ELECTRONICS DEVICES AND CIRCUITS

Digital Devices and Logic Circuits

THE KARNAUGH MAP

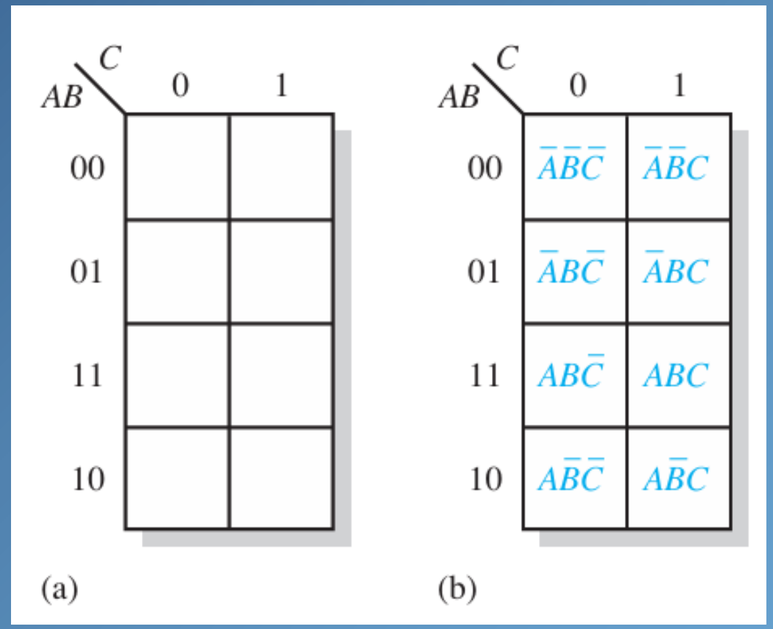
A Karnaugh map is similar to a truth table because it presents all of the possible values of input variables and the resulting output for each value. Instead of being organized into columns and rows like a truth table, the Karnaugh map is an array of cells in which each cell represents a binary value of the input variables. The cells are arranged in a way so that simplification of a given expression is simply a matter of properly grouping the cells. Karnaugh maps can be used for expressions with two, three, four, and five variables, but we will discuss only 3-variable and 4-variable situations to illustrate the principles.

The number of cells in a Karnaugh map, as well as the number of rows in a truth table, is equal to the total number of possible input variable combinations. For three variables, the number of cells is 2^3 = 8. For four variables, the number of cells is 2^4 = 16.

THE KARNAUGH MAP

The 3-Variable Karnaugh Map

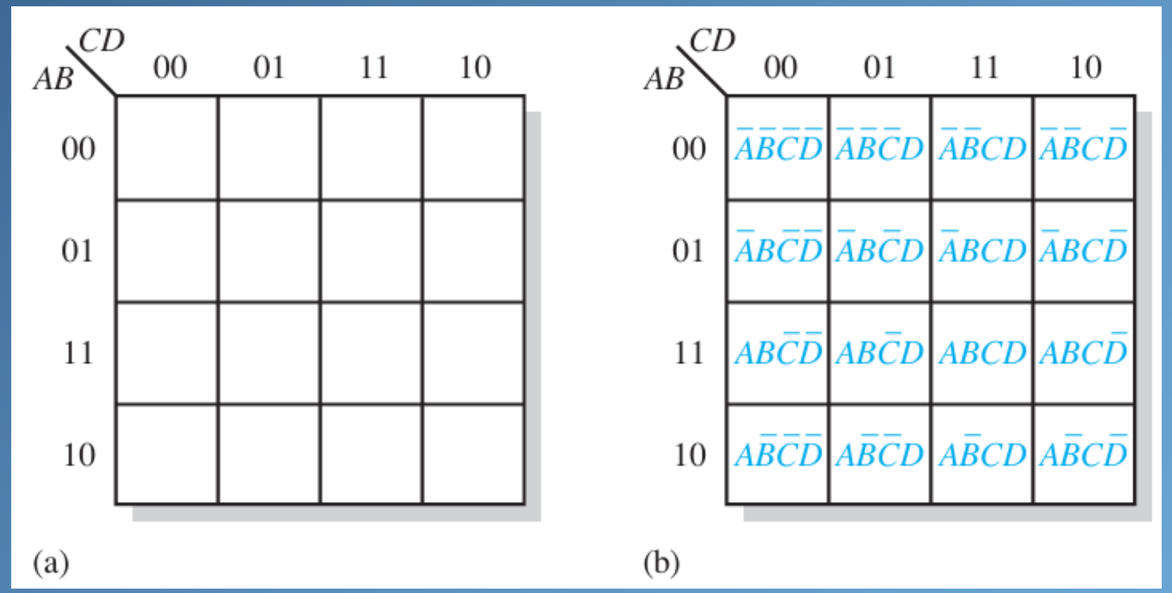
The 3-variable Karnaugh map is an array of eight cells, as shown in Figure 1.34(a). In this case, A, B, and C are used for the variables although other letters could be used. Binary values of A and B are along the left side (notice the sequence) and the values of C are across the top. The value of a given cell is the binary values of A and B at the left in the same row combined with the value of C at the top in the same column. For example, the cell in the upper left corner has a binary value of 000 and the cell in the lower right corner has a binary value of 101. Figure 1.34(b) shows the standard product terms that are represented by each cell in the Karnaugh map.



THE KARNAUGH MAP

The 4-Variable Karnaugh Map

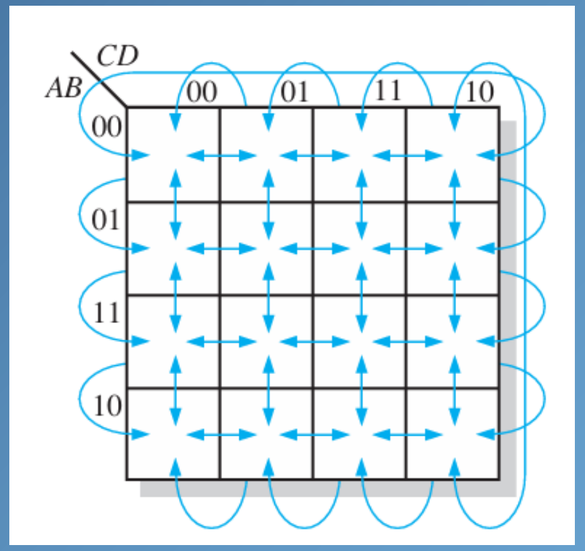
The 4-variable Karnaugh map is an array of sixteen cells, as shown in Figure 1.35(a). Binary values of A and B are along the left side and the values of C and D are across the top. The value of a given cell is the binary values of A and B at the left in the same row combined with the binary values of C and D at the top in the same column. For example, the cell in the upper right corner has a binary value of 0010 and the cell in the lower right corner has a binary value of 1010. Figure 1.35(b) shows the standard product terms that are represented by each cell in the 4-variable Karnaugh map.



THE KARNAUGH MAP

Cell Adjacency

The cells in a Karnaugh map are arranged so that there is only a single-variable change between adjacent cells. Adjacency is defined by a single-variable change. In the 3-variable map the 010 cell is adjacent to the 000 cell, the 011 cell, and the 110 cell. The 010 cell is not adjacent to the 001 cell, the 111 cell, the 100 cell, or the 101 cell. Physically, each cell is adjacent to the cells that are immediately next to it on any of its four sides. A cell is not adjacent to the cells that diagonally touch any of its corners. Also, the cells in the top row are adjacent to the corresponding cells in the bottom row and the cells in the outer left column are adjacent to the corresponding cells in the outer right column. This is called “wrap-around” adjacency because you can think of the map as wrap ping around from top to bottom to form a cylinder or from left to right to form a cylinder. Figure 1.36 illustrates the cell adjacencies with a 4-variable map, although the same rules for adjacency apply to Karnaugh maps with any number of cells.



KARNAUGH MAP SOP MINIMIZATION

Mapping a Standard SOP Expression

For an SOP expression in standard form, a 1 is placed on the Karnaugh map for each product term in the expression. Each 1 is placed in a cell corresponding to the value of a product term. For example, for the product term $A\bar{B}C$, a 1 goes in the 101 cell on a 3-variable map.

When an SOP expression is completely mapped, there will be a number of 1s on the Karnaugh map equal to the number of product terms in the standard SOP expression. The cells that do not have a 1 are the cells for which the expression is 0. Usually, when working with SOP expressions, the 0s are left off the map. The following steps and the illustration in Figure 1.37 show the mapping process.

Step 1: Determine the binary value of each product term in the standard SOP expression. After some practice, you can usually do the evaluation of terms mentally.

Step 2: As each product term is evaluated, place a 1 on the Karnaugh map in the cell having the same value as the product term.

KARNAUGH MAP SOP MINIMIZATION

Sample Problem

Map the following standard SOP expression on a Karnaugh map:

$$\overline{A}\,\overline{B}C + \overline{A}B\overline{C} + AB\overline{C} + ABC$$

Map the standard SOP expression $\overline{A}BC + A\overline{B}C + A\overline{B}\overline{C}$ on a Karnaugh map.

Map the following standard SOP expression on a Karnaugh map:

$$\overline{A}\,\overline{B}CD + \overline{A}B\overline{C}\,\overline{D} + AB\overline{C}D + ABCD + AB\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + A\overline{B}C\overline{D}$$

Map the following standard SOP expression on a Karnaugh map:

$$\overline{A}BC\overline{D} + ABC\overline{D} + AB\overline{C}\,\overline{D} + ABCD$$

KARNAUGH MAP SOP MINIMIZATION

Mapping a Nonstandard SOP Expression

A Boolean expression must first be in standard form before you use a Karnaugh map. If an expression is not in standard form, then it must be converted to standard form by the procedure covered in Section 4–6 or by numerical expansion. Since an expression should be evaluated before mapping anyway, numerical expansion is probably the most efficient approach.

Numerical Expansion of a Nonstandard Product Term

Recall that a nonstandard product term has one or more missing variables. For example, assume that one of the product terms in a certain 3-variable SOP expression is AB. This term can be expanded numerically to standard form as follows. First, write the binary value of the two variables and attach a 0 for the missing variable C: 100. Next, write the binary value of the two variables and attach a 1 for the missing variable C: 101. The two resulting binary numbers are the values of the standard SOP terms AB C and ABC. As another example, assume that one of the product terms in a 3-variable expression is B (remember that a single variable counts as a product term in an SOP expression). This term can be expanded numerically to standard form as follows. Write the binary value of the variable; then attach all possible values for the missing variables A and C as follows:

B
010
011
110
111

KARNAUGH MAP SOP MINIMIZATION

Sample Problem

Map the following SOP expression on a Karnaugh map: $\overline{A} + A\overline{B} + AB\overline{C}$.

Map the SOP expression $BC + \overline{A}\,\overline{C}$ on a Karnaugh map.

Map the following SOP expression on a Karnaugh map:

$$\overline{B}\,\overline{C} + A\overline{B} + AB\overline{C} + A\overline{B}C\overline{D} + \overline{A}\,B\,\overline{C}D + A\overline{B}CD$$

Map the expression $A + \overline{C}D + AC\overline{D} + \overline{A}BC\overline{D}$ on a Karnaugh map.

KARNAUGH MAP SOP MINIMIZATION

Karnaugh Map Simplification of SOP Expressions
The process that results in an expression containing the fewest possible terms with the few est possible variables is called minimization. After an SOP expression has been mapped, a minimum SOP expression is obtained by grouping the 1s and determining the minimum SOP expression from the map.

Grouping the 1s

You can group 1s on the Karnaugh map according to the following rules by enclosing those adjacent cells containing 1s. The goal is to maximize the size of the groups and to minimize the number of groups.

1. A group must contain either 1, 2, 4, 8, or 16 cells, which are all powers of two. In the case of a 3-variable map, 23 = 8 cells is the maximum group.

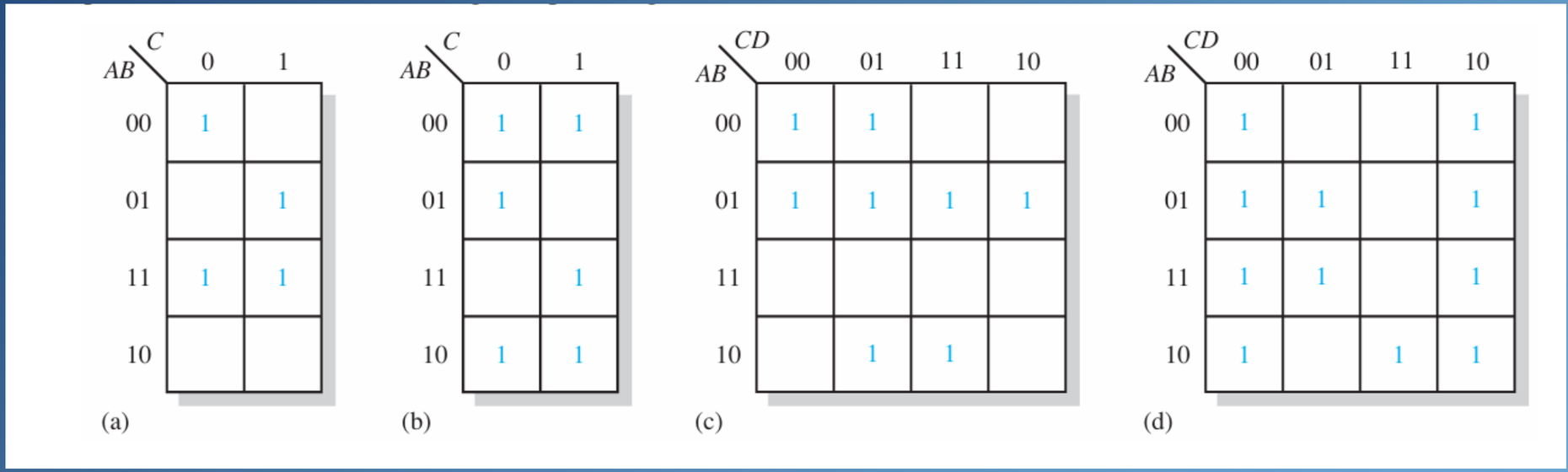
2. Each cell in a group must be adjacent to one or more cells in that same group, but all cells in the group do not have to be adjacent to each other.

3. Always include the largest possible number of 1s in a group in accordance with rule 1.

4. Each 1 on the map must be included in at least one group. The 1s already in a group can be included in another group as long as the overlapping groups include noncommon 1s.

KARNAUGH MAP SOP MINIMIZATION

Sample Problem



AB \ C	0	1
00	1	
01		1
11	1	1
10		

(a)

AB \ C	0	1
00	1	1
01	1	
11		1
10	1	1

(b)

AB \ CD	00	01	11	10
00	1	1		
01	1	1	1	1
11				
10		1	1	

(c)

AB \ CD	00	01	11	10
00	1			1
01	1	1		1
11	1	1		1
10	1		1	1

(d)

KARNAUGH MAP SOP MINIMIZATION

Determining the Minimum SOP Expression from the Map
When all the 1s representing the standard product terms in an expression are properly mapped and grouped, the process of determining the resulting minimum SOP expression begins. The following rules are applied to find the minimum product terms and the mini mum SOP expression:

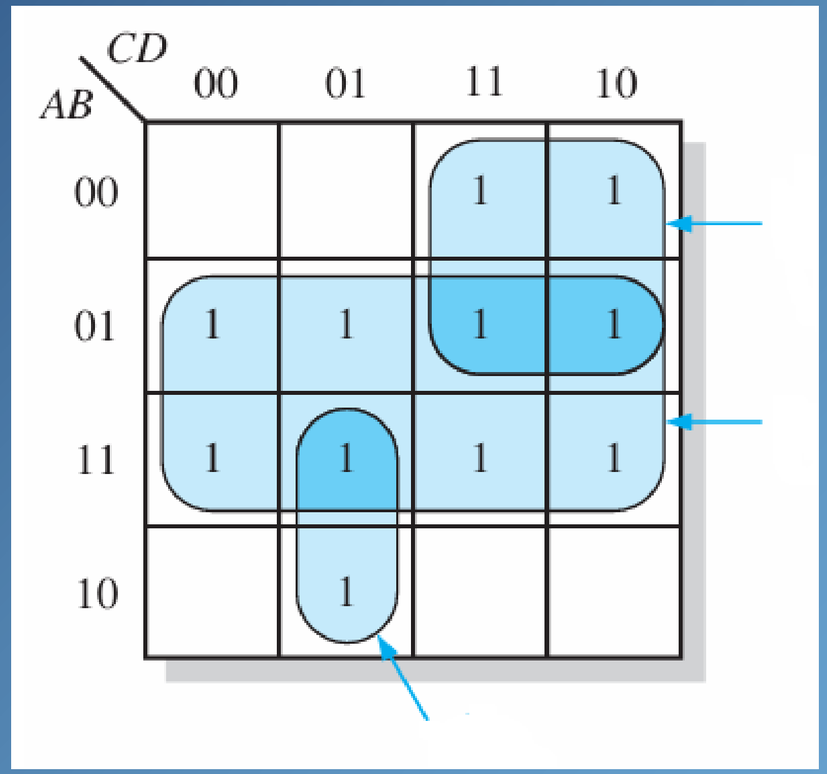
1. Group the cells that have 1s. Each group of cells containing 1s creates one product term composed of all variables that occur in only one form (either uncomplemented or complemented) within the group. Variables that occur both uncomplemented and complemented within the group are eliminated. These are called contradictory variables.

2. Determine the minimum product term for each group. (a) For a 3-variable map: (1) A 1-cell group yields a 3-variable product term (2) A 2-cell group yields a 2-variable product term (3) A 4-cell group yields a 1-variable term (4) An 8-cell group yields a value of 1 for the expression (b) For a 4-variable map: (1) A 1-cell group yields a 4-variable product term (2) A 2-cell group yields a 3-variable product term (3) A 4-cell group yields a 2-variable product term (4) An 8-cell group yields a 1-variable term (5) A 16-cell group yields a value of 1 for the expression

3. When all the minimum product terms are derived from the Karnaugh map, they are summed to form the minimum SOP expression.

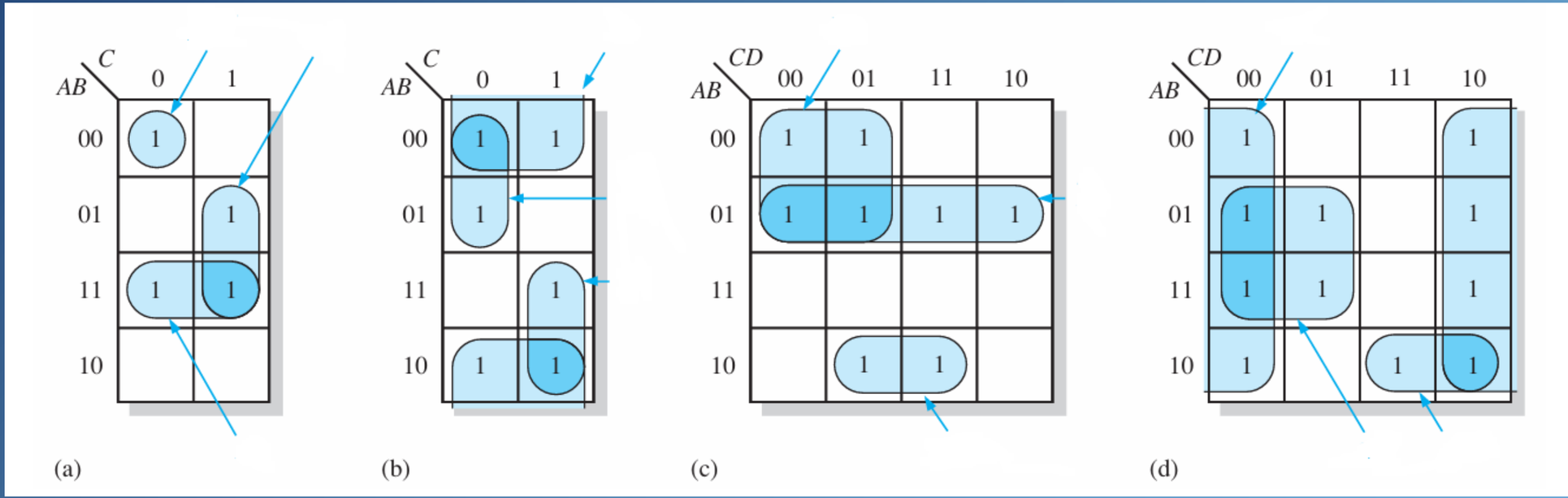
KARNAUGH MAP SOP MINIMIZATION

Sample Problem



KARNAUGH MAP SOP MINIMIZATION

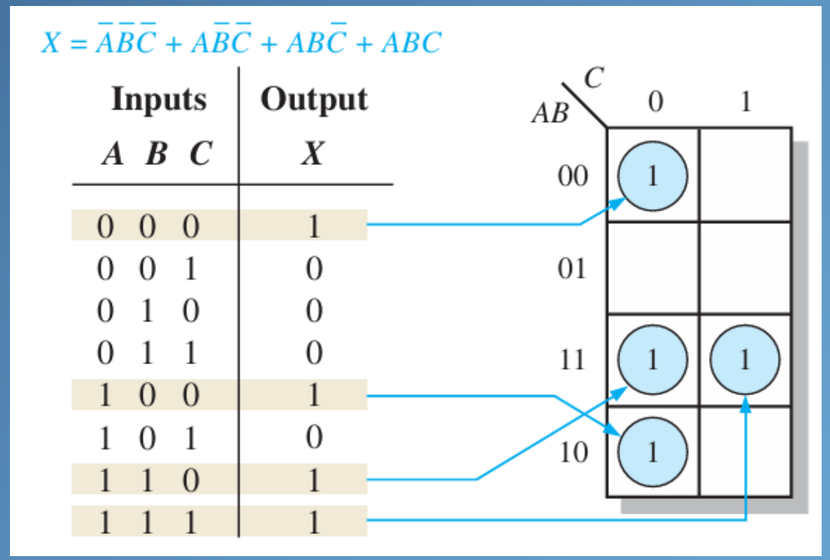
Sample Problem



KARNAUGH MAP SOP MINIMIZATION

Mapping Directly from a Truth Table

You have seen how to map a Boolean expression; now you will learn how to go directly from a truth table to a Karnaugh map. Recall that a truth table gives the output of a Boolean expression for all possible input variable combinations. An example of a Boolean expres sion and its truth table representation is shown in Figure 1.37. Notice in the truth table that the output X is 1 for four different input variable combinations. The 1s in the output column of the truth table are mapped directly onto a Karnaugh map into the cells corresponding to the values of the associated input variable combinations, as shown in Figure 1.37. In the figure you can see that the Boolean expression, the truth table, and the Karnaugh map are simply different ways to represent a logic function.



KARNAUGH MAP SOP MINIMIZATION

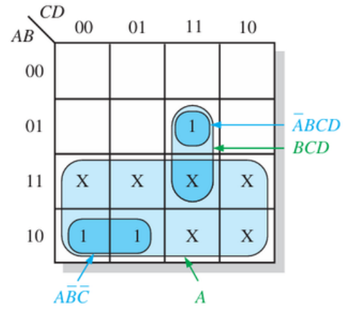
“Don’t Care” Conditions

Sometimes a situation arises in which some input variable combinations are not allowed. For example, recall that in the BCD code, there are six invalid combinations: 1010, 1011, 1100, 1101, 1110, and 1111. Since these unallowed states will never occur in an application involving the BCD code, they can be treated as “don’t care” terms with respect to their effect on the output. That is, for these “don’t care” terms either a 1 or a 0 may be assigned to the output; it really does not matter since they will never occur. The “don’t care” terms can be used to advantage on the Karnaugh map. Figure 1.38 shows that for each “don’t care” term, an X is placed in the cell. When grouping the 1s, the Xs can be treated as 1s to make a larger grouping or as 0s if they cannot be used to advantage. The larger a group, the simpler the resulting term will be. The truth table in Figure 1.38(a) describes a logic function that has a 1 output only when the BCD code for 7, 8, or 9 is present on the inputs. If the “don’t cares” are used as 1s, the resulting expression for the function is A+BCD, as indicated in part (b). If the “don’t cares” are not used as 1s, the resulting expression is AB C+ABCD; so you can see the advantage of using “don’t care” terms to get the simplest expression.

Inputs				Output
A	*B*	*C*	*D*	*Y*
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

(a) Truth table

Don’t cares

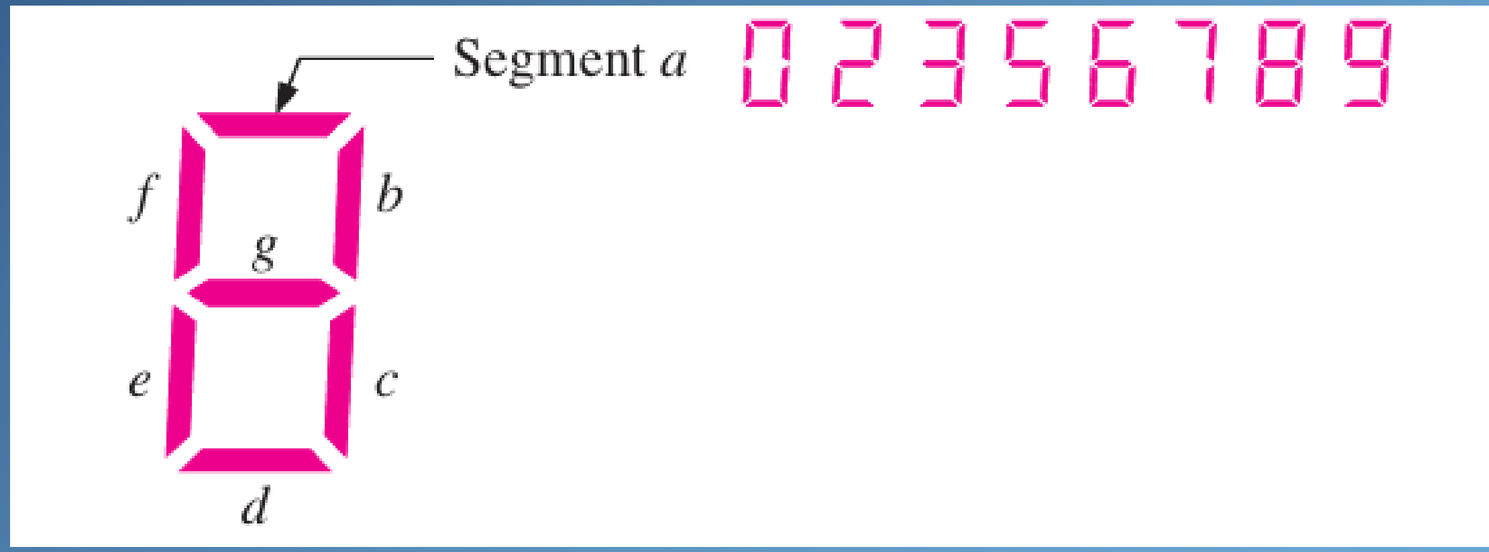


(b) Without “don’t cares” $Y = A\bar{B}\bar{C} + \bar{A}BCD$
With “don’t cares” $Y = A + BCD$

KARNAUGH MAP SOP MINIMIZATION

Sample Problem

In a 7-segment display, each of the seven segments is activated for various digits. For example, segment a is activated for the digits 0, 2, 3, 5, 6, 7, 8, and 9, as illustrated in the Figure below. Since each digit can be represented by a BCD code, derive an SOP expression for segment a using the variables ABCD and then minimize the expression using a Karnaugh map.



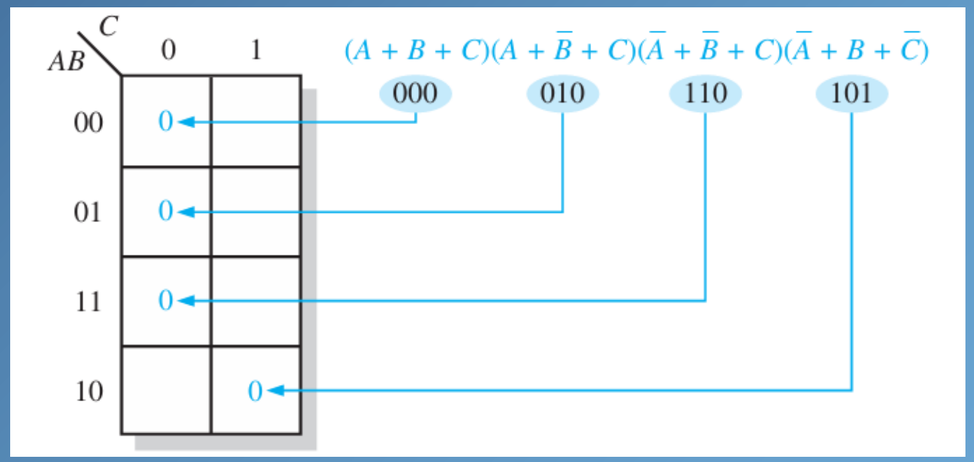
KARNAUGH MAP POS MINIMIZATION

Mapping a Standard POS Expression

For a POS expression in standard form, a 0 is placed on the Karnaugh map for each sum term in the expression. Each 0 is placed in a cell corresponding to the value of a sum term. For example, for the sum term A + B + C, a 0 goes in the 010 cell on a 3-variable map. When a POS expression is completely mapped, there will be a number of 0s on the Karnaugh map equal to the number of sum terms in the standard POS expression. The cells that do not have a 0 are the cells for which the expression is 1. Usually, when working with POS expressions, the 1s are left off. The following steps and the illustration in Figure 1.39 show the mapping process.

Step 1: Determine the binary value of each sum term in the standard POS expression. This is the binary value that makes the term equal to 0.

Step 2: As each sum term is evaluated, place a 0 on the Karnaugh map in the corresponding cell.



KARNAUGH MAP POS MINIMIZATION

Sample Problem

Map the following standard POS expression on a Karnaugh map:

$$(\overline{A} + \overline{B} + C + D)(\overline{A} + B + \overline{C} + \overline{D})(A + B + \overline{C} + D)(\overline{A} + \overline{B} + \overline{C} + \overline{D})(A + B + \overline{C} + \overline{D})$$

Map the following standard POS expression on a Karnaugh map:

$$(A + \overline{B} + \overline{C} + D)(A + B + C + \overline{D})(A + B + C + D)(\overline{A} + B + \overline{C} + D)$$

KARNAUGH MAP POS MINIMIZATION

Karnaugh Map Simplification of POS Expressions

The process for minimizing a POS expression is basically the same as for an SOP expression except that you group 0s to produce minimum sum terms instead of grouping 1s to produce minimum product terms. The rules for grouping the 0s are the same as those for grouping the 1s that you learned in the previous discussion.

KARNAUGH MAP POS MINIMIZATION

Sample Problem

Use a Karnaugh map to minimize the following standard POS expression:

$$(A + B + C)(A + B + \overline{C})(A + \overline{B} + C)(A + \overline{B} + \overline{C})(\overline{A} + \overline{B} + C)$$

Use a Karnaugh map to simplify the following standard POS expression:

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

Use a Karnaugh map to minimize the following POS expression:

$$(B + C + D)(A + B + \overline{C} + D)(\overline{A} + B + C + \overline{D})(A + \overline{B} + C + D)(\overline{A} + \overline{B} + C + D)$$

KARNAUGH MAP POS MINIMIZATION

Converting Between POS and SOP Using the Karnaugh Map

When a POS expression is mapped, it can easily be converted to the equivalent SOP form directly from the Karnaugh map. Also, given a mapped SOP expression, an equivalent POS expression can be derived directly from the map. This provides a good way to compare both minimum forms of an expression to determine if one of them can be implemented with fewer gates than the other. For a POS expression, all the cells that do not contain 0s contain 1s, from which the SOP expression is derived. Likewise, for an SOP expression, all the cells that do not contain 1s contain 0s, from which the POS expression is derived. Example 4–36 illustrates this conversion.

KARNAUGH MAP POS MINIMIZATION

Sample Problem

Using a Karnaugh map, convert the following standard POS expression into a minimum POS expression, a standard SOP expression, and a minimum SOP expression.

$$(\overline{A} + \overline{B} + C + D)(A + \overline{B} + C + D)(A + B + C + \overline{D})(A + B + \overline{C} + \overline{D})(\overline{A} + B + C + \overline{D})(A + B + \overline{C} + D)$$

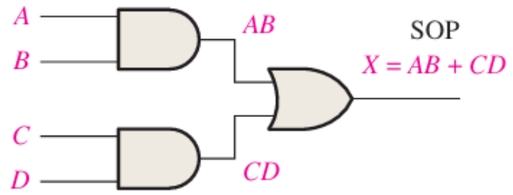
Use a Karnaugh map to convert the following expression to minimum SOP form:

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

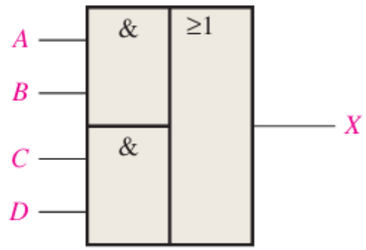
BASIC COMBINATIONAL LOGIC CIRCUITS

AND-OR Logic

Figure 1.39(a) shows an AND-OR circuit consisting of two 2-input AND gates and one 2-input OR gate; Figure 1.39(b) is the ANSI standard rectangular outline symbol. The Boolean expressions for the AND gate outputs and the resulting SOP expression for the output X are shown on the diagram. In general, an AND-OR circuit can have any number of AND gates, each with any number of inputs. The truth table for a 4-input AND-OR logic circuit is shown in Table 1.15. The intermediate AND gate outputs (the AB and CD columns) are also shown in the table.



(a) Logic diagram (ANSI standard distinctive shape symbols)



(b) ANSI standard rectangular outline symbol

Inputs						Output
A	*B*	*C*	*D*	*AB*	*CD*	*X*
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	1	1
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	1	1	0	1	1
1	0	0	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	0	0	0
1	0	1	1	0	1	1
1	1	0	0	1	0	1
1	1	0	1	1	0	1
1	1	1	0	1	0	1
1	1	1	1	1	1	1

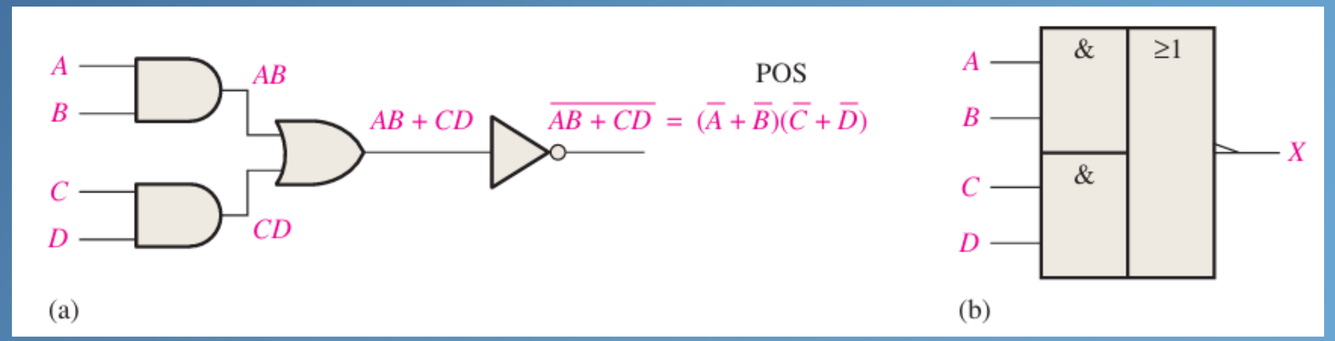
BASIC COMBINATIONAL LOGIC CIRCUITS

AND-OR-Invert Logic

When the output of an AND-OR circuit is complemented (inverted), it results in an AND-OR-Invert circuit. Recall that AND-OR logic directly implements SOP expressions. POS expressions can be implemented with AND-OR-Invert logic. This is illustrated as follows, starting with a POS expression and developing the corresponding AND-OR-Invert (AOI) expression.

$$X = (\overline{A} + \overline{B})(\overline{C} + \overline{D}) = (\overline{AB})(\overline{CD}) = \overline{(\overline{AB})(\overline{CD})} = \overline{\overline{AB}} + \overline{\overline{CD}} = \overline{AB + CD}$$

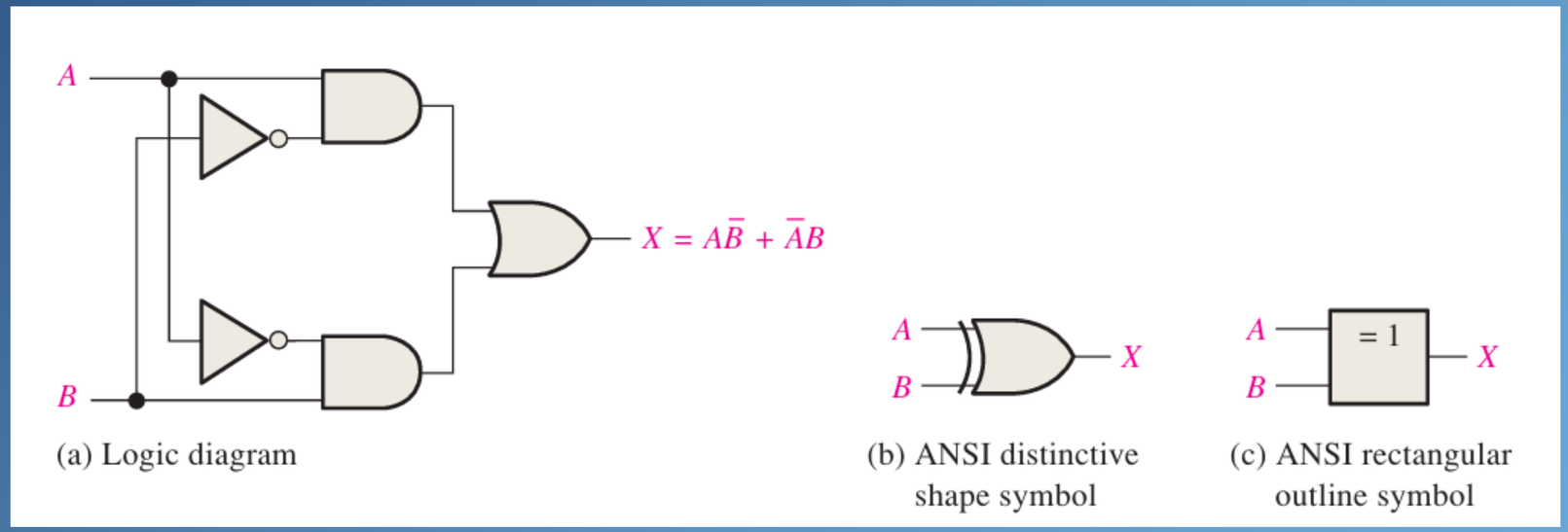
The logic diagram in Figure 1.40(a) shows an AND-OR-Invert circuit with four inputs and the development of the POS output expression. The ANSI standard rectangular outline symbol is shown in part (b). In general, an AND-OR-Invert circuit can have any number of AND gates, each with any number of inputs.



BASIC COMBINATIONAL LOGIC CIRCUITS

Exclusive-OR Logic

The exclusive-OR gate was introduced in the previous discussion. Although this circuit is considered a type of logic gate with its own unique symbol, it is actually a combination of two AND gates, one OR gate, and two inverters, as shown in Figure 1.41(a). The two ANSI standard exclusive-OR logic symbols are shown in parts (b) and (c).



(a) Logic diagram

(b) ANSI distinctive shape symbol

(c) ANSI rectangular outline symbol

BASIC COMBINATIONAL LOGIC CIRCUITS

Exclusive-OR Logic

The output expression for the circuit in Figure 1.41 is

$$X = A\overline{B} + \overline{A}B$$

Evaluation of this expression results in the truth table in Table 1.16. Notice that the output is HIGH only when the two inputs are at opposite levels. A special exclusive-OR operator $\oplus$ is often used, so the expression X = AB + AB can be stated as “X is equal to A exclusive-OR B” and can be written as

$$X = A \oplus B$$

Truth table for an exclusive-OR.

A	*B*	*X*
0	0	0
0	1	1
1	0	1
1	1	0

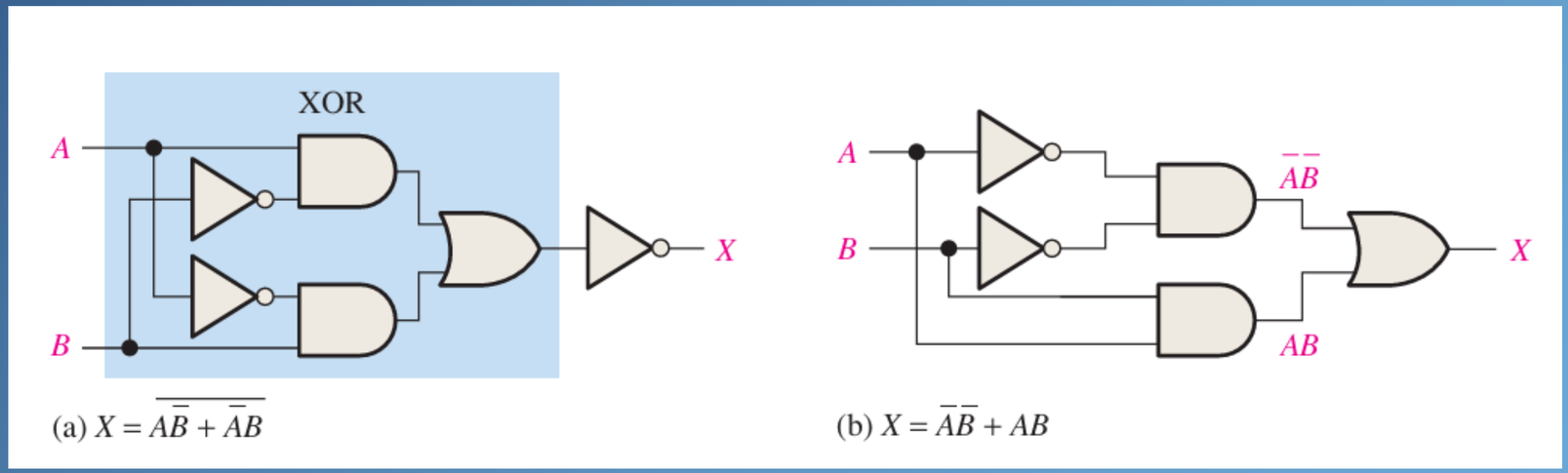
BASIC COMBINATIONAL LOGIC CIRCUITS

Exclusive-NOR Logic

As you know, the complement of the exclusive-OR function is the exclusive-NOR, which is derived as follows:

$$X = \overline{A\overline{B} + \overline{A}B} = \overline{(A\overline{B})}\,\overline{(\overline{A}B)} = (\overline{A} + B)(A + \overline{B}) = \overline{A}\,\overline{B} + AB$$

Notice that the output X is HIGH only when the two inputs, A and B, are at the same level. The exclusive-NOR can be implemented by simply inverting the output of an exclusive OR, as shown in Figure 1.42(a), or by directly implementing the expression A B + AB, as shown in part (b).



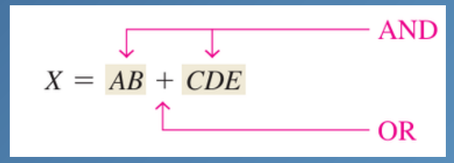
IMPLEMENTING COMBINATIONAL LOGIC

From a Boolean Expression to a Logic Circuit

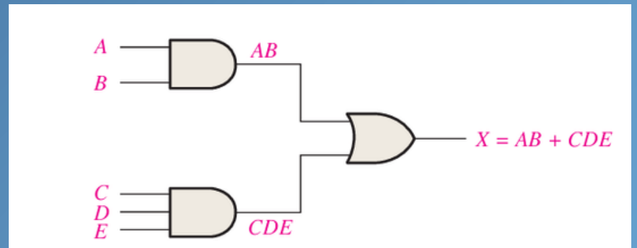
As you know, the complement of the exclusive-OR function is the exclusive-NOR, which is derived as follows:

$$X = AB + CDE$$

A brief inspection shows that this expression is composed of two terms, AB and CDE, with a domain of five variables. The first term is formed by ANDing A with B, and the second term is formed by ANDing C, D, and E. The two terms are then ORed to form the output X. These operations are indicated in the structure of the expression as follows:



Note that in this particular expression, the AND operations forming the two individual terms, AB and CDE, must be performed before the terms can be ORed. To implement this Boolean expression, a 2-input AND gate is required to form the term AB, and a 3-input AND gate is needed to form the term CDE. A 2-input OR gate is then required to combine the two AND terms. The resulting logic circuit is shown in Figure 1.43.



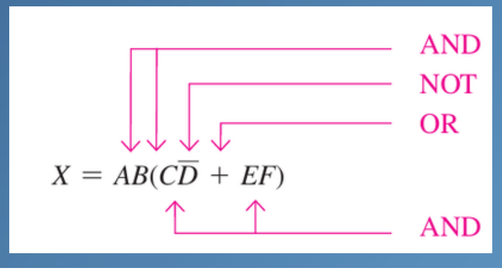
Logic circuit for $X = AB + CDE$.

IMPLEMENTING COMBINATIONAL LOGIC

As another example, let's implement the following expression:

$$X = AB(C\overline{D} + EF)$$

A breakdown of this expression shows that the terms AB and (CD + EF) are ANDed. The term CD + EF is formed by first ANDing C and D and ANDing E and F, and then ORing these two terms. This structure is indicated in relation to the expression as follows:



Before you can implement the final expression, you must create the sum term $C\overline{D} + EF$; but before you can get this term; you must create the product terms $C\overline{D}$ and EF; but before you can get the term $C\overline{D}$, you must create $\overline{D}$. So, as you can see, the logic operations must be done in the proper order.

The logic gates required to implement $X = AB(C\overline{D} + EF)$ are as follows:

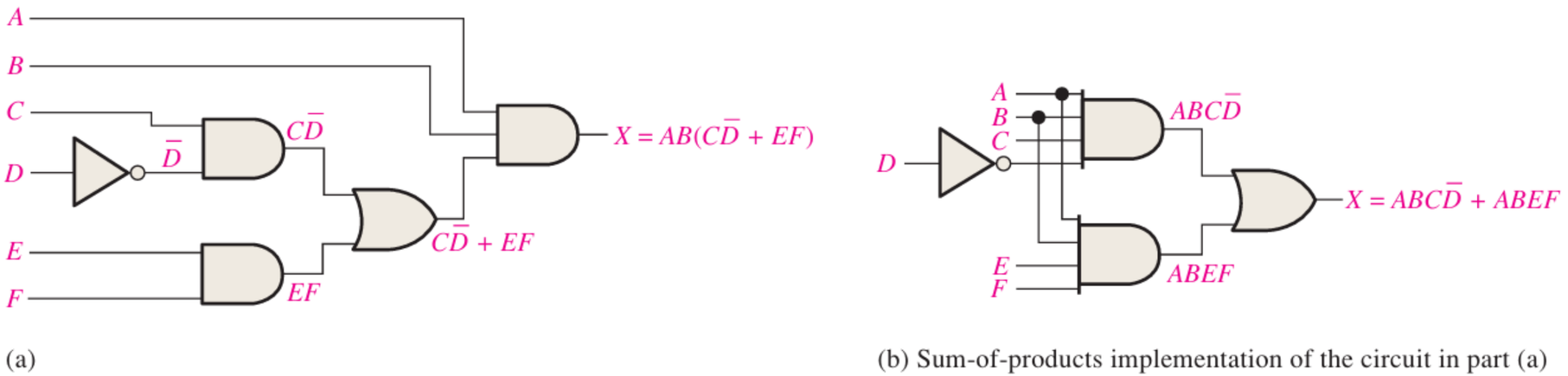
1. One inverter to form $\overline{D}$
2. Two 2-input AND gates to form $C\overline{D}$ and EF
3. One 2-input OR gate to form $C\overline{D} + EF$
4. One 3-input AND gate to form X

IMPLEMENTING COMBINATIONAL LOGIC

The logic circuit for this expression is shown in Figure 5–10(a). Notice that there is a maximum of four gates and an inverter between an input and output in this circuit (from input D to output). Often the total propagation delay time through a logic circuit is a major consideration. Propagation delays are additive, so the more gates or inverters between input and output, the greater the propagation delay time.

Unless an intermediate term, such as $C\overline{D} + EF$ in Figure 5–10(a), is required as an output for some other purpose, it is usually best to reduce a circuit to its SOP form in order to reduce the overall propagation delay time. The expression is converted to SOP as follows, and the resulting circuit is shown in Figure 5–10(b).

$$AB(C\overline{D} + EF) = ABC\overline{D} + ABEF$$



(a)

(b) Sum-of-products implementation of the circuit in part (a)

FIGURE 5–10 Logic circuits for $X = AB(C\overline{D} + EF) = ABC\overline{D} + ABEF$.

IMPLEMENTING COMBINATIONAL LOGIC

From a Truth Table to a Logic Circuit

If you begin with a truth table instead of an expression, you can write the SOP expression from the truth table and then implement the logic circuit. Table 1.16 specifies a logic function. The Boolean SOP expression obtained from the truth table by ORing the product terms for which X = 1 is

$$X = \overline{A}BC + A\overline{B}\,\overline{C}$$

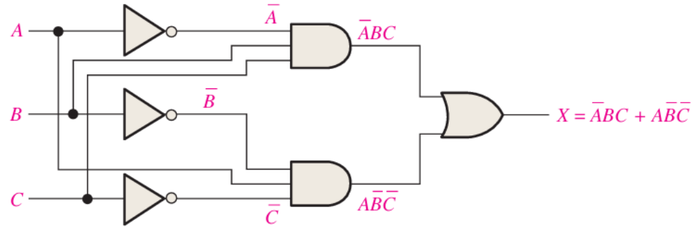
Inputs			Output	
A	*B*	*C*	*X*	Product Term
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\overline{A}BC$
1	0	0	1	$A\overline{B}\,\overline{C}$
1	0	1	0	
1	1	0	0	
1	1	1	0	

If you begin with a truth table instead of an expression, you can write the SOP expression from the truth table and then implement the logic circuit. Table 1.16 specifies a logic function. The Boolean SOP expression obtained from the truth table by ORing the product terms for which X = 1 is

The first term in the expression is formed by ANDing the three variables $\overline{A}$, B, and C. The second term is formed by ANDing the three variables A, $\overline{B}$, and $\overline{C}$.

The logic gates required to implement this expression are as follows: three inverters to form the $\overline{A}$, $\overline{B}$, and $\overline{C}$ variables; two 3-input AND gates to form the terms $\overline{A}BC$ and $A\overline{B}\,\overline{C}$; and one 2-input OR gate to form the final output function, $\overline{A}BC + A\overline{B}\,\overline{C}$.

The implementation of this logic function is illustrated in Figure 5–11.



IMPLEMENTING COMBINATIONAL LOGIC

Sample Problem

Design a logic circuit to implement the operation specified in the truth table of the Table below.

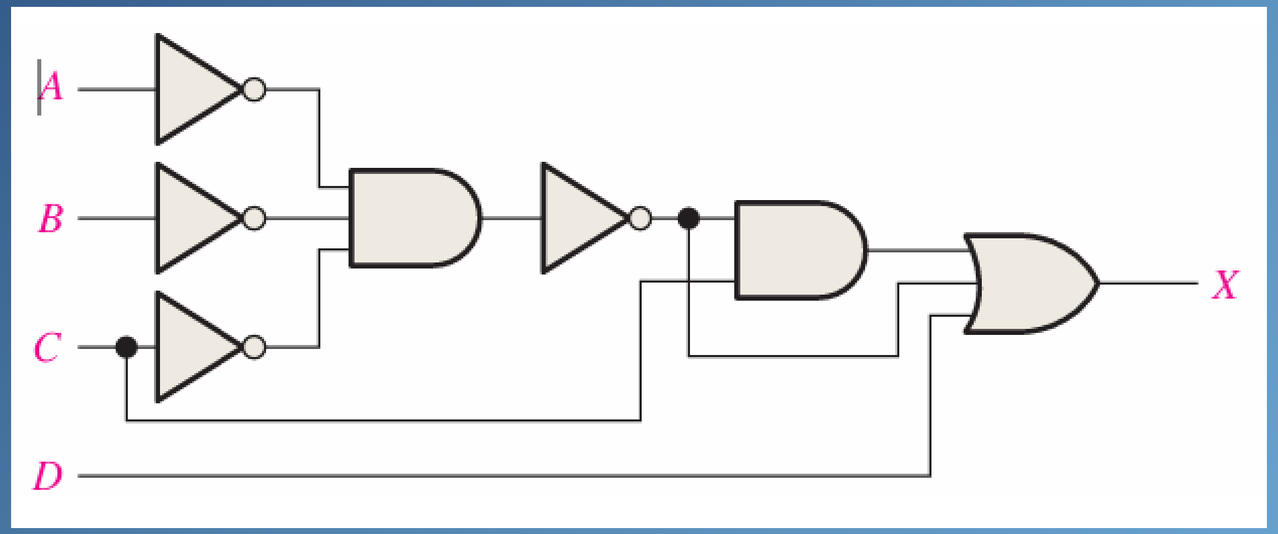
Inputs			Output	
A	*B*	*C*	*X*	Product Term
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\overline{A}BC$
1	0	0	0	
1	0	1	1	$A\overline{B}C$
1	1	0	1	$AB\overline{C}$
1	1	1	0	

Develop a logic circuit with four input variables that will only produce a 1 output when exactly three input variables are 1s.

IMPLEMENTING COMBINATIONAL LOGIC

Sample Problem

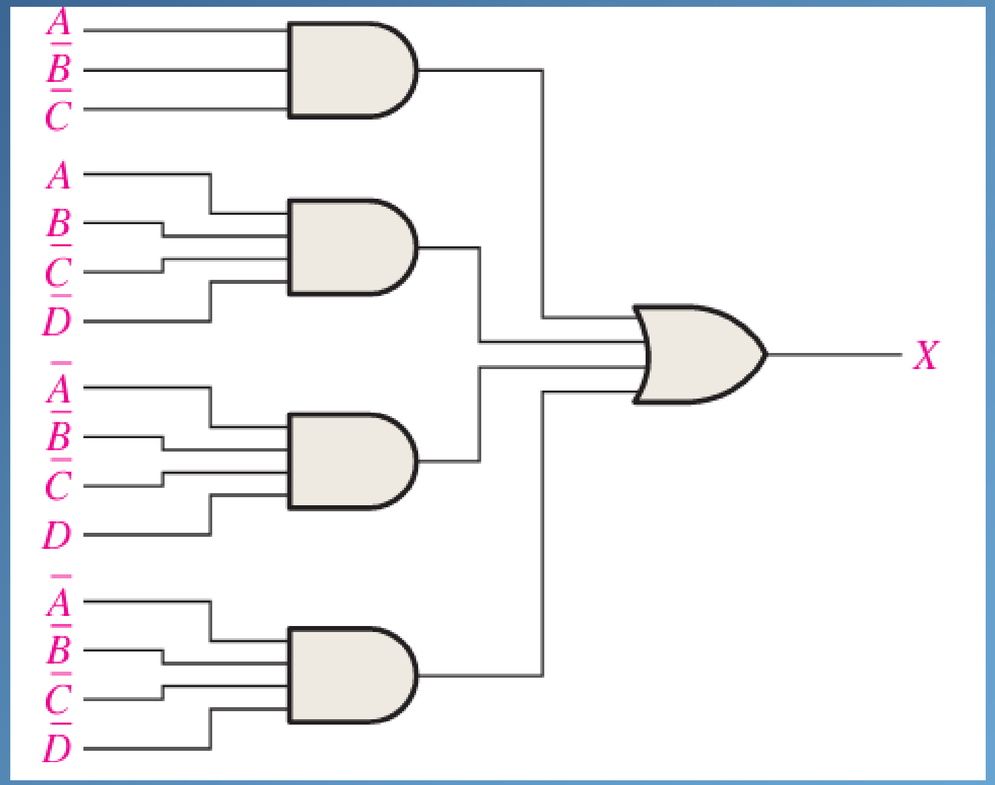
Reduce the combinational logic circuit in the Figure below to a minimum form.



IMPLEMENTING COMBINATIONAL LOGIC

Sample Problem

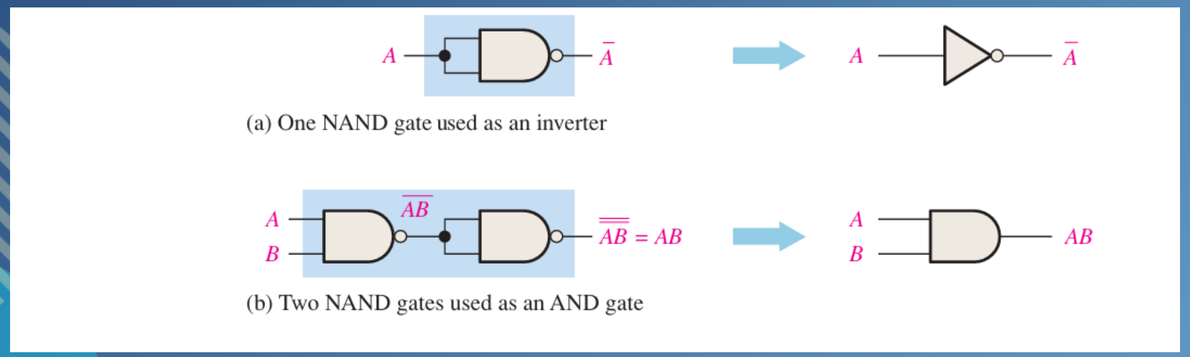
Minimize the combinational logic circuit in the Figure below. Inverters for the complemented variables are not shown.



THE UNIVERSAL PROPERTY OF NAND AND NOR GATES

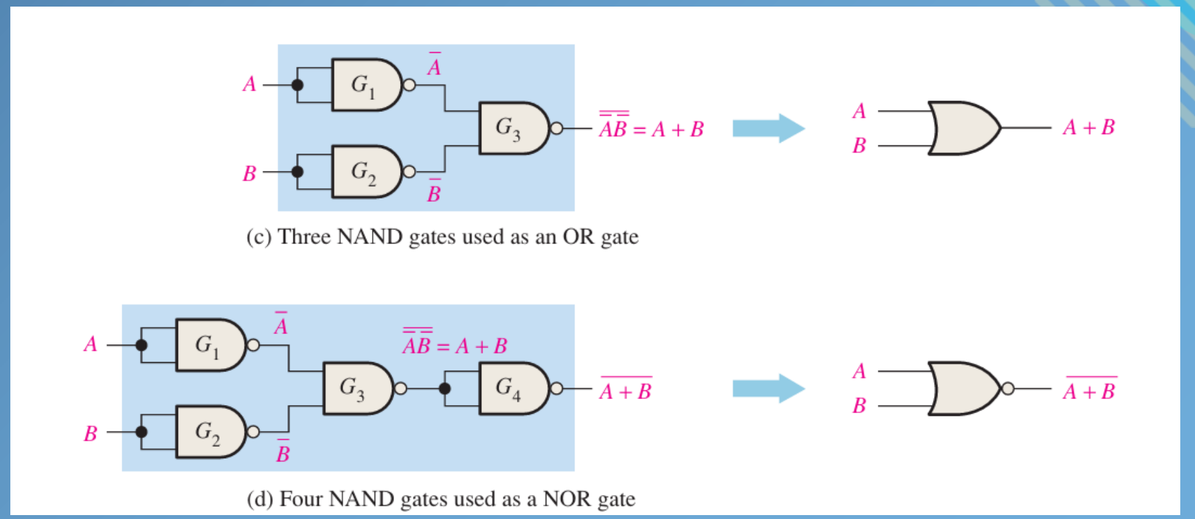
The NAND Gate as a Universal Logic Element

The NAND gate is a universal gate because it can be used to produce the NOT, the AND, the OR, and the NOR functions. An inverter can be made from a NAND gate by connecting all of the inputs together and creating, in effect, a single input, as shown in Figure 1.44(a) for a 2-input gate. An AND function can be generated by the use of NAND gates alone, as shown in Figure 1.44(b). An OR function can be produced with only NAND gates, as illustrated in part (c). Finally, a NOR function is produced as shown in part (d).



(a) One NAND gate used as an inverter

(b) Two NAND gates used as an AND gate



(c) Three NAND gates used as an OR gate

(d) Four NAND gates used as a NOR gate

THE UNIVERSAL PROPERTY OF NAND AND NOR GATES

In Figure 1.44(b), a NAND gate is used to invert (complement) a NAND output to form the AND function, as indicated in the following equation:

$$X = \overline{\overline{AB}} = AB$$

In Figure 1.44(c), NAND gates G1 and G2 are used to invert the two input variables before they are applied to NAND gate G3. The final OR output is derived as follows by application of DeMorgan's theorem:

$$X = \overline{\overline{A}\,\overline{B}} = A + B$$

In Figure 1.44(d), NAND gate G4 is used as an inverter connected to the circuit of part (c) to produce the NOR operation

$$\overline{A + B}.$$

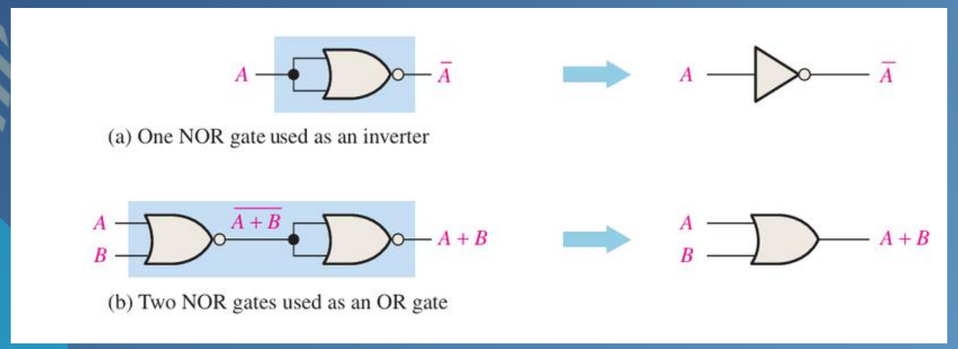
THE UNIVERSAL PROPERTY OF NAND AND NOR GATES

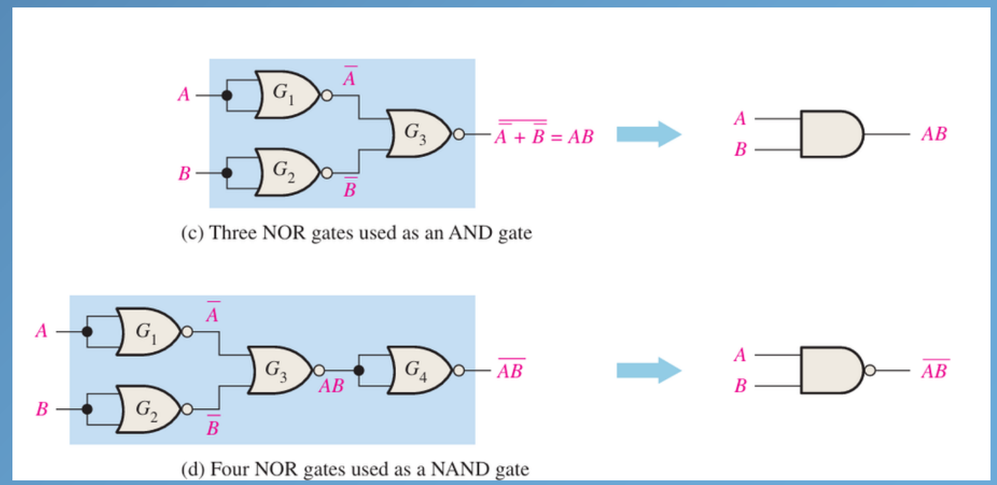
The NOR Gate as a Universal Logic Element

Like the NAND gate, the NOR gate can be used to produce the NOT, AND, OR, and NAND functions. A NOT circuit, or inverter, can be made from a NOR gate by connecting all of the inputs together to effectively create a single input, as shown in Figure 1.45(a) with a 2-input example. Also, an OR gate can be produced from NOR gates, as illustrated in Figure 1.45(b). An AND gate can be constructed by the use of NOR gates, as shown in Figure 1.45(c). In this case the NOR gates G1 and G2 are used as inverters, and the final output is derived by the use of DeMorgan's theorem as follows:

$$X = \overline{\overline{A} + \overline{B}} = AB$$

Figure 1.45(d) shows how NOR gates are used to form a NAND function.





COMBINATIONAL LOGIC USING NAND AND NOR GATES

NAND Logic

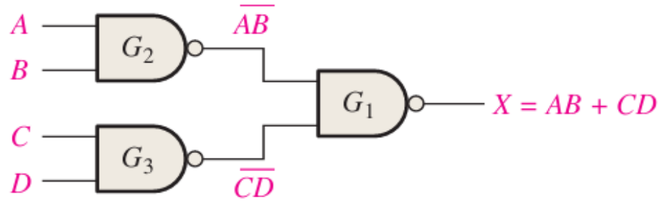
As you have learned, a NAND gate can function as either a NAND or a negative-OR because, by DeMorgan's theorem,

$$\overline{AB} = \overline{A} + \overline{B}$$

NAND — negative-OR

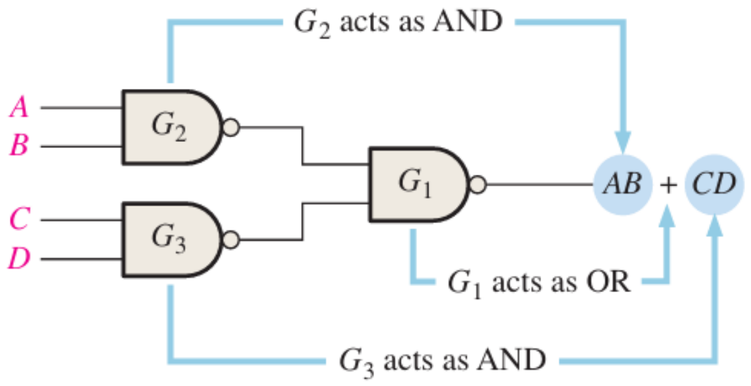
Consider the NAND logic in Figure 1.46. The output expression is developed in the following steps:

$$\begin{aligned} X &= \overline{(\overline{AB})(\overline{CD})} \\ &= \overline{(\overline{A} + \overline{B})(\overline{C} + \overline{D})} \\ &= (\overline{\overline{A} + \overline{B}}) + (\overline{\overline{C} + \overline{D}}) \\ &= \overline{\overline{A}}\,\overline{\overline{B}} + \overline{\overline{C}}\,\overline{\overline{D}} \\ &= AB + CD \end{aligned}$$

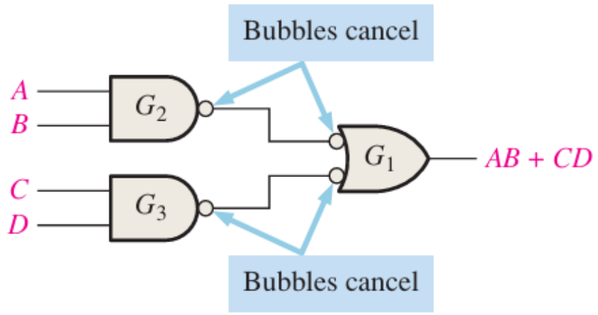


COMBINATIONAL LOGIC USING NAND AND NOR GATES

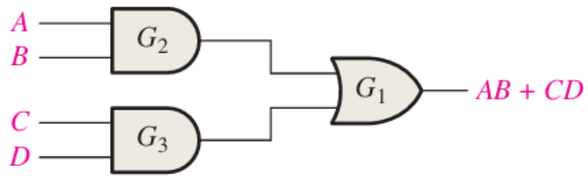
As you can see in Figure 1.46, the output expression, AB + CD, is in the form of two AND terms ORed together. This shows that gates G2 and G3 act as AND gates and that gate G1 acts as an OR gate, as illustrated in Figure 1.47(a). This circuit is redrawn in part (b) with NAND symbols for gates G2 and G3 and a negative-OR symbol for gate G1. Notice in Figure 1.47(b) the bubble-to-bubble connections between the outputs of gates G2 and G3 and the inputs of gate G1. Since a bubble represents an inversion, two connected bubbles represent a double inversion and therefore cancel each other. This inversion cancellation can be seen in the previous development of the output expression AB + CD and is indicated by the absence of barred terms in the output expression. Thus, the circuit in Figure 5–21(b) is effectively an AND-OR circuit, as shown in Figure 1.47(c).



(a) Original NAND logic diagram showing effective gate operation relative to the output expression



(b) Equivalent NAND/Negative-OR logic diagram



(c) AND-OR equivalent

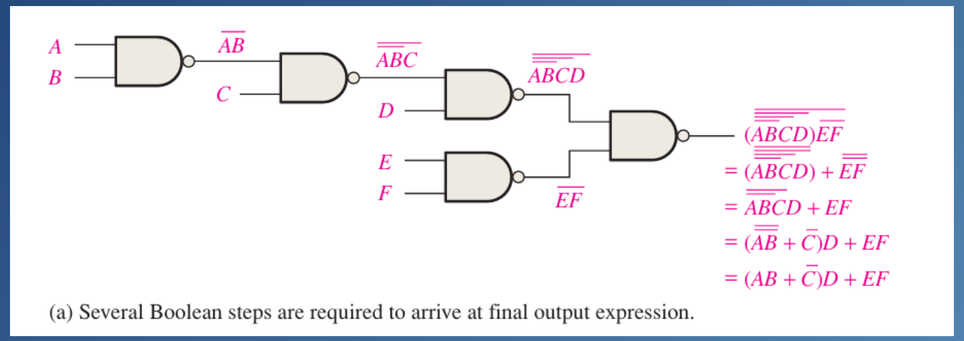
COMBINATIONAL LOGIC USING NAND AND NOR GATES

NAND Logic Diagrams Using Dual Symbols

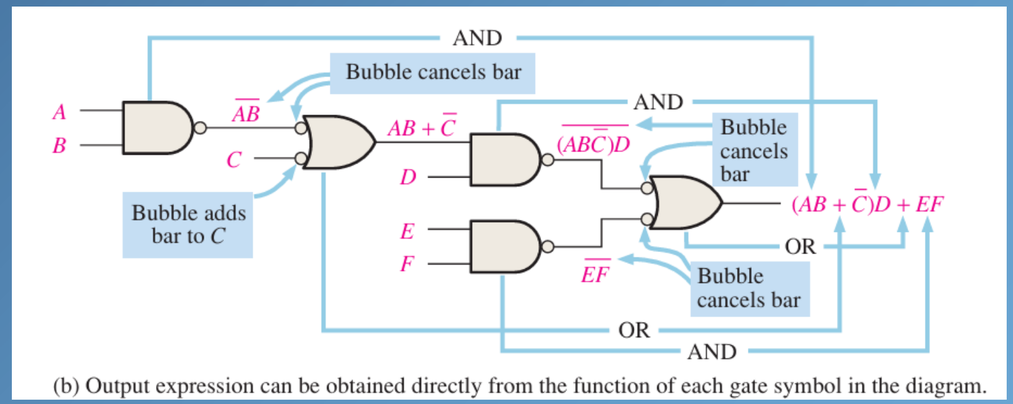
All logic diagrams using NAND gates should be drawn with each gate represented by either a NAND symbol or the equivalent negative-OR symbol to reflect the operation of the gate within the logic circuit. The NAND symbol and the negative-OR symbol are called dual symbols. When drawing a NAND logic diagram, always use the gate symbols in such a way that every connection between a gate output and a gate input is either bubble-to bubble or nonbubble-to-nonbubble. In general, a bubble output should not be connected to a nonbubble input or vice versa in a logic diagram. Figure 1.48 shows an arrangement of gates to illustrate the procedure of using the appropriate dual symbols for a NAND circuit with several gate levels. Although using all NAND symbols as in Figure 1.48(a) is correct, the diagram in part (b) is much easier to “read” and is the preferred method. As shown in Figure 1.48(b), the output gate is represented with a negative-OR symbol. Then the NAND symbol is used for the level of gates right before the output gate and the symbols for successive levels of gates are alternated as you move away from the output.

COMBINATIONAL LOGIC USING NAND AND NOR GATES

NAND Logic Diagrams Using Dual Symbols



(a) Several Boolean steps are required to arrive at final output expression.



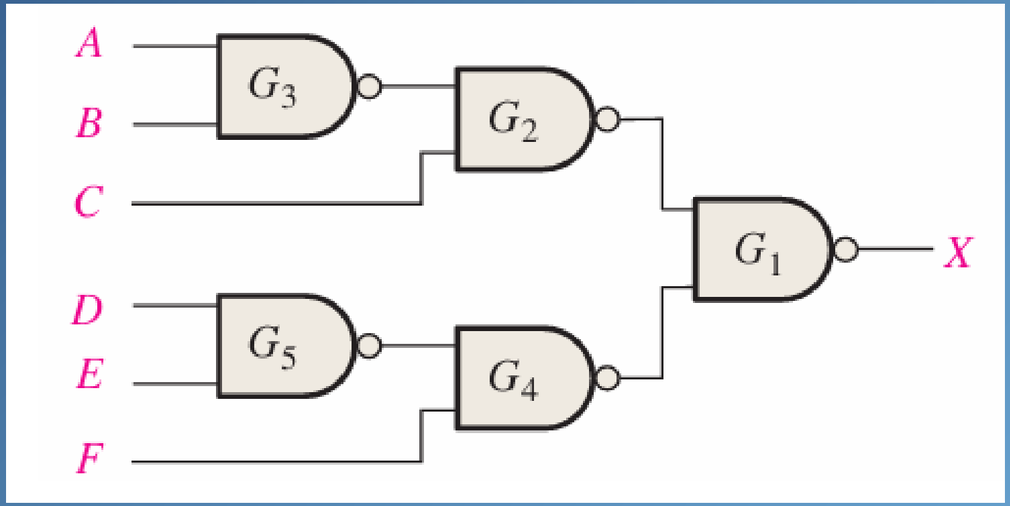
(b) Output expression can be obtained directly from the function of each gate symbol in the diagram.

The shape of the gate indicates the way its inputs will appear in the output expression and thus shows how the gate functions within the logic circuit. For a NAND symbol, the inputs appear ANDed in the output expression; and for a negative-OR symbol, the inputs appear ORed in the output expression, as Figure 1.48(b) illustrates. The dual-symbol dia gram in part (b) makes it easier to determine the output expression directly from the logic diagram because each gate symbol indicates the relationship of its input variables as they appear in the output expression.

IMPLEMENTING COMBINATIONAL LOGIC

Sample Problem

Redraw the logic diagram and develop the output expression for the circuit in the Figure below using the appropriate dual symbols.



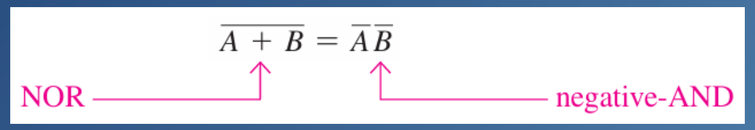
Implement each expression with NAND logic using appropriate dual symbols:

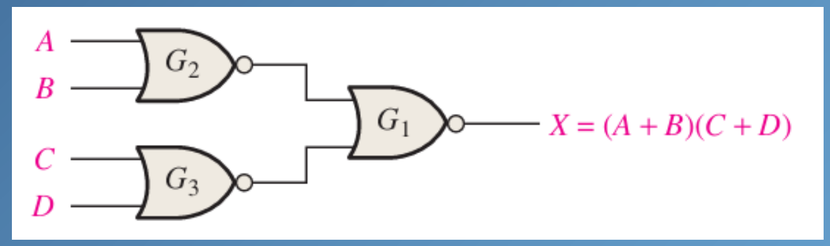
(a) $ABC + DE$ **(b)** $ABC + \overline{D} + \overline{E}$

COMBINATIONAL LOGIC USING NAND AND NOR GATES

NOR Logic

A NOR gate can function as either a NOR or a negative-AND, as shown by DeMorgan's theorem.

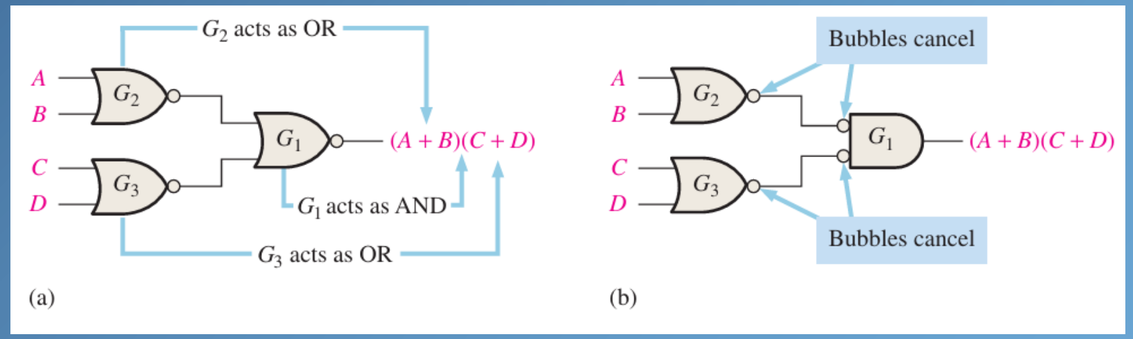




Consider the NOR logic in Figure 1.49. The output expression is developed as follows:

$$X = \overline{\overline{A + B} + \overline{C + D}} = (\overline{\overline{A + B}})(\overline{\overline{C + D}}) = (A + B)C + D)$$

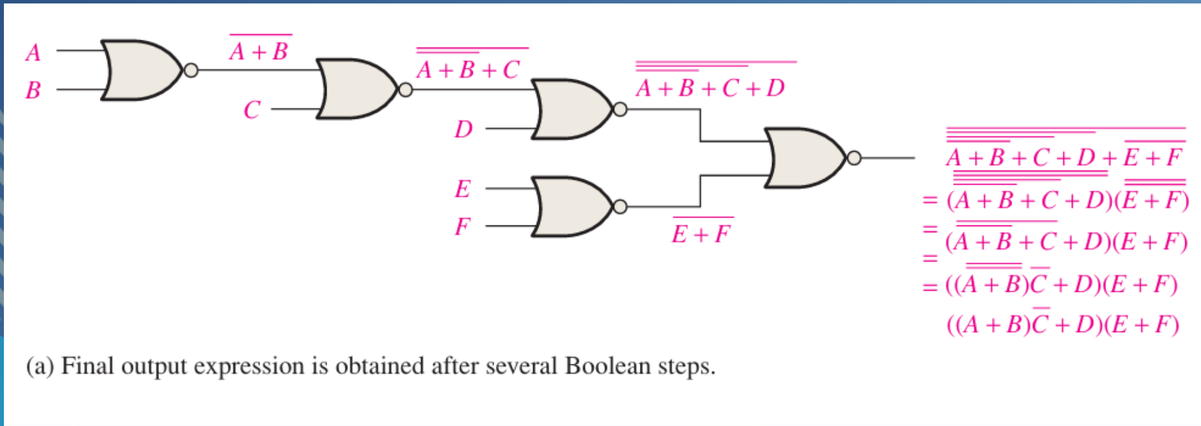
As you can see in Figure 1.49, the output expression (A + B)(C + D) consists of two OR terms ANDed together. This shows that gates G2 and G3 act as OR gates and gate G1 acts as an AND gate, as illustrated in Figure 1.50(a). This circuit is redrawn in part (b) with a negative-AND symbol for gate G1.



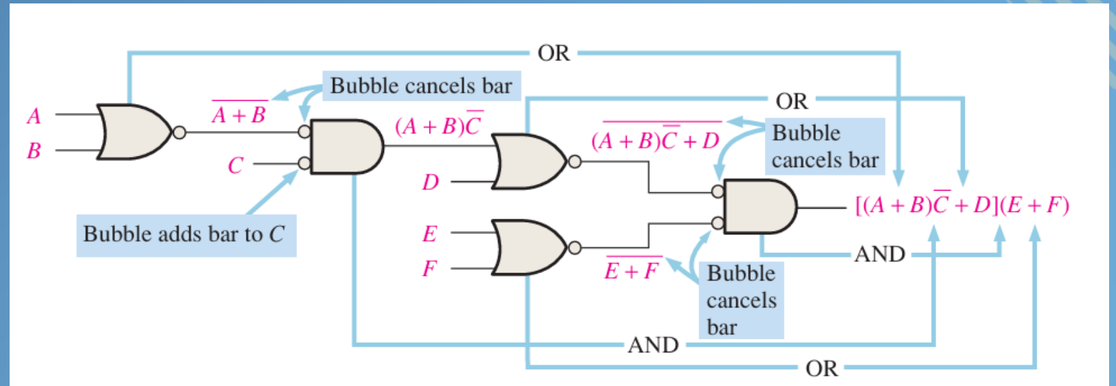
COMBINATIONAL LOGIC USING NAND AND NOR GATES

NOR Logic Diagram Using Dual Symbols

As with NAND logic, the purpose for using the dual symbols is to make the logic diagram easier to read and analyze, as illustrated in the NOR logic circuit in Figure 1.51. When the circuit in part (a) is redrawn with dual symbols in part (b), notice that all output-to-input connections between gates are bubble-to-bubble or nonbubble-to-nonbubble. Again, you can see that the shape of each gate symbol indicates the type of term (AND or OR) that it produces in the output expression, thus making the output expression easier to determine and the logic diagram easier to analyze.



(a) Final output expression is obtained after several Boolean steps.

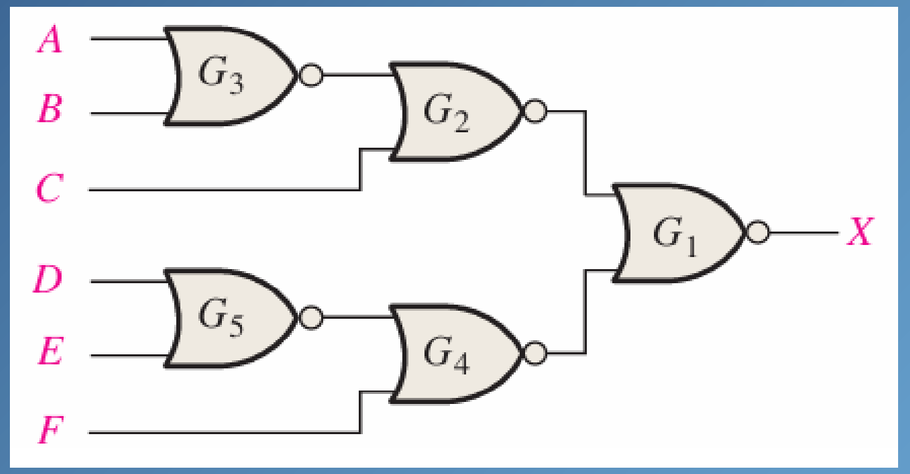


(b) Output expression can be obtained directly from the function of each gate symbol in the diagram.

IMPLEMENTING COMBINATIONAL LOGIC

Sample Problem

Using appropriate dual symbols, redraw the logic diagram and develop the output expression for the circuit in the Figure below.



PULSE WAVEFORM OPERATION

The operation of any gate is the same regardless of whether its inputs are pulsed or constant levels. The nature of the inputs (pulsed or constant levels) does not alter the truth table of a circuit. The examples in this section illustrate the analysis of combinational logic circuits with pulse waveform inputs.

The following is a review of the operation of individual gates for use in analyzing com binational circuits with pulse waveform inputs:

1. The output of an AND gate is HIGH only when all inputs are HIGH at the same time.

2. The output of an OR gate is HIGH only when at least one of its inputs is HIGH.

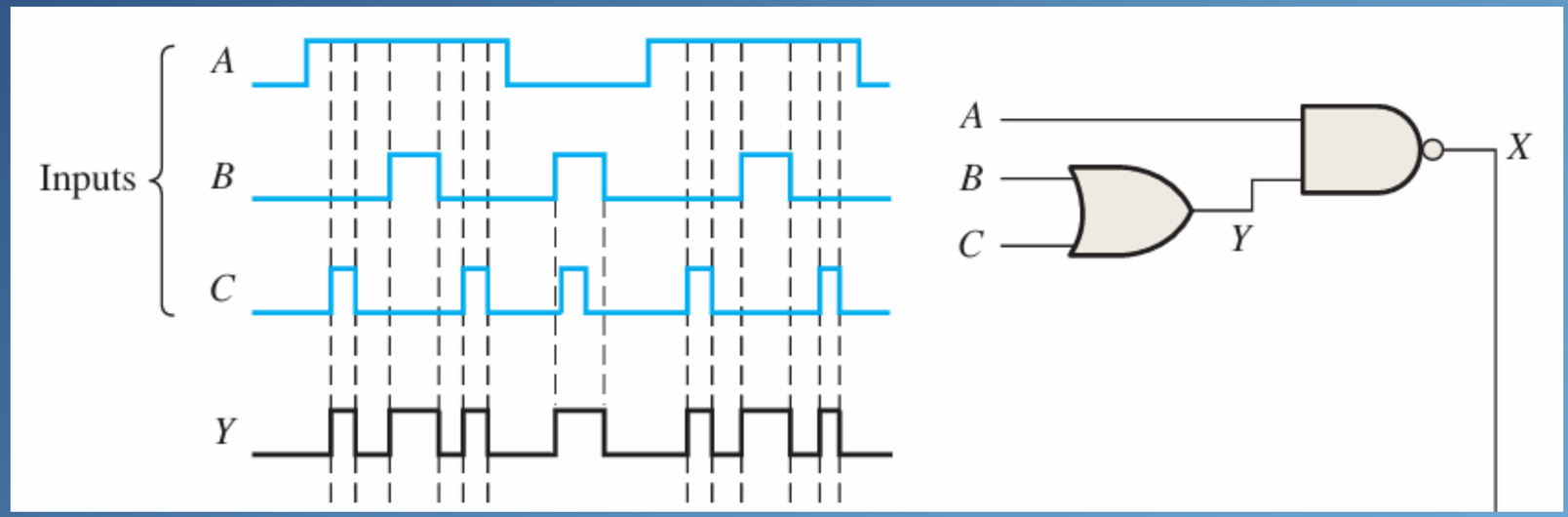
3. The output of a NAND gate is LOW only when all inputs are HIGH at the same time.

4. The output of a NOR gate is LOW only when at least one of its inputs is HIGH.

PULSE WAVEFORM OPERATION

Sample Problem

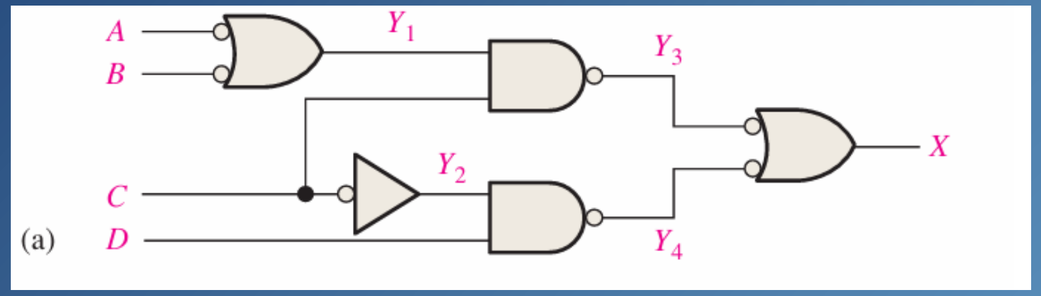
Determine the final output waveform X for the circuit in the Figure below, with input wave forms A, B, and C as shown.

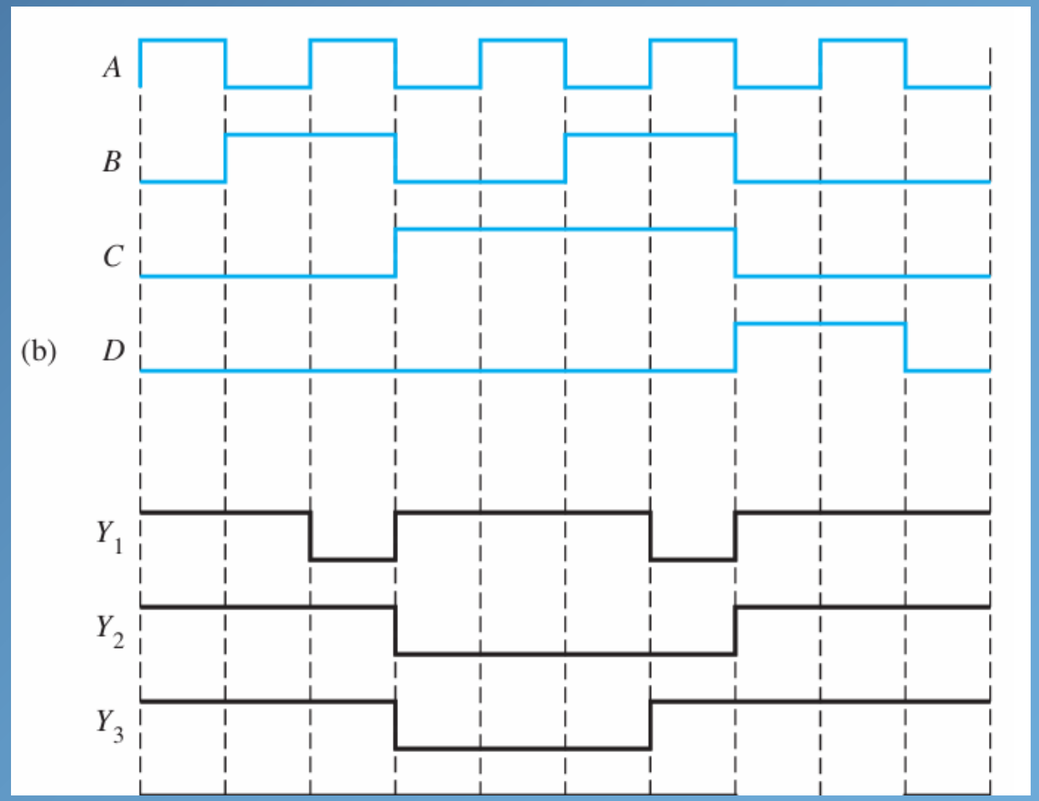


PULSE WAVEFORM OPERATION

Sample Problem

Determine the output waveform X for the logic circuit in Figure 2(a) by first finding the intermediate waveform at each of points Y1, Y2, Y3, and Y4. The input waveforms are shown in Figure 2(b).





ROLE OF DIGITAL ELECTRONICS IN EMBEDDED SYSTEMS

Embedded Systems:

- Dedicated systems designed for specific tasks with optimized performance and minimal resource utilization.
- Found in microcontroller-based devices like appliances, automotive systems, and medical instruments.

Role of Digital Electronics:

- Logic Gates & Flip-Flops: Fundamental building blocks for controlling hardware operations, such as timing and decision-making.
- Boolean Algebra: Simplifies logic design and decision-making processes, enabling efficient operation of embedded systems.

Applications:

- Microcontrollers: Integral to systems like washing machines, anti-lock braking systems (ABS), and wearable health devices.
- Ensures seamless interaction between hardware and software components for task execution.

DIGITAL ELECTRONICS IN IOT

Internet of Things (IoT):

- A vast network of interconnected devices exchanging data over the internet to enhance automation and intelligence.
- Enables real-time monitoring, decision-making, and control in applications like smart cities, healthcare, and industrial automation.

Digital Electronics in IoT:

- Sensors & Actuators: Connected via logic circuits for real-time data collection and response.
- Combinational & Sequential Circuits: Process sensor data and enable intelligent decision-making.
- Embedded Systems with RTOS: Manage real-time tasks with precision, ensuring smooth IoT operations.

Applications:

- Smart Homes: Devices like thermostats, security systems, and smart lighting.
- Industrial Automation: IoT-enabled machines for predictive maintenance and production optimization.

REFERENCES:

- Digital Electronics by Floyd

THANK YOU