Author	Alan Clements
Year	2006
Title of Article/Chapter	2.4 Introduction to Digital Work
Title of Journal/Book	Principles of Computer Hardware
Vol/part/pages	55-67
Publisher	Oxford University Press

ISBN/ISSN: 9780199273133

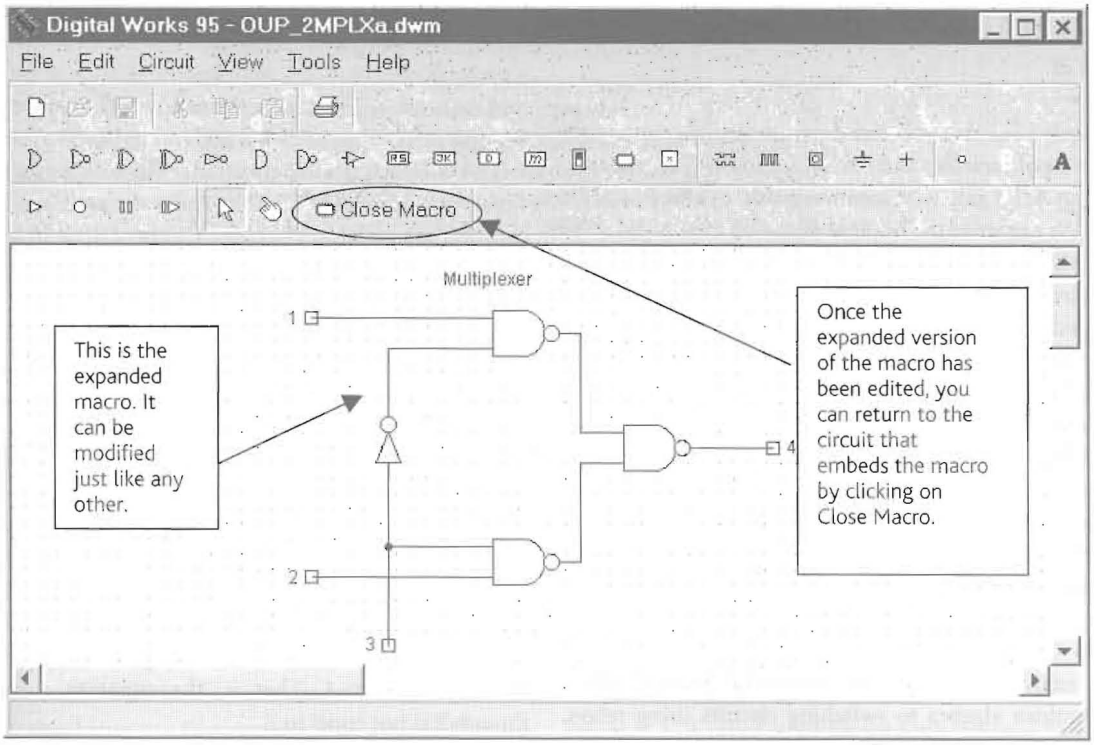
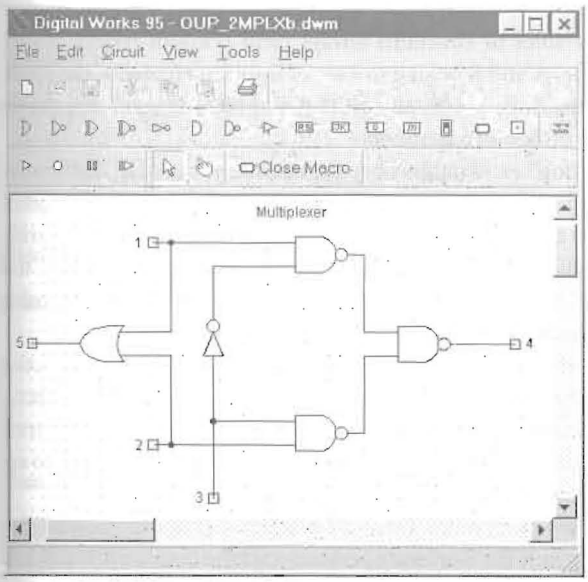
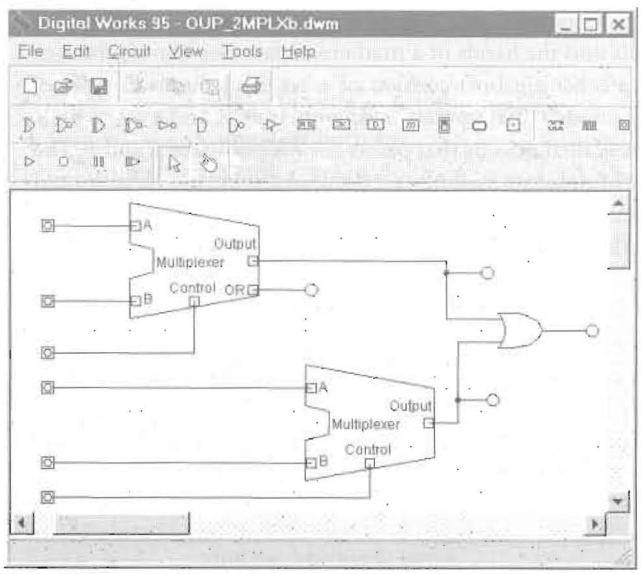


Figure 2.50 Editing the expanded form of the macro.



(a) The modified macro.



(b) The circuit with the modified macro.

Figure 2.51 Example of editing a macro.

2.5 An introduction to Boolean algebra

We've already seen that you can describe circuits containing gates in terms of variables and AND, OR, and NOT operators. Consider an AND gate with input variables A and B, and an output C. We can write the Boolean equation $C = A \cdot B$ which uses variables A, B, and C and the AND operator. In this section we introduce Boolean algebra[1], show how equations are manipulated, and demonstrate how logic circuits can be constructed with only one type of gate. Students requiring only a very basic knowledge of Boolean algebra can omit some of the fine detail that appears later in this section.

George Boole was an English mathematician (1815–1864) who developed a mathematical analysis of logic and published it in his book *An Investigation of the Laws of Thought* in 1854. Boole's algebra of logic would probably have remained a tool of the philosopher, had it not been for the development of electronics in the Twentieth Century.

In 1938 Claude Shannon published a paper entitled 'A symbolic analysis of relays and switching circuits', which applied Boolean algebra to switching circuits using relays. Such circuits were widely used in telephone exchanges and later in digital computers. Today, Boolean algebra is used to design digital circuits and to analyze their behavior.

Digital design is concerned with the conversion of ideas or specifications into hardware and Boolean algebra is a tool that facilitates this process. In particular, Boolean algebra permits an idea to be expressed in a mathematical form and the resulting expression to be simplified and then translated into the real hardware of gates and other logic elements.

Let's begin with a formal definition just in case this book falls into the hands of a mathematician. Boolean algebra (or any other algebra) consists of a set of elements E, a set of functions F that operate on members of E, and a set of basic laws called axioms that define the properties of E and F. The set of elements making up a Boolean algebra are variables and constants that have fixed values of 0 or 1. A Boolean algebra with n variables has a set of 2^n possible permutations of these variables.

Only three functions or operations are permitted in Boolean algebra. The first two are the logical OR represented by a plus (e.g. A + B) and the logical AND represented by a dot (e.g. A·B). Some texts use a ∪ (cup) or a ∨ to denote the logical OR operator and a ∩ (cap) or a ∧ to denote a logical AND operator.

The use of the plus and dot symbols is rather confusing because the same symbols are used for addition and multiplication in everyday life. One reason that these particular symbols have been chosen is that they behave rather like conventional addition and multiplication. Another possible reason Boole chose + and · to represent the logical OR and AND functions is that Boole's background was in probability theory. The chance of throwing a 1 or a 2 with two throws of a single die is 1/6 + 1/6, whereas the chance of throwing a 1 and a 2 is 1/6 × 1/6; that is, the or and and in probability theory also behave like addition and multiplication, respectively.

The third operation permitted in Boolean algebra is that of negation or complementation and is denoted by a bar over a constant or a variable. The complement of 0 (i.e. $\overline{0}$) is 1 and vice versa. The equation $X + Y \cdot \overline{Z} = A$ is read as 'X or Y and not Z equals A'. The priority of an AND operator is higher than that of an OR operator so that the expression means $A = X + (Y \cdot \overline{Z})$ and not $A = (X + Y)\overline{Z}$. Some texts use an asterisk to denote negation and some use a stroke. Thus, we can write NOT(X) as $\overline{X}$ or X* or /X.

The arithmetic operations of subtraction and division do not exist in Boolean algebra. For example, the Boolean expression $X + Y = X + Z$, cannot be rearranged in the form $(X + Y) - X = (X + Z) - X$, which would lead to $Y = Z$. If you don't believe this, then consider the case $X = 1$, $Y = 1$, and $Z = 0$. The left-hand side of the equation yields $X + Y = 1 + 1 = 1$, and the right-hand side yields $X + Z = 1 + 0 = 1$. That is, the equation is valid even though Y is not equal to Z.

2.5.1 Axioms and theorems of Boolean algebra

An axiom or postulate is a fundamental rule that has to be taken for granted (i.e. the axioms of Boolean algebra define the framework of Boolean algebra from which everything else can be derived). The first axiom is called the closure property, which states that Boolean operations on Boolean variables or constants always yield Boolean results. If variables A and B belong to a set of Boolean elements, the operations A · B, A + B, and NOT A and NOT B also belong to the set of Boolean elements.

Boolean variables obey the same commutative, distributive, and associative laws as the variables of conventional algebra. We take these laws for granted when we do everyday arithmetic; for example, the commutative law states that $6 \times 3 = 3 \times 6$. Table 2.13 describes the commutative, distributive, and associative laws of Boolean algebra.

We approach Boolean algebra by first looking at the action of NOT, OR, and AND operations on constants. The effect of these three operations is best illustrated by means of the truth table given in Table 2.14. These rules may be extended to any number of variables.

We can extend Table 2.14, which defines the relationship between the Boolean operators and the constants 0 and 1, to

[1] There are, in fact, an infinite number of Boolean algebras. We are interested only in the Boolean algebra whose variables have binary two-state values.

$A + B = B + A$ $A \cdot B = B \cdot A$	The AND and OR operators are **commutative** so that the order of the variables in a sum or product group does not matter.
$A \cdot (B \cdot C) = (A \cdot B) \cdot C$ $A + (B + C) = (A + B) + C$	The AND and OR operators are **associative** so that the order in which sub-expressions are evaluated does not matter.
$A \cdot (B + C) = A \cdot B + A \cdot C$ $A + B \cdot C = (A + B)(A + C)$	The AND operator behaves like multiplication and the OR operator like addition. The first **distributive** property states that in an expression containing both AND and OR operators the AND operator takes precedence over the OR. The second distributive law, $A + B \cdot C = (A + B)(A + C)$, is not valid in conventional algebra.

Table 2.13 Commutative, distributive, and associative laws of Boolean algebra.

NOT	AND	OR
$\overline{0} = 1$	$0 \cdot 0 = 0$	$0 + 0 = 0$
$\overline{1} = 0$	$0 \cdot 1 = 0$	$0 + 1 = 0$
	$1 \cdot 0 = 0$	$1 + 0 = 1$
	$1 \cdot 1 = 1$	$1 + 1 = 1$

Table 2.14 Basic axioms of Boolean algebra.

AND	OR	NOT
$0 \cdot X = 0$	$0 + X = X$	$\overline{\overline{X}} = X$
$1 \cdot X = X$	$1 + X = 1$	
$X \cdot X = X$	$X + X = X$	
$X \cdot \overline{X} = 0$	$X + \overline{X} = 1$	

Table 2.15 Boolean operations on a constant and a variable.

the relationship between a Boolean operator, a variable, and a literal (see Table 2.15).

We can prove the validity of the equations in Table 2.15 by substituting all the possible values for X (i.e. 0 or 1). For example, consider the axiom $0 \cdot X = 0$. If $X = 1$ we have $0.1 = 0$, which is correct because by definition the output of an AND gate is true if and only if all its inputs are true. Similarly, if $X = 0$ we have $0 \cdot 0 = 0$, which is also correct. Therefore, the expression $0 \cdot X = 0$ is correct for all possible values of X. A proof in which we test a theorem by examining all possibilities is called proof by perfect induction.

The axioms of Boolean algebra could be used to simplify equations, but it would be too tedious to keep going back to first principles. Instead, we can apply the axioms of Boolean algebra to derive some *theorems* to help in the simplification of expressions. Once we have proved a theorem by using the basic axioms, we can apply the theorem to equations.

Theorem 1 $X + X \cdot Y = X$

Proof

$$\begin{aligned} X + X \cdot Y &= X \cdot 1 + X \cdot Y && \text{Using } 1 \cdot X = X \text{ and commutativity} \\ &= X(1 + Y) && \text{Using distributivity} \\ &= X(1) && \text{Because } 1 + Y = 1 \\ &= X \end{aligned}$$

Theorem 2 $X + \overline{X} \cdot Y = X + Y$

Proof

$$\begin{aligned} X + \overline{X} \cdot Y &= (X + X \cdot Y) + \overline{X} \cdot Y && \text{By Theorem 1 } X = X + X \cdot Y \\ &= X + X \cdot Y + \overline{X} \cdot Y \\ &= X + Y(X + \overline{X}) && \text{Remember that } X + \overline{X} = 1 \\ &= X + Y(1) \\ &= X + Y \end{aligned}$$

Theorem 3 $X \cdot Y + \overline{X} \cdot Z + Y \cdot Z = X \cdot Y + \overline{X} \cdot Z$

Proof

$$\begin{aligned} X \cdot Y + \overline{X} \cdot Z + Y \cdot Z &= X \cdot Y + \overline{X} \cdot Z + Y \cdot Z(X + \overline{X}) && \text{Remember that } (X + \overline{X}) = 1 \\ &= X \cdot Y + \overline{X} \cdot Z + X \cdot Y \cdot Z + \overline{X} \cdot Y \cdot Z && \text{Multiply bracketed terms} \\ &= X \cdot Y(1 + Z) + \overline{X} \cdot Z(1 + Y) && \text{Apply distributive rule} \\ &= X \cdot Y(1) + \overline{X} \cdot Z(1) && \text{Because } (1 + Y) = 1 \\ &= X \cdot Y + \overline{X} \cdot Z \end{aligned}$$

Inputs								
X	Y	Z	$\overline{X}$	X·Y	$\overline{X}$·Z	Y·Z	X·Y + $\overline{X}$·Z	X·Y + $\overline{X}$·Z + Y·Z
0	0	0	1	0	0	0	0	0
0	0	1	1	0	1	0	1	1
0	1	0	1	0	0	0	0	0
0	1	1	1	0	1	1	1	1
1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
1	1	0	0	1	0	0	1	1
1	1	1	0	1	0	1	1	1

same

Table 2.16 Proof of Theorem 3 by perfect induction.

We can also prove Theorem 3 by the method of perfect induction. To do this, we set up a truth table and demonstrate that the theorem holds for all possible values of X, Y, and Z (Table 2.16). Because the columns labeled $X \cdot Y + \overline{X} \cdot Z$ and $X \cdot Y + \overline{X} \cdot Z + Y \cdot Z$ in Table 2.16 are identical for all possible inputs, these two expressions must be equivalent.

Theorem 4 $X(X + Y) = X$

Proof

$$\begin{aligned} X(X + Y) &= X \cdot X + X \cdot Y && \text{Multiply by X} \\ &= X + X \cdot Y && \text{Because } X \cdot X = X \\ &= X && \text{By Theorem 1} \end{aligned}$$

Theorem 5 $X(\overline{X} + Y) = X \cdot Y$

Proof

$$\begin{aligned} X(\overline{X} + Y) &= X \cdot \overline{X} + X \cdot Y \\ &= 0 + X \cdot Y && \text{Because } X \cdot \overline{X} = 0 \\ &= X \cdot Y \end{aligned}$$

Theorem 6 $(X + Y)(X + \overline{Y}) = X$

Proof

$$\begin{aligned} (X + Y)(X + \overline{Y}) &= X \cdot X + X \cdot \overline{Y} + X \cdot Y + \overline{Y} \cdot Y \\ &= X + X \cdot \overline{Y} + X \cdot Y && \text{Because } X \cdot X = X, Y \cdot \overline{Y} = 0 \\ &= X(1 + \overline{Y} + Y) \\ &= X \end{aligned}$$

Theorem 7 $(X + Y)(\overline{X} + Z) = X \cdot Z + \overline{X} \cdot Y$

Proof

$$\begin{aligned} (X + Y)(\overline{X} + Z) &= X \cdot \overline{X} + X \cdot Z + \overline{X} \cdot Y + Y \cdot Z && \text{Multiply brackets} \\ &= X \cdot Z + \overline{X} \cdot Y + Y \cdot Z && \text{Because } X \cdot \overline{X} = 0 \\ &= X \cdot Z + \overline{X} \cdot Y && \text{By Theorem 3} \end{aligned}$$

Theorem 8 $(X + Y)(\overline{X} + Z)(Y + Z) = (X + Y)(\overline{X} + Z)$

Proof

$$\begin{aligned} (X + Y)(\overline{X} + Z)(Y + Z) &= (X \cdot Z + \overline{X} \cdot Y)(Y + Z) && \text{By Theorem 7} \\ &= X \cdot Y \cdot Z + X \cdot Z \cdot Z + \overline{X} \cdot Y \cdot Y + \overline{X} \cdot Y \cdot Z \\ &= X \cdot Y \cdot Z + X \cdot Z + \overline{X} \cdot Y + \overline{X} \cdot Y \cdot Z && \text{Because } X \cdot X = 1 \\ &= X \cdot Z(Y + 1) + \overline{X} \cdot Y(1 + Z) \\ &= X \cdot Z + \overline{X} \cdot Y \\ &= (X + Y)(\overline{X} + Z) && \text{By Theorem 7} \end{aligned}$$

We provide an alternative proof for Theorem 8 when we look at de Morgan's theorem later in this chapter.

Theorem 9 $\overline{X \cdot Y \cdot Z} = \overline{X} + \overline{Y} + \overline{Z}$

Proof To prove that $\overline{X \cdot Y \cdot Z} = \overline{X} + \overline{Y} + \overline{Z}$, we assume that the expression is true and test its consequences.
If $\overline{X} + \overline{Y} + \overline{Z}$ is the complement of $X \cdot Y \cdot Z$, then from the basic axioms of Boolean algebra, we have
$(\overline{X} + \overline{Y} + \overline{Z}) \cdot (X \cdot Y \cdot Z) = 0$ and $(\overline{X} + \overline{Y} + \overline{Z}) + (X \cdot Y \cdot Z) = 1$

Subproof 1
$$(\overline{X} + \overline{Y} + \overline{Z}) \cdot X \cdot Y \cdot Z = \overline{X} \cdot X \cdot Y \cdot Z + \overline{Y} \cdot X \cdot Y \cdot Z + \overline{Z} \cdot X \cdot Y \cdot Z$$
$$= \overline{X} \cdot X \cdot (Y \cdot Z) + \overline{Y} \cdot Y \cdot (Y \cdot Z) + \overline{Z} \cdot Z(X \cdot Y)$$
$$= 0$$

Subproof 2

$(\overline{X} + \overline{Y} + \overline{Z}) + X \cdot Y \cdot Z = Y \cdot Z \cdot (X) + \overline{X} + \overline{Y} + \overline{Z}$	Re-arrange equation
$= Y \cdot Z + \overline{X} + \overline{Y} + \overline{Z}$	Use $A \cdot B + \overline{B} = A + \overline{B}$
$= (\overline{Y} + Y \cdot Z) + \overline{X} + \overline{Z}$	Re-arrange equation
$= \overline{Y} + Z + \overline{Z} + \overline{X}$	
$= \overline{Y} + 1 + \overline{X} = 1$	Use $Z + \overline{Z} = 1$

As we have demonstrated that
$(\overline{X} + \overline{Y} + \overline{Z}) \cdot X \cdot Y \cdot Z = 0$ and that
$(\overline{X} + \overline{Y} + \overline{Z}) + X \cdot Y \cdot Z = 1$, it follows that $\overline{X} + \overline{Y} + \overline{Z}$ is the complement of $X \cdot Y \cdot Z$.

Theorem 10 $\overline{X \cdot Y \cdot Z} = \overline{X} + \overline{Y} + \overline{Z}$

Proof One possible way of proving Theorem 10 is to use the method we used to prove Theorem 9. For the sake of variety, we will prove Theorem 10 by perfect induction (see Table 2.17).

Inputs								
X	Y	Z	X+Y+Z	$\overline{X+Y+Z}$	$\overline{X}$	$\overline{Y}$	$\overline{Z}$	$\overline{X} \cdot \overline{Y} \cdot \overline{Z}$
0	0	0	0	1	1	1	1	1
0	0	1	1	0	1	1	0	0
0	1	0	1	0	1	0	1	0
0	1	1	1	0	1	0	0	0
1	0	0	1	0	0	1	1	0
1	0	1	1	0	0	1	0	0
1	1	0	1	0	0	0	1	0
1	1	1	1	0	0	0	0	0

same

Table 2.17 Proof of Theorem 10 by perfect induction.

Theorems 9 and 10 are collectively called de Morgan's theorem. This theorem can be stated as an entire function is complemented by replacing AND operators by OR operators, replacing OR operators by AND operators, and complementing variables and literals. We make extensive use of de Morgan's theorem later.

An important rule in Boolean algebra is called the *principle of duality*. Any expression that is true is also true if AND is replaced by OR (and vice versa) and 1 replaced by 0 (and vice versa). Consider the following examples of duals.

Expression	Dual	
$X = X + X$	$X = X \cdot X$	(replace + by ·)
$1 = X + 1$	$0 = X \cdot 0$	(replace + by · and replace 1 by 0)
$X = X(X + Y)$	$X = X + X \cdot Y$	(replace · by + and replace + by ·)

As you can see, the dual of each expression is also true.

OBSERVATIONS

When novices first encounter Boolean algebra, it is not uncommon for them to invent new theorems that are incorrect (because they superficially look like existing theorems). We include the following observations because they represent the most frequently encountered misconceptions.

Observation 1 $\overline{X}\cdot\overline{Y} + X\cdot Y$ is not equal to 1
$\overline{X}\cdot\overline{Y} + X\cdot Y$ cannot be simplified

Observation 2 $\overline{X}\cdot Y + X\cdot\overline{Y}$ is not equal to 1
$\overline{X}\cdot Y + X\cdot\overline{Y}$ cannot be simplified

Observation 3 $\overline{X\cdot Y}$ is not equal to $\overline{X}\cdot\overline{Y}$

Observation 4 $\overline{X + Y}$ is not equal to $\overline{X} + \overline{Y}$

ALL FUNCTIONS OF TWO VARIABLES—ALL POSSIBLE GATES

This table provides all possible functions of two variables A and B. These two variables have $2^2 = 4$ possible different combinations. We can associate a different function with each of these $4^2 = 16$ values to create all possible functions of two variables; that is, there are only 16 possible types of two-input gate. Some of the functions correspond to functions we've already met. Some functions are meaningless.

Inputs		Functions															
A	B	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Function	Expression	Name
F_0	0	
F_1	$\overline{A + B}$	NOR
F_2	$\overline{A}\cdot B$	
F_3	$\overline{A}$	NOT
F_4	$A\cdot\overline{B}$	
F_5	$\overline{B}$	NOT
F_6	$A \oplus B$	EOR
F_7	$\overline{A\cdot B}$	NAND
F_8	$A\cdot B$	AND
F_9	$\overline{A \oplus B}$	ENOR
F_{10}	B	
F_{11}	$\overline{A}\cdot\overline{B} + A\cdot\overline{B} + A\cdot B = \overline{A\cdot\overline{B}} = \overline{A} + B$	
F_{12}	A	
F_{13}	$\overline{A}\cdot\overline{B} + A\cdot\overline{B} + A\cdot B = \overline{\overline{A}\cdot B} = A + \overline{B}$	
F_{14}	$A + B$	OR
F_{15}	1	

Examples of the use of Boolean algebra in simplifying equations

Having presented the basic rules of Boolean algebra, the next step is to show how it's used to simplify Boolean expressions. By simplifying these equations you can sometimes produce a cheaper version of the logic circuit. The following equations are generally random functions chosen to demonstrate the rules of Boolean algebra.

(a) $X + \overline{Y} + \overline{X}\cdot Y + (X + \overline{Y})\cdot\overline{X}\cdot Y$

(b) $\overline{X}\cdot Y\cdot\overline{Z} + \overline{X}\cdot Y\cdot Z + X\cdot\overline{Y}\cdot Z + X\cdot Y\cdot Z$

(c) $\overline{\overline{X \cdot Y} \cdot \overline{X \cdot Z}}$

(d) $(X + \overline{Y})(\overline{X} + Z)(Y + \overline{Z})$

(e) $(W + X + Y \cdot Z)(\overline{W} + X)(\overline{X} + Y)$

(f) $W \cdot X \cdot \overline{Z} + \overline{X} \cdot Y \cdot Z + W \cdot X \cdot \overline{Y} + X \cdot Y \cdot Z + \overline{W} \cdot Y \cdot Z$

(g) $\overline{W} \cdot X \cdot Z + W \cdot Z + X \cdot Y \cdot \overline{Z} + \overline{W} \cdot X \cdot Y$

(h) $(X + Y + Z)(\overline{X} + Y + Z)(\overline{X} + Y + \overline{Z})$

Solutions

When I simplify Boolean expressions, I try to keep the order of the variables alphabetical, making it easier to pick out logical groupings.

(a) $X + \overline{Y} + \overline{X} \cdot Y + (X + \overline{Y}) \cdot \overline{X} \cdot Y = X + \overline{Y} + \overline{X} \cdot Y + X \cdot \overline{X} \cdot Y + \overline{X} \cdot \overline{Y} \cdot Y$

$= X + \overline{Y} + \overline{X} \cdot Y$ As $\overline{A} \cdot A = 0$

$= X + Y + \overline{Y}$ as $A + \overline{A} \cdot B = A + B$

$= 1$ as $A + \overline{A} = 1$

Note: When a Boolean expression can be reduced to the constant 1, the expression is always true and is independent of the variables.

(b) $\overline{X} \cdot Y \cdot \overline{Z} + \overline{X} \cdot Y \cdot Z + X \cdot \overline{Y} \cdot Z + X \cdot Y \cdot Z = \overline{X} \cdot Y \cdot (\overline{Z} + Z) + X \cdot Z \cdot (\overline{Y} + Y)$

$= \overline{X} \cdot Y \cdot (1) + X \cdot Z \cdot (1)$

$= \overline{X} \cdot Y + X \cdot Z$

(c) $\overline{\overline{X \cdot Y} \cdot \overline{X \cdot Z}} = \overline{\overline{X \cdot Y}} + \overline{\overline{X \cdot Z}}$ By Theorem 9

$= \overline{X} \cdot Y + X \cdot Z$ As $\overline{\overline{F}} = F$

Note: Both expressions in examples (b) and (c) simplify to $X \cdot \overline{Y} + X \cdot Z$, demonstrating that these two expressions are equivalent. These equations are those of the multiplexer with (b) derived from the truth table (Table 2.9) and (c) from the circuit diagram of Fig. 2.14.

(d) $(X + \overline{Y})(\overline{X} + Z)(Y + \overline{Z}) = (X \cdot \overline{X} + X \cdot Z + \overline{X} \cdot \overline{Y} + \overline{Y} \cdot Z) \cdot (Y + \overline{Z})$

$= (X \cdot Z + \overline{X} \cdot \overline{Y} + \overline{Y} \cdot Z) \cdot (Y + \overline{Z})$ As $X \cdot \overline{X} = 0$

$= (X \cdot Z + \overline{X} \cdot \overline{Y}) \cdot (Y + \overline{Z})$ By Theorem 3

$= X \cdot Y \cdot Z + X \cdot \overline{Z} \cdot Z + \overline{X} \cdot \overline{Y} \cdot Y + \overline{X} \cdot \overline{Y} \cdot \overline{Z}$

$= X \cdot Y \cdot Z + \overline{X} \cdot \overline{Y} \cdot \overline{Z}$

(e) $(W + X + Y \cdot Z)(\overline{W} + X)(\overline{X} + Y) = (W \cdot \overline{W} + \overline{W} \cdot X + \overline{W} \cdot Y \cdot Z + W \cdot X + X \cdot X + X \cdot Y \cdot Z)(\overline{X} + Y)$

$= (\overline{W} \cdot X + \overline{W} \cdot Y \cdot Z + W \cdot X + X + X \cdot Y \cdot Z)(\overline{X} + Y)$

$= (X + \overline{W} \cdot Y \cdot Z)(\overline{X} + Y)$

$= X \cdot \overline{X} + X \cdot Y + \overline{W} \cdot \overline{X} \cdot Y \cdot Z + \overline{W} \cdot Y \cdot Y \cdot Z$

$= X \cdot Y + \overline{W} \cdot \overline{X} \cdot Y \cdot Z + \overline{W} \cdot Y \cdot Z$

$= X \cdot Y + \overline{W} \cdot Y \cdot Z(\overline{X} + 1)$

$= X \cdot Y + \overline{W} \cdot Y \cdot Z$

(f) $WX\overline{Z} + \overline{X}YZ + WX\overline{Y} + XYZ + \overline{W}YZ = WX\overline{Z} + YZ(\overline{X} + X + \overline{W}) + WX\overline{Y}$

$= WX\overline{Z} + YZ + WX\overline{Y}$

$= WX(\overline{Y} + \overline{Z}) + YZ$

Note that $\overline{YZ} = \overline{Y} + \overline{Z}$ so we can write

$= W \cdot X(Y + Z) + \overline{\overline{Y} + \overline{Z}}$

$= W \cdot X + Y \cdot Z$ Because $A + \overline{A} \cdot B = A + B$

(g) $\overline{W}XZ + WZ + XY\overline{Z} + \overline{W}XY = Z(\overline{W}X + W) + XY\overline{Z} + \overline{W}XY$

$= Z(X + W) + XY\overline{Z} + \overline{W}XY$

$= XZ + WZ + XY\overline{Z} + \overline{W}XY$

$= X(Z + Y\overline{Z}) + WZ + \overline{W}XY$

$= X(Z + Y) + WZ + \overline{W}XY$

$= XZ + XY + WZ + \overline{W}XY$

$= XZ + XY(1 + \overline{W}) + WZ$

$= XZ + XY + WZ$

(h) $(X + Y + Z)(\overline{X} + Y + Z)(\overline{X} + Y + \overline{Z}) = (Y + Z)(\overline{X} + Y + \overline{Z})$ as $(A + B)(A + \overline{B}) = A$

$= Z(\overline{X} + Y) + Y \cdot \overline{Z}$ as $(A + B)(\overline{A} + C) = A \cdot C + \overline{A} \cdot B$

$= \overline{X} \cdot Z + Y \cdot Z + Y \cdot \overline{Z}$

$= \overline{X} \cdot Z + Y(Z + \overline{Z})$

$= \overline{X} \cdot Z + Y$

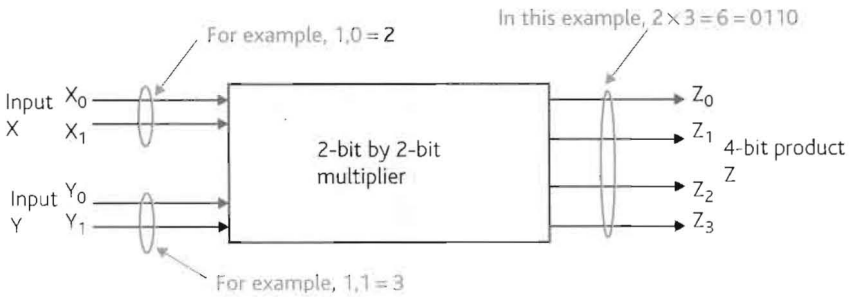


Figure 2.52 A 2-bit multiplier

	Inputs				Output			
	X		Y		Z			
X × Y = Z	X_1	X_0	Y_1	Y_0	Z_3	Z_2	Z_1	Z_0
0 × 0 = 0	0	0	0	0	0	0	0	0
0 × 1 = 0	0	0	0	1	0	0	0	0
0 × 2 = 0	0	0	1	0	0	0	0	0
0 × 3 = 0	0	0	1	1	0	0	0	0
1 × 0 = 0	0	1	0	0	0	0	0	0
1 × 1 = 1	0	1	0	1	0	0	0	1
1 × 2 = 2	0	1	1	0	0	0	1	0
1 × 3 = 3	0	1	1	1	0	0	1	1
2 × 0 = 0	1	0	0	0	0	0	0	0
2 × 1 = 2	1	0	0	1	0	0	1	0
2 × 2 = 4	1	0	1	0	0	1	0	0
2 × 3 = 6	1	0	1	1	0	1	1	0
3 × 0 = 0	1	1	0	0	0	0	0	0
3 × 1 = 3	1	1	0	1	0	0	1	1
3 × 2 = 6	1	1	1	0	0	1	1	0
3 × 3 = 9	1	1	1	1	1	0	0	1

Table 2.18 Truth table for a 2-bit by 2-bit multiplier.

These examples illustrate the art of manipulating Boolean expressions. It's difficult to be sure we have reached an optimal solution. Later we study Karnaugh maps, which provide an approach that gives us confidence that we've reached an optimal solution.

The Design of a 2-bit Multiplier

The following example illustrates how Boolean algebra is applied to a practical problem. A designer wishes to produce a 2-bit by 2-bit binary multiplier. The two 2-bit inputs are X_1, X_0 and Y_1, Y_0 and the four-bit product at the output terminals is Z_3, Z_2, Z_1, Z_0. We have not yet introduced binary arithmetic (see Chapter 4), but nothing difficult is involved here. We begin by considering the block diagram of the system (Fig. 2.52) and constructing its truth table.

The multiplier has four inputs, X_1, X_0, Y_1, Y_0, (indicating a 16-line truth table) and four outputs. Table 2.18 provides a truth table for the binary multiplier. Each 4-bit input represents the product of two 2-bit numbers so that, for example, an input of X_1, X_0, Y_1, Y_0 = 1011 represents the product $10_2 \times 11_2$ or 2×3. The corresponding output is a 4-bit product, which, in this case, is 6 or 0110 in binary form.

From Table 2.18, we can derive expressions for the four outputs, Z_0 to Z_3. Whenever a truth table has m output columns, a set of m Boolean equations must be derived. One equation is associated with each of the m columns. To derive an expression for Z_0, the four *minterms* in the Z_0 column are ORed logically.

$$
\begin{aligned}
Z_0 &= \overline{X}_1 \cdot X_0 \cdot \overline{Y}_1 \cdot Y_0 + \overline{X}_1 \cdot X_0 \cdot Y_1 \cdot Y_0 + X_1 \cdot X_0 \cdot \overline{Y}_1 \cdot Y_0 + X_1 \cdot X_0 \cdot Y_1 \cdot Y_0 \\
&= \overline{X}_1 \cdot X_0 \cdot Y_0(\overline{Y}_1 + Y_1) + X_1 \cdot X_0 \cdot Y_0(\overline{Y}_1 + Y_1) \\
&= \overline{X}_1 \cdot X_0 \cdot Y_0 + X_1 \cdot X_0 \cdot Y_0 \\
&= X_0 \cdot Y_0(\overline{X}_1 + X_1) \\
&= X_0 \cdot Y_0
\end{aligned}
$$

$$
\begin{aligned}
Z_1 &= \overline{X}_1 \cdot X_0 \cdot Y_1 \cdot \overline{Y}_0 + \overline{X}_1 \cdot X_0 \cdot Y_1 \cdot Y_0 + X_1 \cdot \overline{X}_0 \cdot \overline{Y}_1 \cdot Y_0 + X_1 \cdot \overline{X}_0 \cdot Y_1 \cdot Y_0 + X_1 \cdot X_0 \cdot \overline{Y}_1 \cdot Y_0 + X_1 \cdot X_0 \cdot Y_1 \cdot \overline{Y}_0 \\
&= \overline{X}_1 \cdot X_0 \cdot Y_1(\overline{Y}_0 + Y_0) + X_1 \cdot \overline{X}_0 \cdot Y_0(Y_1 + \overline{Y}_1) + X_1 \cdot X_0 \cdot \overline{Y}_1 \cdot Y_0 + X_1 \cdot X_0 \cdot Y_1 \cdot \overline{Y}_0 \\
&= \overline{X}_1 \cdot X_0 \cdot Y_1 + X_1 \cdot \overline{X}_0 \cdot Y_0 + X_1 \cdot X_0 \cdot \overline{Y}_1 \cdot Y_0 + X_1 \cdot X_0 \cdot Y_1 \cdot \overline{Y}_0 \\
&= X_0 \cdot Y_1(\overline{X}_1 + X_1 \cdot \overline{Y}_0) + X_1 \cdot Y_0(\overline{X}_0 + X_0 \cdot \overline{Y}_1) \\
&= X_0 \cdot Y_1(\overline{X}_1 + \overline{Y}_0) + X_1 \cdot Y_0(\overline{X}_0 + \overline{Y}_1) \\
&= \overline{X}_1 \cdot X_0 \cdot Y_1 + X_0 \cdot Y_1 \cdot \overline{Y}_0 + X_1 \cdot \overline{X}_0 \cdot Y_0 + X_1 \cdot \overline{Y}_1 \cdot Y_0
\end{aligned}
$$

$$
\begin{aligned}
Z_2 &= X_1 \cdot \overline{X}_0 \cdot Y_1 \cdot \overline{Y}_0 + X_1 \cdot \overline{X}_0 \cdot Y_1 \cdot Y_0 + X_1 \cdot X_0 \cdot Y_1 \cdot \overline{Y}_0 \\
&= X_1 \cdot \overline{X}_0 \cdot Y_1(\overline{Y}_0 + Y_0) + X_1 \cdot X_0 \cdot Y_1 \cdot \overline{Y}_0 \\
&= X_1 \cdot \overline{X}_0 \cdot Y_1 + X_1 \cdot X_0 \cdot Y_1 \cdot \overline{Y}_0 \\
&= X_1 \cdot Y_1(\overline{X}_0 + X_0 \overline{Y}_0) \\
&= X_1 \cdot Y_1(\overline{X}_0 + \overline{Y}_0) \\
&= X_1 \cdot \overline{X}_0 \cdot Y_1 + X_1 \cdot Y_1 \cdot \overline{Y}_0
\end{aligned}
$$

$$
Z_3 = X_1 \cdot X_0 \cdot Y_1 \cdot Y_0
$$

We have now obtained four simplified sum of products expressions for Z_0 to Z_3; that is,

$$
\begin{aligned}
Z_0 &= X_0 \cdot Y_0 \\
Z_1 &= \overline{X}_1 \cdot X_0 \cdot Y_1 + X_0 \cdot Y_1 \cdot \overline{Y}_0 + X_1 \cdot \overline{X}_0 \cdot Y_0 + X_1 \cdot \overline{Y}_1 \cdot Y_0 \\
Z_2 &= X_1 \cdot \overline{X}_0 \cdot Y_1 + X_1 \cdot Y_1 \cdot \overline{Y}_0 \\
Z_3 &= X_1 \cdot X_0 \cdot Y_1 \cdot Y_0
\end{aligned}
$$

It's interesting to note that each of the above expressions is *symmetric* in X and Y. This is to be expected—if the problem itself is symmetric in X and Y (i.e. $3 \times 1 = 1 \times 3$), then the result should also demonstrate this symmetry. There are many ways of realizing the expressions for Z_0 to Z_3. The circuit of Fig. 2.53 illustrates one possible way.

2.5.2 De Morgan's theorem

Theorems 9 and 10 provide the designer with a powerful tool because they enable an AND function to be implemented by

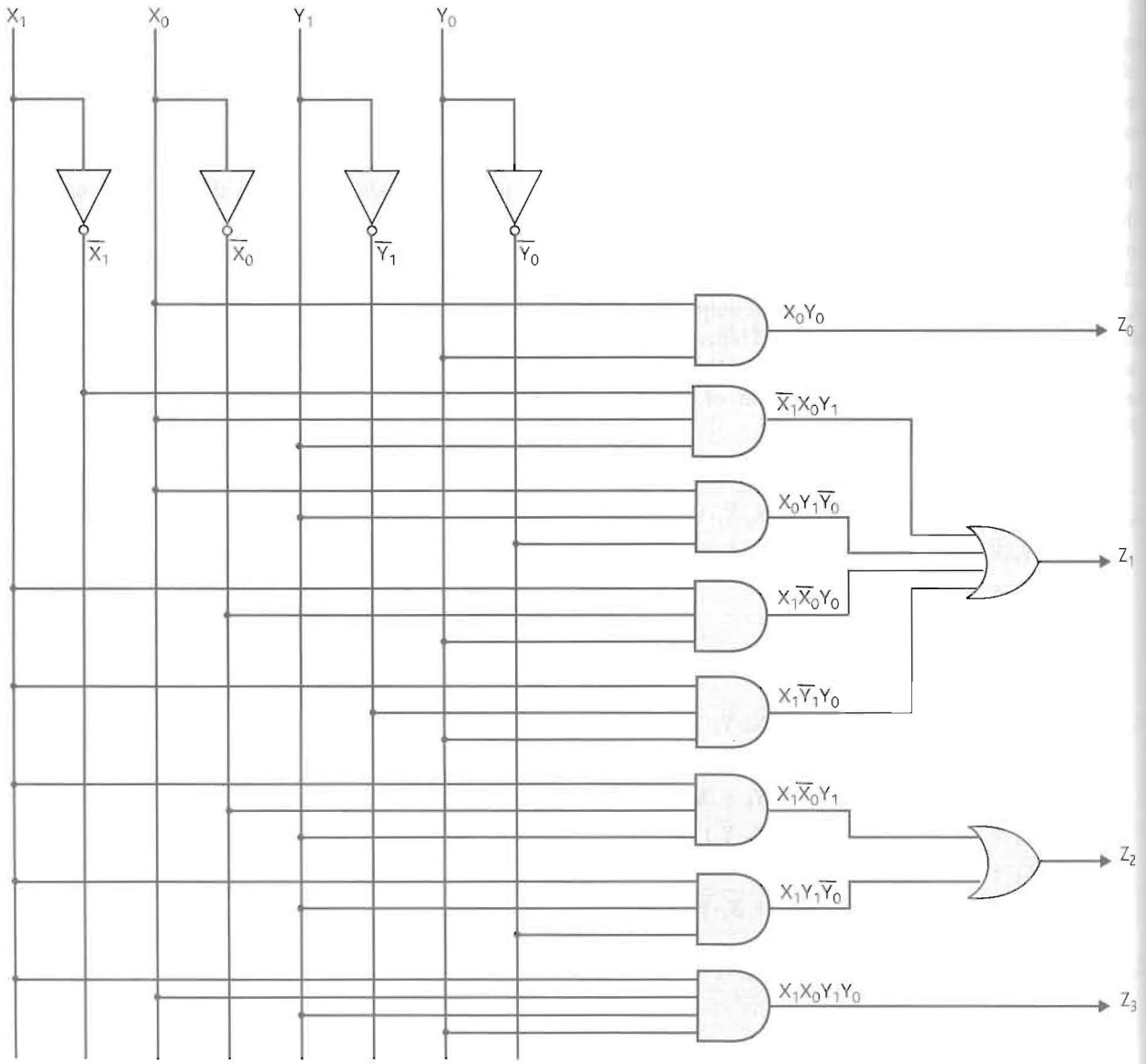


Figure 2.53 Circuit for the two-bit multiplier.

an OR gate and inverter. Similarly, these theorems enable an OR gate to be implemented by an AND gate and inverter. We first demonstrate how de Morgan's theorem is applied to Boolean expressions and then show how circuits can be converted to NAND-only or NOR-only forms. You may wonder why anyone should wish to implement circuits in NAND (or NOR) logic only. There are several reasons for this, but, in general, NAND gates operate at a higher speed than AND gates and NAND gates can be built with fewer components (at the chip level). Later we shall examine in more detail how a circuit can be designed entirely with NAND gates only.

To apply de Morgan's theorem to a function the ANDs are changed into ORs, ORs and the into ANDs and variables (and literals) are complemented. The following examples illustrate the application of de Morgan's theorem.

1. $F = \overline{X \cdot Y + X \cdot Z}$ — We wish to apply de Morgan's theorem to the right-hand side

$= \overline{X \cdot Y} \cdot \overline{X \cdot Z}$ — The + becomes · and variables 'X · Y' and 'X · Z' complemented

$= (\overline{X} + \overline{Y})(\overline{X} + \overline{Z})$ — Variables $\overline{X \cdot Y}$ and $\overline{X \cdot Y}$ are themselves complemented

As you can see, the first step is to replace the OR by an AND operator. The compound variables X · Y and X · Z are complemented to get $\overline{X \cdot Y}$ and $\overline{X \cdot Z}$. The process is continued by applying de Morgan to the two complemented groups (i.e. $\overline{X \cdot Y}$ becomes $\overline{X} + \overline{Y}$ and $\overline{X \cdot Z}$ becomes $\overline{X} + \overline{Z}$).

2. $F = \overline{A \cdot B + C \cdot D + A \cdot D}$

$= \overline{A \cdot B} \cdot \overline{C \cdot D} \cdot \overline{A \cdot D}$ Replace + by · and complement the product terms

$= (\overline{A} + \overline{B})(\overline{C} + \overline{D})(\overline{A} + \overline{D})$ Expand the complemented product terms

3. $F = \overline{A \cdot B \cdot (C + E \cdot D)}$ This is a product term with three elements.

$= \overline{A} + \overline{B} + \overline{C + E \cdot D}$ Replace · by + and complement variables

$= \overline{A} + \overline{B} + \overline{C} \cdot \overline{E \cdot D}$ Evaluate the complemented expression (change + to ·)

$= \overline{A} + \overline{B} + \overline{C} \cdot (\overline{E} + \overline{D})$ Final step, evaluate $\overline{E \cdot D}$

This example demonstrates how you have to keep applying de Morgan's theorem until there are no complemented terms left to evaluate.

4. A proof of Theorem 8 by de Morgan's theorem

$$(X + Y) \cdot (\overline{X} + Y) \cdot (Y + Z) = \overline{\overline{(X + Y) \cdot (\overline{X} + Z) \cdot (Y + Z)}}$$ Complement twice because $X = \overline{\overline{X}}$.

$$= \overline{\overline{X + Y} + \overline{\overline{X} + Z} + \overline{Y + Z}}$$ Remove inner bar by applying de Morgan

$$= \overline{\overline{X} \cdot \overline{Y} + X \cdot \overline{Z} + \overline{Y} \cdot \overline{Z}}$$ Complement the three two-variable groups

$$= \overline{\overline{X} \cdot \overline{Y} + X \cdot \overline{Z}}$$ Use Theorem 3 to simplify

$$= \overline{\overline{X} \cdot \overline{Y}} \cdot \overline{X \cdot \overline{Z}}$$ Remove outer bar, change + to ·

$$= (X + Y)(\overline{X} + Z)$$ Remove bars over two-variable groups

2.5.3 Implementing logic functions in NAND or NOR two logic only

Some gates are *better* than others; for example, the NAND gate is both faster and cheaper than the corresponding AND gate. Consequently, it's often necessary to realize a circuit using one type of gate only. Engineers sometimes implement a digital circuit with one particular type of gate because there is not a uniform range of gates available. For obvious economic reasons manufacturers don't sell a comprehensive range of gates (e.g. two-input AND, three-input AND, . . . , 10-input AND, two-input OR, . . .). For example, there are many types of NAND gate, from the quad two-input NAND to the 13-input NAND, but there are few types of AND gates.

NAND logic We first look at the way in which circuits can be constructed from nothing but NAND gates and then demonstrate that we can also fabricate circuits with NOR gates only. To construct a circuit solely in terms of NAND gates, de Morgan's theorem must be invoked to get rid of all OR operators in the expression. For example, suppose we wish to generate the expression F = A + B + C using NAND gates only. We begin by applying a double negation to the expression, as this does not alter the expression's value but it does give us the opportunity to apply de Morgan's theorem.

$F = A + B + C$ The original expression using OR logic

$F = \overline{\overline{F}} = \overline{\overline{A} + \overline{B} + \overline{C}}$ Double negation has no effect on the value of a function

$F = \overline{\overline{A} \cdot \overline{B} \cdot \overline{C}}$ Apply de Morgan's theorem

We've now converted the OR function into a NAND function. The three NOT functions that generate $\overline{A}$, $\overline{B}$, and $\overline{C}$ can be implemented in terms of NOT gates, or by means of two-input NAND gates with their inputs connected together.

Figure 2.54 shows how the function F = A + B + C can be implemented in NAND logic only. If the inputs of a NAND gate are A and B, and the output is C, then $C = \overline{A \cdot B}$. But if A = B, then $C = \overline{A \cdot A}$ or $C = \overline{A}$. You can better understand this by looking at the truth table for the NAND gate, and imagining the effect of removing the lines A, B = 0, 1 and A, B = 1, 0.

It's important to note that we are not using de Morgan's theorem here to simplify Boolean expressions. We are using de Morgan's theorem to convert an expression into a form suitable for realization in terms of NAND (or NOR) gates.

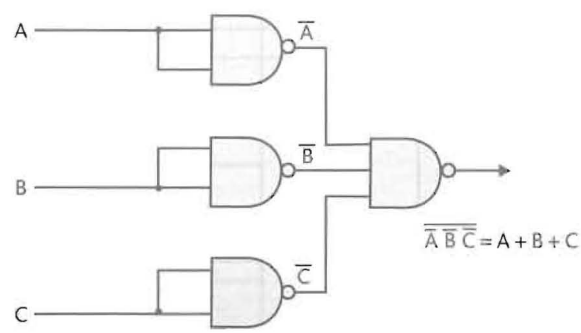


Figure 2.54 Implementing F = A + B + C with NAND logic only.

By applying the same techniques to the 2-bit by 2-bit multiplier we designed earlier we can convert the expressions for the four outputs into NAND-only logic.

$$Z_0 = X_0 \cdot Y_0 = \overline{\overline{X_0 Y_0}} \quad \text{(i.e. NAND gate followed by NOT gate = AND gate)}$$

$$\begin{aligned} Z_1 &= \overline{X}_1 X_0 Y_1 + X_0 Y_1 \overline{Y}_0 + X_1 \overline{X}_0 Y_0 + X_1 \overline{Y}_1 Y_0 \\ &= \overline{\overline{X}_1 X_1 Y_0 + X_0 Y_1 \overline{Y}_0 + X_1 \overline{X}_0 Y_0 + X_1 \overline{Y}_1 Y_0} \\ &= \overline{\overline{X}_1 X_0 Y_1 \cdot X_0 Y_1 \overline{Y}_0 \cdot X_1 \overline{X}_0 Y_0 \cdot X_1 \overline{Y}_1 Y_0} \end{aligned}$$

$$\begin{aligned} Z_2 &= X_1 \overline{X}_0 Y_1 + X_1 Y_1 \overline{Y}_0 \\ &= \overline{\overline{X_1 \overline{X}_0 Y_1 + X_1 Y_1 \overline{Y}_0}} \\ &= \overline{\overline{X_1 \overline{X}_0 Y_1} \cdot \overline{X_1 Y_1 \overline{Y}_0}} \end{aligned}$$

$$\begin{aligned} Z_3 &= X_1 X_0 Y_1 Y_0 \\ &= \overline{\overline{X_1 X_0 Y_1 Y_0}} \end{aligned}$$

Figure 2.55 shows the implementation of the multiplier in terms of NAND logic only. Note that this circuit performs exactly the same function as the circuit of Fig. 2.53.

NOR logic The procedures we've just used may equally be applied to the implementation of circuits using NOR gates

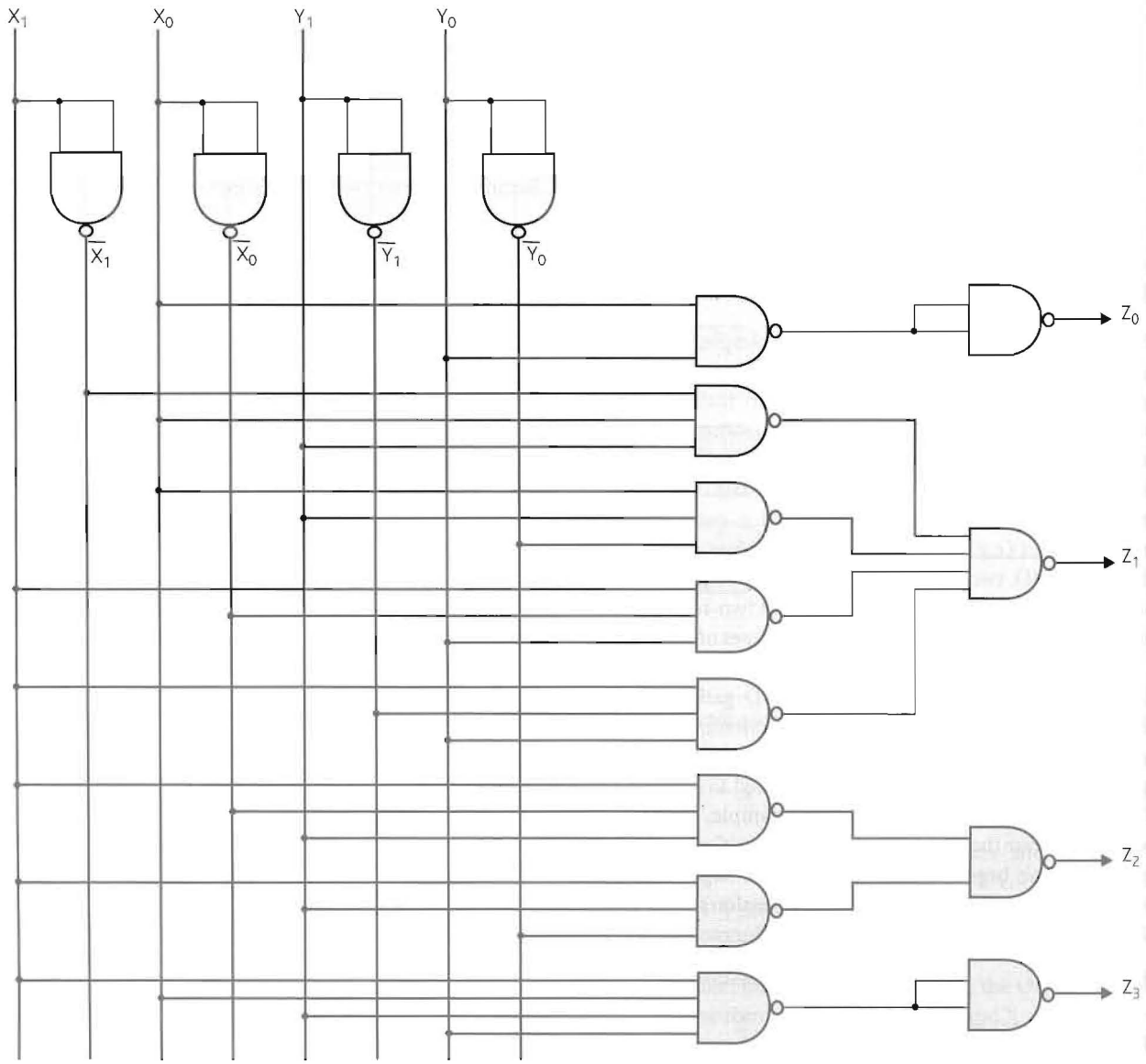


Figure 2.55 Implementing the multiplier circuit in NAND logic only.

only. By way of illustration, the value of Z_3 in the 2-bit multiplier can be converted to NOR logic form in the following way

$$Z_3 = X_1 \cdot X_0 \cdot Y_1 \cdot Y_0$$
$$= \overline{\overline{X_1 \cdot X_0 \cdot Y_1 \cdot Y_0}}$$
$$= \overline{\overline{X}_1 + \overline{X}_0 + \overline{Y}_1 + \overline{Y}_0}$$

Note that negation may be implemented by an inverter or by a NOR gate with its inputs connected together.

As a final example of NAND logic consider Fig. 2.56. A Boolean expression can be expressed in sum-of-products form as $A \cdot B + C \cdot D$. This expression can be converted to NAND logic as

$$\overline{\overline{A \cdot B} \cdot \overline{C \cdot D}}$$

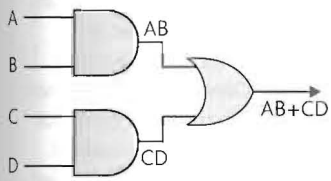
Note how the three-gate circuit in Fig. 2.56(a) can be converted into the three-gate NAND circuit of Fig. 2.56(b).

Fig. 2.57 shows the construction of the two versions of $AB + CD$ in Digital Works. We have provided an LED at each output and manually selectable inputs to enable you to investigate the circuits.

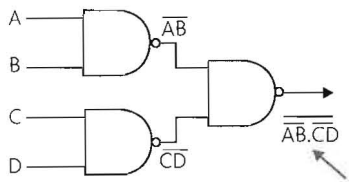
2.5.4 Karnaugh maps

When you use algebraic techniques to simplify a Boolean expression you sometimes reach a point at which you can't proceed, because you're unable to find further simplifications. The *Karnaugh map*, or more simply the *K-map*, is a graphical technique for the representation and simplification of a Boolean expression that shows unambiguously when a Boolean expression has been reduced to its most simple form.

Although the Karnaugh map can simplify Boolean equations with five or six variables, we will use it to solve problems



(A) Realization of AB + CD (AND/OR logic).



$\overline{\overline{AB}.\overline{CD}} = \overline{\overline{AB}} + \overline{\overline{CD}} = AB + CD$

(b) Realization of AB + CD (NAND logic).

Figure 2.56 Implementing $A \cdot B + C \cdot D$ in AND/OR and NAND logic.

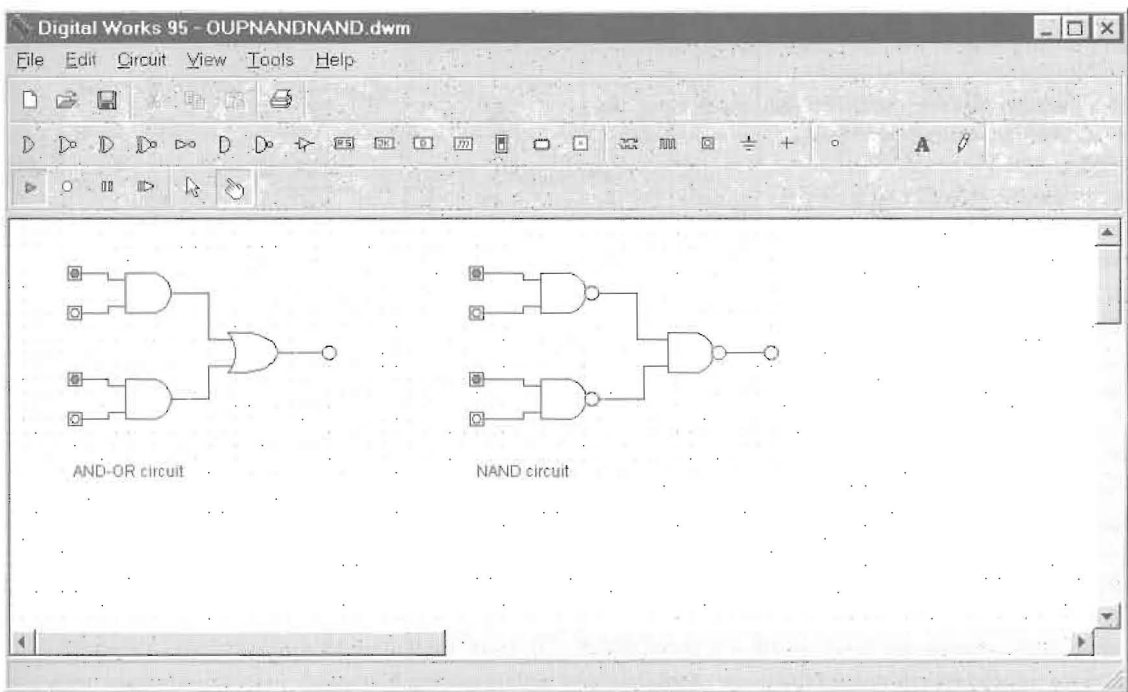


Figure 2.57 Using Digital Works to investigate two circuits.