

# Boolean Algebra

## Boolean Algebra

### Boolean Functions

Convert between Boolean algebra and logical equivalence

Boolean Expressions and Boolean Functions

Boolean functions

Identities of Boolean Algebra

Duality

The Abstract Definition of a Boolean Algebra

### Representing Boolean Functions

Functional Completeness

NAND

NOR (denoted by  $\downarrow$ )

### Logic Gates

Combinations of Gates

Examples of Circuits

Adders

Component 1 : Half adder

Component 2 : Full adder

Combination

### Minimization of Circuits

COMPLEXITY OF BOOLEAN FUNCTION MINIMIZATION

Karnaugh Maps

Don't Care Conditions

The Quine–McCluskey Method

# Boolean Functions

Boolean algebra provides the operations and the rules for working with the set  $\{0, 1\}$ .

The three operations in Boolean algebra that we will use most are

- complementation
  - The complement of an element, denoted with a bar, is defined by
$$\bar{0} = 1 \text{ and } \bar{1} = 0.$$
- Boolean sum
  - The Boolean sum, denoted by  $+$  or by OR, has the following values
  - $1 + 1 = 1, 1 + 0 = 1, 0 + 1 = 1, 0 + 0 = 0.$
- Boolean product
  - The Boolean product, denoted by  $\cdot$  or by AND, has the following values
  - $1 \cdot 1 = 1, 1 \cdot 0 = 0, 0 \cdot 1 = 0, 0 \cdot 0 = 0.$

Unless parentheses are used, the rules of precedence for Boolean operators are: first, all **complements** are computed, followed by all **Boolean products**, followed by all **Boolean sums**.

## Convert between Boolean algebra and logical equivalence

The complement, Boolean sum, and Boolean product correspond to the logical operators,  $\neg$ ,  $\vee$ , and  $\wedge$ , respectively, where 0 corresponds to F (false) and 1 corresponds to T (true). Equalities in Boolean algebra can be directly translated into equivalences of compound propositions. Conversely, equivalences of compound propositions can be translated into equalities in Boolean algebra.

**EXAMPLE 2** Translate  $1 \cdot 0 + \overline{(0 + 1)} = 0$ , the equality found in Example 1, into a logical equivalence.

*Solution:* We obtain a logical equivalence when we translate each 1 into a **T**, each 0 into an **F**, each Boolean sum into a disjunction, each Boolean product into a conjunction, and each complementation into a negation. We obtain

$$(T \wedge F) \vee \neg(T \vee F) \equiv F.$$

Example 3 illustrates the translation from propositional logic to Boolean algebra.

**EXAMPLE 3** Translate the logical equivalence  $(T \wedge T) \vee \neg F \equiv T$  into an identity in Boolean algebra.

*Solution:* We obtain an identity in Boolean algebra when we translate each **T** into a 1, each **F** into a 0, each disjunction into a Boolean sum, each conjunction into a Boolean product, and each negation into a complementation. We obtain

$$(1 \cdot 1) + \bar{0} = 1.$$

## Boolean Expressions and Boolean Functions

Let  $B = \{0, 1\}$ . Then  $B^n = \{(x_1, x_2, \dots, x_n) \mid x_i \in B \text{ for } 1 \leq i \leq n\}$  is the set of all possible  $n$ -tuples of 0s and 1s. The variable  $x$  is called a **Boolean variable** if it assumes values only from  $B$ , that is, if its only possible values are 0 and 1. A function from  $B^n$  to  $B$  is called a **Boolean function of degree  $n$** .

*a mapping from  $n$ -tuples to a single value*

**EXAMPLE 4** The function  $F(x, y) = x\bar{y}$  from the set of ordered pairs of Boolean variables to the set  $\{0, 1\}$  is a Boolean function of degree 2 with  $F(1, 1) = 0$ ,  $F(1, 0) = 1$ ,  $F(0, 1) = 0$ , and  $F(0, 0) = 0$ . We display these values of  $F$  in Table 1.

TABLE 1		
$x$	$y$	$F(x, y)$
1	1	0
1	0	1
0	1	0
0	0	0

## Boolean functions

Boolean functions can be represented using **expressions made up from variables and Boolean operations**

The Boolean expressions in the variables  $x_1, x_2, \dots, x_n$  are defined recursively

$0, 1, x_1, x_2, \dots, x_n$  are Boolean expressions;  
if  $E_1$  and  $E_2$  are Boolean expressions, then  $\overline{E_1}$ ,  $(E_1 E_2)$ , and  $(E_1 + E_2)$  are Boolean expressions.

Each Boolean expression represents a Boolean function. The values of this function are obtained by substituting 0 and 1 for the variables in the expression.

**EXAMPLE 5** Find the values of the Boolean function represented by  $F(x, y, z) = xy + \bar{z}$ .

*Solution:* The values of this function are displayed in Table 2.

TABLE 2					
$x$	$y$	$z$	$xy$	$\bar{z}$	$F(x, y, z) = xy + \bar{z}$
1	1	1	1	0	1
1	1	0	1	1	1
1	0	1	0	0	0
1	0	0	0	1	1
0	1	1	0	0	0
0	1	0	0	1	1
0	0	1	0	0	0
0	0	0	0	1	1

Boolean functions  $F$  and  $G$  of  $n$  variables are equal if and only if  $F(b_1, b_2, \dots, b_n) = G(b_1, b_2, \dots, b_n)$  whenever  $b_1, b_2, \dots, b_n$  belong to  $B$ . **Two different Boolean expressions that represent the same function are called equivalent.** For instance, the Boolean expressions  $xy$ ,  $xy + 0$ , and  $xy \cdot 1$  are equivalent.

- The complement of the Boolean function  $F$  is  $\overline{F}$ 
  - $\overline{F}(x_1, x_2, \dots, x_n) = \overline{F(x_1, x_2, \dots, x_n)}$
- The boolean sum of boolean function  $F$  and  $G$  is  $F + G$ 
  - $(F + G)(x_1, x_2, \dots, x_n) = F(x_1, x_2, \dots, x_n) + G(x_1, x_2, \dots, x_n)$
- The boolean product of boolean function  $F$  and  $G$  is  $F \cdot G$ 
  - $(FG)(x_1, x_2, \dots, x_n) = F(x_1, x_2, \dots, x_n) \cdot G(x_1, x_2, \dots, x_n)$

TABLE 3 The 16 Boolean Functions of Degree Two.																	
$x$	$y$	$F_1$	$F_2$	$F_3$	$F_4$	$F_5$	$F_6$	$F_7$	$F_8$	$F_9$	$F_{10}$	$F_{11}$	$F_{12}$	$F_{13}$	$F_{14}$	$F_{15}$	$F_{16}$
1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
1	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
0	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

**EXAMPLE 7** How many different Boolean functions of degree  $n$  are there?

*Solution:* From the product rule for counting, it follows that there are  $2^n$  different  $n$ -tuples of 0s and 1s. Because a Boolean function is an assignment of 0 or 1 to each of these  $2^n$  different  $n$ -tuples, the product rule shows that there are  $2^{2^n}$  different Boolean functions of degree  $n$ . ◀

# Identities of Boolean Algebra

There are many identities in Boolean algebra. The most important of these are displayed in Table 5. These identities are particularly **useful in simplifying the design of circuits**


TABLE 5 Boolean Identities.	
<i>Identity</i>	<i>Name</i>
$\overline{\overline{x}} = x$	Law of the double complement
$x + x = x$ $x \cdot x = x$	Idempotent laws
$x + 0 = x$ $x \cdot 1 = x$	Identity laws
$x + 1 = 1$ $x \cdot 0 = 0$	Domination laws
$x + y = y + x$ $xy = yx$	Commutative laws
$x + (y + z) = (x + y) + z$ $x(yz) = (xy)z$	Associative laws
$x + yz = (x + y)(x + z)$ $x(y + z) = xy + xz$	Distributive laws
$\overline{(xy)} = \overline{x} + \overline{y}$ $\overline{(x + y)} = \overline{x} \overline{y}$	De Morgan's laws
$x + xy = x$ $x(x + y) = x$	Absorption laws
$x + \overline{x} = 1$	Unit property
$x\overline{x} = 0$	Zero property

**EXAMPLE 9** Translate the distributive law  $x + yz = (x + y)(x + z)$  in Table 5 into a logical equivalence.

*Solution:* To translate a Boolean identity into a logical equivalence, we change each Boolean variable into a propositional variable. Here we will change the Boolean variables  $x$ ,  $y$ , and  $z$  into the propositional variables  $p$ ,  $q$ , and  $r$ . Next, we change each Boolean sum into a disjunction and

each Boolean product into a conjunction. (Note that 0 and 1 do not appear in this identity and complementation also does not appear.) This transforms the Boolean identity into the logical equivalence

$$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r).$$

This logical equivalence is one of the distributive laws for propositional logic in Table 6 in Section 1.3. 

Identities in Boolean algebra can be used to prove further identities. We demonstrate this in Example 10.

**EXAMPLE 10** Prove the **absorption law**  $x(x + y) = x$  using the other identities of Boolean algebra shown in Table 5. (This is called an absorption law because absorbing  $x + y$  into  $x$  leaves  $x$  unchanged.)

*Extra Examples* 

*Solution:* We display steps used to derive this identity and the law used in each step:

$$\begin{aligned} x(x + y) &= (x + 0)(x + y) && \text{Identity law for the Boolean sum} \\ &= x + 0 \cdot y && \text{Distributive law of the Boolean sum over the Boolean product} \\ &= x + y \cdot 0 && \text{Commutative law for the Boolean product} \\ &= x + 0 && \text{Domination law for the Boolean product} \\ &= x && \text{Identity law for the Boolean sum.} \end{aligned}$$




## Duality

The identities in Table 5 come in pairs (except for the law of the double complement and the unit and zero properties). To explain the relationship between the two identities in each pair we use the concept of a dual. **The dual of a Boolean expression is obtained by interchanging Boolean sums and Boolean products and interchanging 0s and 1s.**

*(Note that the complement leaves unchanged !!!)*

**EXAMPLE 11** Find the duals of  $x(y + 0)$  and  $\bar{x} \cdot 1 + (\bar{y} + z)$ .

*Solution:* Interchanging  $\cdot$  signs and  $+$  signs and interchanging 0s and 1s in these expressions produces their duals. The duals are  $x + (y \cdot 1)$  and  $(\bar{x} + 0)(\bar{y}z)$ , respectively. 

The dual of a Boolean function  $F$  represented by a Boolean expression is the function represented by the dual of this expression. This dual function, denoted by  $F^d$ , does not depend on the particular Boolean expression used to



represent  $F$ . An identity between functions represented by Boolean expressions remains valid when the duals of both sides of the identity are taken. (See Exercise 30 for the reason why this is true.) This result, called the duality principle, is useful for obtaining new identities

**EXAMPLE 12** Construct an identity from the absorption law  $x(x + y) = x$  by taking duals.

*Solution:* Taking the duals of both sides of this identity produces the identity  $x + xy = x$ , which is also called an absorption law and is shown in Table 5. ◀

## The Abstract Definition of a Boolean Algebra

it is useful to define Boolean algebras abstractly. Once it is shown that a particular structure is a Boolean algebra, then all results established about Boolean algebras in general apply to this particular structure.

### Definition 1

A *Boolean algebra* is a set  $B$  with two binary operations  $\vee$  and  $\wedge$ , elements 0 and 1, and a unary operation  $\bar{\phantom{x}}$  such that these properties hold for all  $x, y$ , and  $z$  in  $B$ :

$$\left. \begin{array}{l} x \vee 0 = x \\ x \wedge 1 = x \end{array} \right\}$$

Identity laws

$$\left. \begin{array}{l} x \vee \bar{x} = 1 \\ x \wedge \bar{x} = 0 \end{array} \right\}$$

Complement laws

$$\left. \begin{array}{l} (x \vee y) \vee z = x \vee (y \vee z) \\ (x \wedge y) \wedge z = x \wedge (y \wedge z) \end{array} \right\}$$

Associative laws

$$\left. \begin{array}{l} x \vee y = y \vee x \\ x \wedge y = y \wedge x \end{array} \right\}$$

Commutative laws

$$\left. \begin{array}{l} x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z) \\ x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z) \end{array} \right\}$$

Distributive laws

## Representing Boolean Functions


Two important problems of Boolean algebra will be studied in this section

- The first problem is: Given the values of a Boolean function, how can a Boolean expression that represents this function be found?
  - This problem will be solved by showing that any Boolean function can be represented by a Boolean sum of Boolean products of the variables and their complements.
- The second problem is: Is there a smaller set of operators that can be used to represent all Boolean functions?
  - all Boolean functions can be represented using only one operator.

**EXAMPLE 1** Find Boolean expressions that represent the functions  $F(x, y, z)$  and  $G(x, y, z)$ , which are given in Table 1.

TABLE 1					
$x$	$y$	$z$	$F$	$G$	
1	1	1	0	0	
1	1	0	0	1	
1	0	1	1	0	
1	0	0	0	0	
0	1	1	0	0	
0	1	0	0	1	
0	0	1	0	0	
0	0	0	0	0	

**Solution:** An expression that has the value 1 when  $x = z = 1$  and  $y = 0$ , and the value 0 otherwise, is needed to represent  $F$ . Such an expression can be formed by taking the Boolean product of  $x$ ,  $\bar{y}$ , and  $z$ . This product,  $x\bar{y}z$ , has the value 1 if and only if  $x = \bar{y} = z = 1$ , which holds if and only if  $x = z = 1$  and  $y = 0$ .

To represent  $G$ , we need an expression that equals 1 when  $x = y = 1$  and  $z = 0$ , or  $x = z = 0$  and  $y = 1$ . We can form an expression with these values by taking the Boolean sum of two different Boolean products. The Boolean product  $xyz$  has the value 1 if and only if  $x = y = 1$  and  $z = 0$ . Similarly, the product  $\bar{x}\bar{y}z$  has the value 1 if and only if  $x = z = 0$  and  $y = 1$ . The Boolean sum of these two products,  $xyz + \bar{x}\bar{y}z$ , represents  $G$ , because it has the value 1 if and only if  $x = y = 1$  and  $z = 0$ , or  $x = z = 0$  and  $y = 1$ . 


Example 1 illustrates a procedure for constructing a Boolean expression representing a function with given values. Each combination of values of the variables for which the function has the value 1 leads to a Boolean product of the variables or their complements.

## Definition 1

A literal is a Boolean variable or its complement. A minterm(小项) of the Boolean variables  $x_1, x_2, \dots, x_n$  is a Boolean product  $y_1 y_2 \cdots y_n$ , where  $y_i = x_i$  or  $y_i = \bar{x}_i$ . Hence, a minterm is a product of  $n$  literals, with one literal for each variable.

A minterm has the value 1 for one and only one combination of values of its variables

**EXAMPLE 2** Find a minterm that equals 1 if  $x_1 = x_3 = 0$  and  $x_2 = x_4 = x_5 = 1$ , and equals 0 otherwise.

*Solution:* The minterm  $\bar{x}_1 x_2 \bar{x}_3 x_4 x_5$  has the correct set of values. 

The sum of minterms that represents the function is called the **sum-of-products expansion** or the **disjunctive normal form**(析取范式) of the Boolean function.

It is also possible to find a Boolean expression that represents a Boolean function by taking a Boolean product of Boolean sums. The resulting expansion is called the **conjunctive normal form**(合取范式) or **product-of-sums expansion** of the function. These expansions can be found from sum of-products expansions by taking **duals**.

## Functional Completeness

Because every Boolean function can be represented using these operators we say that the set  $\cdot, +, -$  is functionally complete. Can we find a smaller set of functionally complete operators? We can do so if one of the three operators of this set can be expressed in terms of the other two.

This can be done using one of De Morgan's laws.

- eliminate boolean sum

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

- eliminate boolean product

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

- the complement can NOT be eliminated

$$x + x = x, x \cdot x = x$$

it is impossible to express complement( $\bar{x}$ ) in combination of  $+$  and  $\cdot$ .

Can we find a smaller set of functionally complete operators, namely, a set containing just one operator?

Such sets exist.

## NAND

we define NAND (denoted by  $|$ ) as the following

a	b	$a b$
1	1	0
1	0	1
0	1	1
0	0	1

To see that  $\{ | \}$  is functionally complete, because  $\{ \cdot, - \}$  is functionally complete, all that we have to do is show that both of the operators  $\cdot$  and  $-$  can be expressed using just the  $|$  operator.

$$\bar{x} = x|x$$

$$x \cdot y = (x|y)|(x|y) \text{ not NAND is AND}$$

## NOR (denoted by $\downarrow$ )

To see that  $\{\downarrow\}$  is functionally complete, because  $\{+,-\}$  is functionally complete, all that we have to do is show that both of the operators  $+$  and  $-$  can be expressed using just the  $\downarrow$  operator.

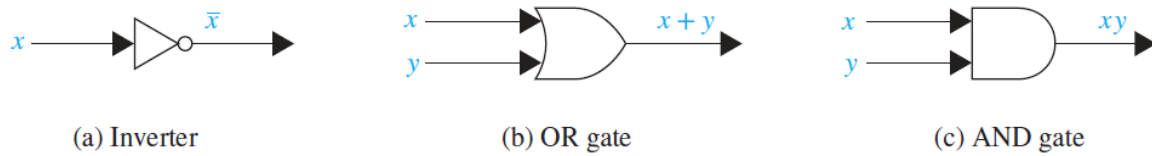
a	b	$a \downarrow b$
1	1	0
0	1	0
1	0	0
0	0	1

$$\bar{x} = x \downarrow x$$

$$x + y = (x \downarrow y) \downarrow (x \downarrow y)$$

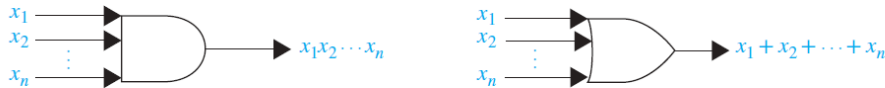
## Logic Gates

Boolean algebra is used to model the circuitry of electronic devices. Each input and each output of such a device can be thought of as a member of the set  $\{0, 1\}$ . A computer, or other electronic device, is made up of a number of circuits. Each circuit can be designed using the rules of Boolean algebra that were studied in Sections 12.1 and 12.2.



**FIGURE 1** Basic types of gates.

We will permit multiple inputs to AND and OR gates. The inputs to each of these gates are shown on the left side entering the element, and the output is shown on the right side. Examples of AND and OR gates with  $n$  inputs are shown in Figure 2.



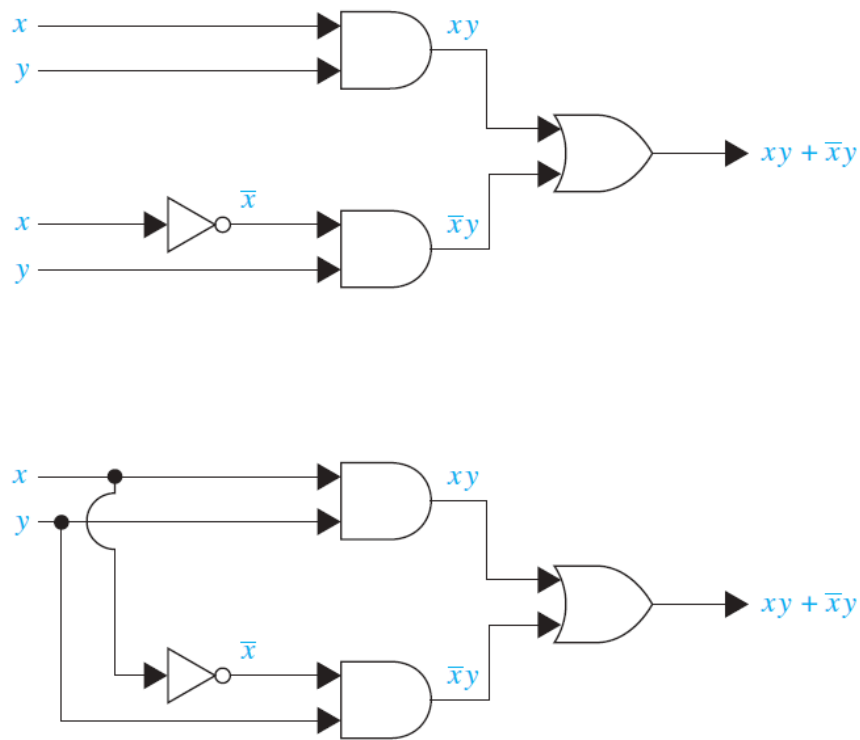
**FIGURE 2** Gates with  $n$  inputs.

## Combinations of Gates

Combinational circuits can be constructed using a combination of inverters, OR gates, and AND gates.

Some gates will share a same input, we can depict this in

- use branchings that indicate all the gates that use a given input.
- indicate this input separately for each gate.

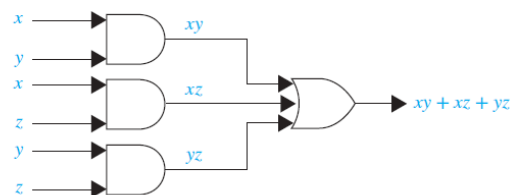


**FIGURE 3** Two ways to draw the same circuit.

## Examples of Circuits

**EXAMPLE 2** A committee of three individuals decides issues for an organization. Each individual votes either yes or no for each proposal that arises. A proposal is passed if it receives at least two yes votes. Design a circuit that determines whether a proposal passes.

**Extra Examples** *Solution:* Let  $x = 1$  if the first individual votes yes, and  $x = 0$  if this individual votes no; let  $y = 1$  if the second individual votes yes, and  $y = 0$  if this individual votes no; let  $z = 1$  if the third individual votes yes, and  $z = 0$  if this individual votes no. Then a circuit must be designed that produces the output 1 from the inputs  $x$ ,  $y$ , and  $z$  when two or more of  $x$ ,  $y$ , and  $z$  are 1. One representation of the Boolean function that has these output values is  $xy + xz + yz$  (see Exercise 12 in Section 12.1). The circuit that implements this function is shown in Figure 5. ◀



**FIGURE 5** A circuit for majority voting.

## Adders

We will illustrate how logic circuits can be used to carry out addition of two positive integers from their binary expansions.

we achieve this by combining some components

## Component 1 : Half adder

*adds two bits, without considering a carry from a previous addition.*

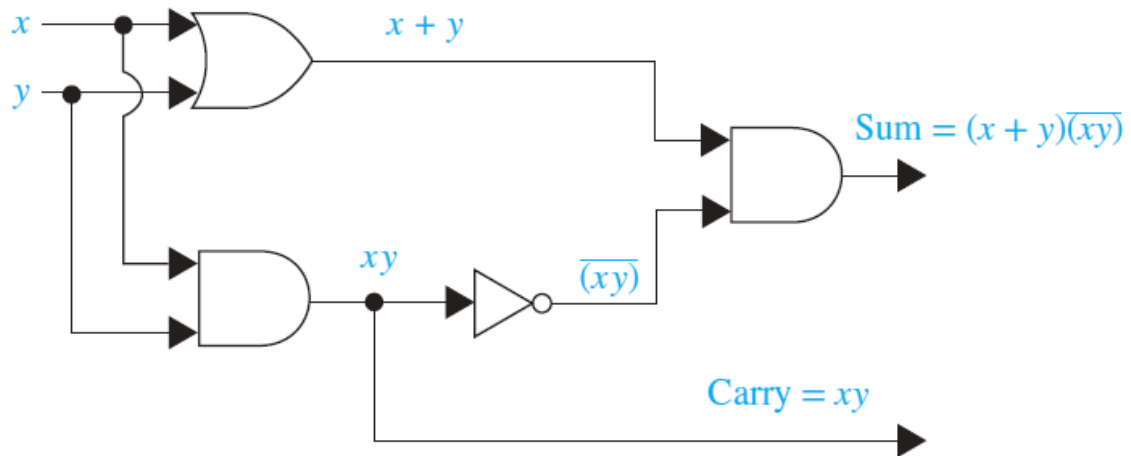
build a circuit that can be used to find  $x + y$ , where  $x$  and  $y$  are two bits. The input to our circuit will be  $x$  and  $y$ , because these each have the value 0 or the value 1. The output will consist of two bits, namely,  $s$  and  $c$ , where  **$s$  is the sum bit** and  **$c$  is the carry bit**.

TABLE 3 Input and Output for the Half Adder.			
<i>Input</i>		<i>Output</i>	
<i>x</i>	<i>y</i>	<i>s</i>	<i>c</i>
1	1	0	1
1	0	1	0
0	1	1	0
0	0	0	0

$$c = x \cdot y$$



$$s = x\bar{y} + \bar{x}y = (x + y)\overline{(xy)}$$



## Component 2 : Full adder

*We use the full adder to compute the sum bit and the carry bit when two bits and a carry are added.*

The inputs to the full adder are the bits  $x$  and  $y$  and the carry  $c_i$ .

The outputs are the sum bit  $s$  and the new carry  $c_{i+1}$ .

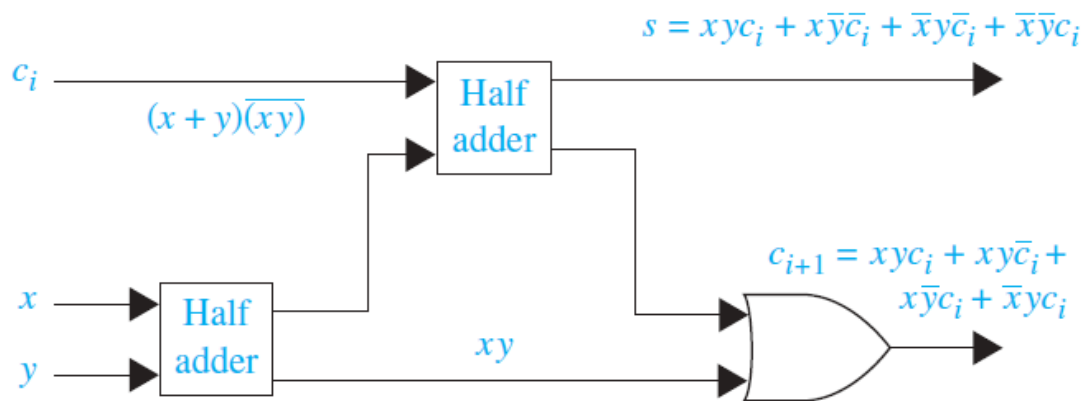
**TABLE 4**  
**Input and**  
**Output for**  
**the Full Adder.**

<i>Input</i>			<i>Output</i>	
$x$	$y$	$c_i$	$s$	$c_{i+1}$
1	1	1	1	1
1	1	0	0	1
1	0	1	0	1
1	0	0	1	0
0	1	1	0	1
0	1	0	1	0
0	0	1	1	0
0	0	0	0	0

$$s = xyc_i + x\bar{y}\bar{c}_i + \bar{x}y\bar{c}_i + \bar{x}\bar{y}c_i$$

$$c_{i+1} = xyc_i + xy\bar{c}_i + x\bar{y}c_i + \bar{x}yc_i$$

However, instead of designing the full adder from scratch, we will use half adders to produce the desired output.



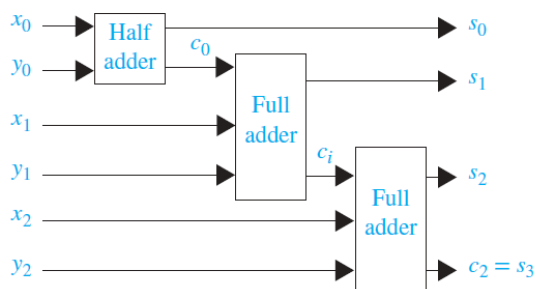
**FIGURE 9 A full adder.**

$$\begin{array}{r} 1 \\ + 1_1 \\ \hline 1 \quad 1 \end{array}$$

- first we assume that there is not a carry, add  $x$  and  $y$  via a half adder
- then we add the output  $s$  and carry  $c_i$  via a half adder
- for the reason that no  $x, y, c_i$  will produce a twice carry, we use a OR gate for  $c$  and  $c_i$  to make  $c_{i+1}$

## Combination

Finally, in Figure 10 we show how full and half adders can be used to add the two three-bit integers  $(x_2x_1x_0)_2$  and  $(y_2y_1y_0)_2$  to produce the sum  $(s_3s_2s_1s_0)_2$ . Note that  $s_3$ , the highest-order bit in the sum, is given by the carry  $c_2$ .



**FIGURE 10 Adding two three-bit integers with full and half adders.**

$$\begin{array}{rrrr}
 & x_2 & x_1 & x_0 \\
 & y_2 & y_1 & y_0 \\
 \hline
 +s_3 & s_2 & s_1 & s_0
 \end{array}$$

## Minimization of Circuits

The efficiency of a combinational circuit depends on the number and arrangement of its gates.

As we have known that all boolean functions can be expressed by the sum-of-product expansions, here we come two procedures that simplify sum-of-products expansions.

- Karnaugh maps (or K-maps)
  - minimize circuits by hand
  - useful in minimizing circuits with up to six variables, although they become rather complex even for five or six variables.
- Quine–McCuskey method
  - automates the process of minimizing combinatorial circuits and can be implemented as a computer program.

## COMPLEXITY OF BOOLEAN FUNCTION MINIMIZATION

- minimizing Boolean functions with many variables is a computationally intensive problem
- It is an NP-complete problem (see Section 3.3 and [Ka93]), so the problem cannot be solved in polynomial-time algorithm.

- The Quine–McCluskey method has exponential complexity
- Since the 1970s a number of newer algorithms have been developed(see [Ha93] and [KaBe04]).However, with the best algorithms yet devised, only circuits with no more than 25 variables can be minimized. Also, heuristic (or rule-of-thumb) methods can be used to substantially simplify, but not necessarily minimize, Boolean expressions with a larger number of literals.

## Karnaugh Maps

Let's begin with 2 variable

- There are four possible minterms in the sum-of-products expansion of a Boolean function in the two variables  $x$  and  $y$ .( $xy, \bar{x}y, x\bar{y}, \bar{x}\bar{y}$ )

	$y$	$\bar{y}$
$x$	$xy$	$x\bar{y}$
$\bar{x}$	$\bar{x}y$	$\bar{x}\bar{y}$

**FIGURE 2**  
**K-maps in two variables.**

- If a minterm present in the expression, place a 1 in the corresponding cell.
- Cells are said to be adjacent if the minterms that they represent differ in exactly one literal. ( $xy$  and  $\bar{x}y$ , similar with bits 010101 and 110101, etc)

**EXAMPLE 1** Find the K-maps for (a)  $xy + \bar{x}y$ , (b)  $x\bar{y} + \bar{x}y$ , and (c)  $x\bar{y} + \bar{x}y + \bar{x}\bar{y}$ .

*Solution:* We include a 1 in a cell when the minterm represented by this cell is present in the sum-of-products expansion. The three K-maps are shown in Figure 3. ◀

	$y$	$\bar{y}$
$x$	1	
$\bar{x}$	1	

(a)

	$y$	$\bar{y}$
$x$		1
$\bar{x}$	1	

(b)

	$y$	$\bar{y}$
$x$		1
$\bar{x}$	1	1

(c)

- Whenever there are 1s in two adjacent cells in the K-map, the minterms represented by these cells can be combined into a product involving just one of the variables. For instance,  $xy$  and  $\bar{x}y$  are represented by adjacent cells and can be combined into  $y$ , because  $xy + \bar{x}y = (\bar{x} + x)y = y$ .

we can also use **Gray code** to represent the variables

Notes that squares on the boundary are adjacent.

	$yz$	$y\bar{z}$	$\bar{y}\bar{z}$	$\bar{y}z$
$x$	1	1	1	1
$\bar{x}$	1		1	1

(c)

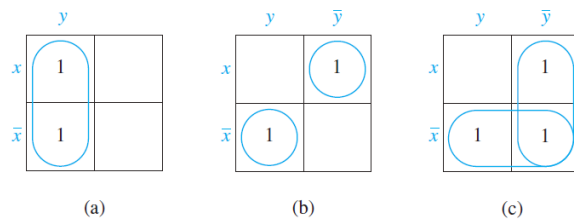
- if 1s are in all four cells, the four minterms can be combined into one term, namely, the Boolean expression 1 that involves none of the variables.
- try to cover all the 1s with the fewest blocks using the largest blocks

first and always using the largest possible blocks.(each block must have the size of

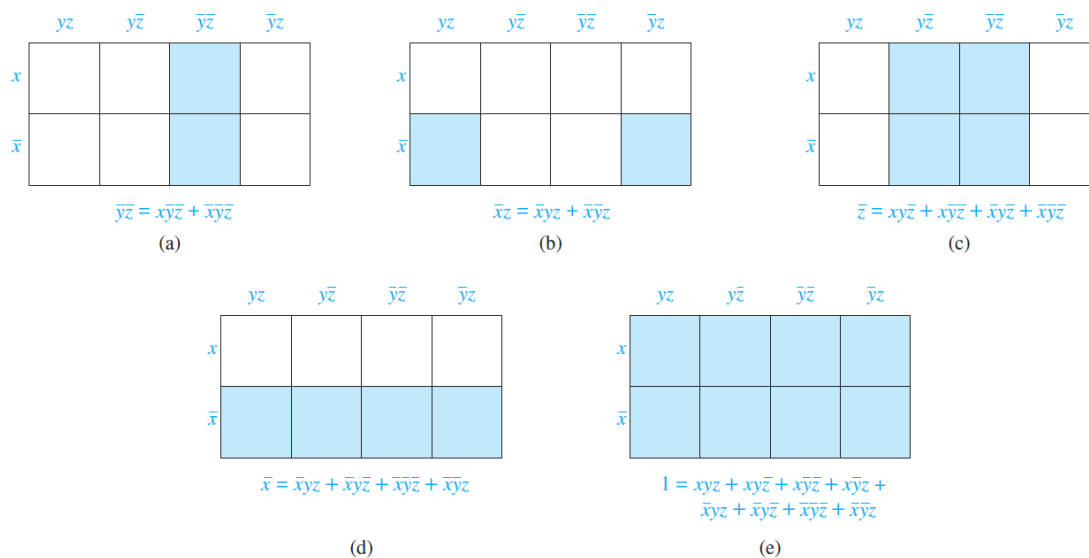
$2^n, n = 0, 1, \dots$ , the number of your variables)

**EXAMPLE 2** Simplify the sum-of-products expansions given in Example 1.

**Solution:** The grouping of minterms is shown in Figure 4 using the K-maps for these expansions. Minimal expansions for these sums-of-products are (a)  $y$ , (b)  $x\bar{y} + \bar{x}y$ , and (c)  $\bar{x} + \bar{y}$ . ◀

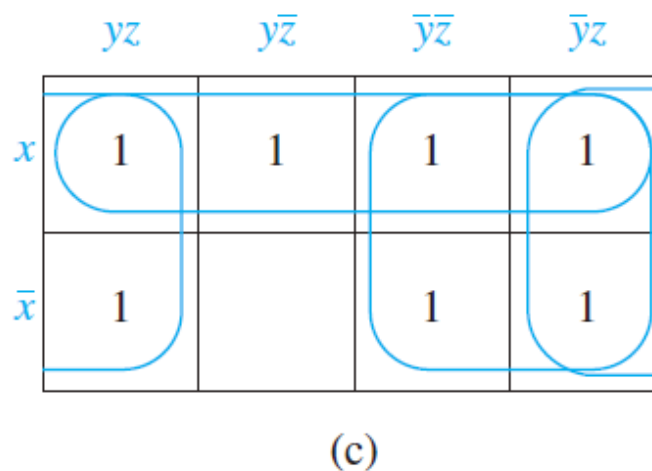


**FIGURE 4** Simplifying the sum-of-products expansions from Example 2.



**FIGURE 6** Blocks in K-maps in three variables.

always use the possible biggest block, even they lapped with each other



use three 4-size block

# Don't Care Conditions

In some circuits we care only about the output for some combinations of input values, because other combinations of input values are not possible or never occur.

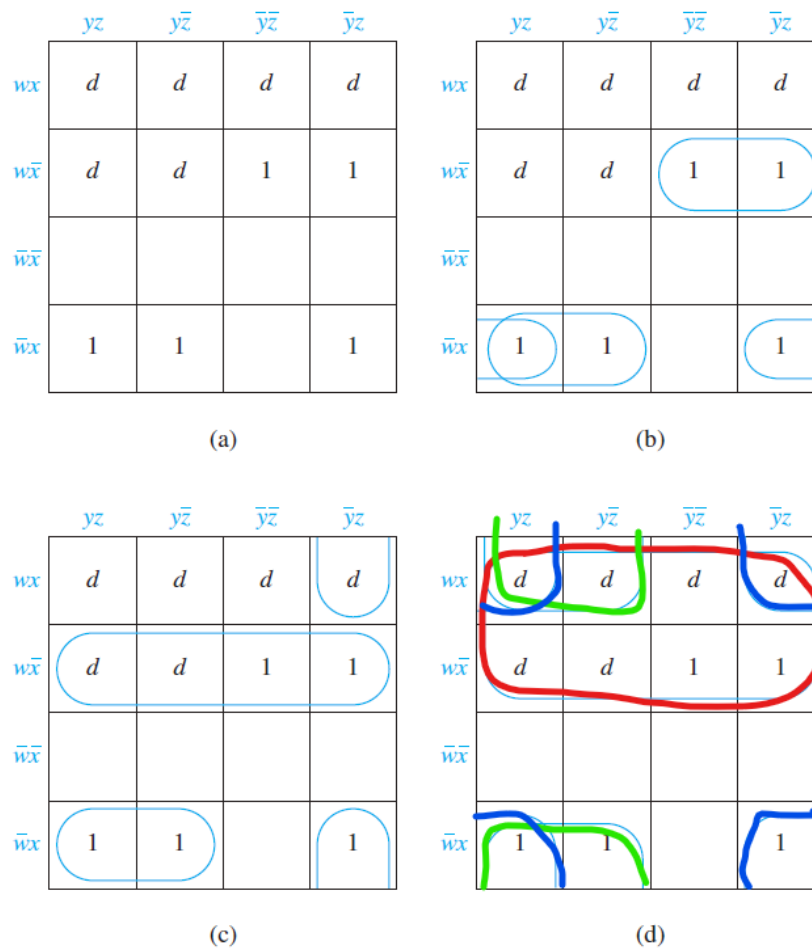
In the minimization process we can assign 1s as values to those combinations of the input values that lead to the largest blocks in the K-map.(we first mark these cells with 'd')

**EXAMPLE 8** One way to code decimal expansions using bits is to use the four bits of the binary expansion of each digit in the decimal expansion. For instance, 873 is encoded as 100001110011. This encoding of a decimal expansion is called a **binary coded decimal expansion**. Because there are 16 blocks of four bits and only 10 decimal digits, there are six combinations of four bits that are not used to encode digits. Suppose that a circuit is to be built that produces an output of 1 if the decimal digit is 5 or greater and an output of 0 if the decimal digit is less than 5. How can this circuit be simply built using OR gates, AND gates, and inverters?

*Solution:* Let  $F(w, x, y, z)$  denote the output of the circuit, where  $wxyz$  is a binary expansion of a decimal digit. The values of  $F$  are shown in Table 1. The K-map for  $F$ , with  $ds$  in the

TABLE 1					
<i>Digit</i>	<i>w</i>	<i>x</i>	<i>y</i>	<i>z</i>	<i>F</i>
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1





**FIGURE 11** The K-map for  $F$  showing its *don't care* positions.

*don't care* positions, is shown in Figure 11(a). We can either include or exclude squares with  $ds$  from blocks. This gives us many possible choices for the blocks. For example, excluding all squares with  $ds$  and forming blocks, as shown in Figure 11(b), produces the expression  $w\bar{x}\bar{y} + \bar{w}xy + \bar{w}xz$ . Including some of the  $ds$  and excluding others and forming blocks, as shown in Figure 11(c), produces the expression  $w\bar{x} + \bar{w}xy + x\bar{y}z$ . Finally, including all the  $ds$  and using the blocks shown in Figure 11(d) produces the simplest sum-of-products expansion possible, namely,  $F(x, y, z) = w + xy + xz$ . ◀

## The Quine–McCuskey Method

the Quine–McCuskey method uses this sequence of steps to simplify a sum-of-products expression.

1. Express each minterm in  $n$  variables by a bit string of length  $n$  with a 1 in the  $i$ -th position if  $x_i$  occurs and a 0 in this position if  $\bar{x}_i$  occurs.
2. Group the bit strings according to the number of 1s in them

3.  $n - 1$  variables that can be formed by taking the Boolean sum. Minterms that can be combined are represented by bit strings that differ in exactly one position. Represent these products in  $n - 1$  variables with strings that have a 1 in the  $i$ -th position if  $x_i$  occurs in the product, a 0 in this position if  $\overline{x_i}$  occurs, and a dash in this position if there is no literal involving  $x_i$  (it is combined)
4. Determine all products in  $n - 2$  variables that can be formed by taking the Boolean sum of the products in  $n - 1$  variables found in the previous step. Products in  $n - 1$  variables that can be combined are represented by bit strings that have a dash in the same position and differ in exactly one position.
5. Continue combining Boolean products into products in fewer variables as long as possible.
6. Find all the Boolean products that arose that were not used to form a Boolean product in one fewer literal
7. Find the smallest set of these Boolean products such that the sum of these products represents the Boolean function. This is done by forming a table showing which minterms are covered by which products. Every minterm must be covered by at least one product.

**EXAMPLE 9** We will show how the Quine–McCluskey method can be used to find a minimal expansion equivalent to

$$xyz + x\bar{y}z + \bar{x}yz + \bar{x}\bar{y}z + \bar{x}\bar{y}\bar{z}.$$

We will represent the minterms in this expansion by bit strings. The first bit will be 1 if  $x$  occurs and 0 if  $\bar{x}$  occurs. The second bit will be 1 if  $y$  occurs and 0 if  $\bar{y}$  occurs. The third bit will be 1 if  $z$  occurs and 0 if  $\bar{z}$  occurs. We then group these terms according to the number of 1s in the corresponding bit strings. This information is shown in Table 2.

Minterms that can be combined are those that differ in exactly one literal. Hence, two terms that can be combined differ by exactly one in the number of 1s in the bit strings that represent them. When two minterms are combined into a product, this product contains two literals. A product in two literals is represented using a dash to denote the variable that does not occur. For instance, the minterms  $x\bar{y}z$  and  $\bar{x}\bar{y}z$ , represented by bit strings 101 and 001, can be combined into  $\bar{y}z$ , represented by the string  $-01$ . All pairs of minterms that can be combined and the product formed from these combinations are shown in Table 3.

Next, all pairs of products of two literals that can be combined are combined into one literal. Two such products can be combined if they contain literals for the same two variables, and literals for only one of the two variables differ. In terms of the strings representing the

TABLE 3

		Step 1		Step 2	
Term	Bit String	Term	String	Term	String
1 $xyz$	111	(1,2) $xz$	1–1	(1,2,3,4) $z$	– –1
2 $x\bar{y}z$	101	(1,3) $yz$	–11		
3 $\bar{x}yz$	011	(2,4) $\bar{y}z$	–01		
4 $\bar{x}\bar{y}z$	001	(3,4) $\bar{x}z$	0–1		
5 $\bar{x}\bar{y}\bar{z}$	000	(4,5) $\bar{x}\bar{y}$	00–		

TABLE 4

	$xyz$	$x\bar{y}z$	$\bar{x}yz$	$\bar{x}\bar{y}z$	$\bar{x}\bar{y}\bar{z}$
$z$	X	X	X	X	
$\bar{x}\bar{y}$				X	X

products, these strings must have a dash in the same position and must differ in exactly one of the other two slots. We can combine the products  $yz$  and  $\bar{y}z$ , represented by the strings  $-11$  and  $-01$ , into  $z$ , represented by the string  $- -1$ . We show all the combinations of terms that can be formed in this way in Table 3.

In Table 3 we also indicate which terms have been used to form products with fewer literals; these terms will not be needed in a minimal expansion. The next step is to identify a minimal set of products needed to represent the Boolean function. We begin with all those products that were not used to construct products with fewer literals. Next, we form Table 4, which has a row for each candidate product formed by combining original terms, and a column for each original term; and we put an X in a position if the original term in the sum-of-products expansion was used to form this candidate product. In this case, we say that the candidate product **covers** the original minterm. We need to include at least one product that covers each of the original minterms. Consequently, whenever there is only one X in a column in the table, the product corresponding to the row this X is in must be used. From Table 4 we see that both  $z$  and  $\bar{x}\bar{y}$  are needed. Hence, the final answer is  $z + \bar{x}\bar{y}$ . ◀