Boolean Algebra and Logic Gates

2-1 BASIC DEFINITIONS

Boolean algebra, like any other deductive mathematical system, may be defined with a set of elements, a set of operators, and a number of unproved axioms or postulates. A *set* of elements is any collection of objects having a common property. If S is a set, and x and y are certain objects, then $x \in S$ denotes that x is a member of the set S, and $y \notin S$ denotes that y is not an element of S. A set with a denumerable number of elements is specified by braces: $A = \{1, 2, 3, 4\}$, i.e., the elements of set A are the numbers 1, 2, 3, and 4. A *binary operator* defined on a set S of elements is a rule that assigns to each pair of elements from S a unique element from S. As an example, consider the relation $a * b = c$. We say that $*$ is a binary operator if it specifies a rule for finding c from the pair (a, b) and also if $a, b, c \in S$. However, $*$ is not a binary operator if $a, b \in S$, whereas the rule finds $c \notin S$.

The postulates of a mathematical system form the basic assumptions from which it is possible to deduce the rules, theorems, and properties of the system. The most common postulates used to formulate various algebraic structures are:

1. *Closure.* A set S is closed with respect to a binary operator if, for every pair of elements of S, the binary opertor specifies a rule for obtaining a unique element of S. For example, the set of natural numbers $N = \{1, 2, 3, 4, \ldots\}$ is closed with respect to the binary operator plus $(+)$ by the rules of arithmetic addition, since for any $a, b \in N$ we obtain a unique $c \in N$ by the operation $a + b = c$. The set of natural numbers is not closed with respect to the binary operator minus $(-)$ by the rules of arithmetic subtraction because $2 - 3 = -1$ and $2, 3 \in N$, while $(-1) \notin N$.

2. *Associative law.* A binary operator $*$ on a set S is said to be associative whenever

$$(x * y) * z = x * (y * z) \qquad \text{for all } x, y, z, \in S$$

3. *Commutative law.* A binary operator $*$ on a set S is said to be commutative whenever

$$x * y = y * x \qquad \text{for all } x, y \in S$$

4. *Identity element.* A set S is said to have an identity element with respect to a binary operation $*$ on S if there exists an element $e \in S$ with the property

$$e * x = x * e = x \qquad \text{for every } x \in S$$

Example: The element 0 is an identity element with respect to operation $+$ on the set of integers $I = \{ \ldots, -3, -2, -1, 0, 1, 2, 3, \ldots \}$ since

$$x + 0 = 0 + x = x \qquad \text{for any } x \in I$$

The set of natural numbers N has no identity element since 0 is excluded from the set.

5. *Inverse.* A set S having the identity element e with respect to a binary operator $*$ is said to have an inverse whenever, for every $x \in S$, there exists an element $y \in S$ such that

$$x * y = e$$

Example: In the set of integers I with $e = 0$, the inverse of an element a is $(-a)$ since $a + (-a) = 0$.

6. *Distributive law.* If $*$ and $\cdot$ are two binary operators on a set S, $*$ is said to be distributive over $\cdot$ whenever

$$x * (y \cdot z) = (x * y) \cdot (x * z)$$

An example of an algebraic structure is a *field.* A field is a set of elements, together with two binary operators, each having properties 1 to 5 and both operators combined to give property 6. The set of real numbers together with the binary operators $+$ and $\cdot$ form the field of real numbers. The field of real numbers is the basis for arithmetic and ordinary algebra. The operators and postulates have the following meanings:

The binary operator $+$ defines addition.
The additive identity is 0.
The additive inverse defines subtraction.
The binary operator $\cdot$ defines multiplication.
The multiplicative identity is 1.
The multiplicative inverse of $a = 1/a$ defines division, i.e., $a \cdot 1/a = 1$.
The only distributive law applicable is that of $\cdot$ over $+$:

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$$

2-2 AXIOMATIC DEFINITION OF BOOLEAN ALGEBRA

In 1854 George Boole introduced a systematic treatment of logic and developed for this purpose an algebraic system now called *Boolean algebra*. In 1938 C. E. Shannon introduced a two-valued Boolean algebra called *switching algebra*, in which he demonstrated that the properties of bistable electrical switching circuits can be represented by this algebra. For the formal definition of Boolean algebra, we shall employ the postulates formulated by E. V. Huntington in 1904.

Boolean algebra is an algebraic structure defined on a set of elements B together with two binary operators + and · provided the following (Huntington) postulates are satisfied:

1. (**a**) Closure with respect to the operator +.
   (**b**) Closure with respect to the operator ·.
2. (**a**) An identity element with respect to +, designated by 0: $x + 0 = 0 + x = x$.
   (**b**) An identity element with respect to ·, designated by 1: $x \cdot 1 = 1 \cdot x = x$.
3. (**a**) Commutative with respect to +: $x + y = y + x$.
   (**b**) Commutative with respect to ·: $x \cdot y = y \cdot x$.
4. (**a**) · is distributive over +: $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$.
   (**b**) + is distributive over ·: $x + (y \cdot z) = (x + y) \cdot (x + z)$.
5. For every element $x \in B$, there exists an element $x' \in B$ (called the complement of x) such that (a) $x + x' = 1$ and (b) $x \cdot x' = 0$.
6. There exists at least two elements $x, y \in B$ such that $x \neq y$.

Comparing Boolean algebra with arithmetic and ordinary algebra (the field of real numbers), we note the following differences:

1. Huntington postulates do not include the associative law. However, this law holds for Boolean algebra and can be derived (for both operators) from the other postulates.
2. The distributive law of + over ·, i.e., $x+(y \cdot z) = (x + y) \cdot (x + z)$, is valid for Boolean algebra, but not for ordinary algebra.
3. Boolean algebra does not have additive or multiplicative inverses; therefore, there are no subtraction or division operations.
4. Postulate 5 defines an operator called *complement* that is not available in ordinary algebra.
5. Ordinary algebra deals with the real numbers, which consitute an infinite set of elements. Boolean algebra deals with the as yet undefined set of elements B, but in the two-valued Boolean algebra defined below (and of interest in our subsequent use of this algebra), B is defined as a set with only two elements, 0 and 1.

Boolean algebra resembles ordinary algebra in some respects. The choice of symbols + and · is intentional to facilitate Boolean algebraic manipulations by persons already familiar with ordinary algebra. Although one can use some knowledge from

ordinary algebra to deal with Boolean algebra, the beginner must be careful not to substitute the rules of ordinary algebra where they are not applicable.

It is important to distinguish between the elements of the set of an algebraic structure and the variables of an algebraic system. For example, the elements of the field of real numbers are numbers, whereas variables such as a, b, c, etc., used in ordinary algebra, are symbols that stand for real numbers. Similarly in Boolean algebra, one defines the elements of the set B, and variables such as x, y, z are merely symbols that represent the elements. At this point, it is important to realize that in order to have a Boolean algebra, one must show:

1. the elements of the set B,
2. the rules of operation for the two binary operators, and
3. that the set of elements B, together with the two operators, satisfies the six Huntington postulates.

One can formulate many Boolean algebras, depending on the choice of elements of B and the rules of operation. In our subsequent work, we deal only with a two-valued Boolean algebra, i.e., one with only two elements. Two-valued Boolean algebra has applications in set theory (the algebra of classes) and in propositional logic. Our interest here is with the application of Boolean algebra to gate-type circuits.

Two-Valued Boolean Algebra

A two-valued Boolean algebra is defined on a set of two elements, $B = \{0, 1\}$, with rules for the two binary operators + and · as shown in the following operator tables (the rule for the complement operator is for verification of postulate 5):

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

x	x'
0	1
1	0

These rules are exactly the same as the AND, OR, and NOT operations, respectively, defined in Table 1-6. We must now show that the Huntington postulates are valid for the set $B = \{0, 1\}$ and the two binary operators defined before.

1. *Closure* is obvious from the tables since the result of each operation is either 1 or 0 and $1, 0 \in B$.
2. From the tables we see that
   (a) $0 + 0 = 0 \qquad 0 + 1 = 1 + 0 = 1$
   (b) $1 \cdot 1 = 1 \qquad 1 \cdot 0 = 0 \cdot 1 = 0$
   which establishes the two *identity elements* 0 for + and 1 for · as defined by postulate 2.

3. The *commutative* laws are obvious from the symmetry of the binary operator tables.
4. **(a)** The *distributive* law $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$ can be shown to hold true from the operator tables by forming a truth table of all possible values of x, y, and z. For each combination, we derive $x \cdot (y + z)$ and show that the value is the same as $(x \cdot y) + (x \cdot z)$.

x	y	z	$y + z$	$x \cdot (y + z)$	$x \cdot y$	$x \cdot z$	$(x \cdot y) + (x \cdot z)$
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

   (b) The *distributive* law of + over · can be shown to hold true by means of a truth table similar to the one above.
5. From the complement table it is easily shown that
   (a) $x + x' = 1$, since $0 + 0' = 0 + 1 = 1$ and $1 + 1' = 1 + 0 = 1$.
   (b) $x \cdot x' = 0$, since $0 \cdot 0' = 0 \cdot 1 = 0$ and $1 \cdot 1' = 1 \cdot 0 = 0$, which verifies postulate 5.
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 1-9. 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 1-9. 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 that 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): $x + x = x$.

$$
\begin{aligned}
x + x &= (x + x) \cdot 1 && \text{by postulate: } 2(b)\\
&= (x + x)(x + x') && 5(a)\\
&= x + xx' && 4(b)\\
&= x + 0 && 5(b)\\
&= x && 2(a)
\end{aligned}
$$

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

$$
\begin{aligned}
x \cdot x &= xx + 0 && \text{by postulate: } 2(a)\\
&= xx + xx' && 5(b)\\
&= x(x + x') && 4(a)\\
&= x \cdot 1 && 5(a)\\
&= x && 2(b)
\end{aligned}
$$

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): $x + 1 = 1$.

$$
\begin{aligned}
x + 1 &= 1 \cdot (x + 1) && \text{by postulate: } 2(b)\\
&= (x + x')(x + 1) && 5(a)\\
&= x + x' \cdot 1 && 4(b)\\
&= x + x' && 2(b)\\
&= 1 && 5(a)
\end{aligned}
$$

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 that have already been proven. Take, for example, the absorption theorem.

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

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

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 DeMorgan'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 DeMorgan'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 DeMorgan'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.

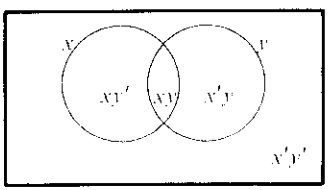


FIGURE 2-1
Venn diagram for two variables

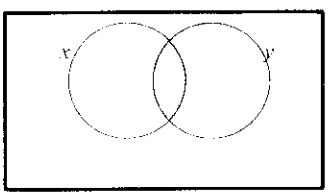
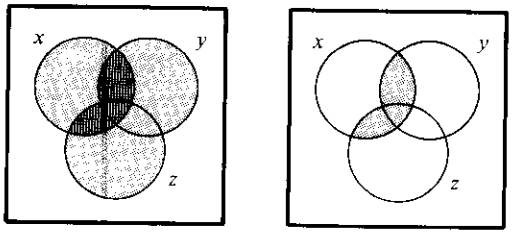


FIGURE 2-2
Venn diagram illustration $x = xy + x$



$x(y + z)$ $xy + xz$

FIGURE 2-3
Venn diagram illustration of the distributive law

2-4 BOOLEAN FUNCTIONS

A binary variable can take the value of 0 or 1. A Boolean function is an expression formed with binary variables, the two binary operators OR and AND, and unary operator NOT, parentheses, and an equal sign. For a given value of the variables, the function can be either 0 or 1. Consider, for example, the Boolean function

$$F_1 = xyz'$$

The function F_1 is equal to 1 if $x = 1$ *and* $y = 1$ *and* $z' = 1$; otherwise $F_1 = 0$. The above is an example of a Boolean function represented as an algebraic expression. A Boolean function may also be represented in a truth table. To represent a function in a truth table, we need a list of the 2^n combinations of 1's and 0's of the n binary variables, and a column showing the combinations for which the function is equal to 1 or 0. As shown in Table 2-2, there are eight possible distinct combinations for assigning bits to three variables. The column labeled F_1 contains either a 0 or a 1 for each of these combinations. The table shows that the function F_1 is equal to 1 only when $x = 1$, $y = 1$, and $z = 0$. It is equal to 0 otherwise. (Note that the statement $z' = 1$ is equivalent to saying that $z = 0$.) Consider now the function

TABLE 2-2
Truth Tables for $F_1 = xyz'$, $F_2 = x + y'z$, $F_3 = x'y'z + x'yz + xy'$, and $F_4 = xy' + x'z$

x	y	z	F_1	F_2	F_3	F_4
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0

$$F_2 = x + y'z$$

$F_2 = 1$ if $x = 1$ or if $y = 0$, while $z = 1$. In Table 2-2, $x = 1$ in the last four rows and $yz = 01$ in rows 001 and 101. The latter combination applies also for $x = 1$. Therefore, there are five combinations that make $F_2 = 1$. As a third example, consider the function

$$F_3 = x'y'z + x'yz + xy'$$

This is shown in Table 2-2 with four 1's and four 0's. F_4 is the same as F_3 and is considered below.

Any Boolean function can be represented in a truth table. The number of rows in the table is 2^n, where n is the number of binary variables in the function. The 1's and 0's combinations for each row is easily obtained from the binary numbers by counting from 0 to $2^n - 1$. For each row of the table, there is a value for the function equal to either 1 or 0. The question now arises, is it possible to find two algebraic expressions that specify the same function? The answer to this question is yes. As a matter of fact, the manipulation of Boolean algebra is applied mostly to the problem of finding simpler expressions for the same function. Consider, for example, the function:

$$F_4 = xy' + x'z$$

From Table 2-2, we find that F_4 is the same as F_3, since both have identical 1's and 0's for each combination of values of the three binary variables. In general, two functions of n binary variables are said to be equal if they have the same value for all possible 2^n combinations of the n variables.

A Boolean function may be transformed from an algebraic expression into a logic diagram composed of AND, OR, and NOT gates. The implementation of the four functions introduced in the previous discussion is shown in Fig. 2-4. The logic diagram includes an inverter circuit for every variable present in its complement form. (The inverter is unnecessary if the complement of the variable is available.) There is an AND gate for each term in the expression, and an OR gate is used to combine two or more terms. From the diagrams, it is obvious that the implementation of F_4 requires fewer gates and fewer inputs than F_3. Since F_4 and F_3 are equal Boolean functions, it is more economical to implement the F_4 form than the F_3 form. To find simpler circuits, one must know how to manipulate Boolean functions to obtain equal and simpler expressions. What constitutes the best form of a Boolean function depends on the particular application. In this section, consideration is given to the criterion of equipment minimization.

Algebraic Manipulation

A *literal* is a primed or unprimed variable. When a Boolean function is implemented with logic gates, each literal in the function designates an input to a gate, and each term is implemented with a gate. The minimization of the number of literals and the number of terms results in a circuit with less equipment. It is not always possible to minimize both simultaneously; usually, further criteria must be available. At the moment, we

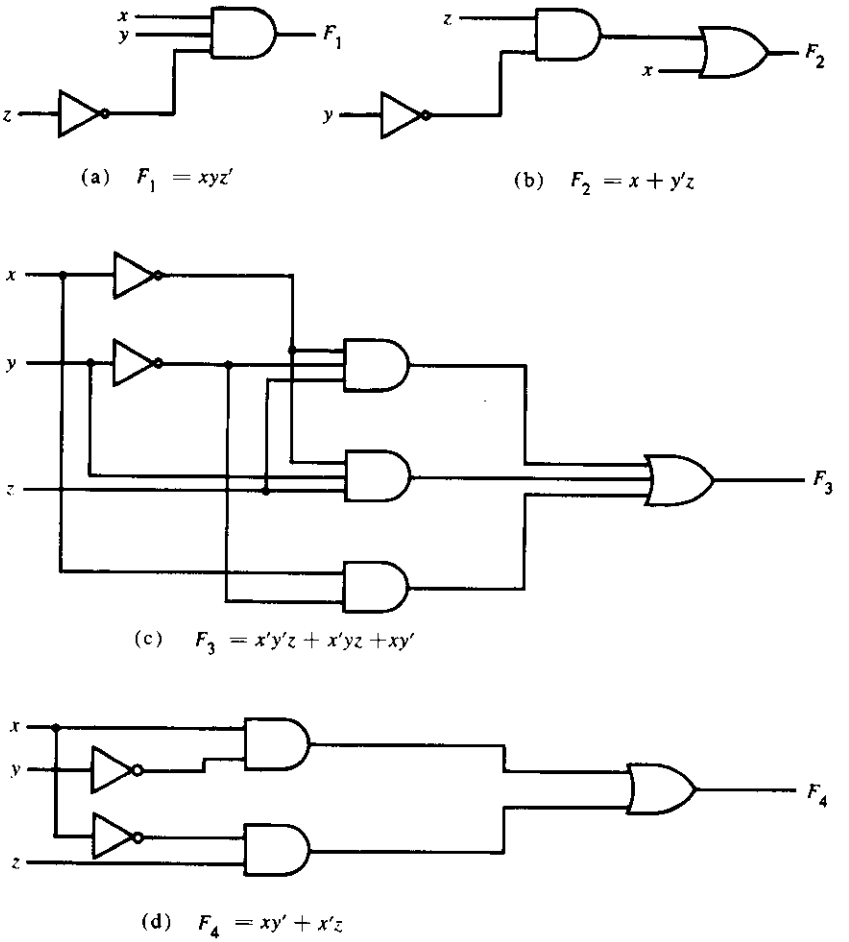


FIGURE 2-4
Implementation of Boolean functions with gates

shall narrow the minimization criterion to literal minimization. We shall discuss other criteria in Chapter 5. The number of literals in a Boolean function can be minimized by algebraic manipulations. Unfortunately, there are no specific rules to follow that will guarantee the final answer. The only method available is a cut-and-try procedure employing the postulates, the basic theorems, and any other manipulation method that becomes familiar with use. The following examples illustrate this procedure.

Example 2-1

Simplify the following Boolean functions to a minimum number of literals.

1. $x + x'y = (x + x')(x + y) = 1 \cdot (x + y) = x + y$
2. $x(x' + y) = xx' + xy = 0 + xy = xy$

3. $x'y'z + x'yz + xy' = x'z(y' + y) + xy' = x'z + xy'$
4. $xy + x'z + yz = xy + x'z + yz(x + x')$
$= xy + x'z + xyz + x'yz$
$= xy(1 + z) + x'z(1 + y)$
$= xy + x'z$
5. $(x + y)(x' + z)(y + z) = (x + y)(x' + z)$ by duality from function 4. ■

Functions 1 and 2 are the duals of each other and use dual expressions in corresponding steps. Function 3 shows the equality of the functions F_3 and F_4 discussed previously. The fourth illustrates the fact that an increase in the number of literals sometimes leads to a final simpler expression. Function 5 is not minimized directly but can be derived from the dual of the steps used to derive function 4.

Complement of a Function

The complement of a function F is F' and is obtained from an interchange of 0's for 1's and 1's for 0's in the value of F. The complement of a function may be derived algebraically through DeMorgan's theorem. This pair of theorems is listed in Table 2-1 for two variables. DeMorgan's theorems can be extended to three or more variables. The three-variable form of the first DeMorgan's theorem is derived below. The postulates and theorems are those listed in Table 2-1.

$$
\begin{aligned}
(A + B + C)' &= (A + X)' && \text{let } B + C = X \\
&= A'X' && \text{by theorem 5(a) (DeMorgan)} \\
&= A' \cdot (B + C)' && \text{substitute } B + C = X \\
&= A' \cdot (B'C') && \text{by theorem 5(a) (DeMorgan)} \\
&= A'B'C' && \text{by theorem 4(b) (associative)}
\end{aligned}
$$

DeMorgan's theorems for any number of variables resemble in form the two variable case and can be derived by successive substitutions similar to the method used in the above derivation. These theorems can be generalized as follows:

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

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

The generalized form ofDeMorgan'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 $F_2 = x(y'z' + yz)$. By applying DeMorgan'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 DeMorgan'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 variabes x and y combined with an AND operation. Since each variable may appear in either form, there are four possible 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.

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

A Boolean function may be expressed algebraically from a given truth table by forming a minterm for each combination of the variables that 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

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

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).

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

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:

$$\begin{aligned} f_2 &= (x + y + z)(x + y + z')(x + y' + z)(x' + y + z) \\ &= M_0M_1M_2M_4 \end{aligned}$$

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 that 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 examples 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:

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

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

$$F = A + B'C$$
$$= ABC + ABC' + AB'C + AB'C' + AB'C + A'B'C$$

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

$$F = A'B'C + AB'C' + AB'C + ABC' + ABC$$
$$= m_1 + m_4 + m_5 + m_6 + m_7$$ ■

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.

An alternate procedure for deriving the minterms of a Boolean function is to obtain the truth table of the function directly from the algebraic expression and then read the minterms from the truth table. Consider the Boolean function given in Example 2-4:

$$F = A + B'C$$

The truth table shown in Table 2-5 can be derived directly from the algebraic expression by listing the eight binary combinations under variables A, B, and C and inserting

TABLE 2-5
Truth Table for $F = A + B'C$

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

1's under F for those combinations where $A = 1$, and $BC = 01$. From the truth table, we can then read the five minterms of the function to be 1, 4, 5, 6, and 7.

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, whereas 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 DeMorgan'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. 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.

A Boolean function can be converted from an algebraic expression to a product of maxterms by using a truth table and the canonical conversion procedure. Consider, for example, the Boolean expression

$$F = xy + x'z$$

First, we derive the truth table of the function, as shown in Table 2-6. The 1's under F in the table are determined from the combination of the variable where $xy = 11$ and

TABLE 2-6
Truth Table for $F = xy + x'z$

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

$xz = 01$. The minterms of the function are read from the truth table to be 1, 3, 6, and 7. The function expressed in sum of minterms is

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

Since there are a total of eight minterms or maxterms in a function of three variable, we determine the missing terms to be 0, 2, 4, and 5. The function expressed in product of maxterm is

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

This is the same answer obtained in Example 2-5.

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, respectively. The product is an AND operation. The use of the words *product* and *sum* stems from the smiilarity 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 2-7. 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 represents the truth table for AND and F_7 represents the truth table for OR. The operator symbols for these functions are $\cdot$ and $+$, respectively.

TABLE 2-7
Truth Tables for the 16 Functions of Two Binary Variables

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Operator symbol			$\cdot$	/		/		$\oplus$	$+$	$\downarrow$	$\odot$	$'$	$\subset$	$'$	$\supset$	$\uparrow$	

The 16 functions listed in truth table form can be expressed algebraically by means of Boolean expressions. This is shown in the first column of Table 2-8. The Boolean expressions listed are simplified to their minimum number of literals.

Although each function can be expressed in terms of the Boolean operators AND, OR, and NOT, there is no reason one cannot assign special operator symbols for expressing the other functions. Such operator symbols are listed in the second column of Table 2-8. However, all the new symbols shown, except for the exclusive-OR symbol, $\oplus$, are not in common use by digital designers.

Each of the functions in Table 2-8 is listed with an accompanying name and a comment that explains the function in some way. The 16 functions listed can be subdivided into three categories:

1. Two functions that produce a constant 0 or 1.
2. Four functions with unary operations: complement and transfer.
3. Ten functions with binary operators that define eight different operations: AND, OR, NAND, NOR, exclusive-OR, equivalence, inhibition, and implication.

TABLE 2-8
Boolean Expressions for the 16 Functions of Two Variables

Boolean functions	Operator symbol	Name	Comments
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	x and y
$F_2 = xy'$	x/y	Inhibition	x but not y
$F_3 = x$		Transfer	x
$F_4 = x'y$	y/x	Inhibition	y but not x
$F_5 = y$		Transfer	y
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	x or y but not both
$F_7 = x + y$	$x + y$	OR	x or y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9 = xy + x'y'$	$x \odot y$	Equivalence	x equals y
$F_{10} = y'$	y'	Complement	Not y
$F_{11} = x + y'$	$x \subset y$	Implication	If y then x
$F_{12} = x'$	x'	Complement	Not x
$F_{13} = x' + y$	$x \supset y$	Implication	If x then y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	Not-AND
$F_{15} = 1$		Identity	Binary constant 1

Any function can be equal to a constant, but a binary function can be equal to only 1 or 0. The complement function produces the complement of each of the binary variables. A function that is equal to an input variable has been given the name *transfer*, because the variable x or y is transferred through the gate that forms the function without changing its value. Of the eight binary operators, two (inhibition and implication) are used by logicians but are seldom used in computer logic. The AND and OR operators have been mentioned in conjunction with Boolean algebra. The other four functions are extensively used in the design of digital systems.

The NOR function is the complement of the OR function and its name is an abbreviation of *not-OR*. Similarly, NAND is the complement of AND and is an abbreviation of *not-AND*. The exclusive-OR, abbreviated XOR or EOR, is similar to OR but excludes the combination of *both* x and y being equal to 1. The equivalence is a function that is 1 when the two binary variables are equal, i.e., when both are 0 or both are 1. The exclusive-OR and equivalence functions are the complements of each other. This can be easily verified by inspecting Table 2-7. The truth table for the exclusive-OR is F_6 and for the equivalence is F_9, and these two functions are the complements of each other. For this reason, the equivalence function is often called exclusive-NOR, i.e., exclusive-OR-NOT.

Boolean algebra, as defined in Section 2-2, has two binary operators, which we have called AND and OR, and a unary operator, NOT (complement). From the definitions,

we have deduced a number of properties of these operators and now have defined other binary operators in terms of them. There is nothing unique about this procedure. We could have just as well started with the operator NOR ($\downarrow$), for example, and later defined AND, OR, and NOT in terms of it. There are, nevertheless, good reasons for introducing Boolean algebra in the way it has been introduced. The concepts of "and," "or," and "not" are familiar and are used by people to express everyday logical ideas. Moreover, the Huntington postulates reflect the dual nature of the algebra, emphasizing the symmetry of + and · with respect to each other.

2-7 DIGITAL LOGIC GATES

Since Boolean functions are expressed in terms of AND, OR, and NOT operations, it is easier to implement a Boolean function with these types of gates. The possibility of constructing gates for the other logic operations is of practial interest. Factors to be weighed when considering the construction of other types of logic gates are (1) the feasibility and economy of producing the gate with physical components, (2) the possibility of extending the gate to more than two inputs, (3) the basic properties of the binary operator such as commutativity and associativity, and (4) the ability of the gate to implement Boolean functions alone or in conjunction with other gates.

Of the 16 functions defined in Table 2-8, two are equal to a constant and four others are repeated twice. There are only ten functions left to be considered as candidates for logic gates. Two, inhibition and implication, are not commutative or associative and thus are impractical to use as standard logic gates. The other eight: complement, transfer, AND, OR, NAND, NOR, exclusive-OR, and equivalence, are used as standard gates in digital design.

The graphic symbols and truth tables of the eight gates are shown in Fig. 2-5. Each gate has one or two binary input variables designated by x and y and one binary output variable designated by F. The AND, OR, and inverter circuits were defined in Fig. 1-6. The inverter circuit inverts the logic sense of a binary variable. It produces the NOT, or complement, function. The small circle in the output of the graphic symbol of an inverter designates the logic complement. The triangle symbol by itself designates a buffer circuit. A buffer produces the *transfer* function but does not produce any particular logic operation, since the binary value of the output is equal to the binary value of the input. This circuit is used merely for power amplification of the signal and is equivalent to two inverters connected in cascade.

The NAND function is the complement of the AND function, as indicated by a graphic symbol that consists of an AND graphic symbol followed by a small circle. The NOR function is the complement of the OR function and uses an OR graphic symbol followed by a small circle. The NAND and NOR gates are extensively used as standard logic gates and are in fact far more popular than the AND and OR gates. This is because NAND and NOR gates are easily constructed with transistor circuits and because Boolean functions can be easily implemented with them.

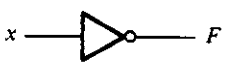
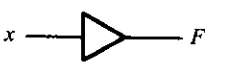
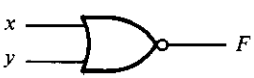
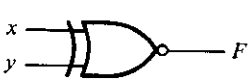
Name	Graphic symbol	Algebraic function	Truth table
AND		$F = xy$	$x\ y \mid F$ $0\ 0 \mid 0$ $0\ 1 \mid 0$ $1\ 0 \mid 0$ $1\ 1 \mid 1$
OR		$F = x + y$	$x\ y \mid F$ $0\ 0 \mid 0$ $0\ 1 \mid 1$ $1\ 0 \mid 1$ $1\ 1 \mid 1$
Inverter		$F = x'$	$x \mid F$ $0 \mid 1$ $1 \mid 0$
Buffer		$F = x$	$x \mid F$ $0 \mid 0$ $1 \mid 1$
NAND		$F = (xy)'$	$x\ y \mid F$ $0\ 0 \mid 1$ $0\ 1 \mid 1$ $1\ 0 \mid 1$ $1\ 1 \mid 0$
NOR		$F = (x + y)'$	$x\ y \mid F$ $0\ 0 \mid 1$ $0\ 1 \mid 0$ $1\ 0 \mid 0$ $1\ 1 \mid 0$
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	$x\ y \mid F$ $0\ 0 \mid 0$ $0\ 1 \mid 1$ $1\ 0 \mid 1$ $1\ 1 \mid 0$
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= x \odot y$	$x\ y \mid F$ $0\ 0 \mid 1$ $0\ 1 \mid 0$ $1\ 0 \mid 0$ $1\ 1 \mid 1$

FIGURE 2-5
Digital logic gates

The exclusive-OR gate has a graphic symbol similar to that of the OR gate, except for the additional curved line on the input side. The equivalence, or exclusive-NOR, gate is the complement of the exclusive-OR, as indicated by the small circle on the output side of the graphic symbol.

Extension to Multiple Inputs

The gates shown in Fig. 2-5, except for the inverter and buffer, can be extended to have more than two inputs. A gate can be extended to have multiple inputs if the binary operation it represents is commutative and associative. The AND and OR operations, defined in Boolean algebra, possess these two properties. For the OR function, we have

$$x + y = y + x \qquad \text{commutative}$$

and

$$(x + y) + z = x + (y + z) = x + y + z \qquad \text{associative}$$

which indicates that the gate inputs can be interchanged and that the OR function can be extended to three or more variables.

The NAND and NOR functions are commutative and their gates can be extended to have more than two inputs, provided the definition of the operation is slightly modified. The difficulty is that the NAND and NOR operators are not associative, i.e., $(x \downarrow y) \downarrow z \neq x \downarrow (y \downarrow z)$, as shown in Fig. 2-6 and below:

$$(x \downarrow y) \downarrow z = [(x + y)' + z]' = (x + y)z' = xz' + yz'$$

$$x \downarrow (y \downarrow z) = [x + (y + z)']' = x'(y + z) = x'y + x'z$$

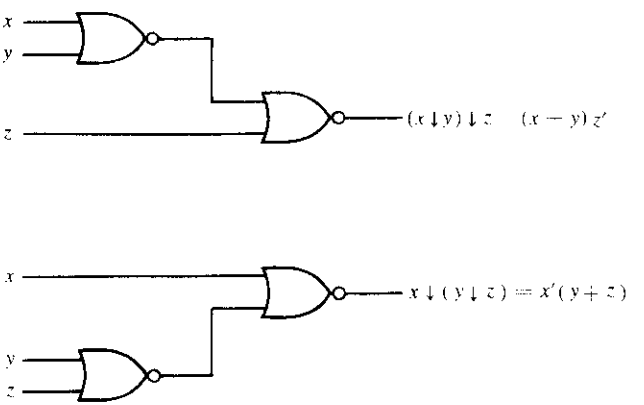


FIGURE 2-6
Demonstrating the nonassociativity of the NOR operator; $(x \downarrow y) \downarrow z \neq x \downarrow (y \downarrow z)$

To overcome this difficulty, we define the multiple NOR (or NAND) gate as a complemented OR (or AND) gate. Thus, by definition, we have

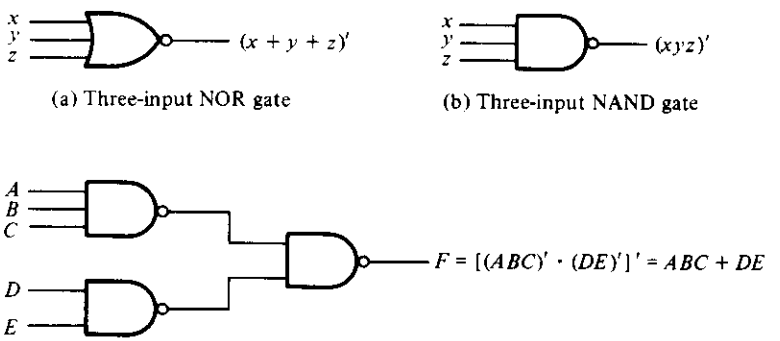
$$x \downarrow y \downarrow z = (x + y + z)'$$
$$x \uparrow y \uparrow z = (xyz)'$$

The graphic symbols for the 3-input gates are shown in Fig. 2-7. In writing cascaded NOR and NAND operations, one must use the correct parentheses to signify the proper sequence of the gates. To demonstrate this, consider the circuit of Fig. 2-7(c). The Boolean function for the circuit must be written as

$$F = [(ABC)'(DE)']' = ABC + DE$$

The second expression is obtained from DeMorgan's theorem. It also shows that an expression in sum of products can be implemented with NAND gates. Further discussion of NAND and NOR gates can be found in Sections 3-6, 4-7, and 4-8.

The exclusive-OR and equivalence gates are both commutative and associative and can be extended to more than two inputs. However, multiple-input exclusive-OR gates are uncommon from the hardware standpoint. In fact, even a 2-input function is usually constructed with other types of gates. Moreoever, the definition of the function must be modified when extended to more than two variables. The exclusive-OR is an *odd* function, i.e., it is equal to 1 if the input variables have an odd number of 1's. The construction of a 3-input exclusive-OR function is shown in Fig. 2-8. It is normally implemented by cascading 2-input gates, as shown in (a). Graphically, it can be represented with a single 3-input gate, as shown in (b). The truth table in (c) clearly indicates that the output F is equal to 1 if only one input is equal to 1 or if all three inputs are equal to 1, i.e., when the total number of 1's in the input variables is *odd*. Further discussion of exclusive-OR can be found in Section 4-9.

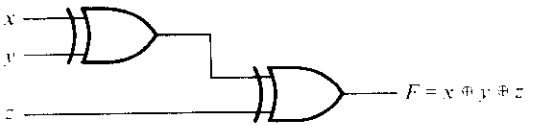


(a) Three-input NOR gate

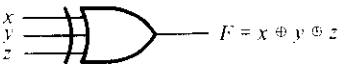
(b) Three-input NAND gate

(c) Cascaded NAND gates

FIGURE 2-7
Multiple-input and cascaded NOR and NAND gates



(a) Using 2-input gates



(b) 3-input gate

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

(c) Truth table

FIGURE 2-8
3-input exclusive-OR gate

2-8 INTEGRATED CIRCUITS

Digital circuits are constructed with integrated circuits. An integrated circuit (abbreviated IC) is a small silicon semiconductor crystal, called a *chip*, containing the electronic components for the digital gates. The various gates are interconnected inside the chip to form the required circuit. The chip is mounted in a ceramic or plastic container, and connections are welded to external pins to form the integrated circuit. The number of pins may range from 14 in a small IC package to 64 or more in a larger package. The size of the IC package is very small. For example, four AND gates are enclosed inside a 14-pin IC package with dimensions of 20 × 8 × 3 millimeters. An entire microprocessor is enclosed within a 64-pin IC package with dimensions of 50 × 15 × 4 millimeters. Each IC has a numeric designation printed on the surface of the package for identification. Vendors publish data books that contain descriptions and all other information about the ICs that they manufacture.

Levels of Integration

Digital ICs are often categorized according to their circuit complexity as measured by the number of logic gates in a single package. The differentiation between those chips that have a few internal gates and those having hundreds or thousands of gates is made by a customary reference to a package as being either a small-, medium-, large-, or very large-scale integration device.

Small-scale integration (SSI) devices contain several independent gates in a single package. The inputs and outputs of the gates are connected directly to the pins in the package. The number of gates is usually fewer than 10 and is limited by the number of pins available in the IC.

Medium-sale integration (MSI) devices have a complexity of approximately 10 to 100 gates in a single package. They usually perform specific elementary digital operations such as decoders, adders, or multiplexers. MSI digital components are introduced in Chapters 5 and 7.

Large-scale integration (LSI) devices contain between 100 and a few thousand gates in a single package. They include digital systems such as processors, memory chips, and programmable logic devices. Some LSI components are presented in Chapters 5 and 7.

Very large-scale integration (VLSI) devices contain thousands of gates within a single package. Examples are large memory arrays and complex microcomputer chips. Because of their small size and low cost, VLSI devices have revolutionized the computer system design technology, giving the designer the capabilities to create structures that previously were uneconomical.

Digital Logic Families

Digital integrated circuits are classified not only by their complexity or logical operation, but also by the specific circuit technology to which they belong. The circuit technology is referred to as a digital logic family. Each logic family has its own basic electronic circuit upon which more complex digital circuits and components are developed. The basic circuit in each technology is a NAND, NOR, or an inverter gate. The electronic components used in the construction of the basic circuit are usually used as the name of the technology. Many different logic families of digital integrated circuits have been introduced commercially. The following are the most popular:

TTL	transistor-transistor logic
ECL	emiter-coupled logic
MOS	metal-oxide semiconductor
CMOS	complementary metal-oxide semiconductor

TTL is a widespread logic family that has been in operation for some time and is considered as standard. ECL has an advantage in systems requiring high-speed operation. MOS is suitable for circuits that need high component density, and CMOS is preferable in systems requiring low power consumption.

The analysis of the basic electronic digital gate circuit in each logic family is presented in Chapter 10. The reader familiar with basic electronics can refer to Chapter 10 at this time to become acquainted with these electronic circuits. Here we restrict the discussion to the general properties of the various IC gates available commercially.

The transistor-transistor logic family evolved from a previous technology that used diodes and transistors for the basic NAND gate. This technology was called DTL for diode-transistor logic. Later the diodes were replaced by transistors to improve the circuit operation and the name of the logic family was changed to TTL.

Emitter-coupled logic (ECL) circuits provide the highest speed among the integrated digital logic families. ECL is used in systems such as supercomputers and signal processors, where high speed is essential. The transistors in ECL gates operate in a nonsaturated state, a condition that allows the achievement of propagation delays of 1 to 2 nanoseconds.

The metal-oxide semiconductor (MOS) is a unipolar transistor that depends upon the flow of only one type of carrier, which may be electrons (n-channel) or holes (p-channel). This is in contrast to the bipolar transistor used in TTL and ECL gates, where both carriers exist during normal operation. A p-channel MOS is referred to as PMOS and an n-channel as NMOS. NMOS is the one that is commonly used in circuits with only one type of MOS transistor. Complementary MOS (CMOS) technology uses one PMOS and one NMOS transistor connected in a complementary fashion in all circuits. The most important advantages of MOS over bipolar transistors are the high packing density of circuits, a simpler procesing technique during fabrication, and a more economical operation because of the low power consumption.

The characteristics of digital logic families are usually compared by analyzing the circuit of the basic gate in each family. The most important parameters that are evaluated and compared are discussed in Section 10-2. They are listed here for reference.

Fan-out specifies the number of standard loads that the output of a typical gate can drive without impairing its normal operation. A standard load is usually defined as the amount of current needed by an input of another similar gate of the same family.

Power dissipation is the power consumed by the gate that must be available from the power supply.

Propagation delay is the average transition delay time for the signal to propagate from input to output. The operating speed is inversely proportional to the propagation delay.

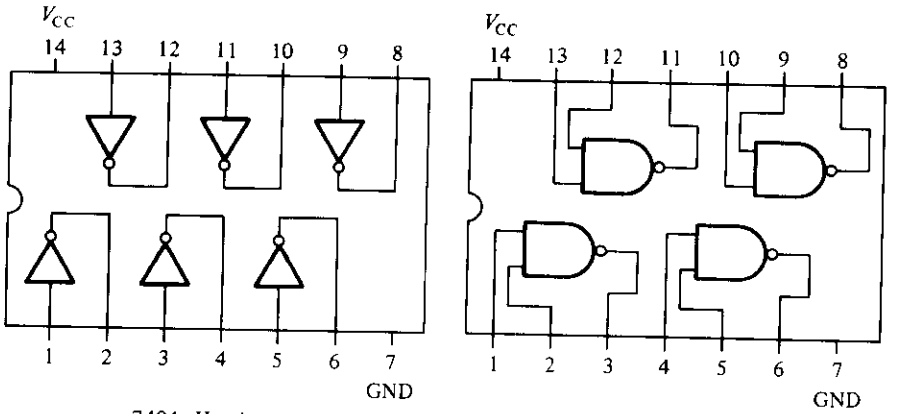
Noise margin is the minimum external noise voltage that causes an undesirable change in the circuit output.

Integrated-Circuit Gates

Some typical SSI circuits are shown in Fig. 2-9. Each IC is enclosed within a 14- or 16-pin package. A notch placed on the left side of the package is used to reference the pin numbers. The pins are numbered along the two sides starting from the notch and continuing counterclockwise. The inputs and outputs of the gates are connected to the package pins, as indicated in each diagram.

TTL IC's are usually distinguished by their numerical designation as the 5400 and 7400 series. The former has a wide operating temperature range, suitable for military use, and the latter has a narrower temperature range, suitable for commercial use. The numeric designation of 7400 series means that IC packages are numbered as 7400, 7401, 7402, etc. Fig. 2-9(a) shows two TTL SSI circuits. The 7404 provides six (hex) inverters in a package. The 7400 provides four (quadruple) 2-input NAND gates. The terminals marked V_{CC} and GRD (ground) are the power-supply pins that require a voltage of 5 volts for proper operation. The two logic levels for TTL are 0 and 3.5 volts.

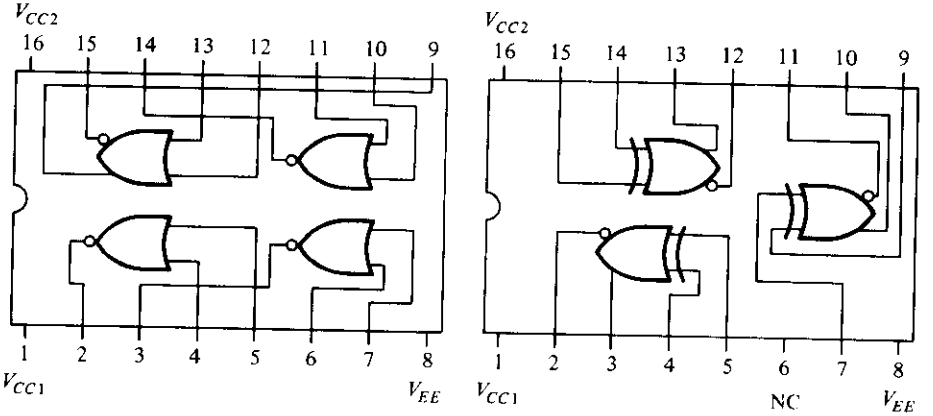
The TTL logic family actually consists of several subfamilies or series. Table 2-9 lists the name of each series and the prefix designation that identifies the IC as being part of that series. As mentioned before, ICs that are part of the standard TTL have an



7404—Hex inverters

7400—Quadruple 2-input NAND gates

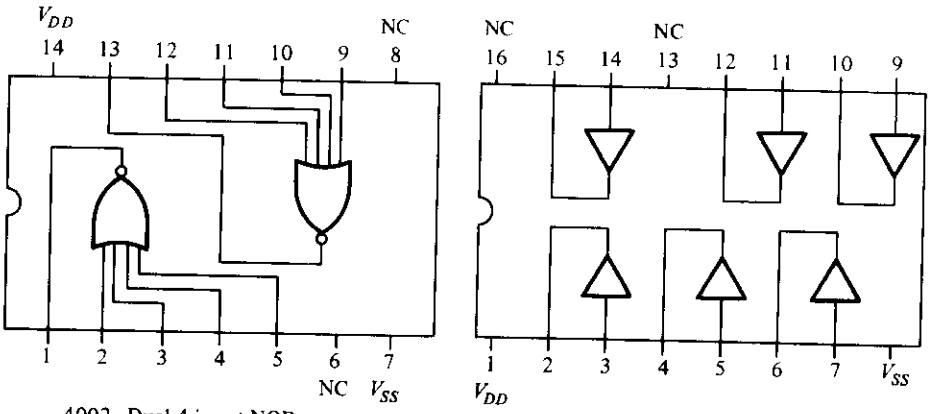
(a) TTL gates.



10102—Quadruple 2-input NOR gates

10107—Triple exclusive-OR/NOR gates

(b) ECL gates.



4002—Dual 4-input NOR gates.

4050—Hex buffers.

(c) CMOS gates.

FIGURE 2-9
Some typical integrated-circuit gates

TABLE 2-9
Various Series of the TTL Logic Family

TTL Series	Prefix	Example
Standard TTL	74	7486
High-speed TTL	74H	74H86
Low-power TTL	74L	74L86
Schottky TTL	74S	74S86
Low-power Schottky TTL	74LS	74LS86
Advanced Schottky TTL	74AS	74AS86
Advanced Low-power Schottky TTL	74ALS	74ALS86

identification number that starts with 74. Likewise, ICs that are part of the high-speed TTL series have an identification number that starts with 74H, ICs in the Schottky TTL series start with 74S, and similarly for the other series. The different characteristics of the various TTL series are listed in Table 10-2 in Chapter 10. The differences between the various TTL series are in their electrical characteristics, such as power dissipation, propagation delay, and switching speed. They do not differ in the pin assignment or logic operation performed by the internal circuits. For example, all the ICs listed in Table 2-9 with an 86 number, no matter what the prefix, contain four exclusive-OR gates with the same pin assignment in each package.

The most common ECL ICs are designated as the 10000 series. Figure 2-9(b) shows two ECL circuits. The 10102 provides four 2-input NOR gates. Note that an ECL gate may have two outputs, one for the NOR function and another for the OR function. The 10107 IC provides three exclusive-OR gates. Here again there are two outputs from each gate; the other output provides the exclusive-NOR function. ECL gates have three terminals for power supply. V_{CC1} and V_{CC2} are usually connected to ground, and V_{EE} to a -5.2-volt supply. The two logic levels for ECL are -0.8 and -1.8 volts.

CMOS circuits of the 4000 series are shown in Fig. 2-9(c). Only two 4-input NOR gates can be accommodated in the 4002 because of pin limitation. The 4050 IC provides six buffer gates. Both ICs have unused terminals marked NC (no connection). The terminal marked V_{DD} requires a power-supply voltage from 3 to 15 volts, whereas V_{SS} is usually connected to ground. The two logic levels are 0 and V_{DD} volts.

The original 4000 series of CMOS circuits was designed independently from the TTL series. Since TTL became a standard in the industry, vendors started to supply other CMOS circuits that are pin compatible with similar TTL ICs. For example, the 74C04 is a CMOS circuit that is pin compatible with TTL 7404. This means that it has six inverters connected to the pins of the package, as shown in Fig. 2-9(a). The CMOS series available commercially are listed in Table 2-10. The 74HC series operates at higher speeds than the 74C series. The 74HCT series is both electrically and pin compatible with TTL devices. This means that 74HCT ICs can be connected directly to TTL ICs without the need of interfacing circuits.

TABLE 2-10
Various Series of the CMOS Logic Family

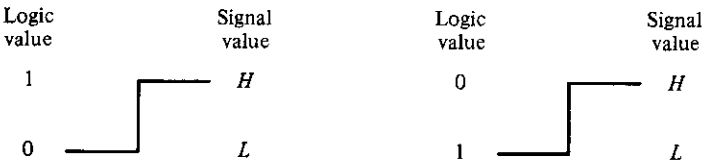
CMOS series	Prefix	Example
Original CMOS	40	4009
Pin compatible with TTL	74C	74C04
High-speed and pin compatible with TTL	74HC	74HCO4
High-speed and electrically compatible with TTL	74HCT	74HCT04

Positive and Negative Logic

The binary signal at the inputs and outputs of any gate has one of two values, except during transition. One signal value represents logic-1 and the other logic-0. Since two signal values are assigned to two logic values, there exist two different assignments of signal level to logic value, as shown in Fig. 2-10. The higher signal level is designated by H and the lower signal level by L. Choosing the high-level H to represent logic-1 defines a positive logic system. Choosing the low-level L to represent logic-1 defines a negative logic system. The terms positive and negative are somewhat misleading since both signals may be positive or both may be negative. It is not the actual signal values that determine the type of logic, but rather the assignment of logic values to the relative amplitudes of the two signal levels.

Integrated-circuit data sheets define digital gates not in terms of logic values, but rather in terms of signal values such as H and L. It is up to the user to decide on a positive or negative logic polarity. Consider, for example, the TTL gate shown in Fig. 2-11(b). The truth table for this gate as given in a data book is listed in Fig. 2-11(a). This specifies the physical behavior of the gate when H is 3.5 volts and L is 0 volt. The truth table of Fig. 2-11(c) assumes positive logic assignment with $H = 1$ and $L = 0$. This truth table is the same as the one for the AND operation. The graphic symbol for a positive logic AND gate is shown in Fig. 2-11(d).

Now consider the negative logic assignment for the same physical gate with $L = 1$ and $H = 0$. The result is the truth table of Fig. 2-11(e). This table represents the OR operation even though the entries are reversed. The graphic symbol for the negative logic OR gate is shown in Fig. 2-11(f). The small triangles in the inputs and output

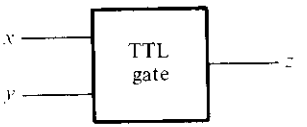


(a) Positive logic (b) Negative logic

FIGURE 2-10
Signal assignment and logic polarity

x	y	z
L	L	L
L	H	L
H	L	L
H	H	H

(a) Truth table with H and L



(b) Gate block diagram

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

(c) Truth table for positive logic

(d) Positive logic AND gate

x	y	z
1	1	1
1	0	1
0	1	1
0	0	0

(e) Truth table for negative logic

(f) Negative logic OR gate

FIGURE 2-11
Demonstration of positive and negative logic

designate a *polarity indicator*. The presence of this polarity indicator along a terminal signifies that negative logic is assumed for the signal. Thus, the same physical gate can operate either as a positive logic AND gate or as a negative logic OR gate.

The conversion from positive logic to negative logic, and vice versa, is essentially an operation that changes 1's to 0's and 0's to 1's in both the inputs and the output of a gate. Since this operation produces the dual of a function, the change of all terminals from one polarity to the other results in taking the dual of the function. The result of this conversion is that all AND operations are converted to OR operations (or graphic symbols) and vice versa. In addition, one must not forget to include the polarity-indicator triangle in the graphic symbols when negative logic is assumed. In this book, we will not use negative logic gates and assume that all gates operate with a positive logic assignment.

REFERENCES

1. BOOLE, G., *An Investigation of the Laws of Thought*. New York: Dover, 1954.
2. SHANNON, C. E., "A Symbolic Analysis of Relay and Switching Circuits." *Trans. AIEE*, **57** (1938), 713–723.
3. HUNTINGTON, E. V., "Sets of Independent Postulates for the Algebra of Logic." *Trans. Am. Math. Soc.*, **5** (1904). 288–309.
4. BIRKHOFF, G., and T. C. Bartee, *Modern Applied Algebra*. New York: McGraw-Hill, 1970.
5. HOHN, F. E., *Applied Boolean Algebra*, 2nd Ed. New York: Macmillan, 1966.
6. WHITESITT, J. E., *Boolean Algebra and Its Application*. Reading, MA: Addison-Wesley, 1961.
7. FRIEDMAN, A. D., and P. R. MENON, *Theory and Design of Switching Circuits*. Rockville, MD: Computer Science Press, 1975.
8. *The TTL Data Book*. Dallas: Texas Instruments, 1988.
9. TOCCI, R. J., *Digital Systems Principles and Applications*, 4th Ed. Englewood Cliffs, NJ: Prentice-Hall, 1988.

PROBLEMS

2-1 Demonstrate by means of truth tables the validity of the following identities:
(a) DeMorgan's theorem for three variables: $(xyz)' = x' + y' + z'$.
(b) The second distributive law: $x + yz = (x + y)(x + z)$.
(c) The consensus theorem: $xy + x'z + yz = xy + x'z$. (This is done algebraically in Example 2-1, part 4.)

2-2 Simplify the following Boolean expressions to a minimum number of literals.
(a) $x'y' + xy + x'y$
(b) $(x + y)(x + y')$
(c) $x'y + xy' + xy + x'y'$
(d) $x' + xy + xz' + xy'z'$
(e) $xy' + y'z' + x'z'$ [use the consensus theorem, Problem 2-1(c)].

2-3 Simplify the following Boolean expressions to a minimum number of literals:
(a) $ABC + A'B + ABC'$
(b) $x'yz + xz$
(c) $(x + y)'(x' + y')$
(d) $xy + x(wz + wz')$
(e) $(BC' + A'D)(AB' + CD')$

2-4 Reduce the following Boolean expressions to the indicated number of literals:
(a) $A'C' + ABC + AC'$ to three literals
(b) $(x'y' + z)' + z + xy + wz$ to three literals
(c) $A'B(D' + C'D) + B(A + A'CD)$ to one literal
(d) $(A' + C)(A' + C')(A + B + C'D)$ to four literals

2-5 Find the complement of $F = x + yz$; then show that $F \cdot F' = 0$ and $F + F' = 1$.

2-6 Find the complement of the following expressions:
(a) $xy' + x'y$
(b) $(AB' + C)D' + E$
(c) $AB(C'D + CD') + A'B'(C' + D)(C + D')$
(d) $(x + y' + z)(x' + z')(x + y)$

2-7 Using DeMorgan's theorem, convert the following Boolean expressions to equivalent expressions that have only OR and complement operations. Show that the functions can be implemented with logic diagrams that have only OR gates and inverters.
(a) $F = x'y' + x'z + y'z$
(b) $F = (y + z')(x + y)(y' + z)$

2-8 Using DeMorgan's theorem, convert the two Boolean expressions listed in Problem 2-7 to equivalent expressions that have only AND and complement operations. Show that the functions can be implemented with only AND gates and inverters.

2-9 Obtain the truth table of the following functions and express each function in sum of minterms and product of maxterms:
(a) $(xy + z)(y + xz)$
(b) $(A' + B)(B' + C)$
(c) $y'z + wxy' + wxz' + w'x'z$

2-10 For the Boolean function F given in the truth table, find the following:
(a) List the minterms of the function.
(b) List the minterms of F'.
(c) Express F in sum of minterms in algebraic form.
(d) Simplify the function to an expression with a minimum number of literals.

x	y	z	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

2-11 Given the following Boolean function:

$$F = xy'z + x'y'z + w'xy + wx'y + wxy$$

(a) Obtain the truth table of the function.
(b) Draw the logic diagram using the original Boolean expression.
(c) Simplify the function to a minimum number of literals using Boolean algebra.
(d) Obtain the truth table of the function from the simplified expression and show that it is the same as the one in part (a).

(e) Draw the logic diagram from the simplified expression and compare the total number of gates with the diagram of part (b).

2-12 Express the following functions in sum of minterms and product of maxterms:
(a) $F(A, B, C, D) = B'D + A'D + BD$
(b) $F(x, y, z) = (xy + z)(xz + y)$

2-13 Express the complement of the following functions in sum of minterms:
(a) $F(A, B, C, D) = \Sigma(0, 2, 6, 11, 13, 14)$
(b) $F(x, y, z) = \Pi(0, 3, 6, 7)$

2-14 Convert the following to the other canonical form:
(a) $F(x, y, z) = \Sigma(1, 3, 7)$
(b) $F(A, B, C, D) = \Pi(0, 1, 2, 3, 4, 6, 12)$

2-15 The sum of all the minterms of a Boolean function of n variables is equal to 1.
(a) Prove the above statement for $n = 3$.
(b) Suggest a procedure for a general proof.

2-16 Convert the following expressions into sum of products and product of sums:
(a) $(AB + C)(B + C'D)$
(b) $x' + x(x + y')(y + z')$

2-17 Draw the logic diagram corresponding to the following Boolean expressions without simplifying them:
(a) $BC' + AB + ACD$
(b) $(A + B)(C + D)(A' + B + D)$
(c) $(AB + A'B')(CD' + C'D)$

2-18 Show that the dual of the exclusive-OR is equal to its complement.

2-19 By substituting the Boolean expression equivalent of the binary operations as defined in Table 2-8, show the following:
(a) The inhibition operation is neither commutative nor associative.
(b) The exclusive-OR operation is commutative and associative.

2-20 Verify the truth table for the three-variable exclusive-OR function listed in Fig. 2-8(c). Do that by listing all eight combinations of x, y, and z; then evaluate $A = x \oplus y$; and then evaluate $F = A \oplus z = x \oplus y \oplus z$.

2-21 TTL SSI come mostly in 14-pin packages. Two pins are reserved for power and the other 12 pins are used for input and output terminals. Determine the number of gates that can be enclosed in one package if it contains the folowing type of gates:
(a) Two-input exclusive-OR gates
(b) Three-input AND gates
(c) Four-input NAND gates
(d) Five-input NOR gates
(e) Eight-input NAND gates

2-22 Show that a positive logic NAND gate is a negative logic NOR gate and vice versa.

2-23 An integrated-circuit logic family has NAND gates with fan-out of 5 and buffer gates with fan-out of 10. Show how the output signal of a single NAND gate can be applied to 50 other NAND-gate inputs without overloading the output gate. Use buffers to satisfy the fan-out requirements.