FUNDAMENTALS OF DIGITAL LOGIC WITH VERILOG DESIGN

SECOND EDITION

Stephen Brown and Zvonko Vranesic
Department of Electrical and Computer Engineering
University of Toronto

McGraw-Hill Higher Education

Boston Burr Ridge, IL Dubuque, IA New York San Francisco St. Louis
Bangkok Bogotá Caracas Kuala Lumpur Lisbon London Madrid Mexico City
Milan Montreal New Delhi Santiago Seoul Singapore Sydney Taipei Toronto

The McGraw-Hill Companies

McGraw-Hill Higher Education

FUNDAMENTALS OF DIGITAL LOGIC WITH VERILOG DESIGN, SECOND EDITION

This book is printed on acid-free paper.

1 2 3 4 5 6 7 8 9 0 DOC/DOC 0 9 8 7

ISBN 978–0–07–338033–9
MHID 0–07–338033–4

Global Publisher: *Raghothaman Srinivasan*
Executive Editor: *Michael Hackett*
Director of Development: *Kristine M. Tibbetts*
Developmental Editor: *Darlene M. Schueller*
Executive Marketing Manager: *Michael Weitz*
Project Manager: *April R. Southwood*
Senior Production Supervisor: *Kara Kudronowicz*
Associate Media Producer: *Christina Nelson*
Lead Media Project Manager: *Judi David*
Associate Design Coordinator: *Brenda A. Rolwes*
Cover Designer: *Studio Montage, St. Louis, Missouri*
(USE) Cover Image: *Royalty-Free/CORBIS*
Senior Photo Research Coordinator: *Lori Hancock*
Compositor: *Techsetters, Inc.*
Typeface: *10/12 Times Roman*
Printer: *R. R. Donnelley Crawfordsville, IN*

Library of Congress Cataloging-in-Publication Data

Brown, Stephen D.
Fundamentals of digital logic with Verilog design / Stephen Brown, Zvonko Vranesic. — 2nd ed.
p. cm.
ISBN 978–0–07–338033–9 — ISBN 0–07–338033–4 (hard copy : alk. paper)
1. Logic circuits—Design and construction—Data processing. 2. Verilog (Computer hardware description language). 3. Computer-aided design. I. Vranesic, Zvonko G. II. Title.

TK7868.L6B76 2008
621.39'2–dc22

2007008622

www.mhhe.com

chapter

2

INTRODUCTION TO LOGIC CIRCUITS

CHAPTER OBJECTIVES

In this chapter you will be introduced to:

- Logic functions and circuits
- Boolean algebra for dealing with logic functions
- Logic gates and synthesis of simple circuits
- CAD tools and the Verilog hardware description language

The study of logic circuits is motivated mostly by their use in digital computers. But such circuits also form the foundation of many other digital systems where performing arithmetic operations on numbers is not of primary interest. For example, in a myriad of control applications actions are determined by some simple logical operations on input information, without having to do extensive numerical computations.

Logic circuits perform operations on digital signals and are usually implemented as electronic circuits where the signal values are restricted to a few discrete values. In *binary* logic circuits there are only two values, 0 and 1. In *decimal* logic circuits there are 10 values, from 0 to 9. Since each signal value is naturally represented by a digit, such logic circuits are referred to as *digital circuits*. In contrast, there exist *analog circuits* where the signals may take on a continuous range of values between some minimum and maximum levels.

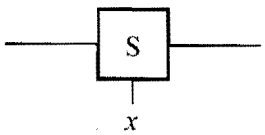
In this book we deal with binary circuits, which have the dominant role in digital technology. We hope to provide the reader with an understanding of how these circuits work, how are they represented in mathematical notation, and how are they designed using modern design automation techniques. We begin by introducing some basic concepts pertinent to the binary logic circuits.

2.1 VARIABLES AND FUNCTIONS

The dominance of binary circuits in digital systems is a consequence of their simplicity, which results from constraining the signals to assume only two possible values. The simplest binary element is a switch that has two states. If a given switch is controlled by an input variable x, then we will say that the switch is open if $x = 0$ and closed if $x = 1$, as illustrated in Figure 2.1*a*. We will use the graphical symbol in Figure 2.1*b* to represent such switches in the diagrams that follow. Note that the control input x is shown explicitly in the symbol. In Chapter 3 we will explain how such switches are implemented with transistors.

Consider a simple application of a switch, where the switch turns a small lightbulb on or off. This action is accomplished with the circuit in Figure 2.2*a*. A battery provides the power source. The lightbulb glows when sufficient current passes through its filament, which is an electrical resistance. The current flows when the switch is closed, that is, when

(a) Two states of a switch



(b) Symbol for a switch

Figure 2.1 A binary switch.

are needed in expressions that are typed using a computer keyboard, which is often done when using CAD tools, it is impractical to use overbars. Instead, either an apostrophe is placed after the variable, or the exclamation mark (!) or the tilde character (~) is placed in front of the variable to denote the complementation. Thus the following are equivalent:

$$\bar{x} = x' = !x = \sim x$$

The complement operation can be applied to a single variable or to more complex operations. For example, if

$$f(x_1, x_2) = x_1 + x_2$$

then the complement of f is

$$\bar{f}(x_1, x_2) = \overline{x_1 + x_2}$$

This expression yields the logic value 1 only when neither x_1 nor x_2 is equal to 1, that is, when $x_1 = x_2 = 0$. Again, the following notations are equivalent:

$$\overline{x_1 + x_2} = (x_1 + x_2)' = !(x_1 + x_2) = \sim(x_1 + x_2)$$

2.3 TRUTH TABLES

We have introduced the three most basic logic operations—AND, OR, and complement—by relating them to simple circuits built with switches. This approach gives these operations a certain "physical meaning." The same operations can also be defined in the form of a table, called a *truth table*, as shown in Figure 2.6. The first two columns (to the left of the heavy vertical line) give all four possible combinations of logic values that the variables x_1 and x_2 can have. The next column defines the AND operation for each combination of values of x_1 and x_2, and the last column defines the OR operation. Because we will frequently need to refer to "combinations of logic values" applied to some variables, we will adopt a shorter term, *valuation*, to denote such a combination of logic values.

x_1	x_2	$x_1 \cdot x_2$	$x_1 + x_2$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1
		AND	OR

Figure 2.6 A truth table for the AND and OR operations.

x_1	x_2	x_3	$x_1 \cdot x_2 \cdot x_3$	$x_1 + x_2 + x_3$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 2.7 Three-input AND and OR operations.

The truth table is a useful aid for depicting information involving logic functions. We will use it in this book to define specific functions and to show the validity of certain functional relations. Small truth tables are easy to deal with. However, they grow exponentially in size with the number of variables. A truth table for three input variables has eight rows because there are eight possible valuations of these variables. Such a table is given in Figure 2.7, which defines three-input AND and OR functions. For four input variables the truth table has 16 rows, and so on. In general, for n input variables the truth table has 2^n rows.

The AND and OR operations can be extended to n variables. An AND function of variables $x_1, x_2, \ldots, x_n$ has the value 1 only if all n variables are equal to 1. An OR function of variables $x_1, x_2, \ldots, x_n$ has the value 1 if at least one, or more, of the variables is equal to 1.

2.4 LOGIC GATES AND NETWORKS

The three basic logic operations introduced in the previous sections can be used to implement logic functions of any complexity. A complex function may require many of these basic operations for its implementation. Each logic operation can be implemented electronically with transistors, resulting in a circuit element called a *logic gate*. A logic gate has one or more inputs and one output that is a function of its inputs. It is often convenient to describe a logic circuit by drawing a circuit diagram, or *schematic*, consisting of graphical symbols representing the logic gates. The graphical symbols for the AND, OR, and NOT gates are shown in Figure 2.8. The figure indicates on the left side how the AND and OR gates are drawn when there are only a few inputs. On the right side it shows how the symbols are augmented to accommodate a greater number of inputs. We will show how logic gates are built using transistors in Chapter 3.

A larger circuit is implemented by a *network* of gates. For example, the logic function from Figure 2.4 can be implemented by the network in Figure 2.9. The complexity of a

Functionally Equivalent Networks

Now consider the network in Figure 2.10*d*. Going through the same analysis procedure, we find that the output g changes in exactly the same way as f does in part (*a*) of the figure. Therefore, $g(x_1, x_2) = f(x_1, x_2)$, which indicates that the two networks are functionally equivalent; the output behavior of both networks is represented by the truth table in Figure 2.10*b*. Since both networks realize the same function, it makes sense to use the simpler one, which is less costly to implement.

In general, a logic function can be implemented with a variety of different networks, probably having different costs. This raises an important question: How does one find the best implementation for a given function? Many techniques exist for synthesizing logic functions. We will discuss the main approaches in Chapter 4. For now, we should note that some manipulation is needed to transform the more complex network in Figure 2.10*a* into the network in Figure 2.10*d*. Since $f(x_1, x_2) = \bar{x}_1 + x_1 \cdot x_2$ and $g(x_1, x_2) = \bar{x}_1 + x_2$, there must exist some rules that can be used to show the equivalence

$$\bar{x}_1 + x_1 \cdot x_2 = \bar{x}_1 + x_2$$

We have already established this equivalence through detailed analysis of the two circuits and construction of the truth table. But the same outcome can be achieved through algebraic manipulation of logic expressions. In the next section we will discuss a mathematical approach for dealing with logic functions, which provides the basis for modern design techniques.

2.5 BOOLEAN ALGEBRA

In 1849 George Boole published a scheme for the algebraic description of processes involved in logical thought and reasoning [1]. Subsequently, this scheme and its further refinements became known as *Boolean algebra*. It was almost 100 years later that this algebra found application in the engineering sense. In the late 1930s Claude Shannon showed that Boolean algebra provides an effective means of describing circuits built with switches [2]. The algebra can, therefore, be used to describe logic circuits. We will show that this algebra is a powerful tool that can be used for designing and analyzing logic circuits. The reader will come to appreciate that it provides the foundation for much of our modern digital technology.

Axioms of Boolean Algebra

Like any algebra, Boolean algebra is based on a set of rules that are derived from a small number of basic assumptions. These assumptions are called *axioms*. Let us assume that Boolean algebra B involves elements that take on one of two values, 0 and 1. Assume that the following axioms are true:

1*a*. $0 \cdot 0 = 0$

1*b*. $1 + 1 = 1$

2*a*. $1 \cdot 1 = 1$

2*b*. $0 + 0 = 0$

3*a*. $0 \cdot 1 = 1 \cdot 0 = 0$

3*b*. $1 + 0 = 0 + 1 = 1$

4*a*. If $x = 0$, then $\bar{x} = 1$

4*b*. If $x = 1$, then $\bar{x} = 0$

Single-Variable Theorems

From the axioms we can define some rules for dealing with single variables. These rules are often called *theorems*. If x is a variable in B, then the following theorems hold:

5*a*. $x \cdot 0 = 0$

5*b*. $x + 1 = 1$

6*a*. $x \cdot 1 = x$

6*b*. $x + 0 = x$

7*a*. $x \cdot x = x$

7*b*. $x + x = x$

8*a*. $x \cdot \bar{x} = 0$

8*b*. $x + \bar{x} = 1$

9. $\bar{\bar{x}} = x$

It is easy to prove the validity of these theorems by perfect induction, that is, by substituting the values $x = 0$ and $x = 1$ into the expressions and using the axioms given above. For example, in theorem 5*a*, if $x = 0$, then the theorem states that $0 \cdot 0 = 0$, which is true according to axiom 1*a*. Similarly, if $x = 1$, then theorem 5*a* states that $1 \cdot 0 = 0$, which is also true according to axiom 3*a*. The reader should verify that theorems 5*a* to 9 can be proven in this way.

Duality

Notice that we have listed the axioms and the single-variable theorems in pairs. This is done to reflect the important *principle of duality*. Given a logic expression, its *dual* is obtained by replacing all + operators with · operators, and vice versa, and by replacing all 0s with 1s, and vice versa. The dual of any true statement (axiom or theorem) in Boolean algebra is also a true statement. At this point in the discussion, the reader will not appreciate why duality is a useful concept. However, this concept will become clear later in the chapter, when we will show that duality implies that at least two different ways exist to express every logic function with Boolean algebra. Often, one expression leads to a simpler physical implementation than the other and is thus preferable.

Two- and Three-Variable Properties

To enable us to deal with a number of variables, it is useful to define some two- and three-variable algebraic identities. For each identity, its dual version is also given. These identities are often referred to as *properties*. They are known by the names indicated below. If x, y, and z are the variables in B, then the following properties hold:

10*a*. $x \cdot y = y \cdot x$ *Commutative*

10*b*. $x + y = y + x$

11*a*. $x \cdot (y \cdot z) = (x \cdot y) \cdot z$ *Associative*

11*b*. $x + (y + z) = (x + y) + z$

x	y	$x \cdot y$	$\overline{x \cdot y}$	$\overline{x}$	$\overline{y}$	$\overline{x} + \overline{y}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0
		LHS		RHS		

Figure 2.11 Proof of DeMorgan's theorem in 15*a*.

12*a*.	$x \cdot (y + z) = x \cdot y + x \cdot z$	*Distributive*
12*b*.	$x + y \cdot z = (x + y) \cdot (x + z)$	
13*a*.	$x + x \cdot y = x$	*Absorption*
13*b*.	$x \cdot (x + y) = x$	
14*a*.	$x \cdot y + x \cdot \overline{y} = x$	*Combining*
14*b*.	$(x + y) \cdot (x + \overline{y}) = x$	
15*a*.	$\overline{x \cdot y} = \overline{x} + \overline{y}$	*DeMorgan's theorem*
15*b*.	$\overline{x + y} = \overline{x} \cdot \overline{y}$	
16*a*.	$x + \overline{x} \cdot y = x + y$	
16*b*.	$x \cdot (\overline{x} + y) = x \cdot y$	
17*a*.	$x \cdot y + y \cdot z + \overline{x} \cdot z = x \cdot y + \overline{x} \cdot z$	*Consensus*
17*b*.	$(x + y) \cdot (y + z) \cdot (\overline{x} + z) = (x + y) \cdot (\overline{x} + z)$	

Again, we can prove the validity of these properties either by perfect induction or by performing algebraic manipulation. Figure 2.11 illustrates how perfect induction can be used to prove DeMorgan's theorem, using the format of a truth table. The evaluation of left-hand and right-hand sides of the identity in 15*a* gives the same result.

We have listed a number of axioms, theorems, and properties. Not all of these are necessary to define Boolean algebra. For example, assuming that the + and · operations are defined, it is sufficient to include theorems 5 and 8 and properties 10 and 12. These are sometimes referred to as Huntington's basic postulates [3]. The other identities can be derived from these postulates.

The preceding axioms, theorems, and properties provide the information necessary for performing algebraic manipulation of more complex expressions.

Example 2.1

Let us prove the validity of the logic equation

$$(x_1 + x_3) \cdot (\overline{x}_1 + \overline{x}_3) = x_1 \cdot \overline{x}_3 + \overline{x}_1 \cdot x_3$$

The left-hand side can be manipulated as follows. Using the distributive property, 12*a*, gives

$$\text{LHS} = (x_1 + x_3) \cdot \overline{x}_1 + (x_1 + x_3) \cdot \overline{x}_3$$

Applying the distributive property again yields

$$\text{LHS} = x_1 \cdot \bar{x}_1 + x_3 \cdot \bar{x}_1 + x_1 \cdot \bar{x}_3 + x_3 \cdot \bar{x}_3$$

Note that the distributive property allows ANDing the terms in parenthesis in a way analogous to multiplication in ordinary algebra. Next, according to theorem 8*a*, the terms $x_1 \cdot \bar{x}_1$ and $x_3 \cdot \bar{x}_3$ are both equal to 0. Therefore,

$$\text{LHS} = 0 + x_3 \cdot \bar{x}_1 + x_1 \cdot \bar{x}_3 + 0$$

From 6*b* it follows that

$$\text{LHS} = x_3 \cdot \bar{x}_1 + x_1 \cdot \bar{x}_3$$

Finally, using the commutative property, 10*a* and 10*b*, this becomes

$$\text{LHS} = x_1 \cdot \bar{x}_3 + \bar{x}_1 \cdot x_3$$

which is the same as the right-hand side of the initial equation.

Example 2.2 Consider the logic equation

$$x_1 \cdot \bar{x}_3 + \bar{x}_2 \cdot \bar{x}_3 + x_1 \cdot x_3 + \bar{x}_2 \cdot x_3 = \bar{x}_1 \cdot \bar{x}_2 + x_1 \cdot x_2 + x_1 \cdot \bar{x}_2$$

The left-hand side can be manipulated as follows

$$\begin{aligned}
\text{LHS} &= x_1 \cdot \bar{x}_3 + x_1 \cdot x_3 + \bar{x}_2 \cdot \bar{x}_3 + \bar{x}_2 \cdot x_3 && \text{using } 10b \\
&= x_1 \cdot (\bar{x}_3 + x_3) + \bar{x}_2 \cdot (\bar{x}_3 + x_3) && \text{using } 12a \\
&= x_1 \cdot 1 + \bar{x}_2 \cdot 1 && \text{using } 8b \\
&= x_1 + \bar{x}_2 && \text{using } 6a
\end{aligned}$$

The right-hand side can be manipulated as

$$\begin{aligned}
\text{RHS} &= \bar{x}_1 \cdot \bar{x}_2 + x_1 \cdot (x_2 + \bar{x}_2) && \text{using } 12a \\
&= \bar{x}_1 \cdot \bar{x}_2 + x_1 \cdot 1 && \text{using } 8b \\
&= \bar{x}_1 \cdot \bar{x}_2 + x_1 && \text{using } 6a \\
&= x_1 + \bar{x}_1 \cdot \bar{x}_2 && \text{using } 10b \\
&= x_1 + \bar{x}_2 && \text{using } 16a
\end{aligned}$$

Being able to manipulate both sides of the initial equation into identical expressions establishes the validity of the equation. Note that the same logic function is represented by either the left- or the right-hand side of the above equation; namely

$$\begin{aligned}
f(x_1, x_2, x_3) &= x_1 \cdot \bar{x}_3 + \bar{x}_2 \cdot \bar{x}_3 + x_1 \cdot x_3 + \bar{x}_2 \cdot x_3 \\
&= \bar{x}_1 \cdot \bar{x}_2 + x_1 \cdot x_2 + x_1 \cdot \bar{x}_2
\end{aligned}$$

As a result of manipulation, we have found a much simpler expression

$$f(x_1, x_2, x_3) = x_1 + \bar{x}_2$$

which also represents the same function. This simpler expression would result in a lower-cost logic circuit that could be used to implement the function.

Examples 2.1 and 2.2 illustrate the purpose of the axioms, theorems, and properties as a mechanism for algebraic manipulation. Even these simple examples suggest that it is impractical to deal with highly complex expressions in this way. However, these theorems and properties provide the basis for automating the synthesis of logic functions in CAD tools. To understand what can be achieved using these tools, the designer needs to be aware of the fundamental concepts.

2.5.1 THE VENN DIAGRAM

We have suggested that perfect induction can be used to verify the theorems and properties. This procedure is quite tedious and not very informative from the conceptual point of view. A simple visual aid that can be used for this purpose also exists. It is called the Venn diagram, and the reader is likely to find that it provides for a more intuitive understanding of how two expressions may be equivalent.

The Venn diagram has traditionally been used in mathematics to provide a graphical illustration of various operations and relations in the algebra of sets. A set s is a collection of elements that are said to be the members of s. In the Venn diagram the elements of a set are represented by the area enclosed by a contour such as a square, a circle, or an ellipse. For example, in a universe N of integers from 1 to 10, the set of even numbers is $E = \{2, 4, 6, 8, 10\}$. A contour representing E encloses the even numbers. The odd numbers form the complement of E; hence the area outside the contour represents $\overline{E} = \{1, 3, 5, 7, 9\}$.

Since in Boolean algebra there are only two values (elements) in the universe, $B = \{0, 1\}$, we will say that the area within a contour corresponding to a set s denotes that $s = 1$, while the area outside the contour denotes $s = 0$. In the diagram we will shade the area where $s = 1$. The concept of the Venn diagram is illustrated in Figure 2.12. The universe B is represented by a square. Then the constants 1 and 0 are represented as shown in parts (a) and (b) of the figure. A variable, say, x, is represented by a circle, such that the area inside the circle corresponds to $x = 1$, while the area outside the circle corresponds to $x = 0$. This is illustrated in part (c). An expression involving one or more variables is depicted by shading the area where the value of the expression is equal to 1. Part (d) indicates how the complement of x is represented.

To represent two variables, x and y, we draw two overlapping circles. Then the area where the circles overlap represents the case where $x = y = 1$, namely, the AND of x and y, as shown in part (e). Since this common area consists of the intersecting portions of x and y, the AND operation is often referred to formally as the *intersection* of x and y. Part (f) illustrates the OR operation, where $x + y$ represents the total area within both circles, namely, where at least one of x or y is equal to 1. Since this combines the areas in the circles, the OR operation is formally often called the *union* of x and y.

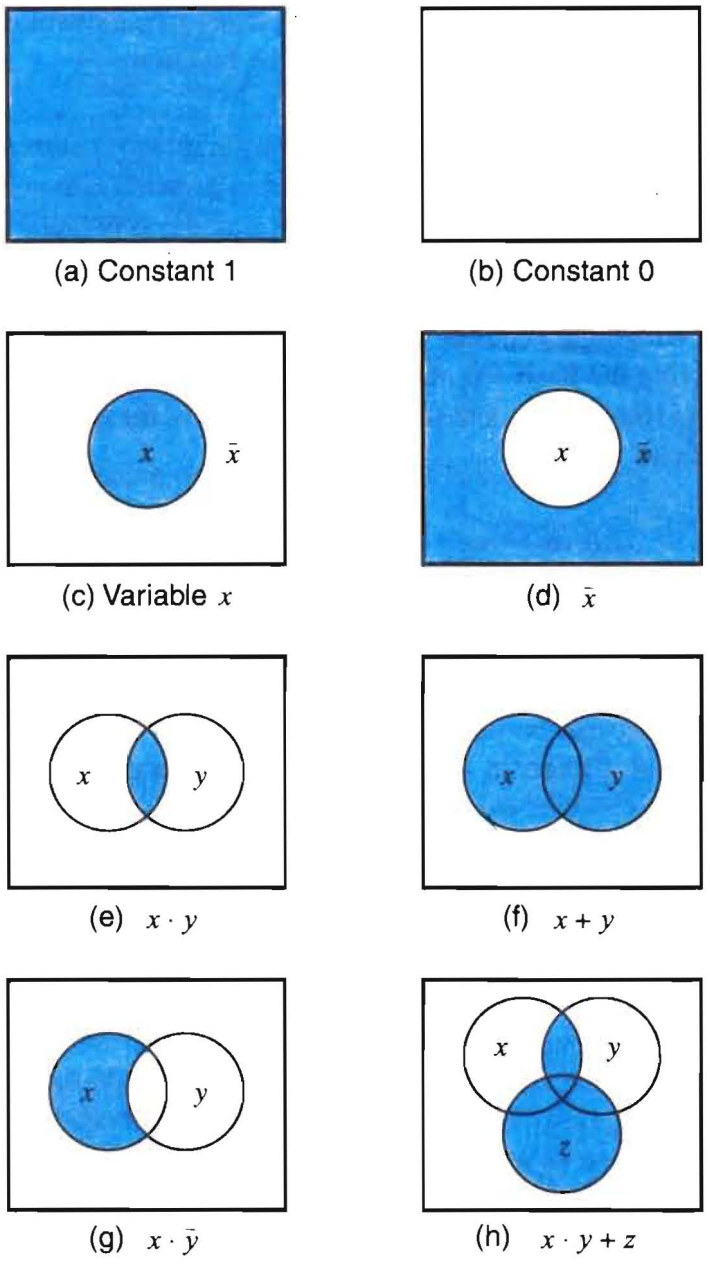


Figure 2.12 The Venn diagram representation.

Part (g) depicts the product term $x \cdot \bar{y}$, which is represented by the intersection of the area for x with that for $\bar{y}$. Part (h) gives a three-variable example; the expression $x \cdot y + z$ is the union of the area for z with that of the intersection of x and y.

To see how we can use Venn diagrams to verify the equivalence of two expressions, let us demonstrate the validity of the distributive property, 12a, in section 2.5. Figure 2.13 gives the construction of the left and right sides of the identity that defines the property

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

Part (a) shows the area where $x = 1$. Part (b) indicates the area for $y + z$. Part (c) gives the diagram for $x \cdot (y + z)$, the intersection of shaded areas in parts (a) and (b). The right-hand

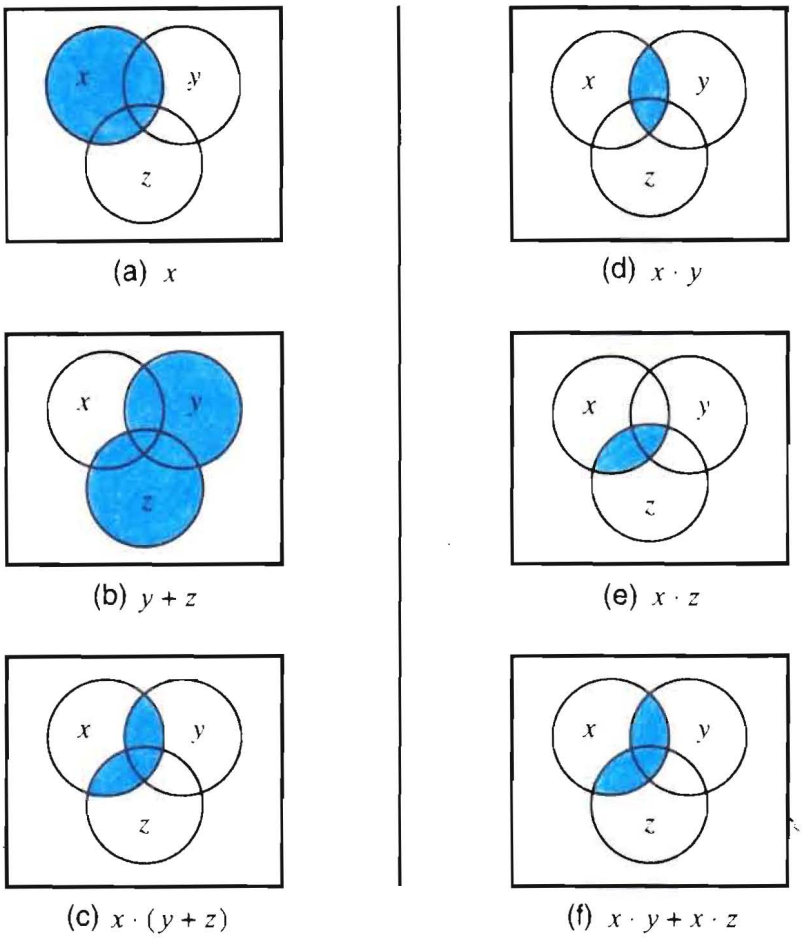


Figure 2.13 Verification of the distributive property $x \cdot (y + z) = x \cdot y + x \cdot z$.

side is constructed in parts (d), (e), and (f). Parts (d) and (e) describe the terms $x \cdot y$ and $x \cdot z$, respectively. The union of the shaded areas in these two diagrams then corresponds to the expression $x \cdot y + x \cdot z$, as seen in part (f). Since the shaded areas in parts (c) and (f) are identical, it follows that the distributive property is valid.

As another example, consider the identity

$$x \cdot y + \bar{x} \cdot z + y \cdot z = x \cdot y + \bar{x} \cdot z$$

which is illustrated in Figure 2.14. Notice that this identity states that the term $y \cdot z$ is fully covered by the terms $x \cdot y$ and $\bar{x} \cdot z$; therefore, this term can be omitted.

The reader should use the Venn diagram to prove some other identities. It is particularly instructive to prove the validity of DeMorgan's theorem in this way.

2.5.2 NOTATION AND TERMINOLOGY

Boolean algebra is based on the AND and OR operations. We have adopted the symbols $\cdot$ and $+$ to denote these operations. These are also the standard symbols for the familiar arithmetic multiplication and addition operations. Considerable similarity exists between the Boolean operations and the arithmetic operations, which is the main reason why the

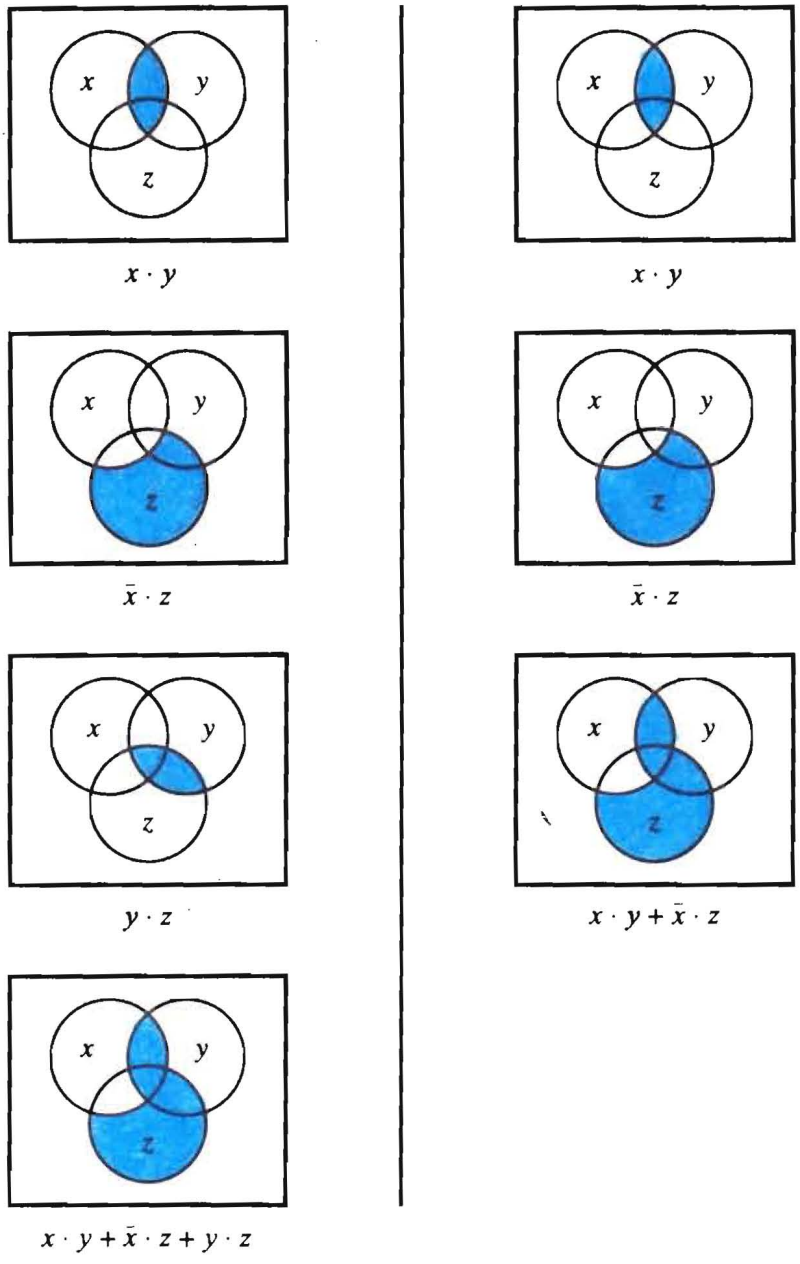


Figure 2.14 Verification of $x \cdot y + \bar{x} \cdot z + y \cdot z = x \cdot y + \bar{x} \cdot z$.

same symbols are used. In fact, when single digits are involved there is only one significant difference; the result of $1 + 1$ is equal to 2 in ordinary arithmetic, whereas it is equal to 1 in Boolean algebra as defined by theorem 7*b* in section 2.5.

When dealing with digital circuits, most of the time the + symbol obviously represents the OR operation. However, when the task involves the design of logic circuits that perform arithmetic operations, some confusion may develop about the use of the + symbol. To avoid such confusion, an alternative set of symbols exists for the AND and OR operations. It is quite common to use the $\wedge$ symbol to denote the AND operation, and the $\vee$ symbol for the

OR operation. Thus, instead of $x_1 \cdot x_2$, we can write $x_1 \wedge x_2$, and instead of $x_1 + x_2$, we can write $x_1 \vee x_2$.

Because of the similarity with the arithmetic addition and multiplication operations, the OR and AND operations are often called the *logical sum* and *product* operations. Thus $x_1 + x_2$ is the logical sum of x_1 and x_2, and $x_1 \cdot x_2$ is the logical product of x_1 and x_2. Instead of saying "logical product" and "logical sum," it is customary to say simply "product" and "sum." Thus we say that the expression

$$x_1 \cdot \bar{x}_2 \cdot x_3 + \bar{x}_1 \cdot x_4 + x_2 \cdot x_3 \cdot \bar{x}_4$$

is a sum of three product terms, whereas the expression

$$(\bar{x}_1 + x_3) \cdot (x_1 + \bar{x}_3) \cdot (\bar{x}_2 + x_3 + x_4)$$

is a product of three sum terms.

2.5.3 PRECEDENCE OF OPERATIONS

Using the three basic operations—AND, OR, and NOT—it is possible to construct an infinite number of logic expressions. Parentheses can be used to indicate the order in which the operations should be performed. However, to avoid an excessive use of parentheses, another convention defines the precedence of the basic operations. It states that in the absence of parentheses, operations in a logic expression must be performed in the order: NOT, AND, and then OR. Thus in the expression

$$x_1 \cdot x_2 + \bar{x}_1 \cdot \bar{x}_2$$

it is first necessary to generate the complements of x_1 and x_2. Then the product terms $x_1 \cdot x_2$ and $\bar{x}_1 \cdot \bar{x}_2$ are formed, followed by the sum of the two product terms. Observe that in the absence of this convention, we would have to use parentheses to achieve the same effect as follows:

$$(x_1 \cdot x_2) + ((\bar{x}_1) \cdot (\bar{x}_2))$$

Finally, to simplify the appearance of logic expressions, it is customary to omit the $\cdot$ operator when there is no ambiguity. Therefore, the preceding expression can be written as

$$x_1 x_2 + \bar{x}_1 \bar{x}_2$$

We will use this style throughout the book.

2.6 SYNTHESIS USING AND, OR, AND NOT GATES

Armed with some basic ideas, we can now try to implement arbitrary functions using the AND, OR, and NOT gates. Suppose that we wish to design a logic circuit with two inputs, x_1 and x_2. Assume that x_1 and x_2 represent the states of two switches, either of which may be open (0) or closed (1). The function of the circuit is to continuously monitor the state