Digital Logic and Computer Design

M. MORRIS MANO

Professor of Engineering
California State University, Los Angeles

Prentice-Hall, Inc., Englewood Cliffs, N.J. 07632

Library of Congress Cataloging in Publication Data

MANO, M. MORRIS (date)
Digital logic and computer design.

Bibliography: p.
Includes index.
1.–Electronic digital computers. 2.–Logic circuits. 3.–Digital integrated circuits. 4.–Logic design. I.–Title.
TK7888.3.M345 621.3815'3 78-21462
ISBN 0-13-214510-3

Printed in the United States of America

10 9 8 7

Editorial/Production Supervision by Lynn S. Frankel
Cover Design by Edsal Enterprise
Manufacturing Buyer: Gordon Osbourne

PRENTICE-HALL INTERNATIONAL, INC., *London*
PRENTICE-HALL OF AUSTRALIA PTY. LIMITED, *Sydney*
PRENTICE-HALL OF CANADA, LTD., *Toronto*
PRENTICE-HALL OF INDIA PRIVATE LIMITED, *New Delhi*
PRENTICE-HALL OF JAPAN, INC., *Tokyo*
PRENTICE-HALL OF SOUTHEAST ASIA PTE. LTD., *Singapore*
WHITEHALL BOOKS LIMITED, *Wellington, New Zealand*

Contents

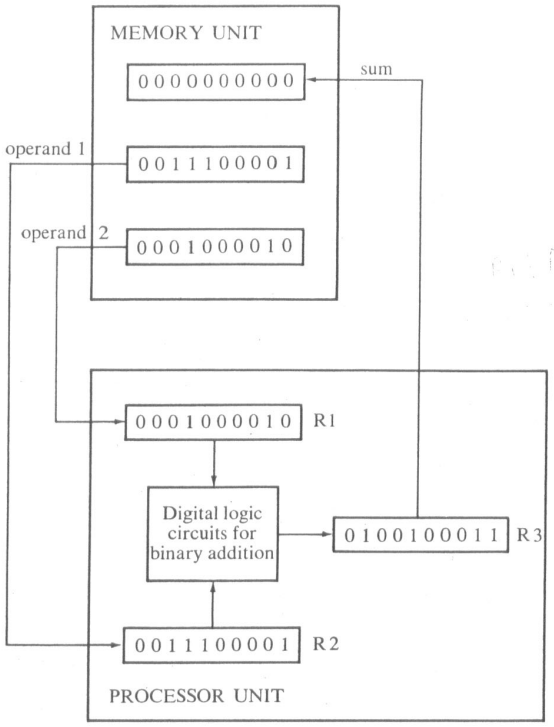


Figure 1-3 Example of binary information processing

processor registers can be transferred back into a memory register for storage until needed again. The diagram shows the contents of two operands transferred from two memory registers into R1 and R2. The digital logic circuits produce the sum, which is transferred to register R3. The contents of R3 can now be transferred back to one of the memory registers.

The last two examples demonstrated the information flow capabilities of a digital system in a very simple manner. The registers of the system are the basic elements for storing and holding the binary information. The digital logic circuits process the information. Digital logic circuits and their manipulative capabilities are introduced in the next section. The subject of registers and register transfer operations is taken up again in Chapter 8.

1-8 BINARY LOGIC

Binary logic deals with variables that take on two discrete values and with operations that assume logical meaning. The two values the variables take may be called by different names (e.g., *true* and *false*, *yes* and *no*, etc.), but for our purpose it is convenient to think in terms of bits and assign the values of 1 and 0. Binary

logic is used to describe, in a mathematical way, the manipulation and processing of binary information. It is particularly suited for the analysis and design of digital systems. For example, the digital logic circuits of Fig. 1-3 that perform the binary arithmetic are circuits whose behavior is most conveniently expressed by means of binary variables and logical operations. The binary logic to be introduced in this section is equivalent to an algebra called Boolean algebra. The formal presentation of a two-valued Boolean algebra is covered in more detail in Chapter 2. The purpose of this section is to introduce Boolean algebra in a heuristic manner and relate it to digital logic circuits and binary signals.

Definition of Binary Logic

Binary logic consists of binary variables and logical operations. The variables are designated by letters of the alphabet such as A, B, C, x, y, z, etc., with each variable having two and only two distinct possible values: 1 and 0. There are three basic logical operations: AND, OR, and NOT.

1. AND: This operation is represented by a dot or by the absence of an operator. For example, $x \cdot y = z$ or $xy = z$ is read "x AND y is equal to z." The logical operation AND is interpreted to mean that $z = 1$ if and only if $x = 1$ *and* $y = 1$; otherwise $z = 0$. (Remember that x, y, and z are binary variables and can be equal either to 1 or 0, and nothing else.)
2. OR: This operation is represented by a plus sign. For example, $x + y = z$ is read "x OR y is equal to z," meaning that $z = 1$ if $x = 1$ *or* if $y = 1$ *or* if both $x = 1$ and $y = 1$. If both $x = 0$ and $y = 0$, then $z = 0$.
3. NOT: This operation is represented by a prime (sometimes by a bar). For example, $x' = z$ (or $\bar{x} = z$) is read "x not is equal to z," meaning that z is what x is not. In other words, if $x = 1$, then $z = 0$; but if $x = 0$, then $z = 1$.

Binary logic resembles binary arithmetic, and the operations AND and OR have some similarities to multiplication and addition, respectively. In fact, the symbols used for AND and OR are the same as those used for multiplication and addition. However, binary logic should not be confused with binary arithmetic. One should realize that an arithmetic variable designates a number that may consist of many digits. A logic variable is always either a 1 or a 0. For example, in binary arithmetic we have $1 + 1 = 10$ (read: "one plus one is equal to 2"), while in binary logic we have $1 + 1 = 1$ (read: "one OR one is equal to one").

For each combination of the values of x and y, there is a value of z specificed by the definition of the logical operation. These definitions may be listed in a compact form using *truth tables*. A truth table is a table of all possible combinations of the variables showing the relation between the values that the variables may take and the result of the operation. For example, the truth tables for the

TABLE 1-6 Truth tables of logical operations

AND			OR			NOT	
x	y	$x \cdot y$	x	y	$x + y$	x	x'
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

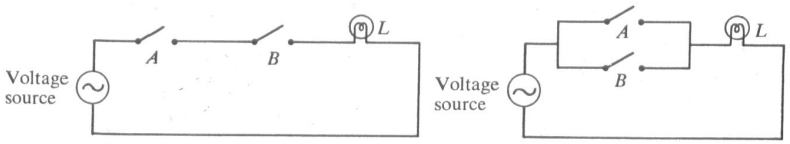
operations AND and OR with variables x and y are obtained by listing all possible values that the variables may have when combined in pairs. The result of the operation for each combination is then listed in a separate row. The truth tables for AND, OR, and NOT are listed in Table 1-6. These tables clearly demonstrate the definitions of the operations.

Switching Circuits and Binary Signals

The use of binary variables and the application of binary logic are demonstrated by the simple switching circuits of Fig. 1-4. Let the manual switches A and B represent two binary variables with values equal to 0 when the switch is open and 1 when the switch is closed. Similarly, let the lamp L represent a third binary variable equal to 1 when the light is on and 0 when off. For the switches in series, the light turns on if *A and B* are closed. For the switches in parallel, the light turns on if *A or B* is closed. It is obvious that the two circuits can be expressed by means of binary logic with the AND and OR operations, respectively:

$$L = A \cdot B \qquad \text{for the circuit of Fig. 1-4(a)}$$
$$L = A + B \qquad \text{for the circuit of Fig. 1-4(b)}$$

Electronic digital circuits are sometimes called *switching circuits* because they behave like a switch, with the active element such as a transistor either conducting (switch closed) or not conducting (switch open). Instead of changing the switch manually, an electronic switching circuit uses binary signals to control the conduction or nonconduction state of the active element. Electrical signals such as



(a) Switches in series – logic AND (b) Switches in parallel – logic OR

Figure 1-4 Switching circuits that demonstrate binary logic

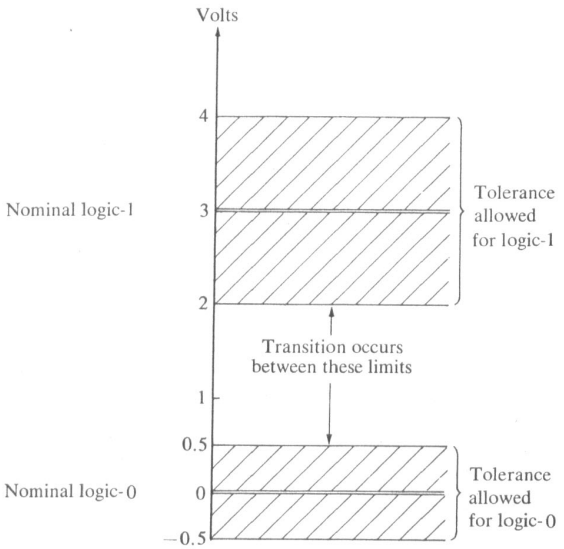
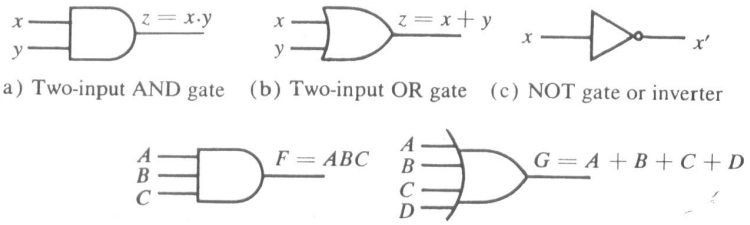


Figure 1-5 Example of binary signals

voltages or currents exist throughout a digital system in either one of two recognizable values (except during transition). Voltage-operated circuits, for example, respond to two separate voltage levels which represent a binary variable equal to logic-1 or logic-0. For example, a particular digital system may define logic-1 as a signal with a nominal value of 3 volts, and logic-0 as a signal with a nominal value of 0 volt. As shown in Fig. 1-5, each voltage level has an acceptable deviation from the nominal. The intermediate region between the allowed regions is crossed only during state transitions. The input terminals of digital circuits accept binary signals within the allowable tolerances and respond at the output terminal with binary signals that fall within the specified tolerances.

Logic Gates

Electronic digital circuits are also called *logic circuits* because, with the proper input, they establish logical manipulation paths. Any desired information for computing or control can be operated upon by passing binary signals through various combinations of logic circuits, each signal representing a variable and carrying one bit of information. Logic circuits that perform the logical operations of AND, OR, and NOT are shown with their symbols in Fig. 1-6. These circuits, called *gates*, are blocks of hardware that produce a logic-1 or logic-0 output signal if input logic requirements are satisfied. Note that four different names have been used for the same type of circuits: digital circuits, switching circuits, logic circuits, and gates. All four names are widely used, but we shall refer to the circuits as



(a) Two-input AND gate (b) Two-input OR gate (c) NOT gate or inverter

(d) Three-input AND gate (e) Four-input OR gate

Figure 1-6 Symbols for digital logic circuits

AND, OR, and NOT gates. The NOT gate is sometimes called an *inverter circuit* since it inverts a binary signal.

The input signals x and y in the two-input gates of Fig. 1-6 may exist in one of four possible states: 00, 10, 11, or 01. These input signals are shown in Fig. 1-7, together with the output signals for the AND and OR gates. The timing diagrams in Fig. 1-7 illustrate the response of each circuit to each of the four possible input binary combinations. The reason for the name "inverter" for the NOT gate is apparent from a comparison of the signal x (input of inverter) and that of x' (output of inverter).

AND and OR gates may have more than two inputs. An AND gate with three inputs and an OR gate with four inputs are shown in Fig. 1-6. The three-input AND gate responds with a logic-1 output if all three input signals are logic-1. The output produces a logic-0 signal if any input is logic 0. The four input OR gate responds with a logic-1 when any input is a logic-1. Its output becomes logic-0 if all input signals are logic-0.

The mathematical system of binary logic is better known as Boolean, or switching, algebra. This algebra is conveniently used to describe the operation of complex networks of digital circuits. Designers of digital systems use Boolean algebra to transform circuit diagrams to algebraic expressions and vice versa. Chapters 2 and 3 are devoted to the study of Boolean algebra, its properties, and manipulative capabilities. Chapter 4 shows how Boolean algebra may be used to express mathematically the interconnections among networks of gates.

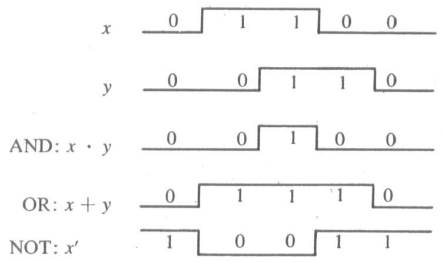


Figure 1-7 Input-output signals for gates (a), (b), and (c) of Fig. 1-6

6. Postulate 6 is satisfied because the two-valued Boolean algebra has two distinct elements 1 and 0 with $1 \neq 0$.

We have just established a two-valued Boolean algebra having a set of two elements, 1 and 0, two binary operators with operation rules equivalent to the AND and OR operations, and a complement operator equivalent to the NOT operator. Thus, Boolean algebra has been defined in a formal mathematical manner and has been shown to be equivalent to the binary logic presented heuristically in Section 1-8. The heuristic presentation is helpful in understanding the application of Boolean algebra to gate-type circuits. The formal presentation is necessary for developing the theorems and properties of the algebraic system. The two-valued Boolean algebra defined in this section is also called "switching algebra" by engineers. To emphasize the similarities between two-valued Boolean algebra and other binary systems, this algebra was called "binary logic" in Section 1-8. From here on, we shall drop the adjective "two-valued" from Boolean algebra in subsequent discussions.

2-3 BASIC THEOREMS AND PROPERTIES OF BOOLEAN ALGEBRA

Duality

The Huntington postulates have been listed in pairs and designated by part (a) and part (b). One part may be obtained from the other if the binary operators and the identity elements are interchanged. This important property of Boolean algebra is called the *duality principle*. It states that every algebraic expression deducible from the postulates of Boolean algebra remains valid if the operators and identity elements are interchanged. In a two-valued Boolean algebra, the identity elements and the elements of the set B are the same: 1 and 0. The duality principle has many applications. If the *dual* of an algebraic expression is desired, we simply interchange OR and AND operators and replace 1's by 0's and 0's by 1's.

Basic Theorems

Table 2-1 lists six theorems of Boolean algebra and four of its postulates. The notation is simplified by omitting the · whenever this does not lead to confusion. The theorems and postulates listed are the most basic relationships in Boolean algebra. The reader is advised to become familiar with them as soon as possible. The theorems, like the postulates, are listed in pairs; each relation is the dual of the one paired with it. The postulates are basic axioms of the algebraic structure and need no proof. The theorems must be proven from the postulates. The proofs of the theorems with one variable are presented below. At the right is listed the number of the postulate which justifies each step of the proof.

TABLE 2-1 Postulates and theorems of Boolean algebra

Postulate 2	(a) $x + 0 = x$	(b) $x \cdot 1 = x$
Postulate 5	(a) $x + x' = 1$	(b) $x \cdot x' = 0$
Theorem 1	(a) $x + x = x$	(b) $x \cdot x = x$
Theorem 2	(a) $x + 1 = 1$	(b) $x \cdot 0 = 0$
Theorem 3, involution	$(x')' = x$	
Postulate 3, commutative	(a) $x + y = y + x$	(b) $xy = yx$
Theorem 4, associative	(a) $x + (y + z) = (x + y) + z$	(b) $x(yz) = (xy)z$
Postulate 4, distributive	(a) $x(y + z) = xy + xz$	(b) $x + yz = (x + y)(x + z)$
Theorem 5, DeMorgan	(a) $(x + y)' = x'y'$	(b) $(xy)' = x' + y'$
Theorem 6, absorption	(a) $x + xy = x$	(b) $x(x + y) = x$

THEOREM 1(a): $x + x = x$.

$x + x = (x + x) \cdot 1$	by postulate: 2(b)
$= (x + x)(x + x')$	5(a)
$= x + xx'$	4(b)
$= x + 0$	5(b)
$= x$	2(a)

THEOREM 1(b): $x \cdot x = x$.

$x \cdot x = xx + 0$	by postulate: 2(a)
$= xx + xx'$	5(b)
$= x(x + x')$	4(a)
$= x \cdot 1$	5(a)
$= x$	2(b)

Note that theorem 1(b) is the dual of theorem 1(a) and that each step of the proof in part (b) is the dual of part (a). Any dual theorem can be similarly derived from the proof of its corresponding pair.

THEOREM 2(a): $x + 1 = 1$.

$x + 1 = 1 \cdot (x + 1)$	by postulate: 2(b)
$= (x + x')(x + 1)$	5(a)
$= x + x' \cdot 1$	4(b)
$= x + x'$	2(b)
$= 1$	5(a)

THEOREM 2(b): $x \cdot 0 = 0$ by duality.

THEOREM 3: $(x')' = x$. From postulate 5, we have $x + x' = 1$ and $x \cdot x' = 0$, which defines the complement of x. The complement of x' is x and is also $(x')'$. Therefore, since the complement is unique, we have that $(x')' = x$.

The theorems involving two or three variables may be proven algebraically from the postulates and the theorems which have already been proven. Take, for example, the absorption theorem.

THEOREM 6(a): $x + xy = x$.

$$
\begin{aligned}
x + xy &= x \cdot 1 + xy && \text{by postulate 2(b)} \\
&= x(1 + y) && \text{by postulate 4(a)} \\
&= x(y + 1) && \text{by postulate 3(a)} \\
&= x \cdot 1 && \text{by theorem 2(a)} \\
&= x && \text{by postulate 2(b)}
\end{aligned}
$$

THEOREM 6(b): $x(x + y) = x$ by duality.

The theorems of Boolean algebra can be shown to hold true by means of truth tables. In truth tables, both sides of the relation are checked to yield identical results for all possible combinations of variables involved. The following truth table verifies the first absorption theorem.

			=
x	y	xy	$x + xy$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

The algebraic proofs of the associative law and De Morgan's theorem are long and will not be shown here. However, their validity is easily shown with truth tables. For example, the truth table for the first De Morgan's theorem $(x + y)' = x'y'$ is shown below.

x	y	$x + y$	$(x + y)'$	x'	y'	$x'y'$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

Operator Precedence

The operator precedence for evaluating Boolean expressions is (1) parentheses, (2) NOT, (3) AND, and (4) OR. In other words, the expression inside the parentheses must be evaluated before all other operations. The next operation that holds precedence is the complement, then follows the AND, and finally the OR. As an example, consider the truth table for De Morgan's theorem. The left side of the

expression is $(x + y)'$. Therefore, the expression inside the parentheses is evaluated first and the result then complemented. The right side of the expression is $x'y'$. Therefore, the complement of x and the complement of y are both evaluated first and the result is then ANDed. Note that in ordinary arithmetic the same precedence holds (except for the complement) when multiplication and addition are replaced by AND and OR, respectively.

Venn Diagram

A helpful illustration that may be used to visualize the relationships among the variables of a Boolean expression is the *Venn diagram*. This diagram consists of a rectangle such as shown in Fig. 2-1, inside of which are drawn overlapping circles, one for each variable. Each circle is labeled by a variable. We designate all points inside a circle as belonging to the named variable and all points outside a circle as not belonging to the variable. Take, for example, the circle labeled x. If we are inside the circle, we say that $x = 1$; when outside, we say $x = 0$. Now, with two overlapping circles, there are four distinct areas inside the rectangle: the area not belonging to either x or y $(x'y')$, the area inside circle y but outside x $(x'y)$, the area inside circle x but outside y (xy'), and the area inside both circles (xy).

Venn diagrams may be used to illustrate the postulates of Boolean algebra or to show the validity of theorems. Figure 2-2, for example, illustrates that the area belonging to xy is inside the circle x and therefore $x + xy = x$. Figure 2-3 illustrates the distributive law $x(y + z) = xy + xz$. In this diagram we have three overlapping circles, one for each of the variables x, y, and z. It is possible to distinguish eight distinct areas in a three-variable Venn diagram. For this particular example, the distributive law is demonstrated by noting that the area intersecting the circle x with the area enclosing y or z is the same area belonging to xy or xz.

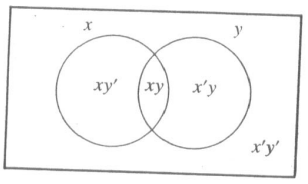


Figure 2-1 Venn diagram for two variables

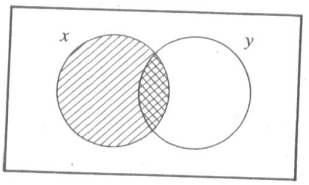


Figure 2-2 Venn diagram illustration $x = xy + x$

able case and can be derived by successive substitutions similar to the method used in the above derivation. These theorems can be generalized as follows:

$$(A + B + C + D + \cdots + F)' = A'B'C'D' \ldots F'$$
$$(ABCD \cdots F)' = A' + B' + C' + D' + \ldots + F'$$

The generalized form of De Morgan's theorem states that the complement of a function is obtained by interchanging AND and OR operators and complementing each literal.

EXAMPLE 2-2: Find the complement of the functions $F_1 = x'yz' + x'y'z$ and $F_2 = x(y'z' + yz)$. Applying De Morgan's theorem as many times as necessary, the complements are obtained as follows:

$$F_1' = (x'yz' + x'y'z)' = (x'yz')'(x'y'z)' = (x + y' + z)(x + y + z')$$
$$F_2' = [x(y'z' + yz)]' = x' + (y'z' + yz)' = x' + (y'z')' \cdot (yz)'$$
$$= x' + (y + z)(y' + z')$$

A simpler procedure for deriving the complement of a function is to take the dual of the function and complement each literal. This method follows from the generalized De Morgan's theorem. Remember that the dual of a function is obtained from the interchange of AND and OR operators and 1's and 0's.

EXAMPLE 2-3: Find the complement of the functions F_1 and F_2 of Example 2-2 by taking their duals and complementing each literal.

1. $F_1 = x'yz' + x'y'z$.
   The dual of F_1 is $(x' + y + z')(x' + y' + z)$.
   Complement each literal: $(x + y' + z)(x + y + z') = F_1'$.
2. $F_2 = x(y'z' + yz)$.
   The dual of F_2 is $x + (y' + z')(y + z)$.
   Complement each literal: $x' + (y + z)(y' + z') = F_2'$.

2-5 CANONICAL AND STANDARD FORMS

Minterms and Maxterms

A binary variable may appear either in its normal form (x) or in its complement form (x'). Now consider two binary variables x and y combined with an AND operation. Since each variable may appear in either form, there are four possible

TABLE 2-3 Minterms and maxterms for three binary variables

			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

combinations: $x'y'$, $x'y$, xy', and xy. Each of these four AND terms represents one of the distinct areas in the Venn diagram of Fig. 2-1 and is called a *minterm* or a *standard product*. In a similar manner, n variables can be combined to form 2^n minterms. The 2^n different minterms may be determined by a method similar to the one shown in Table 2-3 for three variables. The binary numbers from 0 to $2^n - 1$ are listed under the n variables. Each minterm is obtained from an AND term of the n variables, with each variable being primed if the corresponding bit of the binary number is a 0 and unprimed if a 1. A symbol for each minterm is also shown in the table and is of the form m_j, where j denotes the decimal equivalent of the binary number of the minterm designated.

In a similar fashion, n variables forming an OR term, with each variable being primed or unprimed, provide 2^n possible combinations, called *maxterms* or *standard sums*. The eight maxterms for three variables, together with their symbolic designation, are listed in Table 2-3. Any 2^n maxterms for n variables may be determined similarly. Each maxterm is obtained from an OR term of the n variables, with each variable being unprimed if the corresponding bit is a 0 and primed if a 1.* Note that each maxterm is the complement of its corresponding minterm, and vice versa.

A Boolean function may be expressed algebraically from a given truth table by forming a minterm for each combination of the variables which produces a 1 in the function, and then taking the OR of all those terms. For example, the function f_1 in Table 2-4 is determined by expressing the combinations 001, 100, and 111 as $x'y'z$, $xy'z'$, and xyz, respectively. Since each one of these minterms results in $f_1 = 1$, we should have:

$$f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$$

*Some books define a maxterm as an OR term of the n variables, with each variable being unprimed if the bit is a 1 and primed if a 0. The definition adopted in this book is preferable as it leads to simpler conversions between maxterm- and minterm-type functions.

TABLE 2-4 Functions of three variables

x	y	z	Function f_1	Function f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Similarly, it may be easily verified that:

$$f_2 = x'yz + xy'z + xyz' + xyz = m_3 + m_5 + m_6 + m_7$$

These examples demonstrate an important property of Boolean algebra: Any Boolean function can be expressed as a sum of minterms (by "sum" is meant the ORing of terms).

Now consider the complement of a Boolean function. It may be read from the truth table by forming a minterm for each combination that produces a 0 in the function and then ORing those terms. The complement of f_1 is read as:

$$f_1' = x'y'z' + x'yz' + x'yz + xy'z + xyz'$$

If we take the complement of f_1', we obtain the function f_1:

$$\begin{aligned} f_1 &= (x + y + z)(x + y' + z)(x + y' + z')(x' + y + z')(x' + y' + z) \\ &= M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6 \end{aligned}$$

Similarly, it is possible to read the expression for f_2 from the table:

$$\begin{aligned} f_2 &= (x + y + z)(x + y + z')(x + y' + z)(x' + y + z) \\ &= M_0 M_1 M_2 M_4 \end{aligned}$$

These examples demonstrate a second important property of Boolean algebra: Any Boolean function can be expressed as a product of maxterms (by "product" is meant the ANDing of terms). The procedure for obtaining the product of maxterms directly from the truth table is as follows. Form a maxterm for each combination of the variables which produces a 0 in the function, and then form the AND of all those maxterms. Boolean functions expressed as a sum of minterms or product of maxterms are said to be in *canonical form.*

Sum of Minterms

It was previously stated that for n binary variables, one can obtain 2^n distinct minterms, and that any Boolean function can be expressed as a sum of minterms. The minterms whose sum defines the Boolean function are those that give the 1's of the function in a truth table. Since the function can be either 1 or 0 for each minterm, and since there are 2^n minterms, one can calculate the possible functions that can be formed with n variables to be 2^{2^n}. It is sometimes convenient to express the Boolean function in its sum-of-minterms form. If not in this form, it can be made so by first expanding the expression into a sum of AND terms. Each term is then inspected to see if it contains all the variables. If it misses one or more variables, it is ANDed with an expression such as $x + x'$, where x is one of the missing variables. The following example clarifies this procedure.

EXAMPLE 2-4: Express the Boolean function $F = A + B'C$ in a sum of minterms. The function has three variables A, B, and C. The first term A is missing two variables; therefore:

$$A = A(B + B') = AB + AB'$$

This is still missing one variable:

$$\begin{aligned} A &= AB(C + C') + AB'(C + C') \\ &= ABC + ABC' + AB'C + AB'C' \end{aligned}$$

The second term $B'C$ is missing one variable:

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

Combining all terms, we have:

$$\begin{aligned} F &= A + B'C \\ &= ABC + ABC' + AB'C + AB'C' + AB'C + A'B'C \end{aligned}$$

But $AB'C$ appears twice, and according to theorem 1 ($x + x = x$), it is possible to remove one of them. Rearranging the minterms in ascending order, we finally obtain:

$$\begin{aligned} F &= A'B'C + AB'C' + AB'C + ABC' + ABC \\ &= m_1 + m_4 + m_5 + m_6 + m_7 \end{aligned}$$

It is sometimes convenient to express the Boolean function, when in its sum of minterms, in the following short notation:

$$F(A, B, C) = \Sigma\,(1, 4, 5, 6, 7)$$

The summation symbol Σ stands for the ORing of terms; the numbers following it are the minterms of the function. The letters in parentheses following F form a list of the variables in the order taken when the minterm is converted to an AND term.

Product of Maxterms

Each of the 2^{2^n} functions of n binary variables can be also expressed as a product of maxterms. To express the Boolean function as a product of maxterms, it must first be brought into a form of OR terms. This may be done by using the distributive law $x + yz = (x + y)(x + z)$. Then any missing variable x in each OR term is ORed with xx'. This procedure is clarified by the following example.

EXAMPLE 2-5: Express the Boolean function $F = xy + x'z$ in a product of maxterm form. First convert the function into OR terms using the distributive law:

$$\begin{aligned} F &= xy + x'z = (xy + x')(xy + z) \\ &= (x + x')(y + x')(x + z)(y + z) \\ &= (x' + y)(x + z)(y + z) \end{aligned}$$

The function has three variables: x, y, and z. Each OR term is missing one variable; therefore:

$$\begin{aligned} x' + y &= x' + y + zz' = (x' + y + z)(x' + y + z') \\ x + z &= x + z + yy' = (x + y + z)(x + y' + z) \\ y + z &= y + z + xx' = (x + y + z)(x' + y + z) \end{aligned}$$

Combining all the terms and removing those that appear more than once, we finally obtain:

$$\begin{aligned} F &= (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z') \\ &= M_0M_2M_4M_5 \end{aligned}$$

A convenient way to express this function is as follows:

$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

The product symbol, Π, denotes the ANDing of maxterms; the numbers are the maxterms of the function.

Conversion between Canonical Forms

The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function. This is because the original function is expressed by those minterms that make the function equal to 1, while its

complement is a 1 for those minterms that the function is a 0. As an example, consider the function:

$$F(A, B, C) = \Sigma(1, 4, 5, 6, 7)$$

This has a complement that can be expressed as:

$$F'(A, B, C) = \Sigma(0, 2, 3) = m_0 + m_2 + m_3$$

Now, if we take the complement of F' by De Morgan's theorem, we obtain F in a different form:

$$F = (m_0 + m_2 + m_3)' = m_0' \cdot m_2' \cdot m_3' = M_0 M_2 M_3 = \Pi(0, 2, 3)$$

The last conversion follows from the definition of minterms and maxterms as shown in Table 2-3. From the table, it is clear that the following relation holds true:

$$m_j' = M_j$$

That is, the maxterm with subscript j is a complement of the minterm with the same subscript j, and vice versa.

The last example demonstrates the conversion between a function expressed in sum of minterms and its equivalent in product of maxterms. A similar argument will show that the conversion between the product of maxterms and the sum of minterms is similar. We now state a general conversion procedure. To convert from one canonical form to another, interchange the symbols Σ and Π and list those numbers missing from the original form. As another example, the function:

$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

is expressed in the product of maxterm form. Its conversion to sum of minterms is:

$$F(x, y, z) = \Sigma(1, 3, 6, 7)$$

Note that, in order to find the missing terms, one must realize that the total number of minterms or maxterms is 2^n, where n is the number of binary variables in the function.

Standard Forms

The two canonical forms of Boolean algebra are basic forms that one obtains from reading a function from the truth table. These forms are very seldom the ones with the least number of literals, because each minterm or maxterm must contain, by definition, *all* the variables either complemented or uncomplemented.

Another way to express Boolean functions is in *standard* form. In this configuration, the terms that form the function may contain one, two or any

number of literals. There are two types of standard forms: the sum of products and product of sums.

The *sum of products* is a Boolean expression containing AND terms, called *product terms*, of one or more literals each. The *sum* denotes the ORing of these terms. An example of a function expressed in sum of products is:

$$F_1 = y' + xy + x'yz'$$

The expression has three product terms of one, two, and three literals each, respectively. Their sum is in effect an OR operation.

A *product of sums* is a Boolean expression containing OR terms, called *sum terms*. Each term may have any number of literals. The *product* denotes the ANDing of these terms. An example of a function expressed in product of sums is:

$$F_2 = x(y' + z)(x' + y + z' + w)$$

This expression has three sum terms of one, two, and four literals each. The product is an AND operation. The use of the words *product* and *sum* stems from the similarity of the AND operation to the arithmetic product (multiplication) and the similarity of the OR operation to the arithmetic sum (addition).

A Boolean function may be expressed in a nonstandard form. For example, the function:

$$F_3 = (AB + CD)(A'B' + C'D')$$

is neither in sum of products nor in product of sums. It can be changed to a standard form by using the distributive law to remove the parentheses:

$$F_3 = A'B'CD + ABC'D'$$

2-6 OTHER LOGIC OPERATIONS

When the binary operators AND and OR are placed between two variables x and y, they form two Boolean functions $x \cdot y$ and $x + y$, respectively. It was stated previously that there are 2^{2^n} functions for n binary variables. For two variables, $n = 2$ and the number of possible Boolean functions is 16. Therefore, the AND and OR functions are only two of a total of 16 possible functions formed with two binary variables. It would be instructive to find the other 14 functions and investigate their properties.

The truth tables for the 16 functions formed with two binary variables x and y are listed in Table 2-5. In this table, each of the 16 columns F_0 to F_{15} represents a truth table of one possible function for the two given variables x and y. Note that the functions are determined from the 16 binary combinations that can be assigned to F. Some of the functions are shown with an operator symbol. For example, F_1

Copyright United States Copyright Office

Help | Search | History | Titles | Start Over

Public Catalog

Copyright Catalog (1978 to present)

Search Request: Left Anchored Title = digital logic and computer design

Search Results: Displaying 1 of 2 entries

previous next

Labeled View

Digital logic and computer design / M. Morris Mano.

Type of Work: Text
Registration Number / Date: TX0000258870 / 1979-06-01
Title: Digital logic and computer design / M. Morris Mano.
Imprint: Englewood Cliffs, N. J. : Prentice-Hall, c1979.
Description: 612 p.
Copyright Claimant: Prentice-Hall, Inc.
Date of Creation: 1978
Date of Publication: 1979-03-12
ISBN: 0132145103
Names: Mano, M. Morris
Prentice-Hall, Inc.

previous next

Save, Print and Email (Help Page)

Select Download Format Full Record Format for Print/Save

Enter your email address: Email

Help Search History Titles Start Over

Contact Us | Request Copies | Get a Search Estimate | Frequently Asked Questions (FAQs) about Copyright | Copyright Office Home Page | Library of Congress Home Page