Chapter 2

Switching Algebra and Logic Gates

The word *algebra* in the title of this chapter should alert you that more mathematics is coming. No doubt, some of you are itching to get on with digital design rather than tackling more math. However, as your experience in engineering and science has taught you, mathematics is a basic requirement for all fields in these areas. Just as thinking requires knowledge of a language in which concepts can be formulated, so any field of engineering or science requires knowledge of certain mathematical topics in terms of which concepts in the field can be expressed and understood.

The mathematical basis for digital systems is *Boolean algebra.*[1] This chapter starts with a brief exposition of Boolean algebra that lays the groundwork for introducing the building blocks of digital circuits later in the chapter.

1 BOOLEAN ALGEBRA

Boolean algebra, like any other axiomatic mathematical structure or algebraic system, can be characterized by specifying a number of fundamental things:

1. The *domain* of the algebra, that is, the set of *elements* over which the algebra is defined
2. A set of *operations* to be performed on the elements
3. A set of *postulates,* or *axioms,* accepted as premises without proof
4. A set of consequences called *theorems, laws,* or *rules,* which are deduced from the postulates

As in any area of mathematics, it is possible to start from different sets of postulates and still arrive at the same mathematical structure. What is proved as a theorem from one set of postulates can be taken as a postulate in another set, and what was a postulate in the first set can be proved as a theorem from

[1]This designation comes from its originator, the Briton George Boole, who published a work titled *An Investigation of the Laws of Thought* in 1854. This treatise was a fundamental and systematic exposition of logic. The book languished in obscurity for many decades.

Table 1 Huntington's Postulates

1. Closure. There exists a domain B having at least two distinct elements and two binary operators (+) and (•) such that:
- **a.** If x and y are elements, then $x + y$ is an element.
  The operation performed by (+) is called *logical addition*.
- **b.** If x and y are elements, then $x \bullet y$ is an element.
  The operation performed by (•) is called *logical multiplication*.

2. Identity elements. Let x be an element in domain B.
- **a.** There exists an element 0 in B, called the *identity element with respect to* (+), having the property $x + 0 = x$.
- **b.** There exists an element 1 in B, called the *identity element with respect to* (•), having the property that $x \bullet 1 = x$.

3. Commutative law
- **a.** Commutative law with respect to addition: $x + y = y + x$.
- **b.** Commutative law with respect to multiplication: $x \bullet y = y \bullet x$.

4. Distributive law
- **a.** Multiplication is distributive over addition: $x \bullet (y + z) = (x \bullet y) + (x \bullet z)$
- **b.** Addition is distributive over multiplication: $x + (y \bullet z) = (x + y) \bullet (x + z)$

5. Complementation. If x is an element in domain B, then there exists another element x', the *complement* of x, satisfying the properties:
- **a.** $x + x' = 1$
- **b.** $x \bullet x' = 0$

The complement x' performs the *complementation* operation on x.

another set of postulates. So how do we decide on postulates? Clearly, one requirement for a set of postulates is *consistency.* It would not do for the consequences of one postulate to contradict those of another. Another requirement often stated is *independence*. However, independence involves the customary objective of ending up with a *minimal* set of postulates that still permits the derivation of all of the theorems. So long as a mathematical rule is consistent with the others, it can be added as a postulate without harm. If it is dependent on the previous postulates, however, the added rule is derivable from them and so need not be taken as a postulate.

The postulates we shall adopt here are referred to as *Huntington's postulates* and are given in Table 1.[2] Study them carefully. Note that Boolean algebra is like ordinary algebra in some respects but unlike it in others. For example, the distributive law of addition (Postulate 4b) is not valid for ordinary algebra, nor is the complement operation (Postulate 5). On the other hand, the subtraction and division operations of ordinary algebra do not exist in Boolean algebra.

The set of elements in Boolean algebra is called its *domain* and is labeled B. An *m-ary operation* in B is a rule that assigns to each ordered set of m elements a unique element from B. Thus, a *binary* operation involves an ordered *pair* of elements, and a *unary* operation involves just one element. In Boolean algebra two binary operations (logical addition and logical multiplication) and

[2]They were formulated by the British mathematician E. V. Huntington, who made an attempt to systematize the work of George Boole exactly 50 years after the publication of Boole's treatise.

one unary operation (complementation) are defined. Many Boolean algebras with different sets of elements can exist. The terminology *Boolean algebra* is a generic way of referring to them all.

Duality Principle

An examination of Huntington's postulates reveals a certain symmetry: the postulates come in pairs. One of the postulates in each pair can be obtained from the other one

- By interchanging the two binary operators, and
- By interchanging the two identity elements when they appear explicitly.

Thus, one of the commutative laws can be obtained from the other by interchanging the operators (+) and (•). The same is true of the two distributive laws. Consequently, whatever results can be deduced from the postulates should remain valid if

- The operators (+) and (•) are interchanged, and
- The identity elements 0 and 1 are interchanged.

This property of Boolean algebra is referred to as the *duality principle.* Whenever some result (theorem) is deduced from the postulates, the duality principle can be invoked as proof of the dual theorem.

Fundamental Theorems

We will now establish a number of consequences (theorems, rules, or laws) that follow from Huntington's postulates and from the duality principle. The proofs will be carried out step by step, with explicit justification for each step given by referring to the appropriate postulate or previously proved theorem. Two of the general methods of proof used in mathematics are

- Proof by contradiction
- Proof by the principle of (mathematical) induction

A proof by contradiction proceeds by assuming that the opposite of the desired result is true, and then deducing from this assumption a result that contradicts an already-known truth. This means that the opposite of the desired result is not true; therefore, the desired result itself must be true.

A proof by the principle of induction proceeds as follows. A proposition $P(i)$ is claimed to be true for all integers i. To prove the claim, it is necessary to do two things:

- Prove that the claim is true for some small integer, say $i = 1$.
- Assume it to be true for an arbitrary integer k and then show that it must, therefore, be true for the next integer, $k + 1$.

The latter step means that since the result is true for $i = 1$, it must be true for the next integer $i = 2$ $(1 + 1)$; then it must be true for $i = 3$ $(2 + 1)$; and so on for all other integers.

Another general method of proof, which is discussed in the next subsection, is especially valid for a Boolean algebra with only two elements.

Note that the symbol for logical multiplication (•) is often omitted for simplicity, and $x \bullet y$ is written as xy. However, whenever there might be confusion, the operator symbol should be explicitly shown. Confusion can arise, for example, if the name of a logical variable itself consists of multiple characters. Thus, a variable might be called OUT2, designating output number 2, rather than the logical product of O and U and T and 2. In this chapter the variables are given simple names; hence, we will often omit the logical product symbol. Later we will show it explicitly whenever necessary to avoid confusion. Another possible notation when variable names consist of more than one symbol is to enclose the variable names in parentheses. Thus, the parentheses in (OUT2)(OUT3) permit the omission of the logical multiplication symbol. Now on to the theorems.

Theorem 1 Null Law

1a. $x + 1 = 1$
1b. $x \bullet 0 = 0$

Note that each of these laws follows from the other one by duality; hence, only one needs explicit proof. Let's prove the second one.

$x \bullet 0 = 0 + (x \bullet 0)$	Postulate 2a
$= (x \bullet x') + (x \bullet 0)$	Postulate 5b
$= x \bullet (x' + 0)$	Postulate 4a
$= x \bullet x'$	Postulate 2a
$= 0$	Postulate 5b

Theorem 1a follows by duality. ■

Theorem 2 Involution $(x')' = x$
In words, this states that the complement of the complement of an element is that element itself. This follows from the observation that the complement of an element is unique. The details of the proof are left for you (see Problem 1d). ■

Theorem 3 Idempotency

3a. $x + x = x$
3b. $x \bullet x = x$

To prove Theorem 3a,

$x + x = (x + x) \bullet 1$	Postulate 2b
$= (x + x) \bullet (x + x')$	Postulate 5a
$= x + x \bullet x'$	Postulate 4b
$= x + 0$	Postulate 5b
$= x$	Postulate 2a

Theorem 3b is true by duality. ■

Theorem 4 Absorption

4a. $x + xy = x$
4b. $x(x + y) = x$

To prove Theorem 4a,

$$\begin{aligned} x + x \bullet y &= x \bullet 1 + xy && \text{Postulate 2b} \\ &= x \bullet (1 + y) && \text{Postulate 4a} \\ &= x \bullet 1 && \text{Postulate 3a and Theorem 1a} \\ &= x && \text{Postulate 2b} \end{aligned}$$

Theorem 4b is true by duality. ■

Theorem 5 Simplification

5a. $x + x'y = x + y$
5b. $x(x' + y) = xy$

To prove Theorem 5b,

$$\begin{aligned} x(x' + y) &= xx' + xy && \text{Postulate 4a} \\ &= 0 + xy && \text{Postulate 5b} \\ &= xy && \text{Postulate 2a} \end{aligned}$$

Theorem 5a is true by duality. ■

Theorem 6 Associative Law

6a. $x + (y + z) = (x + y) + z = x + y + z$
6b. $x(yz) = (xy)z = xyz$

To prove Theorem 6a requires some ingenuity. First, form the logical product of the two sides of the first equality:

$$A = [x + (y + z)] \bullet [(x + y) + z]$$

Then expand this product using the distributive law, first treating the quantity in the first brackets as a unit to start, and going on from there; and then treating the quantity in the second brackets as a unit to start, and going on from there. The result is $A = x + (y + z)$ in the first case and $A = (x + y) + z$ in the second case. (Work out the details.) The result follows by transitivity (if two quantities are each equal to a third quantity, they must be equal to each other). Since the result is the same no matter how the individual variables are grouped by parentheses, the parentheses are not needed and can be removed.

Theorem 6b follows by duality. ■

Theorem 7 Consensus

7a. $xy + x'z + yz = xy + x'z$
7b. $(x + y)(x' + z)(y + z) = (x + y)(x' + z)$

To prove Theorem 7a,

$$\begin{aligned} xy + x'z + yz &= xy + x'z + yz(x + x') && \text{Postulate 5a} \\ &= xy + x'z + yzx + yzx' && \text{Postulate 4a} \\ &= (xy + xyz) + (x'z + x'zy) && \text{Postulate 3b and Theorem 6a} \\ &= xy + x'z && \text{Theorem 4a} \end{aligned}$$

Theorem 7b is true by duality. ■

Theorem 8 De Morgan's Law

8a. $(x + y)' = x'y'$
8b. $(xy)' = x' + y'$

Prove Theorem 8a by showing that $x'y'$ satisfies both conditions in Postulate 5 of being the complement of $x + y$.

Condition 1

$$\begin{aligned} (x + y) + x'y' &= (x + x'y') + y && \text{Postulate 3a and Theorem 6a} \\ &= (x + y') + y && \text{Theorem 5a} \\ &= x + (y' + y) && \text{Theorem 6a} \\ &= x + 1 && \text{Postulate 5a} \\ &= 1 && \text{Theorem 1a} \end{aligned}$$

Condition 2

$$\begin{aligned} (x + y)(x'y') &= xx'y' + yx'y' && \text{Postulates 3b and 4a} \\ &= 0 \cdot y' + x'(yy') && \text{Postulates 5b and 3b} \\ &= 0 && \text{Postulates 5b and Theorem 1b} \end{aligned}$$

Theorem 8b is true by duality. ■

A number of other important results are left for you to prove in the problem set, but we will use them here as if proved. They include the following:

1. The identity elements 0 and 1 are distinct elements.
2. The identity elements are unique.
3. The inverse of an element is unique.

Exercise 1 Prove that each identity element is the complement of the other one. ◆

Switching Algebra

For the Boolean algebra discussed so far in this book, the domain has not been restricted. That is, no limitation has been placed on the number of elements in the Boolean algebra. From Huntington's postulates, we know that in every Boolean algebra there are two specific elements: the identity elements. Hence, any Boolean algebra has *at least* two elements. In this book, let us henceforth limit ourselves to a *two-element* Boolean algebra.[3]

[3]It is possible to prove that the number of elements in any Boolean algebra is some power of 2: 2^n, for $n \geq 1$.

In a 1937 paper, Claude Shannon implemented a two-element Boolean algebra with a circuit of switches.[4] Now a switch is a device that can be placed in either one of two stable positions: *off* or *on*. These positions can just as well be designated 0 and 1 (or the reverse). For this reason, two-element Boolean algebra has been called *switching algebra*. The identity elements themselves are called the *switching constants*. Similarly, any variables that represent the switching constants are called *switching variables*.

This explains some of the common terminology used in this area, but we have already used some terminology whose source is not evident. The terms *logical multiplication* and *logical addition* were introduced in the first of Huntington's postulates. To explain where the adjective *logical* comes from, we will have to digress slightly.

Over the centuries a number of different algebraic systems have been developed in different contexts. The language used in describing each system and the operations carried out in that system made sense in the context in which the algebra was developed. The algebra of *sets* is one of these; another is a system called *propositional logic,* which was developed in the study of philosophy.

It is possible for different algebraic systems, arising from different contexts, to have similar properties. This possibility is the basis for the following definition.

> *Two algebraic systems are said to be* isomorphic *if they can be made identical by changing the names of the elements and the names and symbols used to designate the operations.*

Propositional logic is concerned with simple propositions—whether or not they are true or false, how the simple propositions can be combined into more complex propositions, and how the truth or falsity of the complex propositions can be deduced from the truth or falsity of the simple ones. A simple proposition is a declarative statement that may be either true or false, but not both. It is said to have two possible *truth values:* true (T) or false (F). Examples are

"The earth is flat." F
"The sum of two positive integers is positive." T

It is not the intention here to pursue this subject in great detail. However, it turns out that two-valued Boolean algebra is isomorphic with propositional logic. Hence, whatever terminology, operations, and techniques are used in logic can be applied to Boolean algebra, and vice versa.

To illustrate, the elements of Boolean algebra (1 and 0) correspond to the truth (T) or falsity (F) of propositions; T and F could be labeled 1 and 0, respectively, or the opposite. Or the elements of Boolean algebra, 1 and 0, could be called "truth values," although the ideas of truth and falsity have no philosophical meaning in Boolean algebra.

---

[4]To *implement* a mathematical expression means to construct a model of a physical system, or the physical system itself, whose performance matches the result of the mathematical operation. Another verb with the same meaning is "to realize." The physical system, or its model, so obtained is said to be an *implementation* or a *realization.*

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

Figure 1 AND truth table.

One proposition is said to be the *negation* of another proposition if it is false whenever the other one is true. ("It is not snowing" is the negation of "It is snowing.") If p is a proposition, then not-p is its negation. This is isomorphic with the complement in Boolean algebra, and the same symbol (prime) can be used to represent it: not-p is written p'. Nobody will be hurt if we use the term *negation* in Boolean algebra to stand for *complement*.

Similar isomorphic relations exist between the operations of Boolean algebra and the connectives that join propositions together. However, further consideration of these will be postponed to the next section.

2 SWITCHING OPERATIONS

A unary operation and two binary operations, with names borrowed from propositional logic, were introduced in Huntington's postulates. For two-element (switching) algebra it is common to rename these operations, again using terms that come from logic.

The AND Operation

Let's first consider logical multiplication (AND) of two variables, xy. The operation will result in different values depending on the values taken on by each of the elements that the variables represent. Thus, if $x = 1$, then from Postulate 2b, $xy = y$; but if $x = 0$, then from Theorem 1, $xy = 0$, independent of y. These results can be displayed in a table (Figure 1) that lists all possible combinations of values of x and y and the corresponding values of xy.

This table is called a *truth table*. Neither the word *truth* nor the name of the operation, AND, makes any sense in terms of Boolean algebra; the terms are borrowed from propositional logic. The operation $x \cdot y$ is like the compound proposition

"The moon is full (x) and the night is young (y)."

This compound proposition is true only if *both* of the simple propositions "the moon is full" and "the night is young" are true; it is false in all other cases. Thus, xy in the truth table is 1 only if both x and y are 1. Study the table thoroughly.

The OR Operation

Besides "and," another way of connecting two propositions is with the connective "or." But this connective introduces some ambiguity. Suppose it is claimed that at 6 o'clock it will be raining or it will be snowing. The compound proposition will

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

Figure 2 OR truth table.

be true if it rains, if it snows, or if *it both rains and snows*—that is, if either or both simple propositions are true. The only time it will be false is if it neither rains nor snows.[5] These results, like the truth table for AND, can be summarized in a truth table for the connective OR. (Confirm the entries in Figure 2.)

Let's see how this compares in switching algebra with logical addition, $x + y$. From Huntington's Postulate 2a, if $y = 0$, then $x + y = x$, so $x + y$ will have whatever value x has. But if $y = 1$, then from Theorem 1a, $x + y = x + 1 = 1$, for both values of x. Verify that these values correspond exactly to the values given in the OR truth table.

The NOT Operation

Finally, the complement operation is isomorphic with negation, or NOT, in logic. It is a simple matter to set up a truth table for this operation; we leave it for you to carry out.

Exercise 2 Construct a truth table for the NOT operator. ◆

Commentary

The isomorphism of switching algebra with propositional logic has introduced a new tool, the truth table, that can be used to establish relationships among switching operations. For example, the theorems proved by application of Huntington's postulates together with previously proved theorems can also be proved from truth tables. We illustrate this by applying the truth tables for the AND, OR, and NOT operations to demonstrate the validity of the first form of De Morgan's law:

$$(x + y)' = x'y'$$

The result is shown in Figure 3. The last two columns are the same for all possible combinations of the switching variables; this proves De Morgan's law.

The procedure just used for establishing De Morgan's law illustrates a general method for proving results in switching algebra.

> *The truth-table method of proving a relationship among switching variables, by verifying that the reltionship is true for all possible combinations of values of the variables, is called the* method of perfect induction.

[5]The ambiguity in the "or" connective is seen in such a statement as "That animal is a cat or a dog." In this case, it is possible for the animal to be a cat or to be a dog, but certainly not both. In such a case, something other than logical addition is needed to describe the connective. This will be discussed in section 5.

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

Figure 3 Truth table for De Morgan's law.

This exhaustive approach can become quite cumbersome if the number of variables is large. Some might consider this approach to be intellectually and aesthetically less satisfying than applying the postulates of Boolean algebra and the theorems already proved. Even so, it is valid.

In De Morgan's law, let each side be called z. Then $z = (x + y)'$ and $z = x'y'$. There are two operations, OR and NOT, on the right side of the first expression. Spurred on by textbooks that they might have consulted in the library, some students might be tempted to think that the way to evaluate the right side is just to figure out which of the two operators "takes precedence." Concentrating on the significance of parentheses in algebraic expressions (both Boolean and ordinary) and the meanings of each Boolean operator should suffice to clarify the matter.

A set of parentheses is a tool used to group some variables into a unit; the operations are to be performed on the unit, not on parts of it within the parentheses. Thus $(x + y)'$ means forming the unit $x + y$ and then taking the complement of this unit. The unit can be given a name, say w. So $(x + y)'$ means w'. In terms of w, you can't even formulate the question of which operation takes precedence and should be performed first! You obviously take the OR of x and y, to get w; and then you take NOT w.

Similarly, for $z = x' \cdot y'$, the question, "Which do I do first—NOT and then AND, or AND first and then NOT?" is meaningless. Perhaps it will be simpler if we define $u = x'$ and $v = y'$; then $z = uv$. There is now no question—we take the AND of two things, u and v, which happen to be the NOT of x and the NOT of y, respectively. Thus, z = (NOT x) AND (NOT y); it couldn't be clearer.

It is common practice, just for simplicity, to omit parentheses around ANDed variables with the understanding that the AND will be performed before other operations on the ANDed unit. Indeed, the same convention—that the times operation is performed before the plus—is used in ordinary algebra as well. So instead of trying to memorize the order in which operations are to be performed in a given expression, concentrate on the meanings of the fundamental AND, OR, and NOT operations. Then you can't go wrong.

3 SWITCHING EXPRESSIONS

Look back at De Morgan's law in Theorem 8. Each side consists of certain switching variables related by means of AND, OR, and NOT operations. Each is an example of a *switching expression*, which we now formally define:

> *A switching expression is a finite relationship among switching variables (and possibly the switching constants 0 and 1), related by the AND, OR, and NOT operations.*

Some simple examples of switching expressions are $xx' + x$, $z(x + y')'$, and $y + 1$. A more complex example of a switching expression is

$$E = (x + yz)(x + y') + (x + y)'$$

where E stands for "expression."

Note that expressions are made up of variables, or their complements, on which various operations are to be performed. For simplicity, we refer to variables or complements of variables as *literals*. The expression E consists of the logical product of two expressions logically added to another term. (When discussing logical sums and products, we will usually drop the adjective *logical* for simplicity. But remember that it is always implied.) The second term in the product is itself a sum of literals, $x + y'$. The first term in the product cannot be described simply as a sum or a product.

A given expression can be put in many equivalent forms by applying Boolean *laws* (that's what we will call the postulates and theorems for short). But, you might ask, what's the point? Why bother to perform a lot of algebra to get a different form? At this time we'll give only a tentative, incomplete answer. Each switching variable in an expression presumably represents a signal; the logical operations are to be implemented (carried out) by means of units of hardware whose overall output is to correspond to the expression in question. If a given expression can be represented in different forms, then different combinations of hardware can be used to give the same overall result. Presumably, some configurations of hardware have advantages over others. More systematic methods for treating different representations of switching expressions will be taken up in Chapter 3; their implementation will be continued in Chapter 4. Here we are only setting the stage.

We return now to expression E, renamed E_1 in what follows. Equivalent expressions can be found by applying specific laws of switching algebra. Applying the distributive law to the product term and De Morgan's law to the last term leads to E_2; then

$E_1 = (x + yz)(x + y') + (x + y)'$	
$E_2 = xx + xy' + xyz + y'yz + x'y'$	
$E_3 = x + x\,(y' + yz) + x'y'$	Theorem 3a, Postulates 4a and 5b
$E_4 = x + x'y'$	Postulate 4a and Theorem 4a
$E_5 = x + y'$	Theorem 5a

A fairly complicated expression has been reduced to a rather simple one. Note that E_2 contains a term yy', which equals the identity element 0. We say the expression is *redundant*. More generally, an expression will be redundant if it contains

- Repeated literals (xx or $x + x$)
- A variable and its complement (xx' or $x + x'$)
- Explicitly shown switching constants (0 or 1)

Redundancies in expressions need never be implemented in hardware; they can be eliminated from expressions in which they show up.

Minterms, Maxterms, and Canonic Forms

Given an expression dependent on n variables, there are two specific and unique forms into which the expression can always be converted. These forms are the subject of this section.

The expression E_1 in the preceding section had mixtures of terms that were products or sums of other terms. Furthermore, although E_1 seemed to be dependent on three variables, one of these variables was redundant. The final, equivalent form was the sum of two terms, each being a single literal.

In the general case, expressions are dependent on n variables. We will consider two nonredundant cases. In one case, an expression consists of nothing but a sum of terms, and each term is made up of a product of literals. Naturally, this would be called a *sum-of-products* (s-of-p) form. The maximum number of literals in a nonredundant product is n. In the second case to be considered, an expression consists of nothing but a product of terms, and each term is made up of a sum of literals; this is the *product-of-sums* (p-of-s) form. Again, the maximum number of literals in a nonredundant sum is n.

Suppose that a product term in a sum-of-products expression, or a sum term in a product-of-sums form, has fewer than the maximum number n of literals. To distinguish such cases from one in which each term is "full," we make the following definition:

> *A sum-of-products or product-of-sums expression dependent on n variables is* canonic *if it contains no redundant literals and each product or sum has exactly n literals.*[6]

Each product or sum term in a canonic expression has as many literals as the number of variables.

EXAMPLE 1

An example of an expression having three variables, in product-of-sums form, is E_1 below (not the previous E_1). It is converted to sum-of-products form as follows, with each step justified by the listed switching laws.

$E_1 = (x' + y' + z)(x + y + z')(x + y + z)$	
$E_2 = (x' + y' + z)[(x + y)(x + y) + (x + y)(z + z') + zz']$	Postulate 4a
$E_3 = (x' + y' + z)(x + y)$	Postulate 1 and Theorem3b
$E_4 = xy' + x'y + xz + yz$	Posulates 4a and 5b, Theorem 7

In going from E_1 to E_4, redundancies were eliminated at each step. The original expression is in product-of-sums form, with the maximum number of literals in each term; hence, it is canonic. The final form, on the other hand, is in sum-of-products form, but it is not canonic; it has only two literals in each term.

Given a noncanonic sum-of-products expression, it is always possible to convert it to canonic form—if there is some reason to do so! The term xy' in expression E_4, for example, has the variable z missing. Hence, multiply the term by $z + z'$, which equals 1 and so does not change the logical value; then expand. The same idea can be used with the other terms. Carrying out these steps on E_4 leads to the following:

$$E_5 = xy'(z + z') + x'y(z + z') + xz(y + y') + yz(x + x')$$
$$E_6 = xy'z + xy'z' + x'yz + x'yz' + xyz$$

[6]Some authors use *canonical* instead of *canonic*.

(You should confirm the last line; note that redundancies that are created in the first line by applying Postulate 4a have to be removed.) This is now in canonic sum-of-products form. ■

In a sum-of-products expression dependent on n variables, in order to distinguish between product terms having *n* literals (the maximum) and those having fewer than *n*, the following definition is made:

A canonic nonredundant product of literals is called a minterm.[7]

That is, a minterm is a nonredundant product of as many literals as there are variables in a given expression. Thus, each term in E_6 is a minterm, and so the entire expression is a sum of minterms.

The same idea applies to a product-of-sums expression. To distinguish between product terms having the maximum number of literals and others, the following definition is made:

A canonic nonredundant sum of literals is called a maxterm.

Each factor in expression E_1 in the preceding development is a maxterm; the entire expression is a product of maxterms. If a product-of-sums expression is not initially canonic, it is possible to convert it to canonic form in a manner similar to the one used for the sum-of-products case, but with the operations interchanged.

Exercise 3 Convert the following expression to a canonic product of maxterms: $E = (x + y')(y' + z')$.
Answer[8]

Generalization of De Morgan's Law

One of the Boolean laws that has found wide application is De Morgan's law. It was first stated as Theorem 8 in terms of two variables. It and its dual form, where the sum and product operations are interchanged, are repeated here:

$$\text{(a) } (x_1 + x_2)' = x_1'x_2' \quad \text{and} \quad \text{(b) } (x_1x_2)' = x_1' + x_2'$$

In words, complementing the sum (product) of two switching variables gives the same result as multiplying (adding) their complements. Suppose we were to increase the number of variables to three; would the result still hold true? That is, we want to perform the following operation: $(A + B + C)'$. Let's rename $A + B$ as D; then what we want is $(D + C)'$. But that's just $D'C'$, by De Morgan's law for two variables; and again by De Morgan's law, $D' = (A + B)' = A'B'$. Hence, the final result is:

$$\text{(a) } (A + B + C)' = A'B'C' \qquad \text{(b) } (ABC)' = A' + B' + C' \tag{1}$$

(The second form follows by duality.)

---

[7]The name doesn't seem to make sense—what is "min" about it? If anything, from the fact that we had to go from terms with two literals to ones with three literals, one would think it should be called "max"! Your annoyance will have to persist until Chapter 3, when the whole thing will be clarified.

[8]$E = (x + y' + z)(x + y' + z')(x' + y' + z')$ ◆

Why stop with three variables? The general case can be written as follows:

$$(x_1 + x_2 + \cdots + x_n)' = x_1'x_2' \cdots x_n' \tag{2}$$

$$(x_1x_2x_3 \cdots x_n)' = x_1' + x_2' + \cdots + x_n' \tag{3}$$

In words, (2) says that the complement of the logical sum of any number of switching variables equals the logical product of the complements of those variables. (In different terms, the NOT of a string of ORs is the AND of the NOTs of the variables in that string.) In (3), we interchange the operations AND and OR. Just writing the generalization doesn't make it true; it is left for you to prove.

Exercise 4 Prove one of the generalized De Morgan laws by mathematical induction. That is, assume it is true for k variables, and show that it is true for $k + 1$. Since we know it is true for two variables, then it will be true for three, (as already proved); since it is true for three, then ... and so on. ◆

Something more general can be concluded from De Morgan's law. In (2) and (3), the generalization involves increasing the number of variables in the theorem. On the left sides, the operations whose NOT is being taken are the sum and product. Suppose now that it is these *operations* that are generalized! That is, on the left, we are to take the complement of some expression that depends on n variables, where the + and • operations are performed in various combinations. Will the result be the same if we take the same expression but interchange the sum and product operations while complementing all the variables? The generalization is

$$E'(x_1, x_2, \ldots, x_n, +, \bullet) = E(x_1', x_2', \ldots, x_n', \bullet, +) \tag{4}$$

Note the order of the sum and product operations on each side; it indicates that the two operations are interchanged on the two sides, wherever they appear. The result can be proved (we won't do it here) by mathematical induction on the number of operations. (You can do it, if you want.)

EXAMPLE 2

The following expression is given: $E = xy'z + x'y'z' + x'yz$. To find the complement of E, we interchange the + and • operations and replace each variable by its complement. Thus,

$$E' = (x' + y + z')\bullet(x + y + z)\bullet(x + y' + z')$$

Let us first rewrite this expression in sum-of-products form by carrying out the product operations on the terms within parentheses and using necessary Boolean algebra to simplify. (You carry out the steps before you check your work in what follows.)

$$\begin{aligned} E' &= (x'y + x'z + yx + y + yz + z'x + z'y)(x + y' + z') \\ &= (x'z + xz' + y)(x + y' + z') \\ &= xy + yz' + x'y'z + xz' \end{aligned}$$

x	y	x'	y'	$x'y'$	E_1: $x + x'y'$	E_2: $x + y'$
0	0	1	1	1	1	1
0	1	1	0	0	0	0
1	0	0	1	0	1	1
1	1	0	0	0	1	1

Figure 4 Truth table for E_1 and E_2

To verify that this expression correctly gives the complement of E, you can take its complement using the same generalization of De Morgan's law and see if the original expression for E results. ■

Exercise 5 Take the complement of E' in the preceding expression, using (4). Put the result in sum-of-products form and verify that the result is the same E as given in Example 2. ◆

4 SWITCHING FUNCTIONS

In the preceding section we saw several examples in which a given switching expression was converted to other, equivalent expressions by applying Boolean laws to it. The equivalent expressions have the same logic values for all combinations of the variable values. The concept of "function" plays a very important role in ordinary algebra. So far, such a concept has not been introduced for switching algebra. We will do so now.

The discussion will start by considering two simple expressions:

$$E_1 = x + x'y' \quad \text{and} \quad E_2 = x + y'$$

Each expression depends on two variables. Collectively, the variables can take on $2^2 = 4$ combinations of values. (For n variables, the number of combinations of values is 2^n.) For any combination of variable values, each expression takes on a value that is found by substituting the variable values into it. When this is done for all combinations of variable values, the result is the truth table in Figure 4. (Confirm all the entries.) The last two columns are the same. This is hardly surprising given Theorem 5a. (Look it up.)

This example illustrates the principle that different expressions can lead to the same truth values for all combinations of variable values.

Exercise 6 Using a truth table, confirm that the expression: $E = xy + xy' + y'$ has the same truth values as E_1 and E_2 in Figure 4. ◆

There must be something more fundamental than equivalent expressions; whatever it is may be identified by its truth values. On this basis, we define a switching function as follows:

> *A switching function is a specific unique assignment of switching values 0 and 1 for all possible combinations of values taken on by the variables on which the function depends.*

x	y	z	f_1	f_2	f_1'	f_2'	$f_1 + f_2'$	$(f_1 f_2)'$
0	0	0	0	1	1	0	0	1
0	0	1	1	0	0	1	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	1	0	0	1	0
1	0	0	0	1	1	0	0	1
1	0	1	0	1	1	0	0	1
1	1	0	1	1	0	0	1	0
1	1	1	1	1	0	0	1	0

Figure 5 Truth tables for several functions.

For a function of n variables there are 2^n possible combinations of values. For each combination of values, the function can take on one of two values. Hence, the number of distinct assignments of two values to 2^n things is 2 to the 2^n power. Hence,

The number of switching functions of n variables is 2 to the 2^n.

On this basis, there are 16 switching functions of two variables and 256 functions of three variables; the number escalates rapidly for more variables.

Now consider the functions $f_1(x, y, z)$ and $f_2(x, y, z)$ whose truth values are shown in Figure 5. Any switching operation (AND, OR, NOT) can be performed on these functions and the results computed using the truth table. Thus, f_1' can be formed by assigning the value 1 whenever f_1 has the value 0, and vice versa. Other combinations of f_1, f_2, or their complements can be formed directly from the truth table. As examples, $f_1 + f_2'$ and $(f_1 f_2)'$ are also shown.

Switching Operations on Switching Functions

It is clear that functions—and, therefore, expressions that represent functions—can be treated as if they were variables. Thus, switching laws apply equally well to switching expressions as to variables representing the switching elements.

Note carefully that there is a difference in meaning between a switching *function* and a switching *expression*. A function is defined by listing its truth values for all combinations of variable values, that is, by its truth table. An expression, on the other hand, is a combination of literals linked by switching operations. For a given combination of variable values, the expression will take on a truth value.

If the truth values taken on by an expression, for all possible combinations of variable values, are the same as the corresponding truth values of the function, then we say that the expression *represents* the function. As observed earlier, it is possible to write more than one expression to represent a specific function. Thus, what is fundamental is the switching *function.* Of all the expressions that can represent a function, we might seek particular ones that offer an advantage in some way or other. This matter will be pursued further in Chapter 3.

Decimal Code	x	y	z	f	Minterms	Maxterms
0	0	0	0	0	$x'y'z'$	$x + y + z$
1	0	0	1	0	$x'y'z$	$x + y + z'$
2	0	1	0	1	$x'yz'$	$x + y' + z$
3	0	1	1	1	$x'yz$	$x + y' + z'$
4	1	0	0	1	$xy'z'$	$x' + y + z$
5	1	0	1	1	$xy'z$	$x' + y + z'$
6	1	1	0	0	xyz'	$x' + y' + z$
7	1	1	1	1	xyz	$x' + y' + z'$

Figure 6 Truth table for example function.

Number of Terms in Canonic Forms

In treating different switching expressions as equivalent in the previous section, Example 1 gave several different expressions dependent on three variables. We say that all these expressions represent the same function. E_1, in canonic product-of-sums form, has three terms as factors in the product—three maxterms. On the other hand, E_6 is in sum-of-products form and has five terms in the sum—five minterms. The sum of the number of minterms and maxterms equals 8, which is 2^3, the number of possible combinations of 3 bits.

This is an interesting observation. Let's pursue it by constructing a truth table of the function represented by expressions E_1 and E_6. First we write the minterms and the maxterms in the order xyz: 000 to 111.

$$E_6 = x'yz' + x'yz + xy'z' + xy'z + xyz$$
$$E_1 = (x + y + z)(x + y + z')(x' + y' + z)$$

The table is shown in Figure 6. Something very interesting can be observed by examining the expressions for the minterms and the maxterms. Suppose we take the complement of the minterm corresponding to 000. By De Morgan's law, it is exactly the first maxterm. Confirm that this is indeed true for each row in the table. From this we conclude that to find the canonic product-of-sums form is *to apply De Morgan's law to each minterm that is not present.*

The result in this example is a general one. Given the canonic sum-of-products form of a switching function, the way to obtain a canonic product-of-sums expression is as follows:

- Apply De Morgan's law to the complement of each minterm that is *absent* in the sum-of-products expression.
- Then form the product of the resulting maxterms.

Conversely, to obtain a sum-of-products expression from a given product-of-sums expression,

- Apply De Morgan's law to each sum term absent from the product;
- Then form the sum of the resulting minterms.

(You can confirm this from the same truth table.)

Exercise 7 Given the product-of-sums expression in E_1 in this section, use the preceding approach to find the corresponding sum-of-products form. ◆

This approach avoids the necessity of carrying out switching algebra to convert from one form to another; instead, given one of the two forms, you can find the other one by mentally carrying out some steps merely by inspection.

Shannon's Expansion Theorem

Previous examples have shown that more than one expression can represent the same switching function. Two specific expressions noted are the sum-of-products and product-of-sums forms. The question arises as to whether a given function can *always* be expressed in these specific standard forms. An answer to this question, which is the subject of this section, was provided by Claude Shannon.[9]

Sum-of-Products Form

Suppose that $f(x_1, x_2, ..., x_n)$ is any switching function of n variables. Shannon showed that one way of expressing this function is

$$f(x_1, x_2, ..., x_n) = x_1 f(1, x_2, ..., x_n) + x_1' f(0, x_2, ..., x_n) \tag{5}$$

On the right side, the function is the sum of two terms, one of them relevant when x_1 takes on the value 1 and the other when x_1 takes on the value 0. The first term is x_1 times what remains of f when x_1 takes on the value 1; the second term is x_1' times what remains of f when x_1 takes on the value 0. The proof of this expression is easy; it follows by perfect induction. That is, if the result is true for all possible values of variable x_1 (there are only two values—1 and 0) it must be true, period. (Confirm that it is true for both values of x_1.)

You can surely see how to proceed: Repeat the process on each of the remaining functions, this time using another variable, say x_2. Continue the process until all variables are exhausted. The result is simply a sum of terms, each of which consists of the product of n literals multiplied by whatever the function becomes when each variable is replaced by either 0 or 1. But the latter is simply the value of the function when the variables collectively take on some specific combination of values; this value is either 0 or 1. So the end result is a sum of all possible nonredundant products of the $2n$ literals (the n variables and their complements), some of which are multiplied by 1 and the remaining ones by 0. The latter are simply absent in the final result. According to Shannon, this will equal the original function. The proof at each step is by perfect induction.

Shannon's expansion theorem in the general case is

$$\begin{aligned} f = {} & a_0 x_1' x_2' \cdots x_n' + a_1 x_1' x_2' \cdots x_{n-1}' x_n + a_2 x_1' x_2' \cdots x_{n-1} x_n' + \cdots \\ & + a_{2^n-2} x_1 x_2 \cdots x_n' + a_{2^n-1} x_1 x_2 \cdots x_n \end{aligned} \tag{6}$$

[9]Claude E. Shannon, "A Symbolic Analysis of Relay and Switching Circuits," *Trans AIEE*, 57 (1938), 713–723.

Each a_i is a constant in which the subscript is the decimal equivalent of the multiplier of a_i viewed as a binary number. Thus, for three variables, a_5 (binary 101) is the coefficient of $x_1x_2'x_3$.

Shannon's expansion theorem in (6) is a major result. It shows that

Any switching function of n variables can be expressed as a sum of products of n literals, one for each variable.

Exercise 8 Let f be a function of two variables, x_1 and x_2. Suppose that f takes on the value 1 for the combinations x_1x_2: 00, 10, 11 and the value 0 for the remaining combination. Determine the expression resulting from Shannon's expansion theorem carried out to completion in this case. As an added task, not dependent on Shannon's theorem, simplify the result if possible.
Answer[10]

In carrying the expansion process in (6) through to completion, a subconscious assumption has been made. That assumption is that, at each step until the last, after some (but not all) variables have been replaced by constants, the function that remains is itself not a constant. That is, what remains still depends on the remaining variables. If, instead, this remaining function reduces to a constant, the process will prematurely terminate and the corresponding term will have fewer than n literals. We have already discussed in the preceding section how to restore any missing literals. Hence, even in such a case, the generalization is valid.

Shannon's theorem constitutes an "existence proof." That is, it proves by a specific procedure that a certain result is true. Once the proof is established, then for any particular example it isn't necessary to follow the exact procedure used to prove the theorem. Now that we know that a function can always be put into the stated form, we can search for easier ways of doing it.

Product-of-Sums Form

Shannon's expansion theorem in (6) is a sum-of-products form. It's easy to surmise that a similar result holds for the product-of-sums form. It isn't necessary to go through an extensive proof for this form. The result can be obtained from (5) and (6) by duality. That is, 0 and 1 are interchanged, as are the sum and product operations. Doing this, the counterparts of (5) and (6) become

$$f(x_1, x_2, \ldots, x_n) = [x_1 + f(0, x_2, \ldots, x_n)][x_1' + f(1, x_2, \ldots, x_n)] \tag{7}$$

$$\begin{aligned} f = {} & (b_0 + x_1' + x_2' + \cdots + x_n')(b_1 + x_1' + \cdots + x_{n-1}' + x_n) \cdots \\ & (b_{n-2} + x_1 + x_2 + \cdots + x_n')(b_{n-1} + x_1 + \mathrm{x}_2 + \cdots + x_n) \cdots \end{aligned} \tag{8}$$

where the b_i are the constants 0, 1. These equations are the duals of (5) and (6). The last one shows that

Any switching function of n variables can be expressed as a product of sums of n literals, one for each variable.

If the expansion should terminate prematurely, the missing literals can always be restored by the approach illustrated in the last section. The constants

[10] $f = x_1'x_2' + x_1x_2' + x_1x_2 = x_1 + x_2'$ ◆

		NOT		AND	OR	XOR	NAND	NOR	XNOR
x	y	x'	y'	xy	$x + y$	$x'y + xy'$	$x' + y'$	$x'y'$	$xy + x'y'$
0	0	1	1	0	0	0	1	1	1
0	1	1	0	0	1	1	1	0	0
1	0	0	1	0	1	1	1	0	0
1	1	0	0	1	1	0	0	0	1

Figure 7 Truth table for basic operations.

will not show up explicitly in an actual function after redundancies are removed. If one of the constants is 1, for example, the entire corresponding factor will be 1, since $1 + x = 1$, and the corresponding factor will not be present. On the other hand, if any constant is 0, since $0 + x = x$, this constant does not show up but the corresponding factor will be present.

Exercise 9 Use the process of Shannon's theorem step by step to put the function in Exercise 8 in a product-of-sums form. Simplify the result, if possible. ◆

5 OTHER SWITCHING OPERATIONS

It was noted earlier that propositional logic is isomorphic with switching algebra. As already observed, the language of propositional logic has permeated the way in which operations and ideas are discussed in switching algebra. We will now borrow some other operations that arise naturally in propositional logic and convert them to our use.

Exclusive OR

In discussing the OR operation in Section 2, an example was given of a compound proposition that sounds as if it should be an OR proposition but isn't: "That animal is a cat or a dog." Although the animal could be a cat or a dog, it can't be *both* a cat and a dog. The operation $x + y$ is "inclusive" OR; it includes the case where both x and y are 1. The "or" in "cat or dog" is different; it has a different meaning in logic. It is called an Exclusive OR, XOR for short, and is given the symbol $\oplus$. Thus, $x \oplus y$ is true when x and y have opposite truth values, but it is false when x and y have the same value. The truth values of $x \oplus y$ are shown in Figure 7. (The table includes the truth values of all the basic logic operations, including some yet to come.)

The Exclusive OR is a function of two variables. The most general sum-of-products form for such a function can be written as follows:

$$x \oplus y = a_0x'y' + a_1x'y + a_2xy' + a_3xy \tag{9}$$

The values of the a_i coefficients can be read from the truth table: 0 for a_0 and a_3, 1 for a_1 and a_2. Thus, (9) reduces to

$$x \oplus y = x'y + xy' \tag{10}$$

Exercise 10 Starting with the sum-of-products form for the XOR in (10), obtain a canonic product-of-sums form.
Answer[11]

NAND, NOR, and XNOR Operations

Besides the NOT operation, we now have three in our repertoire: AND, OR, and XOR. Three additional operations can be created by negating (complementing, or taking the NOT of) these three:

NAND (NOT AND): $(xy)' = x' + y'$ (11)

NOR (NOT OR): $(x + y)' = x'y'$ (12)

XNOR (NOT XOR): $(x \oplus y)' = (x'y + xy')' = xy + x'y'$ (13)

The right side in (13) can be obtained from (10) by negating the truth values of XOR to obtain the values of a_i. (Confirm it.) The truth values for these three are easily obtained from the three operations of which they are the NOTs. All the results are shown in the preceding truth table (Figure 7).

Exercise 11 Discuss the nature of the right sides of (11) and (12) as sum-of-products or product-of-sums forms. ◆

Exercise 12 Apply switching laws to find a product-of-sums form of the right side of (13). ◆

Notice from Figure 7 that XNOR is 1 whenever x and y have the same value. In logic, the function of two variables that equals 1 whenever the two variables have the same value is called an *equivalence* relation. The symbol denoting an equivalence relation is a two-headed arrow, $x \Leftrightarrow y$. Thus, XNOR and equivalence have the same truth values and can be used interchangeably: A XNOR $B = A \Leftrightarrow B = xy + x'y'$.

Comparing two signals (inputs) to see if they are the same is an important operation in many digital systems. Thus, an XNOR gate is a one-bit *comparator;* when its output is 1, we know that the two inputs are the same. We will discuss how to use XNOR gates to compare numbers of more than 1 bit in Chapter 3.

Some of you may have some uneasiness with the introduction of XOR and the other operations in this section. Earlier, a switching expression was defined as one that includes the operators AND, OR, and NOT. Would this mean that something including XOR or the other operations discussed in this section cannot be a switching expression? This apparent contradiction is overcome by noting equations (10) to (13). Each of the operations defined by these operators is expressed in terms of AND, OR, and NOT, so the uneasiness should disappear.

6 UNIVERSAL SETS OF OPERATIONS

Switching algebra was introduced in terms of two binary operations (AND and OR) and one unary operation (NOT). Every switching expression is made up

[11]Add xx' and yy', and then use the distributive law: $(x + y)(x' + y')$. ◆

of variables connected by various combinations of these three operators. This observation leads to the following concept:

> *A set of operations is called* universal *if every switching function can be expressed exclusively in terms of operations from this set.*

(Some use the term *functionally complete* in place of *universal.*) It follows that the set of operations {AND, OR, NOT} is universal.

In the preceding section various other operations were introduced. Two of these are NAND and NOR:

$$x \text{ NAND } y\text{: } (xy)' = x' + y' \tag{14}$$

$$x \text{ NOR } y\text{: } (x + y)' = x'y' \tag{15}$$

We see that the right sides of these expressions are each expressed in terms of only two of the three universal operations (OR and NOT for the first, and AND and NOT for the second). This prompts a general question: Can one of the three operations be removed from the universal set and still leave a universal set? The answer is "yes, indeed" if one of the operations can be expressed in terms of the other two.

Consider the AND operation, xy. By De Morgan's law, it can be written as

$$xy = (x' + y')' \tag{16}$$

The only operations on the right are OR and NOT. Since every AND can be expressed in terms of OR and NOT, the set {AND, OR, NOT} can be expressed in terms of the set {OR, NOT}. Conclusion: The set {OR, NOT} is a universal set.

Exercise 13 By reasoning similar to that just used, show that the set {AND, NOT} is a universal set. ◆

The conclusion just reached is that any switching function can be expressed in terms of only two switching operations. But why stop there—why not try for just one? Let's explore the NAND operation. Since the set {AND, NOT} is universal, if we can express both those operations in terms of NAND, then, {NAND} will be universal! Here we go:

$$x' = x' + x' = (xx)' \qquad \text{Theorems 3a and 8a} \tag{17}$$

$$xy = ((xy)')' = [(xy)'\,(xy)']' \qquad \text{Theorems 2 and 3b} \tag{18}$$

The right sides are expressed in terms of nothing but NAND. Conclusion:

> *Any switching function can be expressed exclusively in terms of NAND operations.*

Exercise 14 Show by a procedure similar to the one just used that {NOR} is a universal set. ◆

The conclusion that results from the preceding exercise is

Any switching function can be expressed exclusively in terms of NOR operations.

The practical point of the preceding discussion is that, in the real world, switching operations are carried out by means of physical devices. If all switching

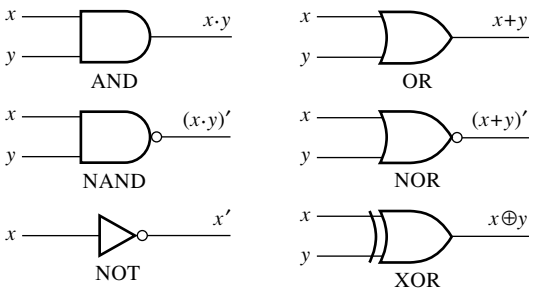

Figure 8 Symbols for logic gates.

functions can be expressed in terms of just one operation, then the physical implementation of any switching function can be carried out with only one type of physical device. This has obvious implications for simplicity, convenience, and cost, which we shall amplify in Chapter 3.

7 LOGIC GATES

Up to this point we have been dealing with rather abstract matters. We have discussed logic operations in an algebra called switching algebra. The variables that are manipulated in switching algebra are abstract switching variables. Nothing has been said so far that relates these switching variables to anything in the real world. Also, no connection has been made between the operations of switching algebra and physical devices.

We are now ready to change all that. For switching algebra to carry out real tasks, physical devices must exist that can carry out the operations of switching algebra as accurately and with as little delay as possible. The switching variables, which take on logic values of 0 and 1, must be identified with real signals characterized by a physical variable, such as voltage.

The generic name given to a physical device that carries out any of the switching operations is *gate.* However, in anticipation of discussing such devices, we will introduce another abstract representation of switching operations, a schematic representation for the real-life devices and for their interconnection in actual switching circuits.

Suppose that each operation in switching algebra is represented by a different schematic symbol, with a distinct terminal for each logic variable. Then any switching expression can be represented by some interconnection of these symbols. Attempts have been made in the past to adopt a set of standard symbols. One set has a uniform shape for all operations (a rectangle), with the operation identified inside.[12] A more common set of symbols for gates uses a distinct shape for each operation, as shown in Figure 8.

The NOT gate is called an *inverter*; each of the others is called by the name of the operation it performs—an AND gate, a NOR gate, and so on. Although each gate (except the inverter) is shown with two input terminals, there may be

[12]The Institute of Electrical and Electronic Engineers has published such standards (see IEEE Standard 91-1984). However, they have not been widely adopted.

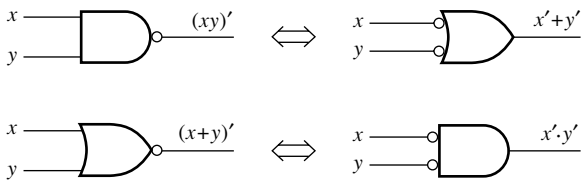

Figure 9 Two equivalent forms of NAND and NOR gates.

more than two inputs. (For the real devices of which these gates are abstractions, there are practical limitations on the number of inputs; these limitations will be discussed shortly.)

There is no problem with having more than two inputs for the two basic Boolean operators AND and OR, because both of these operators satisfy the associative law. The result is the same if we logically add x to $(y + z)$ or add $x + y$ first and then add it to z; unambiguously, the result is $x + y + z$. The same is true of the XOR operation. (Prove this to yourself.) However, the NAND and NOR operations are not associative. The output of a three-input NAND gate is $(xyz)'$; however, this is not the same as the NAND of x and y followed by the NAND with z. That is, $(xyz)' \neq ((xy)'z)'$.

A NAND gate is formed by appending a small circle (called a *bubble*), representing an inverter, to the output line of an AND gate. (The same is true for forming a NOR gate from an OR.) Furthermore, placing a bubble on an input line implies complementing the incoming variable before carrying out the operation performed by the gate.

Alternative Forms of NAND and NOR Gates

Other representations of a NAND gate and a NOR gate can be found by an application of De Morgan's law. For two variables, De Morgan's law is

$$(xy)' = x' + y' \tag{19}$$

$$(x + y)' = x'y' \tag{20}$$

The left sides of (19) and (20) are represented by the NAND and NOR gate symbols in Figure 8. The right side of (19) is an OR operation on inputs that are first inverted. Similarly, the right side of (20) is an AND operation on inverted inputs. These alternative forms of NAND and NOR gates are shown in Figure 9.

As far as ideal logic gates are concerned, the two forms of NAND or NOR gates are equally acceptable. However, when it comes to implementing the logic using real, physical gates, there is a practical difference between the two; we shall discuss this subsequently.

Something else interesting results from taking the inverse of both sides in (19):

$$((xy)')' = xy = (x' + y')'$$

Evidently, an AND gate with inputs x and y can be implemented with a NOR gate whose inputs are the complements of x and y. This result is shown schematically in Figure 10, where the bubbles at the inputs of the NOR gate represent the inverses of the input variables.

Figure 10 Equivalent forms of an AND gate.

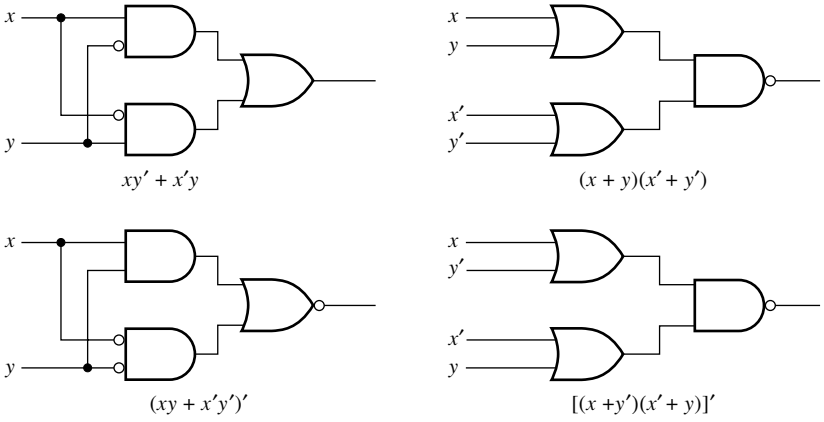

Figure 11 Alternative structures for XOR.

Exercise 15 By a procedure similar to that used above, find an equivalent form of an OR expression and draw the corresponding equivalent gates. ◆

Exclusive-OR Gates

Several different switching expressions can be written to represent the XOR function; some of these were shown earlier in this chapter. Four equivalent forms are as follows:

$$x \oplus y = xy' + x'y \tag{21}$$
$$= (x + y)(x' + y') \tag{22}$$
$$= (xy + x'y')' \tag{23}$$
$$= [(x + y')(x' + y)]' \tag{24}$$

The first two of these are the canonic s-of-p and p-of-s forms for the XOR function; they were given in (10) and in Exercise 9. Confirm the other two forms.

Not counting complementation, each of the expressions on the right involves three switching operations. Hence, each can be represented by three interconnected gates, as shown in Figure 11. Two of these involve the AND or NAND of two ORs, and two involve the OR or NOR of two ANDs. The number of gates and literals is the same in each case.

Commentary

Some observations can be made on the basis of what has just been discussed. *Logic gates* are schematic representations of switching operations. An interconnection of logic gates is just another way of writing a switching expression. We can say that switching algebra and the interconnections of logic gates are

isomorphic systems. When we use schematic symbols called logic gates or draw interconnections of gates to represent switching operations, there need be no implication of anything physical or real.

Suppose, however, that physical devices that perform the operations symbolized by the gates could be constructed, even if they were not perfect. Then we could construct a physical circuit (we might call it a *switching circuit*) that can *implement,* or *realize,* a specific switching expression.[13] The circuit would be an *implementation,* or *realization,* of the expression. This is what makes switching algebra, an abstract mathematical structure, so useful in the physical design of real circuits to carry out desired digital tasks.

As a simple illustration, consider an XOR gate with inputs x and y. We might be interested in comparing the inputs. It is clear from (21) that $x \oplus y = 1$ whenever the values of the two inputs are different. (The opposite would be true for an XNOR gate.) Thus, either gate can play the role of a comparator of 1-bit numbers.

As another illustration, it may not be necessary to explicitly use an inverter to obtain the complement of a switching variable. A certain device that generates a switching variable commonly generates its complement also, so that both are available.[14] Hence, if x and x' both appear in an expression, then in the corresponding circuit, either we can show separate input lines for both of them or we can indicate the complement with a bubble. If you glance back at Figure 11, you will see that each of these approaches is used in half the cases.

8 POSITIVE, NEGATIVE, AND MIXED LOGIC

No matter how electronic gates and circuits are fabricated, the logical values of 0 and 1 are achieved in the real world with values of physical variables—in most cases voltage but sometimes current. The actual voltage values depend on the specific technology; but whatever the two values are, one is higher than the other. Therefore, one of them is designated High (H) and the other Low (L). There is no necessary correlation between the high and low voltage levels and the logic values of 0 and 1. Two schemes are possible, as illustrated in the tables in Figure 12.

Although the 0 and 1 logic values are not numerical values, if we interpret them as such, positive logic (with 1 corresponding to H) appears to be the natural scheme. Negative logic is superfluous; it contributes nothing of value that

Logic Value	Voltage Level
0	L
1	H

(*a*)

Logic Value	Voltage Level
0	H
1	L

(*b*)

Figure 12 Correlations between logic values and voltage levels. (*a*) Positive logic. (*b*) Negative logic.

[13]See footnote 4 for the meanings of *realize* and *implement.*

[14]As will be discussed in Chapter 5, a device called a *flip-flop* has two outputs, one of which is the complement of the other.

is not achievable with positive logic. Hence, there is no useful reason for adopting it. However, it may be useful to adopt positive logic at some terminals of devices in a circuit and negative logic at other terminals. This possibility is referred to as *mixed logic*. Its greatest utility occurs in dealing with real physical circuits. The next few paragraphs will discuss the common terminology in the use of mixed logic.

Two concepts are utilized in addressing the issues surrounding mixed logic. One is the concept of *activity*. The normal state of affairs, it might be assumed, as *inactivity*. The telephone is normally quiet (inactive) until it rings (becomes active), and conversations can then be carried on. Lights are normally off (inactive) until they are activated by a switch. When a microprocessor is performing a READ operation, it is active. Otherwise it isn't doing anything; it is inactive.[15] Thinking of logic 1 in connection with activity and logic 0 with inactivity is just a habit.

At various terminals in a physical system that is designed to carry out digital operations, voltages can have either high (H) or low (L) values. As just discussed, the two ways in which voltage levels and logic values can be associated are $1 \leftrightarrow$ H and $0 \leftrightarrow$ L, or the opposite, $1 \leftrightarrow$ L and $0 \leftrightarrow$ H. What complicates life even more is that at some points in a circuit the association can be made in one way and at other points in the opposite way.

Because of the connection of activity with logic 1, if the association of $1 \leftrightarrow$ H is made, the scheme is said to be "active high." If the association $1 \leftrightarrow$ L is made, it is said to be "active low." But this description employs two steps, two levels of association: Activity is first connected with logic 1; then logic 1, in turn, is associated with a high or a low voltage. Schematically, the train of thought is

$$\text{activity} \rightarrow \text{logic } 1 \leftrightarrow \text{high voltage} = \text{active high}$$
$$\text{activity} \rightarrow \text{logic } 1 \leftrightarrow \text{low voltage} = \text{active low}$$

"Active high" means that logic 1 is associated with a high voltage. More simply, one could say 1-high or 1-H instead of "active high." The adjective *active* just adds distance to the real association to be established between a logic value and a voltage level.

Another concept sometimes used as a synonym for activity is the notion of *asserting*. Thus, "asserted high" and "asserted low" mean the same as "active high" and "active low" which, in the end, mean the association of logic 1 with a high voltage and with a low voltage, respectively. The reasoning goes something like this: To assert is to affirm, to state that something is so; what is asserted might be thought to be "true." In propositional logic, a proposition is either true or false. So when something is asserted, this is equivalent to saying that the proposition is true. The dichotomy true/false in the logic of propositions is associated with switching constants 1 and 0. But there is no one-to-one correspondence between 1-T and 0-F; it is simply customary to associate 1 with "true" and 0 with "false."

[15]One might even think of Newton's first law as exemplifying this contention. Normally, things are in stasis; if you want something to change, you have to become active and exert a force.

Anyway, *if* we associate 1 with "true," and *if* assertion means "true," then assertion will be associated with 1:

$$\text{asserted} \Rightarrow \text{true} \Rightarrow \text{logic 1}$$

Hence, saying something is "asserted high" means that 1 corresponds to a high voltage. The use of "asserted" here adds nothing to the identification of 1 with a high voltage; you might as well say "1-high", or 1-H. In the same way, saying that something is "asserted low" means that 1 corresponds to a low voltage. Here the terminology "asserted" adds nothing to the identification of 1 with a low voltage; you might as well say 1-L.

One final comment on terminology. Sometimes the verb "to assert" is used in a way that does not connect it to "high" or "low." It might be said, for example, that a certain signal must be asserted before something else can happen. In such a usage, there is no commitment to asserting high or asserting low—just to asserting. This terminology does not require a commitment to mixed logic or positive logic; it can be applied in either case. Hence, it can be useful terminology.

When physical devices are used to implement a logic diagram, it is possible to use a different correspondence of voltage levels with logic values at different input and output terminals—even at internal points. For the actual implementation phase of a design, it might be useful to consider what is called mixed logic, in which the correspondence 1-H is made at some device terminals and 1-L at others. To achieve this purpose, special conventions and notations are used to convey the information as to which terminals correspond to 1-H and which to 1-L.

In this book, we will stick with positive logic, so we will not deal with this special notation. Indeed, the gate symbols shown in Figures 8–11 are based on positive logic.

9 SOME PRACTICAL MATTERS REGARDING GATES[16]

Up to this point we have adopted the following viewpoint: The specifications of a logic task, followed by the procedures of switching algebra, result in a switching expression. A schematic diagram containing logic gates is then constructed to realize this expression. Ultimately, we expect to construct the physical embodiment of this diagram using real, physical devices, generically referred to as *hardware*.

The way in which physical gates are designed and built depends on the technology of the day. As already mentioned, the first switching circuits utilized

[16]This section is largely descriptive. Although a couple of exercises are slipped in, these do not require a great deal of effort and creativity to complete. What is discussed is highly relevant to laboratory work. If you wish, you can be a largely passive reader; but we urge you to become engaged, to take notes, to formulate questions, and to refer to other books and manufacturers' handbooks for specific items.

mechanical devices: switches and relays.[17] The first switching devices that were called "gates" were designed with vacuum tubes. Vacuum tubes were later replaced by semiconductor diodes and, later still, by bipolar transistors and then MOSFETs. Each gate was individually constructed with discrete components.

The advent of integrated circuits permitted, first, an entire gate to be fabricated as a unit, and then several independent gates. Such *small-scale integrated* (SSI) units still find use. Soon it became possible to incorporate in the same integrated circuit an interconnection of many gates that together perform certain specific operations; we will discuss a number of such *medium-scale integrated* (MSI) circuits in Chapter 4. In time, an entire circuit consisting of hundreds of thousands of gates—such as a microprocessor—came to be fabricated as a single unit, on a single "chip." Some characteristics of integrated circuits will be discussed here. Design using specific circuits of this type will be carried out in Chapter 8.

Logic Families

As mentioned briefly above, the design and implementation of logic gates in any given era is carried out using the particular devices available at the time. Each type of device operates optimally with specific values of power-supply voltage. Different designs can result in different high and low voltage levels. A set of logic gates using a single design technology is referred to as a *logic family*. Some that were considered "advanced" just four decades ago (such as resistor-transistor logic, or the RTL family) are now obsolete but can still be found in museums.

Different designs are developed and promoted by different manufacturers and are suited to different requirements. The ECL (emitter-coupled logic) family came out of Motorola, while Texas Instruments is the creator of the TTL (transistor-transistor logic) family. The basic TTL design has been modified and improved over time in different ways to enhance one property of a gate (say speed) at the expense of some other property (say power consumption). In this way, different subfamilies are created within the TTL family. A table listing a number of the TTL subfamilies is given in Figure 13.

While the TTL and ECL logic families utilize bipolar transistors as the switching element, CMOS (complementary metal-oxide semiconductor) technology utilizes the MOSFET transistor. The subfamilies of CMOS are listed in Figure 14. A major question arises as to whether the inputs and outputs of CMOS gates can be interconnected with TTL gates without any special conversion circuits. If they can, we say that CMOS logic families are TTL-compatible. It turns out that with a power supply between 3.3 and 5 volts, CMOS families are indeed compatible with TTL.

[17]Although early switches were mechanical devices, the most basic modern electronic devices also act as switches—namely, transistors, both the bipolar junction type and, especially, the MOSFET variety. (See the Appendix for descriptions of MOSFETs and BJTs.) Hence, implementations of switching circuits with switches can be brought up to date utilizing MOSFETs. The logical operations of AND and OR would then be accomplished with series connections and parallel connections of such switches. Since most contemporary suppliers have a vested interest in the currently used technology, it is unlikely that a switch will be made. (Pun intended!) Seriously, though, some are returning to the switch circuits pioneered by Shannon, and some books are beginning to reintroduce such circuits.

Designation	Name	Power	Speed
LS	Low-power Schottky	Low	Slow
ALS	Advanced LS	Low	Moderate
S	Schottky	Medium	Fast
AS	Advanced Schottky	Medium	Fast
L	Low-power	Low	Moderate
F	Fast	High	Very fast

Figure 13 Some subfamilies within the TTL family of gates.

Designation	Name	Power	Speed
HC/HCT	High-speed CMOS	Very low	Moderate
AC/ACT	Advanced CMOS	Low	Fast
C	CMOS	Very low	Moderate
LCX/LVX/LVQ	Low-voltage CMOS	Very low	Moderate
VHC/VHCT	Very-high-speed CMOS	Low	Fast
CD4000	CMOS	Low	Moderate

Figure 14 Some subfamilies within the CMOS family of gates.

ECL circuits, on the other hand, are not compatible with either TTL or CMOS. Special conversion circuits are required to connect ECL gates to TTL or CMOS gates. Except for the CD4000 subfamily, the CMOS families are pin-compatible with the TTL families of Figure 13. That is, two chips with the same functionality have the same pin assignments for inputs, outputs, power, and ground.

How does one decide what subfamily to use in a given case? What is needed is a metric that takes into account both speed and power consumption. The product of these two values might be considered, but that won't work since the goal is to increase one and reduce the other. Instead of speed, we use the delay of a signal in traversing a gate; low delay means high speed. So the product *delay-power* is used as a metric. The lower this product, the better.

In this book, we will not study the operation of the electronic devices with which logic gates are constructed. We will, however, use the TTL and CMOS families of gates for illustrative purposes, but only in a descriptive way.

Input/Output Characteristics of Logic Gates[18]

Physical switching circuits are made up of interconnections of physical logic gates in accordance with a switching expression derived to accomplish a particular digital task. An illustration of a circuit is shown in Figure 15. The output from one gate becomes the input to one or more other gates.

Ideally, there would be no interaction among gates; that is, the operation of gate 4, for example, would have no influence on the proper operation of gate 2.

[18]This subsection is not essential to what follows. Even if your previous knowledge does not permit you to assimilate it thoroughly, you should read it anyway, at least to become exposed to the terminology. See Texas Instruments' *Data Book for Design Engineers* for further information.

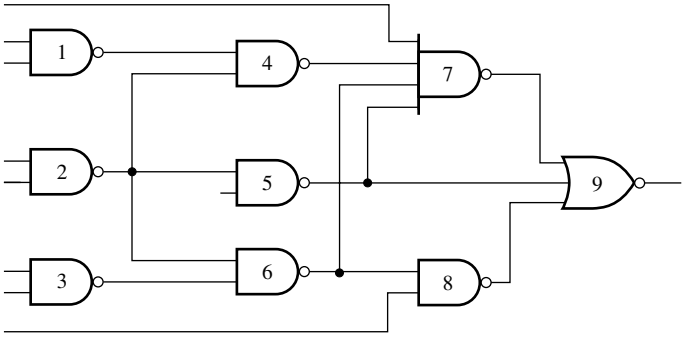

Figure 15 Logic circuit illustrating the loading of gates.

But gate inputs and outputs are connected to electronic devices internal to the gates. When these devices are conducting, currents flow into and out of gate terminals. These currents must necessarily flow through preceding-gate output terminals or succeeding-gate input terminals. Since there are practical limits imposed by the properties of the electronic devices on the level of such currents that can be carried, there are limits on the extent to which gates can be interconnected. Logic designers need not know the details of semiconductor technology and logic-gate implementation, but must understand the input/output characteristics of the technology in use to be able to analyze the interactions between gates.

The interactions between gates are technology specific, so it is necessary to distinguish CMOS and TTL. Let's analyze the input/output characteristics of CMOS first and then TTL. A logic gate output is driven high or low as a function of the input states. From the Appendix we learn that the MOSFET is a voltage-controlled switch. Thus, the inputs to a CMOS logic gate are connected to the gate terminals of MOSFETs. When an input is in one state it closes a MOSFET switch; when in the alternate state it opens a MOSFET switch. The output of a CMOS gate contains one or more MOSFET devices configured to drive the output high for certain input states. If the inputs are such that the output is high, then a set of switches between the output and power supply are closed, connecting the output to the power supply. (The power-supply voltage thus corresponds to logic 1, in positive logic.)

Similarly, a CMOS gate contains one or more MOSFETs to drive the output low when required. These MOSFETs connect the output to ground (corresponding to logic 0) for certain inputs. For our analysis we can assume that one MOSFET is used to pull the output high and one MOSFET is used to pull the output low. The typical input and output MOSFET connections for CMOS technology are shown in Figure 16.

When a MOSFET is switched on, it behaves not as a short circuit, but rather as a source of current. This is an important point. If it did behave as a short circuit, gates would have zero delay and zero power dissipation! We know from the Appendix that the MOSFET has very high input impedance between gate

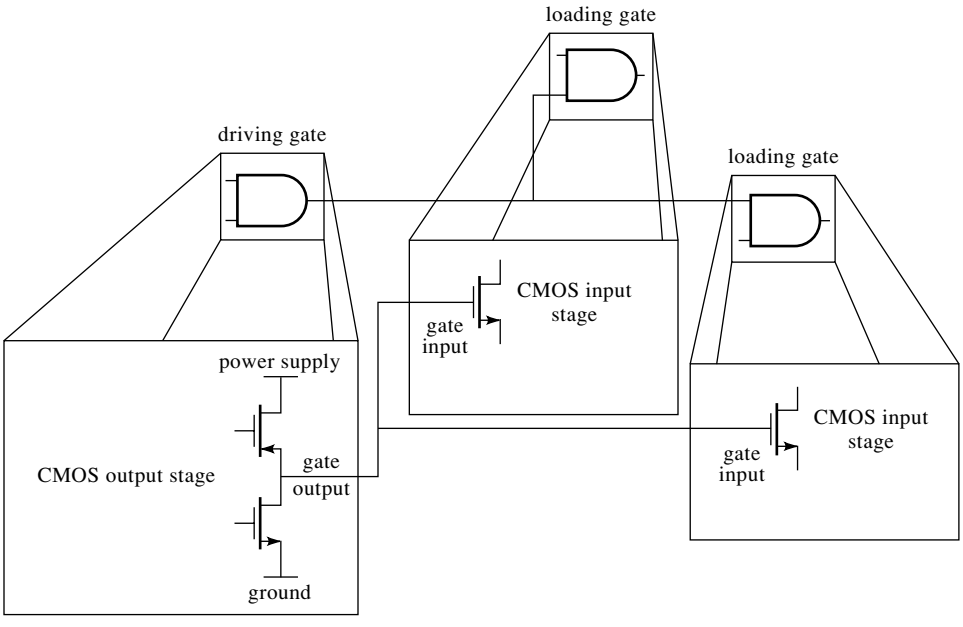

Figure 16 Typical input/output stages of CMOS logic gates.

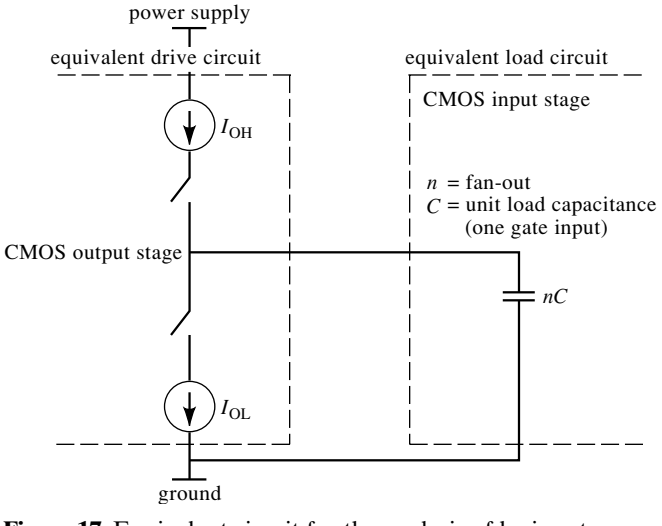

Figure 17 Equivalent circuit for the analysis of logic gate connections in CMOS technology.

and ground. The gate insulator forms the dielectric of a capacitance, and it is this capacitance that creates delay in a gate. A connection from the output of one gate to the inputs of other gates can be modeled as shown in Figure 17.

In a CMOS logic gate the switches shown in Figure 17 are never closed at the same time. Thus, one of the currents flows until the output voltage (which

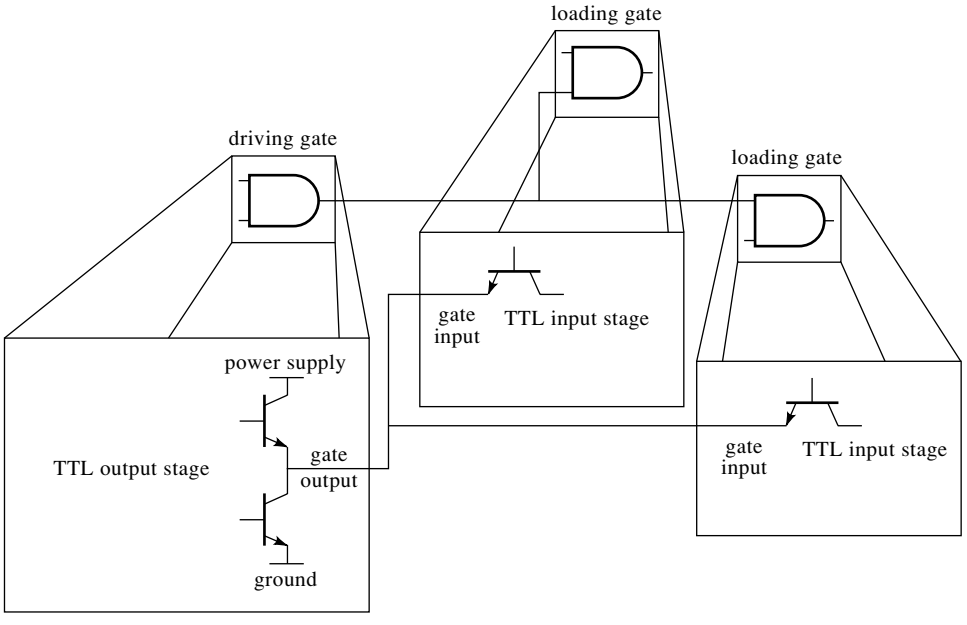

Figure 18 Typical input/output stages of TTL logic gates.

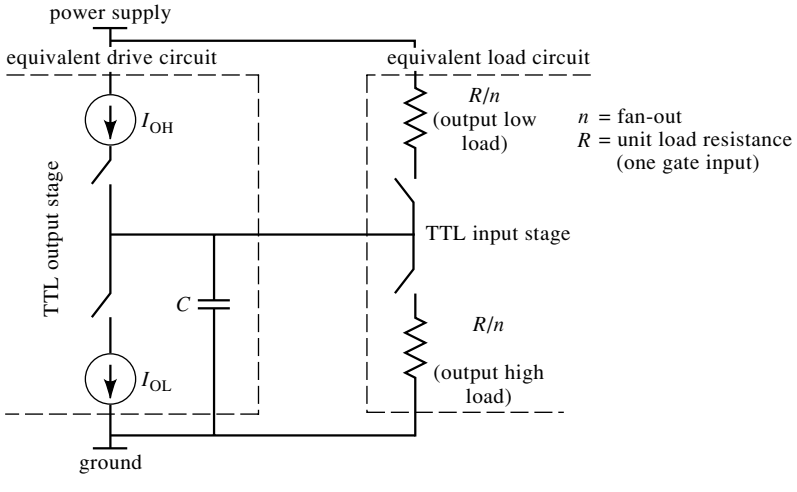

Figure 19 Equivalent circuit for analyzing TTL logic gate connections.

is proportional to the charge on the capacitor) reaches its final value. When the circuit is in steady state, no current is flowing. When the circuit switches state, the capacitance is charged or discharged to the new state. The difference between the input/output interactions of CMOS and TTL is due mainly to the difference in the input stages of the two logic gates. Input/output stages of TTL are shown in Figure 18.

An input of a TTL logic gate is connected to the emitter of a bipolar transistor, so the load seen by a gate output is resistive (as opposed to capacitive for CMOS). The bipolar transistors in the output stage of a TTL gate behave as a current source when switched on, and as an open circuit when switched off. The equivalent circuit for the analysis of gate interaction in TTL is shown in Figure 19.

Fan-out and Fan-in

We can learn some things by making some observations about the structure of Figure 15. The output from gate 1 goes to the input of gate 4 and nowhere else. We say gate 1 *drives* gate 4 and, conversely, that gate 4 *loads* gate 1. The output of gate 2 goes to the inputs of three other gates: gate 2 drives gates 4, 5, and 6, and each of the latter gates loads gate 2. The number of gate inputs driven by, or loading, the output of a single gate is called the *fan-out* of the driving gate. The fan-out of gate 2 in Figure 15, for example, is 3.

In CMOS technology the transistors of a gate output do not have to provide any static current to the inputs of the loading gates. Thus, a CMOS gate can have unlimited fan-out without affecting the logic (voltage level) of the circuit. However, as will be discussed shortly, the fan-out in CMOS technology has a substantial impact on circuit delay.

In TTL circuits a gate output must provide a static current to its fan-out gates. For example, in Figure 15 when the output of gate 2 is low, according to the last paragraph of the preceding subsection, the transistors at the inputs of the driven gates (4, 5, and 6) are conducting. Their currents must all flow through the output transistor of the driving gate (2). This process of returning the current at the input of a gate to ground through the output transistor of the driving gate is called *sinking* the current. For a given family of TTL gates, when the input transistor is conducting, the amount of current is called a *standard load.* With this terminology, gate 2 in Figure 15 must sink three standard loads since its output drives the inputs of three other gates. The proper operation of the output transistor of gate 2 imposes a limit on the number of standard loads that it can sink. Thus, for each TTL circuit design, there is a maximum fan-out (in the mid single digits!), which should not be exceeded.

Continuing the inspection of Figure 15, we notice that most of the gates have two inputs, but gate 9 has three inputs and gate 7 has four inputs. We refer to the number of inputs to a gate as its *fan-in.* Although there is no strong limitation on the fan-in of a gate imposed by its proper operation—as there is on its fan-out—there is, nevertheless, a practical consideration. Gates are manufactured with specific fan-ins, a fixed number of input terminals. If the logic design of a circuit calls for a gate with such a high fan-in that it is not commercially available, the design ought to be changed to utilize gates with available fan-in. There are differences between the fan-in constraints of the CMOS and TTL families, but they are beyond the scope of this book. The fan-in constraint in CMOS is revisited in the subsection on speed and propagation delay.

Buffers

In TTL circuits when the fan-out of a gate is high, the gate will need to sink a lot of current. This may cause the gate to become overloaded and to cease to function properly. In CMOS circuits large fan-out can cause increased delay. To overcome

(*a*) (*b*) **Figure 20** Buffer.

these difficulties, a (noninverting) *buffer* is introduced at the output of the gate; its function is simply to provide increased drive current that permits continued operation with an increased load. The buffer performs no logical operation except the trivial identity operation. That is, its output is the same as its input. The logic symbol for the buffer is shown in Figure 20*a*; it is an inverter without the bubble. It might be made up of two cascaded inverters, as shown in Figure 20*b*. In this way, both a variable and its complement are available to drive a substantial load.

Power Consumption

Whenever there is current in a physical circuit, some of the associated electrical power will be converted to heat. If the circuit is to continue functioning, this heat must be dissipated so as not to result in excessive heating of the devices in the circuit. The design of gates and integrated circuits—including power-supply requirements, the number and type of transistors, and other factors—will have a lot to do with the power consumption. It is possible to design circuits specifically for low power consumption. You must know by now, however, that you can't get something for nothing. Low power consumption can be achieved at a price—usually at the expense of reduced speed. Review the tables for the TTL family in Figure 13 and the CMOS family in Figure 14; notice the range of power dissipation based on the variety of designs.

Noise Margin

Each logic family has specific nominal voltages corresponding to its high level and its low level. (For the TTL family, the nominal high value is 3.3 V and the nominal low value is 0.5 V.) However, digital circuits operate reliably even when actual voltage levels deviate considerably from their nominal values. Moreover, the high output level of a gate is not necessarily the same as its high input level. What is considered a "high" input or output voltage must extend over a range of values and similarly for "low" values.

The inputs and outputs of the gates under discussion are signals that we *intend* them to have. But there are many ways in which unwanted signals can be generated in a circuit and thus change input and output values from what are intended. We refer to such extraneous signals produced in a system as *noise*. Noise can be generated by a variety of mechanisms within the environment in which a circuit operates, from atmospheric radiation to interference from the 60 Hz power system. Noise can also be generated within the electronic devices in the system itself.

When intended signals are accompanied by noise, the actual signals will be modified versions of the desired ones. Provision must be made in the design of the circuit to enable it to continue to operate in the presence of noise up to some anticipated level. That is, there should be a certain immunity to noise. A

measure of the amount of noise that will be tolerated before a circuit malfunctions is the *noise margin, N.* We will not pursue the details of this topic. Suffice it to say that what are considered high voltage levels and low voltage levels in the operation of gates are not fixed values but ranges of values. So long as input and output voltages stay within the range of values specified by the manufacturer, even though contaminated by noise, the gates will operate as if the voltages had their nominal values.

Speed and Propagation Delay

The speed at which logic circuits operate determines how rapidly they complete a task. Limitations on the speed arise from two sources:

- The delay encountered by a signal in going through a single gate
- The number of *levels* in the circuit, that is, the number of gates a signal encounters in going from an input to the output. We call the sequence of gates from an input to an output in a circuit a *logic path.*

Exercise 16 Reexamine Figure 15. Specify the number of levels that each input encounters on its way to the output.
Answer[19]

The delay in a TTL gate stems largely from the fact that transistors within the gate require a nonzero time to switch from a cut off state to a conducting condition and vice versa. This delay, for the most part, is independent of the load seen by a gate. Thus, in TTL circuits it can be assumed that a gate has a fixed delay, and the total delay from a logic input to a logic output can be estimated by accumulating the delays of the gates in the logic path from input to output.

The delay in a CMOS gate comes not only from the time required for transistors to switch between conducting and nonconducting states, but also from the time required to charge and discharge the load capacitance of the fan-out gates. Let's call the delay due to the switching of state of the transistors in a gate the *intrinsic delay* of a gate, and the delay due to load capacitance the *extrinsic delay.* The intrinsic delay is a strong function of the fan-in of a gate; the extrinsic delay is a strong function of its fan-out. Gates with large fan-in have a greater intrinsic delay than gates with small fan-in.

In CMOS it can be advantageous to increase the number of gate levels to keep the fan-in low in order to decrease the delay of a circuit. For example, three realizations of an eight-input AND gate are shown in Figure 21. In CMOS technology (due to the characteristics of the MOSFET) it is likely that the realizations shown in Figures 21*b* and 21*c* have lower delays than the one in Figure 21*a*; this, however, depends on the characteristics of the specific subfamily of CMOS used for the implementation. The extrinsic delay is caused by

[19]Six signals encounter the maximum of four levels, one signal encounters three levels, and two signals encounter one level. ◆

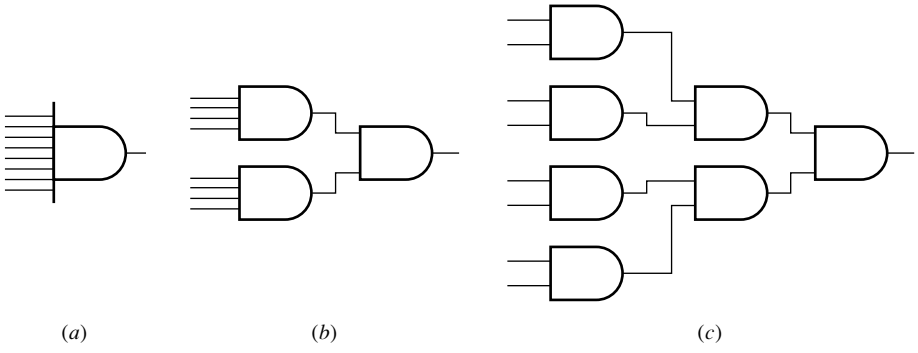

Figure 21 Three alternative realizations of an eight-input AND gate.

the physical constraint imposed by the capacitance: it takes time for the current of the driving gate to charge or discharge the load capacitance to the desired voltage level. The delay in a CMOS gate cannot be estimated accurately by simply counting the number of levels of gates in a logic path. This delay for a given load capacitance can be obtained from manufacturers' data sheets, which contain curves of measured delay as a function of load capacitance.

The transient response shown in Figure 22 illustrates the integrated effect of all the transistors and other components in a gate. The symbols t_{pLH} and t_{pHL} are typically used to denote the low-to-high and high-to-low propagation delay through a gate, respectively. The propagation delay time is the time elapsed between the application of an input signal and the output response. The delay times t_{pLH} and t_{pHL} are not necessarily the same for a specific gate. The delay of a logic path can be determined by the accumulation of t_{pLH} and t_{pHL} for each gate in the path. The symbols t_r and t_f denote, respectively, the rise and fall times of a signal and are defined as the time required for a signal to make a transition from 10 percent to 90 percent of its final value.

10 INTEGRATED CIRCUITS

Historically, both analog and digital circuits, from the simplest device to the most complex, were constructed with discrete components interconnected by conducting wires called *leads*. During the 1960s a method was developed for fabricating all the components that make up a circuit as a single unit on a silicon "chip." The generic name for this process is "integrating" the circuit. Thus, an *integrated circuit* (IC) is a circuit that performs some function; it consists of multiple electrical components (including transistors, resistors, and others), all properly interconnected on a semiconductor chip during the fabrication process.[20]

[20]If you visit a historical museum, among the collections you will find such objects as millenia-old bone knives, flint arrowheads, and hammers formed by a stone lashed with bark to a tree branch. Thousands of years old, these objects are now referred to as *primitive.* In the same way, single gates, buffers, and inverters are described as *primitive devices,* even though they were created only a few decades ago. That will tell you something about the rapidity of change in contemporary times.

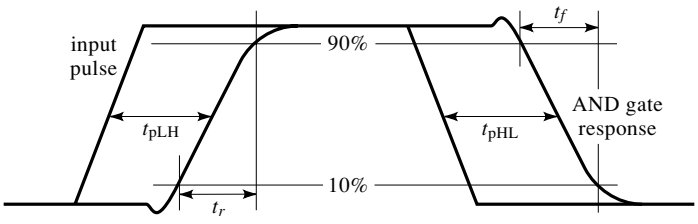

Figure 22 Waveforms of two-input AND gate output in response to a pulse at one input, with the other input held at logic 1.

The complexity of a digital circuit fabricated as an IC ranges from just a few gates to an entire microprocessor on a single chip. The following classifications give a measure of the complexity.

- SSI (*small-scale integration*): ICs with up to a dozen gates per chip, often multiple copies of the same type of gate.
- MSI (*medium-scale integration*): ICs with a few hundred gates per chip, interconnected to perform some specific function.
- LSI (*large-scale integration*): ICs with a few thousand gates per chip.
- VLSI (*very-large-scale integration*): ICs with more than about 10,000 gates per chip.[21]

Each integrated circuit is packaged as a single unit with a number of terminals (pins) available for connecting inputs, outputs, and power supplies. A few typical SSI integrated circuits are illustrated in Figure 23. Consult manufacturers' handbooks for others.

Some Characteristics of ICs

Digital circuits using ICs have a number of highly significant characteristics:

Size and space. The starting point for an integrated circuit is a semiconductor *wafer* about 100 mm in diameter and only 0.15 mm thick. Each wafer is subdivided into many hundreds of rectangular areas called *chips.* A complete circuit is fabricated on each chip. Although each circuit is mounted in a standard-size packet, a considerable size advantage is obtained.

Reliability. Very refined manufacturing and testing techniques are used in the production of ICs. This, together with the fact that all components and their interconnections are fabricated at the same time in the initial process, gives integrated circuits more reliability. For the same reason, since multiple copies of the same type of components (say transistors) are fabricated at the same time, it is much easier to obtain "matched" components, both in terms of parameter values and sensitivity to temperature. Such matched components are often required in a circuit design.

[21]Someone in Britain suggested a further category: VLSII, standing for VLSI, indeed!

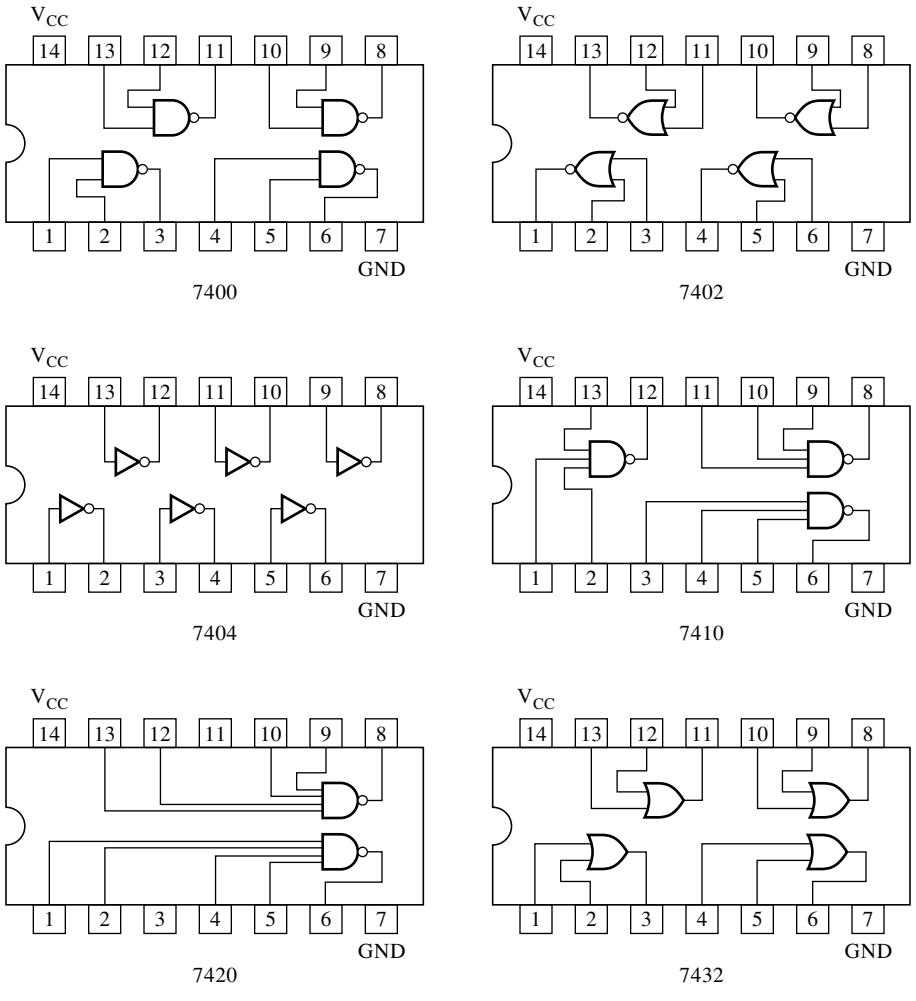

Figure 23 Typical SSI integrated circuits.

Power requirements. The heat generated in electrical circuits must be removed if the circuits are not to overheat. (In early computers, so much heat was generated that as much room for air conditioning equipment was needed to cool the computer as for the huge computer itself.) The problem is greatly reduced if the power consumption of the circuit is low. The much-reduced power requirement of the ICs and their small size thus have a symbiotic relationship. With a lower power demand, circuits can be packed more densely without fear of overheating.

Cost. The process of fabricating a single transistor is not much different from that of fabricating a gate or an SSI integrated circuit; the same chip can be used as the beginning point. The number of details of manufacturing operations to be performed is similar.

The packaging cost for a single transistor is only marginally less than the packaging cost for a more complicated IC. The same considerations apply to

two different ICs with different numbers of gates: the cost of an IC with 30–40 gates is only slightly lower than the cost for one with 80–90 gates. Thus, the cost of an IC is not directly proportional to the number of individual components packed in a package; steps in the manufacturing process cost only marginally more if the number of gates in the package is doubled, for example.

Design Economy

When logic circuits were constructed with discrete ("primitive") gates, costs could be reduced by reducing the number of gates required to perform a particular function. Procedures for finding such minimal circuits became very highly developed. Such minimization procedures still play a role, especially in connection with logic design using what are called *programmable logic devices* (PLDs), as we will see in Chapters 4 and 8. We will therefore devote some attention to them in the following chapter.

The cost of a circuit designed with SSI ICs, however, depends not simply on the gate count but also on the IC package count.[22] Thus, adding a gate or two to a design may cost nothing extra if SSI packages with unused gates are already present in the design. Or suppose that a particular task can be achieved with a 25-gate circuit in two SSI packages, and a 40-gate package (with gates having the appropriate number of inputs) is available; it might be less costly to use the single package in the design and "waste" the unused gates rather than using the two packages with fewer gates.

In some cases another part of the circuit to be designed may call for a few gates of the kind left unused in the design just described. These previously "wasted" gates could now be used for this purpose, with no further hardware cost except for the cost of making pin connections, something that would be required anyway. In any case, design economy no longer depends on gate count but on integrated circuit package count and of the number of connections that have to be made in constructing the circuit.

One way in which to economize on package count, for example, is to avoid using inverters (e.g., 74LS04s). Instead, use an available fraction of a NAND, NOR, or XOR package. Consider the following expressions:

$$A' = (A + A + \cdots + A)' = (A + 0 + \cdots + 0)' \tag{25}$$

$$A' = (A \bullet A \bullet \cdots \bullet A)' = (A \bullet 1 \bullet \cdots \bullet 1)' \tag{26}$$

$$A' = A \oplus 1 \tag{27}$$

From (26) we note that the complement of a variable is the output of a NAND gate with any number of inputs all connected to that variable, or one input connected to that variable and all the rest connected to a high voltage (logic 1, assuming positive logic). Thus, if one-quarter of a 74LS00 (or one-third of a 74LS10) is unused in a package whose remainder already forms part of a circuit, this fraction of the package can be used to implement an inverter at no further hardware cost.

[22] In practice, multiple chips in a design are mounted on a *printed-circuit board* (PCB). So design economy involves not only the IC count but the PCB count as well. Such practical matters are easily learned when one is engaged in practice and need not concern us here.

Exercise 17 In a similar way, interpret the expressions in (25) and (27) for implementing an inverter. ◆

As we shall see in Chapter 4, many combinational-circuit units that perform some rather complex functions have been available for some time in MSI integrated circuits. There is no longer any need to design such functions by combining individual gates in SSI packages.

Application-Specific ICs

The integrated circuits and chips discussed so far have been general purpose. The SSI chips illustrated in Figure 23, for example, the MSI chips in which the applications to be described in Chapter 4 are implemented, or even the LSI chips that embody larger units can be used in the design of a wide variety of applications. A more recent development is the possibility of designing a chip for one particular application. If only a few of the resulting chips will ever be used, then it would make no economic sense to spend the required engineering design resources. On the other hand, if the number of copies needed is large, it may be economically feasible to design an entire integrated circuit just for this particular application. It should be no surprise that this type of IC is called an *application-specific integrated circuit (ASIC).*

Two kinds of costs are associated with an ASIC. Once the unit is designed and debugged, there will be production costs. Of course, the actual design and debugging itself will entail a substantial cost, but this will not depend on later production costs; it is a *nonrecurrent engineering (NRE) cost.* Why undertake an enterprise requiring high NRE costs unless there is some cost or performance advantage? Indeed, ASICs do provide such advantages: generally, the system uses use fewer chips, has smaller physical size, and consequently lower power consumption and smaller time delays, and thus smaller delay-power products. We will not pursue this topic any further.

11 WIRED LOGIC

It is sometimes convenient to perform logic functions using special circuit configurations and connections rather than logic gates (for example, simply by connecting two or more wires together). Two types of gates are described in this section that enable the use of such special configurations and connections. The usefulness of these gates is probably not obvious to you at this point, but it will be when we get to Chapter 4. We consider them here because their implementation depends on the transistor circuits described in the previous section.

Tristate (High-Impedance) Logic Gates

Consider the equivalent circuit for the output stage of a CMOS gate as shown in Figure 16. In a conventional CMOS gate the logic inputs to the gate control the state of the switches (the output transistors). For example, in a two-input NAND gate the switch between the output and ground is closed if both inputs are logic 1; otherwise the switch between the output and the power supply is closed. The

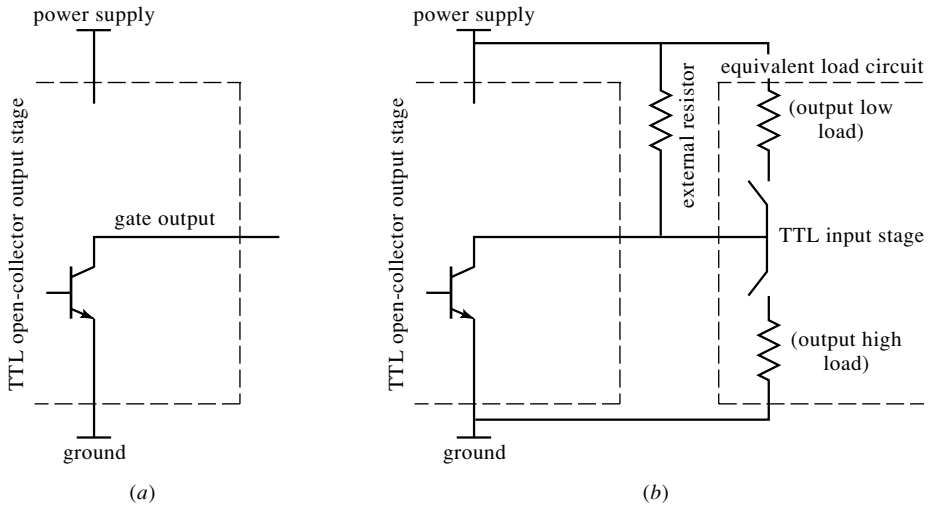

Figure 24 (*a*) The output stage of an open-collector TTL gate. (*b*) The circuit configuration for the open-collector gate.

two switches are never closed at the same time, since this would result in an output voltage level that does not correspond to either a logic 1 or a logic 0. (In a conventional gate the output is always driven to one of the valid logic levels.

Is there any advantage to opening both switches at the same time? If both of the switches are open, then the output is not driven at all; that is, it is floating. In this state the output has a very high impedance, so it is called the *high-impedance* (or high-*Z*) state. A gate can be given the capability of entering this third state by adding an additional input that, independent of the logic inputs, can open both output switches. This additional input is often called an *enable* input. When it is active the gate behaves as a conventional gate, but when it is inactive the gate is in the high-impedance state. Such a gate is called a high-impedance or *tristate* gate. The outputs of two tristate gates can be connected directly to one another if they are never enabled at the same time. The operation of a tristate TTL logic gate is similar.

Exercise 18 Explain the behavior of two conventional logic gates if their outputs are connected directly to one another. (Does it make a difference whether the gates are TTL or CMOS?) ◆

Open-Collector and Open-Drain Logic Gates

Suppose a new gate is constructed from a conventional TTL gate (Figure 18) in which one of the transistors in the output stage is removed. The new gate would have the ability to drive its output to one logic state but not the other. Let's remove the transistor that drives the output high. The new circuit is shown in Figure 24*a*. Is such a circuit useful? Perhaps, but only if there is some way to drive the output high when necessary. This can be accomplished using an external resistor as shown in Figure 24*b*.

When the inputs of the gate are such that the output is driven low, it is driven low through the TTL gate output; the resistor acts as an additional load in parallel with the fan-out gates. The resistance value must not be so small that it overloads the gate. When the inputs are such that the output should be driven high, the ouput is not driven by the TTL gate; the external resistor provides a path for current to drive the fan-out gates instead.

Notice that when the output is driven high, the external resistor is in series with the equivalent input stages of the fan-out gates; thus, its value should be chosen such that an output voltage corresponding to a valid logic level is achieved. This kind of gate is called an *open-collector* gate because the collector of the output transistor is an open circuit without any internal connection on the chip. A similar gate can be constructed in CMOS and is called an *open-drain* gate because, internal to the chip, the drain terminal of the output transistor is not connected.

The usefulness of this circuit is not obvious. Furthermore, we need more components than we started with (an external resistor). However, if we connect the outputs of two or more such gates, we have a circuit that always provides a valid logic level as long as the external resistor has an appropriate value. If one or more of the gates pull the output low, then the external resistor acts as an additional load. If none of the gates pull the output low, then the external resistor provides a conductive path for current and drives the output high. For this reason the connection is called a *wired-AND*. If one or more of the outputs wired together correspond to a logic 0, then, assuming positive logic, the signal does likewise; otherwise, it corresponds to a logic 1.

CHAPTER SUMMARY AND REVIEW

This chapter introduced Boolean algebra and the results that follow from its use. It also introduced the primitive devices, called gates, on which logic design is based. The following topics were included.

- Huntington's postulates
- Boolean algebra
  - Involution
  - Idempotency
  - Absorption
  - Associative law
  - Consensus
- De Morgan's law
- Switching operations: AND, OR, NOT, NAND, NOR, XOR, XNOR
- Switching expressions
  - Sum-of-products form
  - Product-of-sums form
  - Canonic form
- Minterms and maxterms
- Switching functions
- Shannon's expansion theorem: s-of-p and p-of-s
- Universal operations

- Logic gates
- Equivalent forms of NAND and NOR gates
- Positive, negative, and mixed logic
- Families of logic gates: TTL, CMOS
- Gate properties
  - Fan-out
  - Fan-in
  - Power consumption
  - Propagation delay
  - Speed
  - Noise margin
- Integrated circuits: SSI, MSI, LSI, VLSI, ASIC
- Wired logic
  - Tristate gates
  - Open-collector gates
  - Open-drain gates.

PROBLEMS

1 Prove the following theorems of Boolean algebra.

a. The identity elements are distinct; that is, they are not the same.
b. The identity elements are unique; that is, there are no others.
c. The inverse of an element is unique; that is, there aren't two different inverses.
d. The complement of an element is unique. From this, prove $(x')' = x$.

2 Use the distributive law and any other laws of switching algebra to place each of the following expressions in simplest sum-of-products form. (These are useful results to remember.)

a. $f_1 = (A + B)(A + C)$
b. $f_2 = (A + B)(A' + C)$

3 Complete the details of the proof of the associative law (Theorem 6) outlined in the text.
4 Prove the consensus theorem by *perfect induction,* that is, by showing it to be true for all possible values of the variables.
5 Prove that a Boolean algebra of three distinct elements, say {0, 1, 2}, does not exist. (If all of Huntington's postulates were satisfied, then it *would* exist.)
6 The following set of four elements is under consideration as the elements of a Boolean algebra: {0, 1, *a, b*}. The identity elements that satisfy Postulate 2 are to be 0 and 1. Fill in the tables for (+) and (•) in Figure P6 such that the algebraic system defined by these tables will be a Boolean algebra. Find the complement of each element, and confirm that all of Huntington's postulates are satisfied.
7 The operations of a four-element algebraic system are given in Figure P7. Determine whether this system is a Boolean algebra. If so, which are the identity elements?
8 In a natural-food restaurant, fruit is offered for dessert but only in certain combinations. One choice is either peaches or apples or both. Another choice is either cherries and apples or neither. A third choice is peaches, but if you choose peaches, then you must also take bananas. Define Boolean variables for all of the fruits and write a logical expression that specifies the fruit available for dessert. Then simplify the expression.
9 Write a logical expression that represents the following proposition: The collector current in a bipolar transistor is proportional to the base-emitter voltage v_{BE}, provided the transistor is neither saturated nor cut off.

(+)	0	a	b	1
0				
a				
b				
1				

(a)

(•)	0	a	b	1
0				
a				
b				
1				

(b)

Figure P6

x	x'
a	c
b	d
c	a
d	b

(a)

(+)	a	b	c	d
a	a	a	a	a
b	a	b	b	a
c	a	b	c	d
d	a	a	d	d

(b)

(•)	a	b	c	d
a	a	b	c	d
b	b	b	c	c
c	c	c	c	c
d	d	c	c	d

(c)

Figure P7

10 Construct a truth table for each function represented by the following expressions.

a. $E = (A' + B)(A' + B' + C)$
b. $E = (((A' + B)' + C)' + A)'$
c. $E = xz' + yz + xy'$

11 Carry out the proof of Shannon's expansion theorem given as (6) in the text.

12 Use one or both of the distributive laws (repeatedly if necessary) to place each of the expressions below in product-of-sums form.

a. $f_1 = x + wyz'$
b. $f_2 = AC + B'D'$
c. $f_3 = xy' + x'y$
d. $f_4 = xy' + wuv + xz$
e. $f_5 = BC' + AD'E$
f. $f_6 = ABC' + ACD' + AB'D + BC'D$

13 Construct a truth table for each function represented by the following expressions.

a. $E = xy + (x + z')(x + y' + z') + xy'z$
b. $E = (w + x'z' + y)(y' + z') + wxy'z$
c. $E = (AB'C + BC'D) + AB'D + BCD'$

14 Apply De Morgan's law (repeatedly if necessary, but without any simplifying manipulations) to find the complement of each of the following expressions.

a. $f = AB(C + D') + A'C'(BD' + B'D)$
b. $f = [AC' + (B' + D)(A + C')][BC + A'D(CE' + A)]$

15 Verify the following expression using the rules of Boolean algebra.

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

16 Using the rules of Boolean algebra, simplify the expressions that follow to the fewest total number of literals.

a. $f = AB' + ABC + AC'D$
b. $f = wyz + xy + xz' + yz$
c. $f = B + AD + BC + [B + A(C + D)]'$

17 Use switching algebra to simplify the following expressions as much as possible.

- **a.** $xyz' + xy'z' + x'y$
- **b.** $(wx')'(w + y)(x'y'z')'$
- **c.** $x'(y + wy'z') + x'y'(w'z' + z)$
- **d.** $(w + x)(w' + x + yz')(w + y')$

18
- **a.** Carry out appropriate switching operations on the following expressions to arrive at noncanonic sum-of-products forms. Then restore missing variables to convert them to canonic form.
- **b.** In each case, using the distributive law and other Boolean laws, convert the noncanonic sum-of-products form to product-of-sums form.
- **c.** In each case, restore the missing variables to the forms determined in part *a* to convert them to canonic form.
- **d.** In each case, construct the truth table and confirm the canonic sum-of-products form.
- **e.** In each case, by applying De Morgan's law to the complements of the minterms absent from the canonic sum-of-products form, find the canonic product-of-sums form.

$$E_1 = (x + y')(y + z')$$
$$E_2 = (x'y + x'z)'$$
$$E_3 = (x + yz')(y + xz')$$
$$E_4 = (B + D')(A + D')(B' + D')$$
$$E_5 = (AB)'(B + C'D)(A + D')$$

19 Use switching algebra to prove the following relationships.

- **a.** $x' \oplus y = x \oplus y'$
- **b.** $x' \oplus y' = x \oplus y$
- **c.** $x'y \oplus xy' = x \oplus y$

20 Use appropriate laws of switching algebra to convert the expression $(x \oplus y)(x \oplus z)$ to the form $A \oplus B$; specify A and B in terms of x, y, z.

21
- **a.** Express the Exclusive-OR function, $x \oplus y$, in terms of only NAND functions.
- **b.** Express $x \oplus y$ in terms of only NOR functions.

22 Prove that the Exclusive-OR operation is associative. That is, $(x \oplus y) \oplus z = x \oplus (y \oplus z)$.

23 Show that {NOR} is a universal set of operations.

24 The *implication* function $f = (x \Rightarrow y)$ is the statement "If x is true, then y is true." A switching expression for the implication function is $f = x' + y$. This function can be implemented by a two-input OR gate in which one of the input lines has a bubble. It might be called an *implication* gate.

- **a.** Construct a truth table for the implication function. Are any of the rows puzzling?
- **b.** Show that the implication function constitutes a universal set; that is, any switching expression can be represented in terms of implication functions only.
- **c.** Write the following expression in terms of implication functions only: $f = AC + BC'$.

25 Each of the following expressions includes all three of the Boolean operations. Using appropriate Boolean theorems, convert each expression to one that includes

- **a.** Only AND and NOT operations
- **b.** Only OR and NOT operations

c. Only NAND operations
d. Only NOR operations

$$E_1 = AB' + AC + B'C$$
$$E_2 = (x' + y')(y' + z)(y + z')$$
$$E_3 = xy' + (y + z')(x + z)$$

26 **a.** Use the result of Figure 2 in the text to convert a two-level AND-OR circuit consisting of two double-input AND gates followed by an OR gate into an all-NAND circuit.
b. Use the result of Exercise 15 in the text to convert a two-level OR-AND circuit consisting of two double-input OR gates followed by an AND gate into an all-NOR circuit.

27 Show that the OR operation in the first form of Shannon's theorem can be replaced by the Exclusive-OR.
28 Determine the relationship between the Exclusive-OR function and its dual.
29 The objective of this problem is to convert one kind of two-input gate to a different kind of gate but with inverted inputs. Let the inputs be A and B. Use an appropriate Boolean theorem to carry out this objective for the following gate types:

AND: AB
OR: $A + B$
NAND: $(AB)'$
NOR: $(A + B)'$

30 An OR gate has two inputs A and B. B is obtained from A after A goes through three consecutive inverters. Assume that each inverter has a 1 μs delay. Suppose that A has been 1 for some time and then drops to 0. Draw the waveform of the resulting output.
31 A standard load for an LS-TTL gate is 3 mA of current; such a gate can drive three standard loads of the same kind of gate. A standard load for an S-TTL gate is 7mA. How many S-TTL gates can be driven by an LS-TTL gate?
32 Assuming the logic-high and logic-low voltage levels of two CMOS and TTL subfamilies are compatible, what are the consequences of driving a TTL gate with a CMOS gate and vice versa?
33 The intrinsic delay of a gate is 0.5 ns and the extrinsic delay is 0.2 ns per load. What is the fan-out limitation of this gate, given that its delay cannot exceed 1.2 ns?
34 Can the outputs of conventional gates be connected to the outputs of open-collector gates? If so, show such a circuit and describe its operation. If not, explain why they cannot be connected together.
35 What is the consequence of connecting many tristate outputs together? How many tristate outputs can be connected without affecting the proper functionality of a circuit?
36 What is the consequence of connecting many open-collector outputs together? How many open-collector outputs can be connected without affecting the proper functionality of a circuit?
37 Can the outputs of open-collector and tristate gates be connected together? Explain why or why not.