ECE 2372 Modern Digital Design July 9

Reading assignment for today: Ch 2: 2-2 thru and including 2-3

On page 43 in your textbook is a figure showing the common logic gate types and schematic symbols, their Boolean Algebra equation and their truth tables. I have included that table at the end of this document for your convenience. Note for the NOT gate, NAND gate, NOR gate, and XNOR gates there is a "bubble" on the output signal. The bubble on a signal path means that signal is inverted at that point.

Note for example the AND gate at the top of the page and the NAND gate about halfway down the page. The symbols look the same except there is a bubble on the output pin of the NAND gate. Notice the truth tables for these two gates on the righthand side of the figure. Note that if the took the function of the truth table for the AND gate, and inverted the results, we would have the same truth table as for the NAND gate. In fact, the NAND gate (NOT AND) is in fact the inverse function of the AND gate, and can be achieved by simply inverting the output of an AND gate. 0 = FALSE and 1 = TRUE.

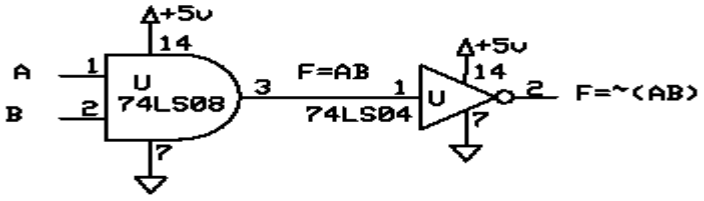
AND

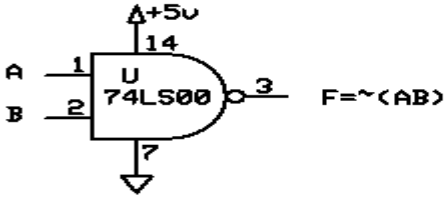
A	B	F= AB	F=~(AB)
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

NAND

A	B	F=~(AB)
0	0	1
0	1	1
1	0	1
1	1	0

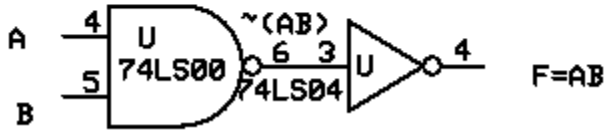
Notice we have two truth tables above. The leftmost truth table is for the AND function of two variables, A and B, and is represented as F = AB. Sometimes this is written as F = A·B but for convenience in writing or typing, the dot is often left out and it is simply written as AB. In the leftmost truth table above, we have included an extra column on the righthand side for the inverse of the function F = AB. It is written as F = ~(AB). The ~ symbol is often used in typing to indicate inversion, or NOT of the signal. Now notice the rightmost truth table for the NAND function. NAND stands for NOT AND. The NAND function is in fact the inverse of the AND function. Notice the rightmost column on the NAND truth table F=~(AB) exactly matches the NOT AND (F=~(AB)) column of the leftmost truth table, and that column is simply the inverse of the F=AB column to the left. Notice the two circuits below





achieve the same logical function, NOT AND of the two variables A, B. In the diagram above, I have chosen standard logic gate components available from electronics component vendors such as Mouser Electronics and Digikey. These are from the 7400 series of logic gates. In particular, the 74LS (low power Schotcky series). The 74LS00 is a NAND gate, the 74LS08 is an AND gate, and the 74LS04 is a NOT or INVERTER gate. I went ahead and showed the power supply connections for the gates. They do require power to run since they internally have transistors inside the chips. A lot of times the power and ground connections are not shown, they are simply assumed. We will have plenty of logic circuit diagrams without always showing the power and ground connections, but be aware those connections are required in actual circuits, and each and every logic chip should also have its own individual power supply bypass capacitor. You will learn more about that in future classes such as Electronics and the lab classes.

We can by combining gate combinations achieve other logical functions. For example, immediately below we see a method to form the AND function using a NAND gate and an INVERTER gate.



Let's look at the truth table for this case.

AND

A	B	F= AB
0	0	0
0	1	0
1	0	0
1	1	1

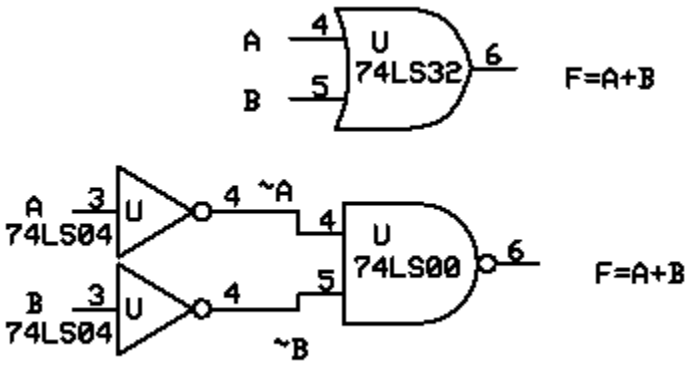
NOT NAND

A	B	F=~(AB)	F= ~(~(AB))
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	0

Notice the rightmost column F= ~(~(AB)) matches exactly with the F = AB column in the AND truth table. The two inversions cancel out.

Now look at the case below where we first invert the A and B signals before feeding them into a NAND gate. We achieve F = A + B by inverting A and B individually prior to feeding the NAND gate.

We see the OR function implemented using a 74LS32 logic gate, but also an implementation of the same function using two INVERTERS and a NAND gate



Let's look at the truth table for this case.

OR			NAND of ~A, ~B				
A	B	F= A + B	A	B	A	B	F=~(~A~B)
0	0	0	0	0	1	1	0
0	1	1	0	1	1	0	1
1	0	1	1	0	0	1	1
1	1	1	1	1	0	0	1

Notice the rightmost column for each truth table. They both match. F = A + B is the same as F = ~(~A~B).

Ordinarily we will simply use the appropriate gate for the logic function we need, but it is useful to understand that logic gates can be combined to form alternate functionality. This is important to understand just for the sake of understanding the subject, but also can be handy in a practical design if for example you need an OR function but don't have any spare OR gates available, but do have a spare NAND and a couple of INVERTER gates handy. On well know computer manufacturer of the early 1970's era used only NAND gates in their computer designs. They realized they could create any logic function from NAND gates, and that also simplified their inventory of spare components for repair.

Boolean Algebra

The idea of a special algebra for logic circuit systems may seem daunting at first. However, Boolean Algebra can be quite handy for specifying a logic circuit design an for simplifying a design to a more efficient form prior to implementation with actual logic gates. The rules of this algebra have a lot in common with regular algebra. One simplification in Boolean Algebra is that variables can only have one of two values, either 0 or 1. Some basic identities of Boolean Algebra are given below.

X + 0 = X

X + 1 = 1

X + X = X

X + ~X = 1

~(~X) = X

X · 1 = X

X · 0 = 0

X · X = X

X · ~X = 0

Further, commutative, associative, distributive and DeMorgan's apply

X + Y = Y + X

X + (Y + Z) = (X + Y) + Z

X(Y + Z) = XY + XZ

~(X +Y) = ~X · ~Y

XY = YX

X(YZ) = (XY)Z

X +YZ = (X + Y)(X +Z)

~(XY) = ~X + ~Y

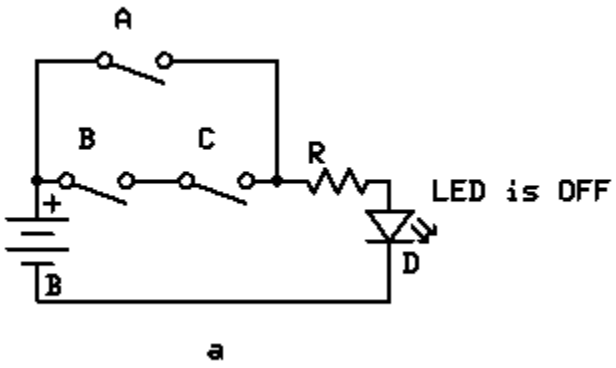
On page 50 in the scanned material below, is a table showing the proof of DeMorgan's Theorem. You should work thru that table to develop an understanding of DeMorgan's Theorem. DeMorgan's Theorem can sometimes be use to simplify a design.

To develop a better foundation for working with Boolean Algebra, it is useful to consider a logic function defined by a truth table, and then see how that compares to showing the same function in Boolean Algebraic form.

Let's initially consider a function of 3 variables, A, B, and C. We could conveniently write out the basic idea as Z = F(A,B,C). Here we are saying that some variable Z is equal to some function F of the three variables A, B, and C. Let's first set this up in a truth table as shown below:

Binary code			Z= F(A,B,C)
2^2	2^1	2^0	
4's	2's	1's	
A	B	C	
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Let's set this up with the following scenario: There is a locked door that can be opened if both B and C are available with their keys at the same time. We also have person A who can open the door at any time with a special key. The table is filled in above to represent these cases. Notice that the function is TRUE for the case where B and C are both TRUE, and it is also TRUE for any case where A is TRUE. This logic function can be represented by the circuit diagram shown below:



a

Note that any time A is in the ON position, the LED will turn ON. Likewise, if both B and C switches are in the ON position, the LED will turn ON. And, further, if all the switches A, B, and C are ON, the LED will be ON. We can write this out in the form of an equation:

Z = F(A,B,C) = ~ABC + A~B~C + A~BC + AB~C + ABC

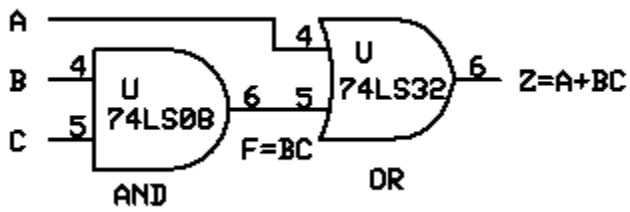
With some algebraic manipulation, we can simplify this equation. By applying some of the identities from the information given earlier:

~ABC can be combined with ABC to eliminate the A term as follows:
Z = F(A,B,C) = (~ABC + ABC) + A~B~C + A~BC + AB~C
This yields
Z = F(A,B,C) = (~A + A)(BC) + A~B~C + A~BC + AB~C

Which yields
Z = F(A,B,C) = (1)(BC) + A~B~C + A~BC + AB~C
Which reduces to
Z = F(A,B,C) = (BC) + A~B~C + A~BC + AB~C
We can further combine some remaining terms
Z = F(A,B,C) = (1))(BC) + (A~B~C + AB~C) + A~BC
To obtain
Z = F(A,B,C) =(BC) + (A~C + AC) (~B+B) + A~BC
Yielding
Z = F(A,B,C) = (BC) + (A~C + AC) (1) + A~BC
Further
Z = F(A,B,C) = (BC) + (A~C + AC) + A~BC
Reduces to
Z = F(A,B,C) = (BC) + A(~C + C) + A~BC
Which reduces to
Z = F(A,B,C) = (BC) + A(1) + A~BC
Leaving
Z = F(A,B,C) = (BC) + A + A~BC
But, A OR'ed with anything else reduces to just A, so we obtain finally:
Z = F(A,B,C) = BC + A

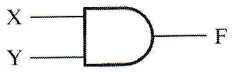
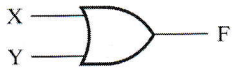
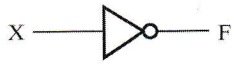
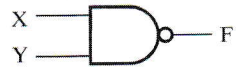
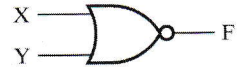
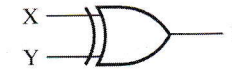
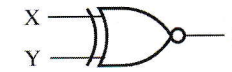
So, Z, a function of three variables A,B, and C is reduced to BC + A

We can implement this as a logic circuit shown here:



This solution is in the form Sum of Products (SOP). Note the equation: Z = F(A,B,C) = A + BC
It may not be quite so obvious in this simple example, but if we treat the + symbol as indicating a "sum" (although it represents the OR function here), and we treat BC as a product (B·C) (written as BC), then the resulting equation is a sum of products. It might be more obvious in a more complex case such as: Z = F(A,B,C) = ABC + ~AB + BC

We will stop here for today. Look at the scanned material below in preparation for the next class (and some is related to today's class as well). In particular, tomorrow we will begin to look at the SOP form using minterms, and the POS (Product of Sums) form using Maxterms, and learn some alternative formats for specifying a logic function. And we will learn some additional methods for simplifying logic equations using graphical tools known as K-maps. You should read up thru to section 2-5 on page 71.

Name	Distinctive-Shape Graphics Symbol	Algebraic Equation	Truth Table
AND		$F = XY$	X Y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = X + Y$	X Y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT (inverter)		$F = \overline{X}$	X | F 0 | 1 1 | 0
NAND		$F = \overline{X \cdot Y}$	X Y | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = \overline{X + Y}$	X Y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
Exclusive-OR (XOR)		$F = X\overline{Y} + \overline{X}Y$ $= X \oplus Y$	X Y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
Exclusive-NOR (XNOR)		$F = XY + \overline{X}\overline{Y}$ $= X \oplus Y$	X Y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

□ **FIGURE 2-3**
Commonly Used Logic Gates

outputs, function F is a multiple output function with multiple variables and equations required to represent its outputs. Circuit gates are interconnected by wires that carry logic signals. Logic circuits of this type are called *combinational logic circuits*, since the variables are "combined" by the logical operations. This is in contrast to the sequential logic to be treated in Chapter 4, in which variables are stored over time as well as being combined.

There is only one way that a Boolean function can be represented in a truth table. However, when the function is in algebraic equation form, it can be expressed in a variety of ways. The particular expression used to represent the function dictates the interconnection of gates in the logic circuit diagram. By manipulating a Boolean expression according to Boolean algebraic rules, it is often possible to obtain a simpler expression for the same function. This simpler expression reduces both the number of gates in the circuit and the numbers of inputs to the gates. To see how this is done, we must first study the basic rules of Boolean algebra.

Basic Identities of Boolean Algebra

Table 2-6 lists the most basic identities of Boolean algebra. The notation is simplified by omitting the symbol for AND whenever doing so does not lead to confusion. The first nine identities show the relationship between a single variable X, its complement $\overline{X}$, and the binary constants 0 and 1. The next five identities, 10 through 14, have counterparts in ordinary algebra. The last three, 15 through 17, do not apply in ordinary algebra, but are useful in manipulating Boolean expressions.

The basic rules listed in the table have been arranged into two columns that demonstrate the property of duality of Boolean algebra. The *dual* of an algebraic expression is obtained by interchanging OR and AND operations and replacing 1s by 0s and 0s by 1s. An equation in one column of the table can be obtained from the corresponding equation in the other column by taking the dual of the expressions on both sides of the equals sign. For example, relation 2 is the dual of relation 1 because the OR has been replaced by an AND and the 0 by 1. It is important to note that most of the time the dual of an expression is not equal to the original expression, so that an expression usually cannot be replaced by its dual.

□ **TABLE 2-6**
Basic Identities of Boolean Algebra

1. $X + 0 = X$		2. $X \cdot 1 = X$	
3. $X + 1 = 1$		4. $X \cdot 0 = 0$	
5. $X + X = X$		6. $X \cdot X = X$	
7. $X + \overline{X} = 1$		8. $X \cdot \overline{X} = 0$	
9. $\overline{\overline{X}} = X$			
10. $X + Y = Y + X$		11. $XY = YX$	Commutative
12. $X + (Y + Z) = (X + Y) + Z$		13. $X(YZ) = (XY)Z$	Associative
14. $X(Y + Z) = XY + XZ$		15. $X + YZ = (X + Y)(X + Z)$	Distributive
16. $\overline{X + Y} = \overline{X} \cdot \overline{Y}$		17. $\overline{X \cdot Y} = \overline{X} + \overline{Y}$	DeMorgan's

The nine identities involving a single variable can be easily verified by substituting each of the two possible values for X. For example, to show that $X + 0 = X$, let $X = 0$ to obtain $0 + 0 = 0$, and then let $X = 1$ to obtain $1 + 0 = 1$. Both equations are true according to the definition of the OR logic operation. Any expression can be substituted for the variable X in all the Boolean equations listed in the table. Thus, by identity 3 and with $X = AB + C$, we obtain

$$AB + C + 1 = 1$$

Note that identity 9 states that double complementation restores the variable to its original value. Thus, if $X = 0$, then $\overline{X} = 1$ and $\overline{\overline{X}} = 0 = X$.

Identities 10 and 11, the commutative laws, state that the order in which the variables are written will not affect the result when using the OR and AND operations. Identities 12 and 13, the associative laws, state that the result of applying an operation over three variables is independent of the order that is taken, and therefore, the parentheses can be removed altogether, as follows:

$$X + (Y + Z) = (X + Y) + Z = X + Y + Z$$

$$X(YZ) = (XY)Z = XYZ$$

These two laws and the first distributive law, identity 14, are well known from ordinary algebra, so they should not pose any difficulty. The second distributive law, given by identity 15, is the dual of the ordinary distributive law and does not hold in ordinary algebra. As illustrated previously, each variable in an identity can be replaced by a Boolean expression, and the identity still holds. Thus, consider the expression $(A + B)(A + CD)$. Letting $X = A$, $Y = B$, and $Z = CD$, and applying the second distributive law, we obtain

$$(A + B)(A + CD) = A + BCD$$

The last two identities in Table 2-6,

$$\overline{X + Y} = \overline{X} \cdot \overline{Y} \text{ and } \overline{X \cdot Y} = \overline{X} + \overline{Y}$$

are referred to as DeMorgan's theorem. This is a very important theorem and is used to obtain the complement of an expression and of the corresponding function. DeMorgan's theorem can be illustrated by means of truth tables that assign all the possible binary values to X and Y. Table 2-7 shows two truth tables that verify the

□ **TABLE 2-7**
Truth Tables to Verify DeMorgan's Theorem

(a) X	Y	X + Y	$\overline{X + Y}$	(b) X	Y	$\overline{X}$	$\overline{Y}$	$\overline{X} \cdot \overline{Y}$
0	0	0	1	0	0	1	1	1
0	1	1	0	0	1	1	0	0
1	0	1	0	1	0	0	1	0
1	1	1	0	1	1	0	0	0

first part of DeMorgan's theorem. In (a), we evaluate $\overline{X + Y}$ for all possible values of X and Y. This is done by first evaluating $X + Y$ and then taking its complement. In (b), we evaluate $\overline{X}$ and $\overline{Y}$ and then AND them together. The result is the same for the four binary combinations of X and Y, which verifies the identity of the equation.

Note the order in which the operations are performed when evaluating an expression. In part (b) of the table, the complement over a single variable is evaluated first, followed by the AND operation, just as in ordinary algebra with multiplication and addition. In part (a), the OR operation is evaluated first. Then, noting that the complement over an expression such as $X + Y$ is considered as specifying NOT $(X + Y)$, we evaluate the expression within the parentheses and take the complement of the result. It is customary to exclude the parentheses when complementing an expression, since a bar over the entire expression joins it together. Thus, $\overline{(X + Y)}$ is expressed as $\overline{X + Y}$ when designating the complement of $X + Y$.

DeMorgan's theorem can be extended to three or more variables. The general DeMorgan's theorem can be expressed as

$$\overline{X_1 + X_2 + \ldots + X_n} = \overline{X}_1\overline{X}_2 \ldots \overline{X}_n$$

$$\overline{X_1X_2 \ldots X_n} = \overline{X}_1 + \overline{X}_2 + \ldots + \overline{X}_n$$

Observe that the logic operation changes from OR to AND or from AND to OR. In addition, the complement is removed from the entire expression and placed instead over each variable. For example,

$$\overline{A + B + C + D} = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$$

Algebraic Manipulation

Boolean algebra is a useful tool for simplifying digital circuits. Consider, for example, the Boolean function represented by

$$F = \overline{X}YZ + \overline{X}Y\overline{Z} + XZ$$

The implementation of this equation with logic gates is shown in Figure 2-8(a). Input variables X and Z are complemented with inverters to obtain $\overline{X}$ and $\overline{Z}$. The three terms in the expression are implemented with three AND gates. The OR gate forms the logical OR of the terms. Now consider a simplification of the expression for F by applying some of the identities listed in Table 2-6:

$$\begin{aligned} F &= \overline{X}YZ + \overline{X}Y\overline{Z} + XZ \\ &= \overline{X}Y(Z + \overline{Z}) + XZ && \text{by identity 14} \\ &= \overline{X}Y \cdot 1 + XZ && \text{by identity 7} \\ &= \overline{X}Y + XZ && \text{by identity 2} \end{aligned}$$

The expression is reduced to only two terms and can be implemented with gates as shown in Figure 2-8(b). It is obvious that the circuit in (b) is simpler than the one in (a) yet, both implement the same function. It is possible to use a truth table to verify that the two implementations are equivalent. This is shown in Table 2-8. As

expression for the function in Figure 2-8(a) has three terms and eight literals; the one in Figure 2-8(b) has two terms and four literals. By reducing the number of terms, the number of literals, or both in a Boolean expression, it is often possible to obtain a simpler circuit. Boolean algebra is applied to reduce an expression for the purpose of obtaining a simpler circuit. For highly complex functions, finding the best expression based on counts of terms and literals is very difficult, even by the use of computer programs. Certain methods, however, for reducing expressions are often included in computer tools for synthesizing logic circuits. These methods can obtain good, if not the best, solutions. The only manual method for the general case is a cut-and-try procedure employing the basic relations and other manipulations that become familiar with use. The following examples use identities from Table 2-6 to illustrate a few of the possibilities:

1. $X + XY = X \cdot 1 + XY = X(1 + Y) = X \cdot 1 = X$
2. $XY + X\overline{Y} = X(Y + \overline{Y}) = X \cdot 1 = X$
3. $X + \overline{X}Y = (X + \overline{X})(X + Y) = 1 \cdot (X + Y) = X + Y$

Note that the intermediate steps $X = X \cdot 1$ and $X \cdot 1 = X$ are often omitted because of their rudimentary nature. The relationship $1 + Y = 1$ is useful for eliminating redundant terms, as is done with the term XY in this same equation. The relation $Y + \overline{Y} = 1$ is useful for combining two terms, as is done in equation 2. The two terms being combined must be identical except for one variable, and that variable must be complemented in one term and not complemented in the other. Equation 3 is simplified by means of the second distributive law (identity 15 in Table 2-6). The following are three more examples of simplifying Boolean expressions:

4. $X(X + Y) = X \cdot X + X \cdot Y = X + XY = X(1 + Y) = X \cdot 1 = X$
5. $(X + Y)(X + \overline{Y}) = X + Y\overline{Y} = X + 0 = X$
6. $X(\overline{X} + Y) = X\overline{X} + XY = 0 + XY = XY$

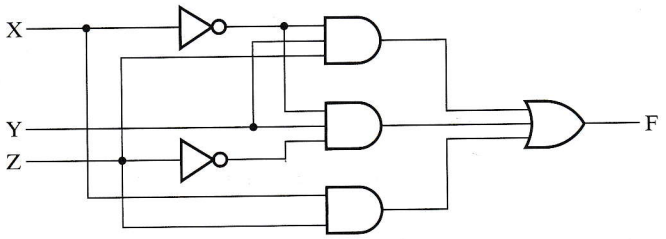
The six equalities represented by the initial and final expressions are theorems of Boolean algebra proved by the application of the identities from Table 2-6. These theorems can be used along with the identities in Table 2-6 to prove additional results and to assist in performing simplification.

Theorems 4 through 6 are the duals of equations 1 through 3. Remember that the dual of an expression is obtained by changing AND to OR and OR to AND throughout (and 1s to 0s and 0s to 1s if they appear in the expression). The *duality principle* of Boolean algebra states that a Boolean equation remains valid if we take the dual of the expressions on both sides of the equals sign. Therefore, equations 4, 5, and 6 can be obtained by taking the dual of equations 1, 2, and 3, respectively.

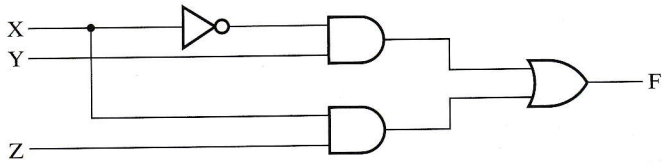
Along with the results just given in equations 1 through 6, the following *consensus theorem* is useful when simplifying Boolean expressions:

$$XY + \overline{X}Z + YZ = XY + \overline{X}Z$$

The theorem shows that the third term, YZ, is redundant and can be eliminated. Note that Y and Z are associated with X and $\overline{X}$ in the first two terms and appear



(a) $F = \overline{X}YZ + \overline{X}Y\overline{Z} + XZ$



(b) $F = \overline{X}Y + XZ$

□ **FIGURE 2-8**
Implementation of Boolean Function with Gates

□ **TABLE 2-8**
Truth Table for Boolean Function

X	Y	Z	(a) F	(b) F
0	0	0	0	0
0	0	1	0	0
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1

expressed in Figure 2-8(a), the function is equal to 1 if $X = 0$, $Y = 1$, and $Z = 1$; if $X = 0$, $Y = 1$, and $Z = 0$; or if X and Z are both 1. This produces the four 1s for F in part (a) of the table. As expressed in Figure 2-8(b), the function is equal to 1 if $X = 0$ and $Y = 1$ or if $X = 1$ and $Z = 1$. This produces the same four 1s in part (b) of the table. Since both expressions produce the same truth table, they are equivalent. Therefore, the two circuits have the same output for all possible binary combinations of the three input variables. Each circuit implements the same function, but the one with fewer gates and/or fewer gate inputs is preferable because it requires fewer components.

When a Boolean equation is implemented with logic gates, each term requires a gate, and each variable within the term designates an input to the gate. We define a *literal* as a single variable within a term that may or may not be complemented. The

together in the term that is eliminated. The proof of the consensus theorem is obtained by first ANDing YZ with $(X + \overline{X}) = 1$ and proceeds as follows:

$$\begin{aligned} XY + \overline{X}Z + YZ &= XY + \overline{X}Z + YZ(X + \overline{X}) \\ &= XY + \overline{X}Z + XYZ + \overline{X}YZ \\ &= XY + XYZ + \overline{X}Z + \overline{X}YZ \\ &= XY(1 + Z) + \overline{X}Z(1 + Y) \\ &= XY + \overline{X}Z \end{aligned}$$

The dual of the consensus theorem is

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

The following example shows how the consensus theorem can be applied in manipulating a Boolean expression:

$$\begin{aligned} (A + B)(\overline{A} + C) &= A\overline{A} + AC + \overline{A}B + BC \\ &= AC + \overline{A}B + BC \\ &= AC + \overline{A}B \end{aligned}$$

Note that $A\overline{A} = 0$ and $0 + AC = AC$. The redundant term eliminated in the last step by the consensus theorem is BC.

Complement of a Function

The complement representation for a function F, $\overline{F}$, is obtained from an interchange of 1s to 0s and 0s to 1s for the values of F in the truth table. The complement of a function can be derived algebraically by applying DeMorgan's theorem. The generalized form of this theorem states that the complement of an expression is obtained by interchanging AND and OR operations and complementing each variable and constant, as shown in Example 2-2.

EXAMPLE 2-2 Complementing Functions

Find the complement of each of the functions represented by the equations $F_1 = \overline{X}YZ + X\overline{Y}Z$ and $F_2 = X(\overline{Y}\,\overline{Z} + YZ)$. Applying DeMorgan's theorem as many times as necessary, we obtain the complements as follows:

$$\begin{aligned} \overline{F}_1 &= \overline{\overline{X}YZ + X\overline{Y}Z} = (\overline{\overline{X}YZ}) \cdot (\overline{X\overline{Y}Z}) \\ &= (X + \overline{Y} + Z)(X + Y + \overline{Z}) \\ \overline{F}_2 &= \overline{X(\overline{Y}\,\overline{Z} + YZ)} = \overline{X} + (\overline{\overline{Y}\,\overline{Z} + YZ}) \\ &= \overline{X} + \overline{\overline{Y}\,\overline{Z}} \cdot \overline{YZ} \\ &= \overline{X} + (Y + Z)(\overline{Y} + \overline{Z}) \end{aligned}$$

A simpler method for deriving the complement of a function is to take the dual of the function equation and complement each literal. This method follows from the

exactly once, either complemented or uncomplemented, is called a *minterm*. Its characteristic property is that it represents exactly one combination of binary variable values in the truth table. It has the value 1 for that combination and 0 for all others. There are 2^n distinct minterms for n variables. The four minterms for the two variables X and Y are $\overline{X}\,\overline{Y}$, $\overline{X}Y$, $X\overline{Y}$, and XY. The eight minterms for the three variables X, Y, and Z are listed in Table 2-9. The binary numbers from 000 to 111 are listed under the variables. For each binary combination, there is a related minterm. Each minterm is a product term of exactly n literals, where n is the number of variables. In this example, $n = 3$. A literal is a complemented variable if the corresponding bit of the related binary combination is 0 and is an uncomplemented variable if it is 1. A symbol m_j for each minterm is also shown in the table, where the subscript j denotes the decimal equivalent of the binary combination corresponding to the minterm. This list of minterms for any given n variables can be formed in a similar manner from a list of the binary numbers from 0 through 2^n-1. In addition, the truth table for each minterm is given in the right half of the table. These truth tables clearly show that each minterm is 1 for the corresponding binary combination and 0 for all other combinations. Such truth tables will be helpful later in using minterms to form Boolean expressions.

A sum term that contains all the variables in complemented or uncomplemented form is called a *maxterm*. Again, it is possible to formulate 2^n maxterms with n variables. The eight maxterms for three variables are listed in Table 2-10. Each maxterm is a logical sum of the three variables, with each variable being complemented if the corresponding bit of the binary number is 1 and uncomplemented if it is 0. The symbol for a maxterm is M_j, where j denotes the decimal equivalent of the binary combination corresponding to the maxterm. In the right half of the table, the truth table for each maxterm is given. Note that the value of the maxterm is 0 for the corresponding combination and 1 for all other combinations. It is now clear where the terms "minterm" and "maxterm" come from: a minterm is a function, not equal to 0, having the minimum number of 1s in its truth table; a maxterm is a function, not equal to 1, having the maximum of 1s in its truth table. Note from Table 2-9

☐ **TABLE 2-9**
Minterms for Three Variables

X	Y	Z	Product Term	Symbol	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7
0	0	0	$\overline{X}\,\overline{Y}\,\overline{Z}$	m_0	1	0	0	0	0	0	0	0
0	0	1	$\overline{X}\,\overline{Y}Z$	m_1	0	1	0	0	0	0	0	0
0	1	0	$\overline{X}Y\overline{Z}$	m_2	0	0	1	0	0	0	0	0
0	1	1	$\overline{X}YZ$	m_3	0	0	0	1	0	0	0	0
1	0	0	$X\overline{Y}\,\overline{Z}$	m_4	0	0	0	0	1	0	0	0
1	0	1	$X\overline{Y}Z$	m_5	0	0	0	0	0	1	0	0
1	1	0	$XY\overline{Z}$	m_6	0	0	0	0	0	0	1	0
1	1	1	XYZ	m_7	0	0	0	0	0	0	0	1

and Table 2-10 that a minterm and maxterm with the same subscript are the complements of each other; that is, $M_j = \overline{m}_j$ and $m_j = \overline{M}_j$. For example, for $j = 3$, we have

$$M_3 = X + \overline{Y} + \overline{Z} + = \overline{\overline{X}YZ} = \overline{m}_3$$

A Boolean function can be represented algebraically from a given truth table by forming the logical sum of all the minterms that produce a 1 in the function. This expression is called a *sum of minterms*. Consider the Boolean function F in Table 2-11(a). The function is equal to 1 for each of the following binary combinations of the variables X, Y, and Z: 000, 010, 101 and 111. These combinations correspond to minterms 0, 2, 5, and 7. By examining Table 2-11 and the truth tables for these minterms in Table 2-9, it is evident that the function F can be expressed algebraically as the logical sum of the stated minterms:

$$F = \overline{X}\,\overline{Y}\,\overline{Z} + \overline{X}Y\overline{Z} + X\overline{Y}Z + XYZ = m_0 + m_2 + m_5 + m_7$$

This can be further abbreviated by listing only the decimal subscripts of the minterms:

$$F(X, Y, Z) = \Sigma m(0, 2, 5, 7)$$

□ **TABLE 2-10**
Maxterms for Three Variables

X	Y	Z	Sum Term	Symbol	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7
0	0	0	$X + Y + Z$	M_0	0	1	1	1	1	1	1	1
0	0	1	$X + Y + \overline{Z}$	M_1	1	0	1	1	1	1	1	1
0	1	0	$X + \overline{Y} + Z$	M_2	1	1	0	1	1	1	1	1
0	1	1	$X + \overline{Y} + \overline{Z}$	M_3	1	1	1	0	1	1	1	1
1	0	0	$\overline{X} + Y + Z$	M_4	1	1	1	1	0	1	1	1
1	0	1	$\overline{X} + Y + \overline{Z}$	M_5	1	1	1	1	1	0	1	1
1	1	0	$\overline{X} + \overline{Y} + Z$	M_6	1	1	1	1	1	1	0	1
1	1	1	$\overline{X} + \overline{Y} + \overline{Z}$	M_7	1	1	1	1	1	1	1	0

□ **TABLE 2-11**
Boolean Functions of Three Variables

(a) X	Y	Z	F	$\overline{F}$	(b) X	Y	Z	E
0	0	0	1	0	0	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	1	0	0	1	0	1
0	1	1	0	1	0	1	1	0
1	0	0	0	1	1	0	0	1
1	0	1	1	0	1	0	1	1
1	1	0	0	1	1	1	0	0
1	1	1	1	0	1	1	1	0