND gates (similar to the tree of Figure 2.9). The number of levels in the tree will be \( \lceil \log_2 n \rceil \). Similarly a fan-out of \( n \) can be realized by a tree with \( \lceil \log_2 n \rceil \) levels where each gate in the tree has one input and a fanout index of 2.

Despite the importance of multiple level realizations, there does not exist any computationally efficient general procedure for obtaining optimal realizations of functions with more than two levels. We shall now consider a design procedure which leads to multiple level realizations via decomposition of combinational functions.

### 2.3 DECOMPOSITION OF COMBINATIONAL FUNCTIONS

A function \( f(x_1, x_2, \ldots, x_n) \) is decomposable if \( f \) can be realized as a composition of functions each of which has fewer than \( n \) variables.
That is, \( f(x_1, x_2, \ldots, x_n) = F(g_1, g_2, \ldots, g_k) \), as shown in Figure 2.11. For example, the function \( f(x_1, x_2, x_3, x_4) = x_1 x_3 + (x_2 + x_4) x_4 \) can be decomposed as shown in Figure 2.12.

Decompositions of combinational functions \([1, 3]\) can be classified as **disjunctive** or **nondisjunctive**. In a disjunctive decomposition the external input variables to any component subfunction \( g_i \) are independent of the input variables of any other component subfunction \( g_j \). Let \( X = \{x_1, x_2, \ldots, x_n\} \) be the set of input variables. Thus, if \( f(x) = f(g_1(A_1), g_2(A_2), \ldots, g_k(A_k)) \), where \( A_1, A_2, \ldots, A_k \) are sets of input variables such that \( A_i \cap A_j = \emptyset \) for any \( i, j \) and \( \bigcup_{i=1}^{k} A_i = A_1 \cup A_2 \cup \ldots \cup A_k = X \), the decomposition is disjunctive. If, for some \( i, j \), \( 1 \leq i, j \leq n, A_i \cap A_j \neq \emptyset \), the decomposition is nondisjunctive.

### 2.3.1 Simple Disjunctive Decompositions

A simple disjunctive decomposition has only a single subfunction \( g_1 \) and \( f(x) = f(x_1, x_2, \ldots, x_n) = f(g_1(A_1), A_2) \) where \( A_1 \cap A_2 = \emptyset \), \( A_1 \cup A_2 = X \).

**Example 2.5** Let \( f(x) = f(x_1, x_2, x_3, x_4) = x_1 x_2 x_3 + x_1 x_3 x_4 \). Figure 2.13(a) shows a nondisjunctive decomposition of \( f(x) \) and Figure 2.13(b) shows a simple disjunctive decomposition of \( f(x) \) where \( A_1 = \{x_1, x_2, x_3\} \), \( A_2 = \{x_4\} \) and \( g_1(A_1) = x_1 x_2 + x_4 \). \( F(g_1(A_1), A_2) = x_3 \cdot g_1(x_1, x_2, x_4) \).

In order to determine properties of combinational functions which permit simple disjunctive decompositions of the form \( F(g(A_1), A_2) \), let us consider a Karnaugh map of a function \( f \) arranged as shown in Figure 2.14(a), where the rows of the map correspond to the variables in the set \( A_2 \) and the columns to the variables in the set \( A_1 \). For \( A_1 = \{x_1, x_2, x_3\} \) and \( A_2 = \{x_4, x_5\} \) the map is shown in Figure 2.14(b). The Karnaugh map arranged in this way will be called a **decomposition map** for the variable sets \( (A_1, A_2) \).

If \( A_1 \) has \( q \) input variables (denoted by \( |A_1| = q \)) and \( A_2 \) has \( r \) input variables (\( |A_2| = r \)) then \( |X| = n = r + q \). Each row of the decomposition map represents \( 2^q \) points of the \( n \)-cube and each column represents a set of \( 2^r \) points. The set of points represented by any row all have the same value of the variable set \( A_2 \) and the set of points represented by any column all have the same value of the variable set \( A_1 \). Now suppose a function \( f(x) \) which has a simple disjunctive decomposition \( f(x) = F(g_1(A_1), A_2) \) is plotted on a decomposition map for the variable sets \( (A_1, A_2) \). The following theorem gives a necessary and sufficient condition for this map to define a simple disjunctive decomposition in the variable sets \( (A_1, A_2) \).

**Theorem 2.2** A completely specified combinational function \( f(x_1, x_2, \ldots, x_n) \) has a simple disjunctive decomposition of the form \( f(x) = \)
of 0's and 1's. Therefore, \( g(C_1) \neq g(C_2), g(C_2) \neq g(C_3), g(C_1) \neq g(C_3) \). But then \( g \) cannot be a binary function. Contradiction.

**Sufficiency:** Suppose there are only two different columns \( C_1, C_2 \). Let \( g(A_i) \) be defined as follows:
- \( g(A_i) = 1 \) for all columns identical to \( C_1 \)
- \( g(A_i) = 0 \) for all columns identical to \( C_2 \) (or vice versa).

The map can then be reduced to a map with \( q + 1 \) variables by combining all the columns identical to \( C_1 \) as the \( g = 1 \) map half and the columns identical to \( C_2 \) as the \( g = 0 \) map half. The resulting map can be used to define \( F(g(A_1), A_2) \).

**Example 2.6** Consider the decomposition map of Figure 2.15 for a function \( f(x_1,x_2,x_3,x_4,x_5) \). Since there are only two distinct column patterns, there exists a simple disjunctive decomposition of \( f \) of the form \( f = F(g(x_1,x_2,x_3),x_4,x_5) \).

![Figure 2.14](image1)

**Figure 2.14** Map for decompositions of the form \( F(g(A_1), A_2) \).

![Figure 2.15](image2)

**Figure 2.15** Decomposition map for Example 2.6

If \( g = 1 \) for those columns with pattern 1001 from top to bottom, then \( g = \bar{x}_1 \bar{x}_2 x_3 + x_1 x_2 \). The map of Figure 2.16(a) is obtained by combining identical columns of Figure 2.15. The decomposition \( F = \bar{g} x_4 + \bar{x}_4 x_5 + g x_5 \) is obtained from this map. The function \( g \) is determined from the map of Figure 2.16(b) in which \( g = 1 \) in columns corresponding to columns of Figure 2.15 with pattern 1001 and \( g = 0 \) for columns with pattern 1110. Thus, \( g = \bar{x}_1 \bar{x}_2 x_3 + x_1 x_2 \). The decomposed realization is shown in Figure 2.17. \( \square \)
to the decomposition $f(x_1, x_2, \ldots, x_n) = x_i f(x_1, x_2, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) + \bar{x}_i f(x_1, x_2, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$ which is known as Shannon's expansion theorem. This is always possible and we shall not consider it further.

Figure 2.16 Simplified decomposition map of Example 2.6

Figure 2.17 Decomposed realization of $f(x_1, x_2, x_3, x_4, x_5)$

If all the columns of a decomposition map are identical, it follows that the function is independent of the variables in the set $A_1$. Thus, the decomposition is trivial. If the decomposition table has two distinct columns, but the column patterns consist of all 0's or all 1's, then the function is independent of the variables in the set $A_2$.

The decomposition maps on the sets of variables $(A_1, A_2)$ can also be used to determine decompositions of the form $F(A_1, g(A_2))$. Such a decomposition exists if the decomposition map has two distinct rows. Thus, it is not necessary to construct the decomposition maps for both $(A_1, A_2)$ and $(A_2, A_1)$. With this interpretation there are $2^{n-1} - 1$ possible decomposition maps for an $n$-variable function. Figure 2.18 shows the decomposition maps for $n = 3$ and $n = 4$. In order to determine a simple disjunctive decomposition of an $n$-variable function, if it exists, Theorem 2.2 can be applied exhaustively to each of these $2^{n-1} - 1$ maps.

A special case of Theorem 2.2 is obtained when $A_1 = \{x_i\}$ and $A_2 = X - A_1$. The decomposition map will have only two columns leading

Figure 2.18 Decomposition maps for (a) 3-variable (b) 4-variable functions
Example 2.7 Consider the function \( f(x_1, x_2, x_3) = \bar{x}_1 \bar{x}_2 \bar{x}_3 + x_1 x_3 + x_2 x_3 \). This function is plotted in the three possible decomposition maps in Figure 2.19. Figure 2.19(a) does not define a simple disjunctive decomposition. Figure 2.19(c) defines the decomposition \( F(g_1(x_1, x_3), x_2) \) where \( g_1(x_1, x_2) = x_1 + x_2 \) and \( F(g_2(x_3)) = x_3 \bar{g}_2 \). Figure 2.19(b) violates Theorem 2.2 and hence does not define a simple disjunctive decomposition.

For incompletely specified functions a decomposition map will have unspecified entries and hence Theorem 2.2 must be modified to be applicable to this case. We define two columns (rows) of a decomposition map to be compatible (denoted by \( \sim \)) if the don't care entries can be filled in such a way that the columns (rows) become identical. If a decomposition map contains three columns and three rows of which all pairs are incompatible (denoted by \( \neq \)), then a simple disjunctive decomposition of the function cannot be obtained with variable partitioning defined by the decomposition map. A simple disjunctive decomposition can be obtained if the set of rows (or columns) can be partitioned into two disjoint subsets such that all elements of each subset are pairwise compatible. A systematic procedure for determining the existence of such a partition will be presented in Chapter 3 (Procedure 3.2).
Example 2.8 From the decomposition map of \( f(x_1, x_2, x_3, x_4) \) shown in Figure 2.20, we see that \( C_1 \neq C_2 \), \( C_1 \neq C_3 \), and \( C_1 \neq C_4 \). Therefore, the function \( f \) has no simple disjunctive decomposition of the form \( F(g(x_3, x_4), x_1, x_2) \). However, the don't care entries may be specified

\[
\begin{array}{cccc}
  & C_1 & C_2 & C_3 & C_4 \\
 r_1 & 0 & 1 & - & 0 \\
 r_3 & 1 & - & 1 & 0 \\
 r_4 & 1 & 0 & 1 & 0 \\
 x_1 & - & - & 0 & 0 \\
 x_2 & & & & \\
 x_3 & & & & \\
 x_4 & & & & \\
\end{array}
\]

For large \( n \), the determination of a simple disjunctive decomposition may require a considerable amount of computation, since the process involves an exhaustive search.

2.3.2 Complex Disjunctive Decompositions

A disjunctive decomposition which is not simple (i.e., has more than one subfunction) will be called a complex disjunctive decomposition \([1,3,7]\).

Complex disjunctive decompositions can be shown to be extensions of simple disjunctive decompositions. An iterative disjunctive decomposition is of the general form

\[
f(x) = F(g_m(\ldots g_2(g_1(A_1), A_2), A_3), \ldots, A_m),
\]

and is based on a repeated decomposition of an original simple disjunctive decomposition on the variables \( A_1, A_2, \ldots, A_{m-1}, A_m \).

A multiple disjunctive decomposition is of the form \( f(x) = F[g_1(A_1), g_2(A_2), \ldots, g_m(A_m)] \). The relation between a complex disjunctive decomposition and simple disjunctive decompositions is shown by the following theorems.

**Theorem 2.3** For a given combinational function \( f(x) \) there exists a multiple disjunctive decomposition \( f(x) = F[g_1(A_1), g_2(A_2), \ldots, g_m(A_m)] \) if and only if there exist simple disjunctive decompositions of the form

\[
\begin{align*}
  f(x) &= F_1[g_1(A_1), A_2, A_3, \ldots, A_m] \\
  f(x) &= F_2[g_2(A_2), A_1, A_3, \ldots, A_m] \\
  f(x) &= F_m[g_m(A_m), A_1, A_2, \ldots, A_{m-1}]
\end{align*}
\]

**Proof** See Reference [3].

**Theorem 2.4** For a given combinational function \( f(x) \) there exists an iterative disjunctive decomposition \( f(x) = F(g_m(\ldots g_3(g_2(g_1(A_1), A_2), A_3), \ldots, A_m) \) if and only if there exists a simple disjunctive de-
composition of the form

\[ f(x) = F_m(g_m'(A_1, A_2, ..., A_{m-1}, A_m)) \]

and a simple disjunctive decomposition of \( g_m' \),

\[ g_m' = F_{m-1}(g_{m-1}'(A_1, A_2, ..., A_{m-2}, A_{m-1})) \]

and a simple disjunctive decomposition of \( g_{m-1}' \), etc.

**Proof:** Exercise. \( \square \)

**Example 2.9** Let \( f = x_1 \bar{x}_2 \bar{x}_3 \bar{x}_4 + \bar{x}_1 x_2 \bar{x}_3 \bar{x}_4 + x_1 x_4 + \bar{x}_1 \bar{x}_2 x_4 + x_1 x_2 x_4 \). From the decomposition map of Figure 2.23(a) we obtain the simple disjunctive decomposition

\[ f = F(g_1(x_1, x_2, x_3, x_4)) \]

where

\[ g_1 = x_3 + x_1 x_2 + \bar{x}_1 \bar{x}_2 \]

These two simple disjunctive decompositions together yield the iterative disjunctive decomposition \( f = F(F_1(g_2(x_1, x_2, x_3, x_4))) \), shown in Figure 2.24. \( \square \)

**Figure 2.23** Decomposition maps of (a) a combinational function and (b) a subfunction of the decomposition

**Figure 2.24** Realization of an iterative disjunctive decomposition

**Example 2.10** Let \( f = x_1 \bar{x}_2 x_4 + x_1 \bar{x}_2 x_4 + \bar{x}_1 x_2 x_3 + \bar{x}_1 x_2 x_4 \). The decomposition map of Figure 2.25 has two distinct rows and two distinct
columns and therefore yields the following two decompositions of the function $f$.

![Diagram of Decomposition Map]

Figure 2.25 Decomposition map of function of Example 2.10

\[ f = F_i(g_i(x_1, x_2), x_3, x_4) \]

where \( g_i(x_1, x_2) = x_1 \bar{x}_2 + \bar{x}_1 x_2 \)

\[ F_i(g_i(x_1, x_2), x_3, x_4) = g_1 x_3 + g_1 x_4 \]

\[ f = F_i(g_2(x_3, x_4), x_1, x_2) \]

where \( g_2 = x_3 + x_4 \)

and \( F_2 = g_2 x_1 \bar{x}_2 + g_2 \bar{x}_1 x_2 \).

From Theorem 2.3 there exists a multiple disjunctive decomposition

\[ f = F_i(g_1(x_1, x_2), g_2(x_3, x_4)) \]

The function $F$ can be determined by plotting $f$ as a function of $g_i$ on a Karnaugh map. In this example, $F = g_1 \cdot g_2$.

2.3.3 Nondisjunctive Decompositions

In a nondisjunctive decomposition, the input variables of different component subfunctions have some common elements. Thus, if $f(x) = F(g_1(A_1), g_2(A_2), \ldots, g_k(A_k))$ where $\bigcup_{i=1}^k A_i = X$ and there exist $i$ and $j$, $i \neq j$, such that $A_i \cap A_j \neq \emptyset$, this represents a nondisjunctive decomposition. A combinational function $f(x)$ has a simple nondisjunctive decomposition if $f(x) = F(g(A_1), A_2)$ where $A_1 \cup A_2 = X$ and $A_1 \cap A_2 = A_{12} \neq \emptyset$.

Let $B_1 = A_1 - A_{12}$ and $B_2 = A_2 - A_{12}$. If $|A_{12}| = p_1$, $|B_1| = p_1$, and $|B_2| = p_2$, then $p_1 + p_2 = n = |X|$. For each of the $2^n$ values of $A_{12}$ we can define a decomposition map with $2^{p_1}$ rows and $2^{p_2}$ columns. In order for $f(x)$ to have a simple nondisjunctive decomposition, each of these maps must define a simple disjunctive decomposition.

**Theorem 2.5** A combinational function $f(x)$ has a simple nondisjunctive decomposition of the form $f(x) = F(g_1(A_1), A_2)$ if and only if each of the $2^n$ maps defined for a fixed value of $A_{12} = A_1 \cap A_2$ with rows corresponding to $A_2 = A_{12}$ and columns corresponding to $A_1 = A_{12}$ has at most two distinct rows or columns.

**Proof:** See Reference [3].

**Example 2.11** Let $f = \bar{x}_2 x_2 \bar{x}_4 + \bar{x}_2 x_3 x_4 + x_2 \bar{x}_3 \bar{x}_4 + x_2 \bar{x}_4 \bar{x}_5$. In order to obtain a decomposition with $A_1 = \{x_1, x_2, x_3\}$ and $A_2 = \{x_2, x_4, x_5\}$ we construct decomposition maps for all possible combinations of inputs in the set $A_{12} = A_1 \cap A_2 = \{x_2\}$. Decomposition maps for $x_2 = 0$ and $x_2 = 1$ are shown in Figure 2.26. Let us denote $f(x_1, 0, x_3, x_4, x_5)$ as $f_0$ and $f(x_1, 1, x_3, x_4, x_5)$ as $f_1$. From the decomposition maps we obtain the following decompositions of $f_0$ and $f_1$:

\[ f_0(x_1, x_3, x_4, x_5) = F_0(g_0(x_1, x_3), x_4, x_5) \]

where $g_0 = \bar{x}_1 + \bar{x}_3$ and $F_0 = \bar{x}_4 g_0 + x_4 \bar{g}_0$

\[ f_1(x_1, x_3, x_4, x_5) = F_1(g_1(x_1, x_3), x_4, x_5) \]

where $g_1 = \bar{x}_1 + x_1$ and $F_1 = \bar{x}_4 g_1 + x_4 \bar{g}_1$.

By Shannon's expansion theorem,

\[ f = \bar{x}_2 f_0(x_1, x_3, x_4, x_5) + x_2 f_1(x_1, x_3, x_4, x_5) \]

\[ \bar{x}_2 f_0 = \bar{x}_2 (\bar{x}_4 g_0 + x_4 \bar{g}_0) \text{ and } x_2 f_1 = x_2 (x_4 g_1 + \bar{x}_4 \bar{g}_1) \]

\[ \bar{x}_2 g_0 = \bar{x}_2 x_3 + x_2 \bar{x}_1 + \bar{x}_3 \]

\[ \bar{x}_2 g_1 = \bar{x}_2 (x_4 g_0 + x_4 \bar{g}_0) = \bar{x}_2 (x_4 + \bar{g}_0) \bar{x}_2 + \bar{g}_1 \]

By Shannon's expansion theorem,
2.4.1 Symmetric Functions

Symmetric functions [4] are an interesting class of combinational functions which have simple canonical realizations using bilateral devices such as relays and contacts.

A function \( f(x_1, x_2, \ldots, x_n) \) is symmetric in variables \( x_i \) denoted by \( x_i \sim x_j \) if \( f(x_1, x_2, \ldots, x_i, \ldots, x_j, \ldots, x_n) = f(x_1, x_2, \ldots, x_j, \ldots, x_i, \ldots, x_n) \).

That is, interchanging variables \( x_i \) and \( x_j \) on a Karnaugh map of \( f \) results in an identical map, or equivalently, replacing all appearances of \( x_i \) by \( x_j \) and \( x_j \) by \( x_i \) in a Boolean expression for \( f \) results in an equivalent expression. Similarly, \( x_i \sim x_j \) if \( f(x_1, x_2, \ldots, x_i, \ldots, x_j, \ldots, x_n) = f(x_1, x_2, \ldots, x_j, \ldots, x_i, \ldots, x_n) \).

Example 2.12 Consider the function \( f = \bar{x}_2 + \bar{x}_1 \bar{x}_3 + x_1 x_2 \). For this function \( x_i \sim x_j \) as shown by interchanging \( x_i \) and \( x_j \) in the Boolean expression for \( f \) or from the two Karnaugh maps of Figure 2.27(a).

\[
\begin{align*}
\text{The decomposition can be expressed as} & \\
f(x_1, x_2, x_3, x_4, x_5) &= F(g(x_1, x_2, x_3), x_2, x_4, x_5) \\
\text{where } F &= \bar{x}_2 \bar{x}_3 \bar{g} + \bar{x}_2 x_4 \bar{g} + x_2 \bar{x}_4 \bar{g} + \bar{x}_2 \bar{x}_4 \bar{x}_5 \bar{g} \\
\text{and } g &= \bar{x}_2 \bar{x}_3 + \bar{x}_1 x_2 + x_2 x_3 \quad \Box
\end{align*}
\]

Complex nondisjunctive decompositions can also be defined as in the disjunctive case discussed earlier. However, obtaining such decompositions will involve an exhaustive search of decomposition maps of all nondisjoint subsets of the set of all input variables, and is not computationally feasible in general.

2.4. SPECIAL CLASSES OF COMBINATIONAL FUNCTIONS

In this section we shall study several classes of combinational functions and their properties. Each of these classes has some desirable property, or has simple realizations in certain technologies.
Proof: Exercise. □

A function \( f(x_1, x_2, \ldots, x_n) \) is totally symmetric if for every pair of variables \( x_i, x_j \), \( x_i = x_j \) or \( [x_i \rightarrow x_j] \). For the function \( f \) of Example 2.12, \( x_1 \neq x_2 \) and \( x_1 \neq x_3 \). Therefore, \( f \) is not totally symmetric. The following lemmas characterize the class of totally symmetric functions and symmetry preserving logical operations.

Lemma 2.2 A function \( f(x_1, x_2, \ldots, x_n) \) is totally symmetric if and only if there is a set of integers \( \{ a_1, a_2, \ldots, a_p \} \), \( 0 \leq a_i \leq n \), such that \( f = 1 \) if and only if exactly \( a_i \) of the variables \( (x_1^+, x_2^+, \ldots, x_n^+) \) are equal to \( 1 \) for some \( j = 1, 2, \ldots, p \) (where \( x_j^+ \) denotes \( x_j \) or \( \bar{x}_j \)). This function will be denoted by \( S_{a_1, a_2, \ldots, a_p}(x_1^+, x_2^+, \ldots, x_n^+) \).

Proof: Suppose \( f \) is totally symmetric and it has a 1-point with exactly \( a_i \) of the variables equal to one. Then \( f = 1 \) when any set of exactly \( a_i \) variables are equal to 1 and any pair of variables may be interchanged without affecting the value of the function.

Similarly, if \( f = 1 \) for all points with exactly \( a_i \) variables of \( (x_1^+, x_2^+, \ldots, x_n^+) \) equal to 1 for \( j = 1, 2, \ldots, p \) then \( f \) is totally symmetric. □

Lemma 2.3 (a) \( S_{a_1, a_2, \ldots, a_p}(x_1, x_2, \ldots, x_n) + S_{b_1, b_2, \ldots, b_q}(x_1, x_2, \ldots, x_n) \) \( = S_{c_1, c_2, \ldots, c_k}(x_1, x_2, \ldots, x_n) \), where \( \{ c_1, c_2, \ldots, c_k \} \) \( = \{ a_1, a_2, \ldots, a_p \} \cup \{ b_1, b_2, \ldots, b_q \} \).

(b) \( S_{a_1, a_2, \ldots, a_p}(x_1, x_2, \ldots, x_n) \cdot S_{b_1, b_2, \ldots, b_q}(x_1, x_2, \ldots, x_n) \) \( = S_{c_1, c_2, \ldots, c_k}(x_1, x_2, \ldots, x_n) \), where \( \{ c_1, c_2, \ldots, c_k \} \) \( = \{ a_1, a_2, \ldots, a_p \} \cap \{ b_1, b_2, \ldots, b_q \} \).

(c) \( S_{a_1, a_2, \ldots, a_p}(x_1, x_2, \ldots, x_n) = S_{b_1, b_2, \ldots, b_q}(x_1, x_2, \ldots, x_n) \) \( \text{where} \) \( \{ b_1, b_2, \ldots, b_q \} = \{ 0, 1, \ldots, n \} \) \( - \{ a_1, a_2, \ldots, a_p \} \).

(d) \( S_{a_1, a_2, \ldots, a_p}(x_1, x_2, \ldots, x_n) = S_{b_1, b_2, \ldots, b_q}(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n) \) \( \text{where} \) \( b_i = n - a_i, i = 1, 2, \ldots, p \).

Proof: Exercise. □

Using Shannon's expansion theorem, any combinational function of \( n \) variables may be expressed as:

\[ \overline{x}_1 + x_1 \overline{x}_2 + \overline{x}_3 + \overline{x}_4 \overline{x}_5 + \overline{x}_6 \overline{x}_7 + \overline{x}_8 \overline{x}_9 \overline{x}_{10} \overline{x}_{11} \overline{x}_{12} \overline{x}_{13} \overline{x}_{14} \overline{x}_{15} \overline{x}_{16} \times b + 1 = x_1 \overline{x}_2 + \overline{x}_3. \]
Lemma 2.6  For the function \( f(x_1, x_2, \ldots, x_n) \) (a) \( x_1 \sim x_2 \) if and only if \( f(0,1, x_3, x_4, \ldots, x_n) = f(1,0, x_3, x_4, \ldots, x_n) \), and (b) \( x_1 \sim \bar{x}_2 \) if and only if \( f(0,0, x_3, x_4, \ldots, x_n) = f(1,1, x_3, x_4, \ldots, x_n) \).

Proof: (a) By Shannon's expansion theorem, \( f(x_1, x_2, \ldots, x_n) = x_1 x_2 f(1,1, x_3, \ldots, x_n) + x_1 \bar{x}_2 f(1,0, x_3, \ldots, x_n) + \bar{x}_1 x_2 f(0,1, x_3, \ldots, x_n) + \bar{x}_1 \bar{x}_2 f(0,0, x_3, \ldots, x_n) \). If \( x_1 \sim x_2 \), then we can interchange \( x_1 \) and \( x_2 \) without changing the function. If we do this to the right side of the above equation, we obtain the expression \( x_1 x_2 f(1,1, x_3, \ldots, x_n) + \bar{x}_1 x_2 f(1,0, x_3, \ldots, x_n) + x_1 \bar{x}_2 f(0,1, x_3, \ldots, x_n) + \bar{x}_1 \bar{x}_2 f(0,0, x_3, \ldots, x_n) \). This expression is equal to the original expression from which it was obtained if and only if \( f(0,1, x_3, x_4, \ldots, x_n) = f(1,0, x_3, x_4, \ldots, x_n) \).

(b) Exercise.

Lemmas 2.4 – 2.6 can be used to determine symmetry by first applying Lemma 2.4 to determine possible variables of symmetry and then applying Lemmas 2.5 and/or 2.6 to reduce the original problem to smaller problems. However, after using Lemma 2.5, we are still faced with the problem of determining symmetry for two \((n-1)\)-variable functions. In contrast, after Lemma 2.6, we need only determine the equivalence (not symmetry) of two \((n-2)\)-variable functions. Therefore, Lemma 2.6 seems to be more useful.

Example 2.13  Let \( f(x_1, x_2, x_3, x_4) \) be a function whose 1-points are listed in the table of Figure 2.28.

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 2.28  List of 1-points of the function of Example 2.13

If \( p_i \) is the number of 1-points with \( x_i = 1 \) and \( q_i \) is the number of 1-points with \( x_i = 0 \), then

\[
\begin{align*}
p_1 &= 4 \\
p_2 &= 4 \\
p_3 &= 3 \\
p_4 &= 4 \\
q_1 &= 3 \\
q_2 &= 3 \\
q_3 &= 4 \\
q_4 &= 3
\end{align*}
\]

From Lemma 2.4 we can conclude that \( x_1 \) may be symmetric to \( x_2 \) (since \( p_1 = p_2 \)) but \( x_1 \not\sim \bar{x}_2 \) (since \( p_1 \neq q_2 \)). Similarly \( x_1 \) may be symmetric to \( \bar{x}_3 \) and \( x_4 \). From Lemma 2.6 if \( x_1 \sim x_2 \) then \( f(0,1, x_3, x_4) = f(1,0, x_3, x_4) \). The 1-points of these two functions are listed below:

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Since the sets of 1-points are not identical, \( x_1 \not\sim \bar{x}_2 \), and the function is not totally symmetric.

This result could also have been obtained by applying Lemma 2.5.

If \( f \) were totally symmetric, then \( f(0, x_3, \ldots, x_n) \) and \( f(1, x_2, \ldots, x_n) \) must be totally symmetric. The 1-points of \( f(1, x_2, x_3, x_4) \) are:

<table>
<thead>
<tr>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 2.29  Values of \( p_i, q_i \) for 1-points of \( f(1, x_2, x_3, x_4) \) of Example 2.13

Applying Lemma 2.4, we see that \( x_2 \not\sim x_3 \) and \( x_2 \not\sim \bar{x}_3 \) since \( p_3 \neq p_2 \) and \( p_3 \neq q_2 \). Therefore, \( f(1, x_2, x_3, x_4) \) is not totally symmetric. Hence, \( f(x_1, x_2, x_3, x_4) \) is not totally symmetric.

Example 2.14  Consider the function \( f \) whose 1-points are listed in the table of Figure 2.30.
Sufficiency: Let $P_1(f)$ and $P_2(f)$ be the functions obtained by permuting the inputs as in conditions (1) and (2) of the theorem. Since $P_1(f) = f$ and $P_2(f) = P_2$, these permutations can be applied repeatedly in any sequence of $P_1$ and $P_2$ without changing the function. Any pair of variables $x_i$ and $x_j$ can be interchanged by applying an appropriate sequence of permutations $P_1$ and $P_2$. Therefore, $f(x_1, x_2, ..., x_i, ..., x_j, ..., x_n) = f(x_1, x_2, ..., x_j, ..., x_i, ..., x_n)$ and $x_i \sim x_j$ for all $i, j$, $1 \leq i, j \leq n$. 

This theorem is useful for finding symmetric functions in the uncomplicated variable set. For the function $f$ of Example 2.14, we note (by permuting columns 3 and 4 of Figure 2.30) that $f(x_1, x_2, x_3, x_4) \neq f(x_3, x_2, x_1, x_4)$, violating condition (1) of Theorem 2.6. In order to determine whether the function is totally symmetric, we have to determine whether the conditions of the theorem can be satisfied by complementing some of the input variables. The function of Example 2.14 can be shown to be totally symmetric by treating it as a function of $x_1$, $x_2$, $x_3$, and $x_4$ and applying Theorem 2.6.

A canonical contact network for realizing any symmetric function is shown in Figure 2.31 [2]. Associated with each contact is a variable that controls the opening and closing of the contact as discussed in Section 1.2. Any point in the circuit has a value of 1 if and only if there is a path from that point to the point labeled B in Figure 2.31, along which all contacts are closed. Since the OR function can be realized in contact networks by connecting contacts in parallel, any symmetric function $S_{a_1, a_2, ..., a_n}$ can be realized by connecting together the points labeled $S_{a_1, S_{a_2, ..., S_{a_n}}}$. Figure 2.32 shows a realization of $S_{a_1, a_2, ..., a_n}$. Such canonical realizations of symmetric functions may be implemented with contacts or other types of bidirectional devices such as metal oxide semiconductor (MOS) field effect transistors (FET).

24.2 Unate Functions

An important property of combinational functions is unateness [10]. Let $x_i$ and $x_j$ be points in the $n$-cube defined by the variables $(x_1, x_2, ..., x_n)$. For a function $f(x_1, x_2, ..., x_n)$, $f(x_i) \geq f(x_j)$ if $f(x_i) = f(x_j) = f(x_j) = 0$. The function $f$ is said to be positive in a variable $x_i$, if for all $2^n - 1$ possible combinations of values of the remaining $n - 1$ variables,

$$f(x_1, x_2, ..., x_{i-1}, 1, x_{i+1}, ..., x_n) \geq f(x_1, x_2, ..., x_{i-1}, 0, x_{i+1}, ..., x_n).$$
Similarly, \( f \) is negative in the variable \( x_i \) if
\[
f(x_1, x_2, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) \geq f(x_1, x_2, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n).
\]

**Lemma 2.7** A completely specified combinational function \( f(x_1, x_2, \ldots, x_n) \) is positive in a variable \( x_i \) if and only if any minimal sum of products expression for \( f \) does not contain the literal \( \overline{x}_i \). Similarly, \( f \) is negative in the variable \( x_i \) if and only if no minimal sum of products expression for \( f \) contains the literal \( x_i \).

**Proof:** Exercise. \( \Box \)

A function \( f(x_1, x_2, \ldots, x_n) \) is said to be positive if it is positive in all its variables, \( x_i \). Similarly, a function which is negative in all its variables is called a negative function. A function \( f(x_1, x_2, \ldots, x_n) \) is unate if for every \( x_i, i = 1, 2, \ldots, n \), \( f \) is either positive or negative in the variable \( x_i \).

Let \( x_i = (a_1, a_2, \ldots, a_n) \) and \( x_j = (b_1, b_2, \ldots, b_n) \). Then \( x_i \geq x_j \) if \( a_k \geq b_k \) for all \( k, 1 \leq i \leq n \). Thus, for a positive function, \( f(x_i) \geq f(x_j) \) if \( x_i \geq x_j \). Similarly, for a negative function \( f(x_i) \leq f(x_j) \) if \( x_i \leq x_j \).

**Theorem 2.7** A completely specified function \( f(x_1, x_2, \ldots, x_n) \) is unate if and only if any minimal sum of products expression contains either the literal \( x_i \) or \( \overline{x}_i \) but not both, for all \( x_i, i = 1, 2, \ldots, n \).

**Proof:** Exercise. \( \Box \)

**Example 2.15** The function \( f_1 = x_1 x_2 + x_3 x_4 \) is unate. However \( f_2 = x_1 x_2 x_3 + \overline{x}_2 x_4 \) is not unate since both \( x_2 \) and \( \overline{x}_2 \) appear in the minimal sum of products. \( \Box \)

A consequence of Theorem 2.7 is that a unate function can be realized without using any inverters, assuming that each variable or its complement (but not both) is available (depending on whether the function is positive or negative in the particular variable).

### 2.4.3 Threshold Functions

A function \( f(x_1, x_2, \ldots, x_n) \) is a threshold function [14] if there exists a set of numbers \( \{w_1, w_2, \ldots, w_n\} \) (called weights) and a number \( T \) (called the threshold) such that \( f(x_1, x_2, \ldots, x_n) = 1 \) if and only if \( \sum_{i=1}^{n} w_i x_i \geq T \).
where \( x_i = 0 \) or 1 and the multiplication and summation are arithmetic (rather than Boolean).

In some technologies, threshold functions can be realized using a single device called a threshold element which is represented as shown in Figure 2.33. Threshold functions are of interest primarily for this reason.

\[ \begin{align*}
&x_1 \rightarrow w_1 \\
&x_2 \rightarrow w_2 \\
&\vdots \\
&x_n \rightarrow w_n
\end{align*} \]

Figure 2.33 Representation of a threshold element

The following two lemmas show that the set of threshold functions is a proper subset of the set of unate functions.

Lemma 2.8 All threshold functions are unate.

Proof: Assume \( f(x_1, x_2, \ldots, x_n) \) is a threshold function with weights \( w_1, w_2, \ldots, w_n \). If \( f \) is not unate, there is a minimal sum of products expression in which some variable \( x_i \) appears complemented and uncomplemented and \( f \) can be represented as

\[
f = x_i f_{i1}(x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) + \bar{x}_i f_{i0}(x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)
\]

where \( f_{i1} = f(x_1, x_2, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) \) and \( f_{i0} = f(x_1, x_2, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) \).

If \( w_i \leq 0 \), then any 1-point of \( f_{i1} \) is also a 1-point of \( f_{i0} \). Therefore, \( f = x_i f_{i0} + f_{i1} \) and \( f \) is negative in \( x_i \). Contradiction. If \( w_i \geq 0 \), then any 1-point of \( f_{i0} \) is also a 1-point of \( f_{i1} \). Therefore, \( f = x_i f_{i0} + f_{i0} \) and \( f \) is positive in \( x_i \). Contradiction. This proves that there is no variable \( x_i \) such that both \( x_i \) and \( \bar{x}_i \) appear in a minimal sum of products expression of \( f \). Hence, by Theorem 2.5, \( f \) is unate. \( \square \)

From the proof of Lemma 2.8 it follows that if \( f \) is a threshold function with \( w_i > 0 \), \( f \) is positive in \( x_i \) and if \( w_i < 0 \), \( f \) is negative in \( x_i \).

Lemma 2.9 Not all unate functions are threshold functions.

Proof: Consider the unate function \( f = x_1 x_2 + x_3 x_4 \) and assume that it is a threshold function with weights \( w_1, w_2, w_3, w_4 \) and threshold \( T \). Since \( f(1,1,0,0) = 1 \), \( w_1 + w_2 \geq T \) and since \( f(1,0,1,0) = 0 \), \( w_1 + w_3 < T \). From these inequalities, we can conclude that \( w_1 > w_2 \). Similarly, \( f(0,1,0) = 1 \) implies \( w_2 + w_4 \geq T \) and \( f(0,1,0,1) = 0 \) implies \( w_3 + w_4 < T \). From these inequalities, we conclude that \( w_3 > w_4 \). This contradicts \( w_3 < w_4 \) and proves that \( f \) is not a threshold function. \( \square \)

For a threshold function \( f(x_1, x_2, \ldots, x_n) \), the \( 2^n \) points of the n-cube can be partitioned into two disjoint sets, the 1-points and the 0-points by a hyperplane defined by the equation

\[
\sum_{k=1}^{n} w_k x_k = \frac{T}{2}
\]

Because the equation is linear, the points on one side of this hyperplane are 1-points and the points on the other side are 0-points. Threshold functions are also called linearly separable functions because of the above property.

For each 1-point of \( f \), \( x_i = (x_{i1}, x_{i2}, \ldots, x_{in}) \) and each 0-point of \( f \), \( x_i = (x_{i1}, x_{i2}, \ldots, x_{in}) \). \( \sum_{k=1}^{n-1} w_k x_k > \sum_{k=1}^{n} w_k x_k \). Thus, if \( f \) has \( k_0 \) 0-points and \( k_1 \) 1-points, we can derive a set of \( k_0, k_1 \) linear constraints (inequalities) on the weights \( w_i \) which can be successfully satisfied if and only if \( f \) is a threshold function.

For a unate function \( f(x_1, x_2, \ldots, x_n) \) a 1-point \( x_i \) is minimal if changing any positive variable from 1 to 0 or any negative variable from 0 to 1 transforms \( x_i \) to \( x_i' \) where \( x_i' \) is a 0-point of \( f \). Similarly, a 0-point \( x_i \) is minimal if changing any positive variable from 0 to 1 or any negative variable from 1 to 0 transforms \( x_i \) to \( x_i' \) where \( x_i' \) is a 1-point. Let \( M_0 \) be the set of all minimal 1-points and \( M_0 \) be the set of all minimal 0-points. Each pair of elements \( (p_i, p_j) \), \( p_i \in M_0, p_j \in M_0 \), defines a linear inequality in the \( w \)-variables. If these constraints are satisfied the constraint defined by any 1-point and any 0-point will also be satisfied. (This follows since for any 1-point \( x_k \) if \( x_k \in M_0 \) there exists a 1-point \( x_i \in M_0 \) such that \( \sum_{k=1}^{n} w_k x_k > \sum_{k=1}^{n} w_k x_k \) where \( w_k \leq 0 \) if \( f \) is positive in \( x_q \), and for any 0-point \( x_m \) \( x_m \in M_0 \) there exists a 0-point \( x_q \in M_0 \) such that \( \sum_{k=1}^{n} w_k x_k < \sum_{k=1}^{n} w_k x_k \) where \( w_q \leq 0 \) if \( f \) is negative in \( x_q \).

Procedure 2.3

1. For a unate function \( f(x_1, x_2, \ldots, x_n) \) determine the sets \( M_0 \) of minimal 1-points and \( M_0 \) of maximal 0-points
2. Each pair of elements \( (p_i, p_j) \), \( p_i \in M_1, p_j \in M_0 \), defines a linear
inequality in the \( w \)-variables. If \( m_0 \) and \( m_1 \) represent the number of elements in \( M_0 \) and \( M_1 \), respectively, there will be \( m_0 \cdot m_1 \) inequalities. Solve these inequalities, if possible, to determine \( \{ w_1, w_2, \ldots, w_n \} \). The value of the threshold is given by \( T = \min_{p \in M_i} \sum_{j=1}^{n} w_j x_j \) where \( p_i = (x_{i1}, x_{i2}, \ldots, x_{in}) \).

The proof of Lemma 2.9 illustrates the application of Procedure 2.3 to show that a given function is not a threshold function. If all constraints can be satisfied by some values of \( w_i \) and \( T \), the function is a threshold function since for these values of \( w_i \) and \( T \), 0-points not adjacent to the hyperplane will have \( \Sigma w_i x_i < T \), and 1-points not adjacent to the hyperplane will have \( \Sigma w_i x_i > T \).

Example 2.16 Consider the unate function

\[
 f(x_1, x_2, x_3, x_4) = x_2 + \overline{x}_3 + x_1 x_4
\]

The 1-point \((0,1,1,0)\) is a minimal 1-point because changing \( x_3 \) to 0 will make \( f = 0 \). Similarly changing \( x_3 \) to 1 will cause the 1-point \( (0,0,0,0) \) to become a 0-point. The following sets of minimal 1-points, \( M_1 \), and maximal 0-points, \( M_0 \), are obtained in this manner:

\[
 M_1 = \{(0,1,1,0), (0,0,0,0), (1,0,1,1)\}
\]

\[
 M_0 = \{(0,0,1,1), (1,0,1,0)\}
\]

From \( M_1 \) and \( M_0 \), we obtain the following sets of constraints.

\[
 w_2 + w_3 \geq w_4 + w_4 \quad (1)
\]

\[
 w_2 + w_3 \geq w_1 + w_3 \quad (2)
\]

\[
 0 \geq w_3 + w_4 \quad (3)
\]

\[
 0 \geq w_1 + w_3 \quad (4)
\]

\[
 w_1 + w_3 + w_4 \geq w_4 \quad (5)
\]

\[
 w_1 + w_3 + w_4 \geq w_1 + w_3 \quad (6)
\]

Constraint (5) implies \( w_1 > 0 \). To satisfy this constraint, we set \( w_1 = +1 \). To satisfy constraint (4), we set \( w_3 = -2 \). To satisfy (6) and (3), we set \( w_4 = 1 \), and to satisfy (2) and (1), we set \( w_2 = 2 \). The value of \( T \) is found from the elements of \( M_1 \).

\[
 \begin{align*}
 w_2 + w_3 & \geq T \quad (7) \\
 0 & \geq T \quad (8) \\
 w_1 + w_3 + w_4 & \geq T \quad (9)
\end{align*}
\]

The value \( T = 0 \) satisfies all of these inequalities for \((w_1, w_2, w_3, w_4) = (1, 2, -2, 1)\).

As the number of variables in the function increases, the set of inequalities may become too large to solve manually as in the above example. However, linear programming techniques can be used to solve larger sets of linear inequalities.

Special classes of functions and corresponding properties of interest vary with changes in technology. Another interesting property of functions for current technologies is the amount of fanout required to realize a function. As we shall see, fanout presents difficulties in magnetic bubble logic. Also the degree of difficulty in testing a circuit seems to depend on the total amount of fanout in the circuit. For this reason it may have to be realized by a circuit with minimum fanout. In a fanout-free circuit each input and the output of each gate is an input to at most one gate. Hayes [6] has shown that the class of fanout-free functions (those functions which can be realized by fanout-free circuits) is a proper subset of unate functions and has developed a simple test procedure to determine whether a unate function is fanout free. The more general problem of determining minimum total fanout realizations of arbitrary functions has not been solved.

2.5 COMPLETE SETS OF LOGIC PRIMITIVES

A combinational circuit may be considered to be an interconnection of simpler single output combinational circuits, which we shall call logic primitives [11]. For example, the sum-of-products realization of a function may be thought of as being comprised of the logic primitives, AND, OR and NOT. Although combinational functions can be realized by circuits containing feedback [8], we shall consider only feedback-free realizations. A basic question is: Given a set of logic primitives, can they be interconnected to realize an arbitrary combinational function?

A set of logic primitives that can be used to realize any combinational function is called logically complete.

A set of logic primitives is said to be strongly complete if any arbitrary combinational function including the constants 0 and 1 can be realized.
by interconnecting a finite number of primitives from the set, assuming that only the unimplemented input variables \((x_1, \ldots, x_n)\) are available as inputs. A set of primitives that is sufficient for realizing all combinational functions if the constants 0 and 1 are available as inputs is called weak complete.*

Any combinational function can be realized in the sum-of-products form using two input ORs, two input ANDs, and NOTs, and these constitute a strong complete set of primitives. The AND itself can be realized using ORs and NOTs and, therefore, the latter two functions form a strong complete set. Similarly, AND and NOT also form a strong complete set. The 2-input NAND itself constitutes a strong complete set, because the OR (and AND) and NOT functions can be realized using NANDs. The exclusive-OR and AND functions together constitute a weak complete set, since the NOT element can be realized as \(x = x \oplus x\), but the constant 1 cannot be realized.

A set of logic primitives can be determined to be complete or incomplete depending upon whether or not it can realize the OR (or AND) and the NOT functions. However, it may be difficult to determine this except for primitives with only a few inputs. It is, therefore, desirable to determine the completeness of a set of logic primitives from the properties of the functions comprising the set. To this end, the following properties of combinational functions will be useful.

1. **Zero-preservation.** A function \(f(x_1, x_2, \ldots, x_n)\) is zero-preserving if \(f(0,0,\ldots,0) = 0\).

2. **One-preservation.** A function \(f(x_1, x_2, \ldots, x_n)\) is one-preserving if \(f(1,1,\ldots,1) = 1\).

3. **Self-duality.** The dual \(f_d\) of a function \(f(x_1, x_2, \ldots, x_n)\) was defined in Chapter 1 as \(f_d(x_1, x_2, \ldots, x_n) = f(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)\). A function \(f(x_1, x_2, \ldots, x_n)\) is self-dual if \(f_d(x_1, x_2, \ldots, x_n) = f(x_1, x_2, \ldots, x_n)\).

4. **Positive unateness (monotonicity)** was defined in Section 2.3 and may be restated as follows: Let \(a = (a_1, a_2, \ldots, a_n)\) and \(b = (b_1, b_2, \ldots, b_n)\) be any two points in the \(n\)-cube. Then, \(f(x_1, x_2, \ldots, x_n)\) is positive unate if and only if \(f(a) \geq f(b)\) for all \(a \geq b\). Recall that \(a \geq b\) if and only if \(a_i \geq b_i\) for all \(i, 1 \leq i \leq n\).

5. **Linearity.** A function \(f(x_1, x_2, \ldots, x_n)\) is said to be linear if it can be expressed as \(f(x_1, x_2, \ldots, x_n) = a_0 \oplus a_1 x_1 \oplus a_2 x_2 \oplus \cdots \oplus a_n x_n\) where \(a_i = 0, 1\) for \(i = 0, 1, 2, \ldots, n\).

*We can also define strong \(c\)-complete and weak \(c\)-complete sets of functions in which complemented as well as unimplemented input variables are available (see Problem 2.22).

---

**Lemma 2.10** Let \(F = \{f_1, f_2, \ldots, f_k\}\) be a set of primitives, all of which have some common property \(P\) from the set of five properties given above. Any combinational function realized by interconnecting the primitives will also have the property \(P\).

**Proof:** We shall present the proof of the lemma for linearity only. The lemma can be proved for the other properties in a similar manner. Without loss of generality, consider a circuit of the form shown in Figure 2.34, where \(f_1, \ldots, f_k\) and \(g\) are linear functions of their inputs, and \(x_i \in \{x_1, x_2, \ldots, x_n\}\). If this circuit realizes \(f(x_1, x_2, \ldots, x_n)\), then \(z = f(x_1, x_2, \ldots, x_n) = g(x_1, \ldots, x_m, f_1(x_{i_1}, \ldots, x_{i_p}), \ldots, f_k(x_{k_1}, \ldots, x_{k_q}))\). Since \(g\) is a linear function, \(z = a_0 \oplus a_1 f_1(x_{i_1}, \ldots, x_{i_p}) \oplus \cdots \oplus a_k f_k(x_{k_1}, \ldots, x_{k_q}) \oplus b_1 x_1 \oplus \cdots \oplus b_m x_m\). Since each \(f_i\) is also a linear function, \(z = a_0 \oplus a_1 c_{i_1} c_{i_2} x_{i_1} \oplus \cdots \oplus c_{i_p} x_{i_p} \oplus \cdots \oplus a_k c_{k_1} c_{k_2} \cdots \oplus b_1 x_1 \oplus \cdots \oplus b_m x_m\). Each constant \(a_i, b_i, \text{ and } c_i\) are either 0 or 1. Therefore, the output of the circuit can be expressed as \(z = d_0 \oplus d_1 x_1 \oplus \cdots \oplus d_n x_n\), where \(d_i = 0, 1\). Any number of inputs of the circuit can be replaced by linear functions, and the output can be shown to be a linear function. Therefore, Lemma 2.10 holds for any acyclic circuit of linear functions. \(\Box\)

**Lemma 2.11** If the constants 0 and 1 are available as inputs, the NOT operation can be realized by any function which is not positive unate.

**Proof:** Let \(f(x_1, x_2, \ldots, x_n)\) be a function that is not positive unate. This implies that there exist some \(a = (a_1, a_2, \ldots, a_n)\) and \(b = (b_1, b_2, \ldots, b_n)\) such that \(a \geq b\) but \(f(a) = 0\) and \(f(b) = 1\). Let \(X\) be the set of input variables that have the value 1 in \(a\) and 0 in \(b\). Since \(a > b\), \(X\) will contain at least one element and all variables \(x_i \in X\) have the same value in \(a\) and \(b\). The complement \(\bar{y}\) of an input \(y\) can be
realized by connecting the input \( y \) to all \( x_i \in X \) and setting \( x_i = a_i = b_i \) for all \( x_i \notin X \).

**Lemma 2.12** If the constants 0 and 1 are available as inputs, the AND or OR operation can be realized by any nonlinear function.

**Proof:** Any combinational function can be expressed as a sum of minterms. Since at most one minterm can be 1 at any time, the logical sums can be replaced by exclusive ORs. Replacing every complemented variable, \( \bar{x}_i \) in each term by \( 1 \oplus x_i \) and simplifying each term, we obtain a representation (called the Reed-Muller expansion \([12],[16]\)) of the function in the form \( a_0 \oplus \Sigma a_i x_{i_1} x_{i_2} \ldots x_{i_p} \), where \( \{ x_{i_1}, x_{i_2}, \ldots, x_{i_p} \} \) is a subset of the variables \( x_1, x_2, \ldots, x_n \), \( a_i = 0,1 \) and \( \Sigma \) represents the modulo 2 sum. (See Problem 1.11.) If the function is linear, each nonzero product term in the expansion will contain only one variable \( x_i \).

If \( f(x_1, x_2, \ldots, x_n) \) is nonlinear, the expansion of the function must contain at least one term containing two or more variables. Let one such term be of the form \( p(x_{i_1}, x_{i_2}) \), where \( p \) may be 1 or a product of other variables. Select one such term with the fewest number of variables and set all the variables contained in \( p \) to 1. The variables other than \( x_{i_1}, x_{i_2} \) and those in \( p \) are assigned to the value 0. Let the function so obtained be denoted by \( h(x_{i_1}, x_{i_2}) \), which will be of the form \( h(x_{i_1}, x_{i_2}) = a_0 \oplus a_1 x_{i_1} \oplus a_2 x_{i_2} \oplus a_3 x_{i_1} x_{i_2} \). For \( a_0 = a_1 = a_2 = 0 \), \( h(x_{i_1}, x_{i_2}) = x_{i_1} \) and for \( a_0 = 0, a_1 = a_2 = 1, h(x_{i_1}, x_{i_2}) = x_{i_1} + x_{i_2} \). The remaining six combinations of values for \( a_0, a_1 \) and \( a_2 \) yield the functions \( \bar{x}_{i_1} \bar{x}_{i_2}, x_{i_1} \bar{x}_{i_2}, \bar{x}_{i_1} x_{i_2}, x_{i_1} x_{i_2}, \bar{x}_{i_1} + x_{i_2}, x_{i_1} + \bar{x}_{i_2} \), and \( \bar{x}_{i_1} + \bar{x}_{i_2} \). None of these six functions is positive unate. If \( a_0, a_1 \), and \( a_2 \) have one of these six combinations of values, Lemma 2.12 may be used to construct NOT functions, which can be used to complement the appropriate inputs or the output enabling us to realize AND or OR (by De Morgan's law).

**Theorem 2.9** (Post's theorem \([15]\)) A set of logic primitives \( F \) is strong complete if and only if \( \{ F \} \) contains a function that is (1) not zero-preserving, (2) not one-preserving, (3) not self-dual, (4) not positive unate, and (5) nonlinear. (A different function in \( f \) may be used for each of the 5 properties.)

**Proof:** The necessity of the conditions follows from Lemma 2.10. The sufficiency can be proved by showing that the constants 0 and 1 can be realized and then applying Theorem 2.8.

There are two cases to be considered: (1) there is a function \( f_1 \) which is not zero-preserving but is one-preserving, and a function \( f_2 \) which is not one-preserving but is zero-preserving, and (2) there is a function \( f_3 \) which is neither zero-preserving nor one-preserving.

In case (1), the constant 1 can be produced by connecting all input terminals of \( f_1 \) to the same variable. Similarly, the constant 0 can be produced by connecting all inputs of \( f_2 \) to the same variable.

In case (2), \( f_1(0, 0, \ldots, 0) = 1 \) and \( f_2(1, 1, \ldots, 1) = 0 \). By connecting all input terminals of this primitive to the same input variable, we can realize the NOT function. Now we can use any non-self-dual function \( f_3 \) (which may be the same function used for realizing the NOT) to realize the constants 0 and 1, as follows: Since \( f_3 \) is non-self-dual, there exists some input combination \( (a_1, a_2, \ldots, a_n) \) such that \( f_3(a_1, a_2, \ldots, a_n) = f_3(\bar{a}_1, \bar{a}_2, \ldots, \bar{a}_n) \), where \( a_i = 0, 1 \), \( 1 \leq i \leq n \). Without loss of generality, let us assume that \( a_i = 0 \) for \( 1 \leq i \leq k \) and \( a_i = 1 \) for \( k + 1 \leq i \leq n \). The constants 0 and 1 can be realized using the NOT function (realized as described earlier) and the non-self-dual function \( f_3 \) as shown.

**Figure 2.35** Realization of \( x \cdot y \) using a complex module

construction in the proof of Lemma 2.12, \( f(x_1, x_2, 1, 0) = h(x_1, x_2) = 1 \oplus x_1 x_2 = x_1 + x_2 \). Since \( h(x_1, x_2) \) is not positive unate, we can use it for realizing NOT by setting \( x_1 \) or \( x_2 \) to 1. Figure 2.35 shows how the product \( x \cdot y \) may be realized using a module with output \( f = 1 \oplus x_1 x_2 x_3 \oplus x_2 x_3 x_4 \).
in Figure 2.36. The constants 0 and 1 will be obtained at A and B, but which of the two will be 0 and which will be 1 will depend upon the function $f_3$. If $f_3(a_1, a_2, ..., a_n) = 0(1)$, then $A = 1(0)$ and $B = 0(1)$.

With the constants 0 and 1 realized by either of the above methods, the OR and AND can be realized using the nonlinear function as in the proof of Lemma 2.12. The NOT can be realized using a function that preserves neither 0 nor 1 (as in case 2 above) or by using a nonpositive-uneate function and the constants 0 and 1 as in the proof of Lemma 2.11.

A strong (weak) complete set of logic primitives is called a strong (weak) basis if no proper subset of the primitives is also complete.

**Theorem 2.10** The maximum number of primitives in a strong basis is four and in a weak basis, two.

**Proof:** A strong basis must satisfy the conditions of Theorem 2.9. However, a function which is not zero-preserving is also either not one-preserving or nonself-dual. (Proof: Exercise) Therefore, the maximum number of primitives in a strong basis is four. Since there exist nonlinear functions which are also positive unate (e.g., AND), it follows from Theorem 2.8 that the maximum number of primitives in a weak basis is two.

It is also of interest to consider the relative efficiency of different sets of logically complete modules in realizing arbitrary functions. It seems especially difficult to use a module with output $f$ to realize $f^d$, the dual of $f$. Thus it requires five 3-input NAND gates to realize a 3-input NOR. For this reason it seems likely that sets of primitives which contain dual pairs may be relatively efficient. Thus the complete set {NAND, AND} may not be as good as {NAND, NOR}. However, this hypothesis has not been verified and little has been achieved in measuring efficiency or obtaining the efficient realizations using sets of complex modules [13].

**Sources**

The minimization problem for multiple output combinational functions was considered by McCluskey and Schorr [9]. The material on combinational function decomposition is primarily due to Ashenhurst [1], and was extended for incompletely specified functions by Hight [7]. A good reference on this subject is a book by Curtis [3]. The subject of symmetric functions has been considered by many people among them Caldwell [2], Harrison [5], and Epstein [4]. The material on unate functions is due to McNaughton [10] and that on threshold functions to Paul and McCluskey [14]. Hayes [6] considered fanout free functions. The material on logical completeness is due to Post [15] and is also discussed by Mukhopadhyay [11]. Opsahl [13] considered the problem of obtaining efficient realizations from logically complete module sets.

**REFERENCES**


PROBLEMS

2.1 For each of the following sets of functions find a minimal 2-level realization.

a) \( f_1(x_1,x_2,x_3) = \Sigma m_0,m_1,m_2,m_6 \)
\( f_2(x_1,x_2,x_3) = \Sigma m_2,m_3,m_4,m_6 \)

b) \( f_1(x_1,x_2,x_3) = \Sigma m_0,m_1 + \Sigma m_2,m_7 \)
\( f_2(x_1,x_2,x_3) = \Sigma m_2,m_3,m_4 + \Sigma m_7 \)

c) \( f_1(x_1,x_2,x_3,x_4) = \Sigma m_2,m_7,m_{12},m_{13} + \Sigma m_2 \)
\( f_2(x_1,x_2,x_3,x_4) = \Sigma m_0,m_1,m_2,m_3 + \Sigma m_7 \)
\( f_3(x_1,x_2,x_3,x_4) = \Sigma m_1,m_2,m_4,m_{12} + \Sigma m_{13} \)

2.2 The circuit of Figure 2.37 is a minimal 2-level realization of the set of functions \( \{f_1,f_2\} \) where \( f_1(x_1,x_2,x_3,x_4) = \Sigma m_0,m_1,m_2 + \Sigma \text{don't care} m_7 \). Is \( f_2(x_1,x_2,x_3,x_4) \) uniquely determined? Prove this or determine the \( f_2 \) with the greatest number of unspecified entries.

2.3 For each of the following functions find a decomposed realization of the form specified, if possible.

a) \( f(x_1,x_2,x_3,x_4) = \bar{x}_1 \bar{x}_2 x_3 \bar{x}_4 + \bar{x}_1 \bar{x}_2 \bar{x}_3 x_4 + x_1 \bar{x}_2 \bar{x}_4 + x_1 x_2 x_4 + x_2 \bar{x}_3 x_4 + \bar{x}_2 \bar{x}_3 \bar{x}_4 \)
   as \( F(g(x_1,x_3),x_2,x_4) \).

b) \( f(x_1,x_2,x_3,x_4) = \Sigma m_0,m_1,m_2,m_3,m_4,m_6,m_{10},m_{13},m_{15} \) as \( F(g(x_1,x_3),x_2,x_4) \).

c) \( f(x_1,x_2,x_3,x_4) = x_1 x_2 \bar{x}_4 + x_1 x_4 \bar{x}_3 + \bar{x}_1 x_4 + \bar{x}_2 \bar{x}_3 \bar{x}_4 \)
   as \( F(g(x_1,x_2,x_3),x_4) \), where \( g(x_1,x_2,x_3) = F'(g'(x_2,x_3),x_1) \).

2.4 For the function specified by the Karnaugh map of Figure 2.38 what must be the entries denoted by \( a,b,c \) in order for \( f \) to have a simple disjunctive decomposition of the form \( F(g(x_1,x_2),x_3,x_4) \).
2.5 Which of the following are symmetric functions?

a) \( f(x_1, x_2, x_3, x_4) = \Sigma m_1, m_2, m_4, m_6, m_7, m_8, m_{13}, m_{14} \)

b) \( f(x_1, x_2, x_3) = \Sigma m_1, m_2, m_4, m_7 \)

c) \( f(x_1, x_2, x_3) = \Sigma m_1, m_4, m_7 \)

d) \( f(x_1, x_2, x_3, x_4) = \Sigma m_1, m_2, m_4, m_6, m_7, m_8, m_{13}, m_{14} \)

2.6 a) If \( f_1(x_1, x_2, \ldots, x_n) \) and \( f_2(x_1, x_2, \ldots, x_n) \) are both symmetric, which, if any, of the following are always symmetric,

\[ f_1 \cdot f_2, f_1 + f_2, f_1 \oplus f_2 \]

and which are never symmetric?

b) Repeat (a) if \( f_1 \) is symmetric but \( f_2 \) is not.

c) Repeat (a) if neither \( f_1 \) nor \( f_2 \) is symmetric.

2.7 Under what conditions are \( f(x_1, x_2, \ldots, x_n) \) and the dual of \( f \) both symmetric functions?

2.8 For the function \( f \) specified by the Karnaugh map of Figure 2.39, what must the entries denoted by \( a, b, \) and \( c \) be in order for \( f \) to be symmetric?

![Karnaugh Map](image)

2.9 Prove that the circuits of Figures 2.40 (a) and (b) can be made to realize the same function by an appropriate choice of \( w_i \) and find the smallest such value of \( w_i \) as a function of \( T_1, T_2, w_{11}, w_{12} \).

2.10 a) Prove that all prime implicants of a completely specified unate function are essential.

2.11 a) Consider a threshold function \( f(x_1, x_2, \ldots, x_n) \) with weights \( w_i, i = 1, \ldots, n \), and threshold \( T \). If all \( w_i \) are equal, \( f \) is called a voting function. Prove that all voting functions are symmetric.

b) Are all symmetric functions voting functions? If not, can you specify constraints on the numbers \( k_1, k_2 \ldots \) etc. in order that \( S_{k_1, k_2, k_3} \) \( (x_1, x_2, \ldots, x_n) \) be a voting function?

2.12 If \( f(x_1, x_2, \ldots, x_n) \) is a threshold function, is the dual of \( f \) always a threshold function?
2.13 a) If \( f(x_1, x_2, ..., x_n) \) is a threshold function, and \( x_p \) is not a member of the set \( \{x_1, x_2, ..., x_n\} \) which, if any of the following are threshold functions?

\[ \bar{x}_p f, \bar{x}_p \bar{f}, x_p \bar{f} \]

b) Repeat (a) if \( x_p \) is a member of the set \( \{x_1, x_2, ..., x_n\} \).

2.14 A function \( f(x) \) is said to be dual comparable if \( f \), the dual of \( f \), is such that \( f + f = f \) or \( f + f = \bar{f} \).

a) Are all threshold functions dual comparable?

b) Are all dual comparable functions threshold functions?

2.15 How many symmetric functions of \( n \) variables are there?

2.16 How many unate functions of \( n \) variables are there?

2.17 Let \( f(x_1, x_2, ..., x_n) \) be a Boolean function, where the values of the variables are either 0 or 1 and where there exists a set of weights \( w_1, w_2, ..., w_n \) \((w_i = +1 \text{ or } -1, 1 \leq i \leq n)\) and a real number \( T \) such that

\[ f(x_1, x_2, ..., x_n) = 1 \text{ if } \sum_{i=1}^{n} w_i x_i \geq T \text{ and} \]

\[ f(x_1, x_2, ..., x_n) = 0 \text{ if } \sum_{i=1}^{n} w_i x_i < T. \]

This function is obviously a special type of threshold function. Is \( f(x_1, x_2, ..., x_n) \) also a totally symmetric function? If so, prove, and indicate the variables of symmetry and the value of \( T \). If not, display some such \( f(x_1, x_2, ..., x_n) \) which is not totally symmetric.

2.18 The Fibonacci numbers are a sequence defined by the relation

\[ F_n = F_{n-1} + F_{n-2} \]

where \( F_1 \) is the \( i \)th number in the sequence. The first numbers in the sequence are 1, 1, 2, 3, 5, 8, ...

Consider the set of functions

\[ F = x_1 + x_{n-1} (x_{n-2} + x_{n-3}) + x_{n-5} (\ldots + x_2 (x_1)) \ldots \]

(n odd)

Show that all functions in this set are threshold functions with weights \( w_n = F_n \) (the \( n \)th Fibonacci number)

and find the value of the threshold \( T \).

Also show that this is true for the dual of \( f \) and find the value of \( T \).

2.19 (a) Prove that for a 3-variable function \( f(x_1, x_2, x_3) \) if \( f \) is non 0-preserving, non 1-preserving and non self-dual then \( f \) is non-linear.

(b) Use (a) to determine how many 3-variable functions \( f \) are such that \( \{f\} \) is strong complete.

2.20 For each of the following sets of 3-variable functions determine whether it is strong complete, weak complete, a strong basis, a weak basis:

(a) \( \{f_1, f_2, f_3, f_4\} \) where \( f_2 = \Sigma m_0, m_1, m_2, m_3, m_4 \)

(b) \( \{f_1, f_2\} \) where \( f_1 = \Sigma m_0, m_1, m_5 \)

(c) \( \{f_1, f_2, f_3\} \) where \( f_3 = \Sigma m_0, m_1, m_4 \).

2.21 A 4-input module realizing a function \( f(x_1, x_2, x_3, x_4) \) is to be designed and must satisfy the following constraints:

(i) \( f(0,0,0,0) = \bar{x}_2 + x_1 \)

(ii) \( f(0,1,0,1) = \bar{x}_4 \)

(iii) \( f(1,0,1,0) = \bar{x}_2 \)

(iv) \( f(1,1,1,0) = 1 \).

(a) Is \( f \) weak complete?

(b) Is \( f \) strong complete?

(c) If possible, specify a function \( f \) which is strong complete and satisfies these conditions and use \( f \) to realize the function \( g = x_1 \bar{x}_2 + \bar{x}_1 x_2 \).

2.22 A set of functions \( F \) is strong c-complete if using inputs \( x_1 \) and \( \bar{x}_1 \), all functions can be realized and \( F \) is weak c-complete if using inputs \( x_1, \bar{x}_1, 0, 1 \) all functions can be realized. Determine necessary and sufficient conditions for strong c-completeness and weak c-completeness.