Digital Logic Circuits

IN THIS CHAPTER

1-1 Digital Computers

digital

The digital computer is a digital system that performs various computational tasks. The word *digital* implies that the information in the computer is represented by variables that take a limited number of discrete values. These values are processed internally by components that can maintain a limited number of discrete states. The decimal digits 0, 1, 2, ..., 9, for example, provide 10 discrete values. The first electronic digital computers, developed in the late 1940s, were used primarily for numerical computations. In this case the discrete elements are the digits. From this application the term *digital computer* has emerged. In practice, digital computers function more reliably if only two states are used. Because of the physical restriction of components, and because human logic tends to be binary (i.e., true-or-false, yes-or-no statements), digital components that are constrained to take discrete values are further constrained to take only two values and are said to be *binary*.

bit

Digital computers use the binary number system, which has two digits: 0 and 1. A binary digit is called a *bit*. Information is represented in digital computers in groups of bits. By using various coding techniques, groups of bits can be made to represent not only binary numbers but also other discrete

symbols, such as decimal digits or letters of the alphabet. By judicious use of binary arrangements and by using various coding techniques, the groups of bits are used to develop complete sets of instructions for performing various types of computations.

In contrast to the common decimal numbers that employ the base 10 system, binary numbers use a base 2 system with two digits: 0 and 1. The decimal equivalent of a binary number can be found by expanding it into a power series with a base of 2. For example, the binary number 1001011 represents a quantity that can be converted to a decimal number by multiplying each bit by the base 2 raised to an integer power as follows:

$$1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 75$$

The seven bits 1001011 represent a binary number whose decimal equivalent is 75. However, this same group of seven bits represents the letter K when used in conjunction with a binary code for the letters of the alphabet. It may also represent a control code for specifying some decision logic in a particular digital computer. In other words, groups of bits in a digital computer are used to represent many different things. This is similar to the concept that the same letters of an alphabet are used to construct different languages, such as English and French.

A computer system is sometimes subdivided into two functional entities: hardware and software. The hardware of the computer consists of all the electronic components and electromechanical devices that comprise the physical entity of the device. Computer software consists of the instructions and data that the computer manipulates to perform various data-processing tasks. *program* A sequence of instructions for the computer is called a *program*. The data that are manipulated by the program constitute the *data base*.

A computer system is composed of its hardware and the system software available for its use. The system software of a computer consists of a collection of programs whose purpose is to make more effective use of the computer. The programs included in a systems software package are referred to as the *operating system*. They are distinguished from application programs written by the user for the purpose of solving particular problems. For example, a high-level language program written by a user to solve particular data-processing needs is an application program, but the compiler that translates the high-level language program to machine language is a system program. The customer who buys a computer system would need, in addition to the hardware, any available software needed for effective operation of the computer. The system software is an indispensable part of a total computer system. Its function is to compensate for the differences that exist between user needs and the capability of the hardware.

computer hardware   The hardware of the computer is usually divided into three major parts, as shown in Fig. 1-1. The central processing unit (CPU) contains an arithmetic
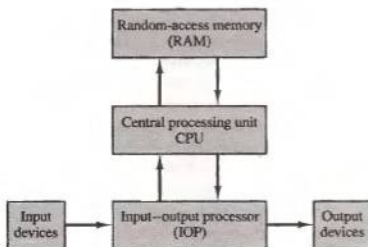
Figure 1-1 Block diagram of a digital computer.

and logic unit for manipulating data, a number of registers for storing data, and control circuits for fetching and executing instructions. The memory of a computer contains storage for instructions and data. It is called a random-access memory (RAM) because the CPU can access any location in memory at random and retrieve the binary information within a fixed interval of time. The input and output processor (IOP) contains electronic circuits for communicating and controlling the transfer of information between the computer and the outside world. The input and output devices connected to the computer include keyboards, printers, terminals, magnetic disk drives, and other communication devices.

This book provides the basic knowledge necessary to understand the hardware operations of a computer system. The subject is sometimes considered from three different points of view, depending on the interest of the investigator. When dealing with computer hardware it is customary to distinguish between what is referred to as computer organization, computer design, and computer architecture.

computer organization
Computer organization is concerned with the way the hardware components operate and the way they are connected together to form the computer system. The various components are assumed to be in place and the task is to investigate the organizational structure to verify that the computer parts operate as intended.

computer design
Computer design is concerned with the hardware design of the computer. Once the computer specifications are formulated, it is the task of the designer to develop hardware for the system. Computer design is concerned with the determination of what hardware should be used and how the parts should be connected. This aspect of computer hardware is sometimes referred to as *computer implementation*.

computer architecture
Computer architecture is concerned with the structure and behavior of the computer as seen by the user. It includes the information formats, the instruc-

tion set, and techniques for addressing memory. The architectural design of a computer system is concerned with the specifications of the various functional modules, such as processors and memories, and structuring them together into a computer system.

The book deals with all three subjects associated with computer hardware. In Chapters 1 through 4 we present the various digital components used in the organization and design of computer systems. Chapters 5 through 7 cover the steps that a designer must go through to design and program an elementary digital computer. Chapters 8 and 9 deal with the architecture of the central processing unit. In Chapters 11 and 12 we present the organization and architecture of the input–output processor and the memory unit.

1-2   Logic Gates

Binary information is represented in digital computers by physical quantities called *signals*. Electrical signals such as voltages exist throughout the computer in either one of two recognizable states. The two states represent a binary variable that can be equal to 1 or 0. For example, a particular digital computer may employ a signal of 3 volts to represent binary 1 and 0.5 volt to represent binary 0. The input terminals of digital circuits accept binary signals of 3 and 0.5 volts and the circuits respond at the output terminals with signals of 3 and 0.5 volts to represent binary input and output corresponding to 1 and 0, respectively.

Binary logic deals with binary variables and with operations that assume a logical meaning. It is used to describe, in algebraic or tabular form, the manipulation and processing of binary information. The manipulation of binary information is done by logic circuits called *gates*. Gates are blocks of hardware that produce signals of binary 1 or 0 when input logic requirements are satisfied. A variety of logic gates are commonly used in digital computer systems. Each gate has a distinct graphic symbol and its operation can be described by means of an algebraic expression. The input–output relationship of the binary variables for each gate can be represented in tabular form by a *truth table*.

gates

The names, graphic symbols, algebraic functions, and truth tables of eight logic gates are listed in Fig. 1-2. Each gate has one or two binary input variables designated by A and B and one binary output variable designated by x. The AND gate produces the AND logic function: that is, the output is 1 if input A and input B are both equal to 1; otherwise, the output is 0. These conditions are also specified in the truth table for the AND gate. The table shows that output x is 1 only when both input A and input B are 1. The algebraic operation symbol of the AND function is the same as the multiplication symbol of ordinary arithmetic. We can either use a dot between the variables or

AND

| Name	Graphic symbol	Algebraic function	Truth table
AND	A, B → x	$x = A \cdot B$ or $x = AB$	$A\ B \mid x$ $0\ 0 \mid 0$ $0\ 1 \mid 0$ $1\ 0 \mid 0$ $1\ 1 \mid 1$
OR	A, B → x	$x = A + B$	$A\ B \mid x$ $0\ 0 \mid 0$ $0\ 1 \mid 1$ $1\ 0 \mid 1$ $1\ 1 \mid 1$
Inverter	A → x	$x = A'$	$A \mid x$ $0 \mid 1$ $1 \mid 0$
Buffer	A → x	$x = A$	$A \mid x$ $0 \mid 0$ $1 \mid 1$
NAND	A, B → x	$x = (AB)'$	$A\ B \mid x$ $0\ 0 \mid 1$ $0\ 1 \mid 1$ $1\ 0 \mid 1$ $1\ 1 \mid 0$
NOR	A, B → x	$x = (A + B)'$	$A\ B \mid x$ $0\ 0 \mid 1$ $0\ 1 \mid 0$ $1\ 0 \mid 0$ $1\ 1 \mid 0$
Exclusive-OR (XOR)	A, B → x	$x = A \oplus B$ or $x = A'B + AB'$	$A\ B \mid x$ $0\ 0 \mid 0$ $0\ 1 \mid 1$ $1\ 0 \mid 1$ $1\ 1 \mid 0$
Exclusive-NOR or equivalence	A, B → x	$x = (A \oplus B)'$ or $x = A'B' + AB$	$A\ B \mid x$ $0\ 0 \mid 1$ $0\ 1 \mid 0$ $1\ 0 \mid 0$ $1\ 1 \mid 1$

Figure 1-2   Digital logic gates.

concatenate the variables without an operation symbol between them. AND gates may have more than two inputs, and by definition, the output is 1 if and only if all inputs are 1.

OR

The OR gate produces the inclusive-OR function; that is, the output is 1 if input A or input B or both inputs are 1; otherwise, the output is 0. The algebraic symbol of the OR function is $+$, similar to arithmetic addition. OR gates may have more than two inputs, and by definition, the output is 1 if any input is 1.

inverter

The inverter circuit inverts the logic sense of a binary signal. It produces the NOT, or complement, function. The algebraic symbol used for the logic complement is either a prime or a bar over the variable symbol. In this book we use a prime for the logic complement of a binary variable, while a bar over the letter is reserved for designating a complement microoperation as defined in Chap. 4.

The small circle in the output of the graphic symbol of an inverter designates a logic complement. A triangle symbol by itself designates a buffer circuit. A buffer does not produce any particular logic function since the binary value of the output is the same as the binary value of the input. This circuit is used merely for power amplification. For example, a buffer that uses 3 volts for binary 1 will produce an output of 3 volts when its input is 3 volts. However, the amount of electrical power needed at the input of the buffer is much less than the power produced at the output of the buffer. The main purpose of the buffer is to drive other gates that require a large amount of power.

NAND

The NAND function is the complement of the AND function, as indicated by the graphic symbol, which consists of an AND graphic symbol followed by a small circle. The designation NAND is derived from the abbreviation of NOT-AND. The NOR gate is the complement of the OR gate and uses an OR graphic symbol followed by a small circle. Both NAND and NOR gates may have more than two inputs, and the output is always the complement of the AND or OR function, respectively.

NOR

exclusive-OR

The exclusive-OR gate has a graphic symbol similar to the OR gate except for the additional curved line on the input side. The output of this gate is 1 if any input is 1 but excludes the combination when both inputs are 1. The exclusive-OR function has its own algebraic symbol or can be expressed in terms of AND, OR, and complement operations as shown in Fig. 1-2. The exclusive-NOR is the complement of the exclusive-OR, as indicated by the small circle in the graphic symbol. The output of this gate is 1 only if both inputs are equal to 1 or both inputs are equal to 0. A more fitting name for the exclusive-OR operation would be an odd function; that is, its output is 1 if an odd number of inputs are 1. Thus in a three-input exclusive-OR (odd) function, the output is 1 if only one input is 1 or if all three inputs are 1. The exclusive-OR and exclusive-NOR gates are commonly available with two inputs, and only seldom are they found with three or more inputs.

1-3   Boolean Algebra

Boolean function

Boolean algebra is an algebra that deals with binary variables and logic operations. The variables are designated by letters such as A, B, x, and y. The three basic logic operations are AND, OR, and complement. A Boolean function can be expressed algebraically with binary variables, the logic operation symbols, parentheses, and equal sign. For a given value of the variables, the Boolean function can be either 1 or 0. Consider, for example, the Boolean function

$$F = x + y'z$$

truth table

The function F is equal to 1 if x is 1 or if both y' and z are equal to 1; F is equal to 0 otherwise. But saying that $y' = 1$ is equivalent to saying that $y = 0$ since y' is the complement of y. Therefore, we may say that F is equal to 1 if $x = 1$ or if $yz = 01$. The relationship between a function and its binary variables can be represented in a truth table. To represent a function in a truth table we need a list of the 2^n combinations of the n binary variables. As shown in Fig. 1-3(a), there are eight possible distinct combinations for assigning bits to the three variables x, y, and z. The function F is equal to 1 for those combinations where $x = 1$ or $yz = 01$; it is equal to 0 for all other combinations.

logic diagram

A Boolean function can be transformed from an algebraic expression into a logic diagram composed of AND, OR, and inverter gates. The logic diagram for F is shown in Fig. 1-3(b). There is an inverter for input y to generate its complement y'. There is an AND gate for the term $y'z$, and an OR gate is used to combine the two terms. In a logic diagram, the variables of the function are taken to be the inputs of the circuit, and the variable symbol of the function is taken as the output of the circuit.

The purpose of Boolean algebra is to facilitate the analysis and design of digital circuits. It provides a convenient tool to:

1. Express in algebraic form a truth table relationship between binary variables.

Figure 1-3   Truth table and logic diagram for $F = x + y'z$.

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1
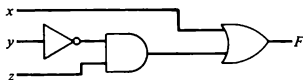
(a) Truth table   (b) Logic diagram

2. Express in algebraic form the input–output relationship of logic diagrams.

3. Find simpler circuits for the same function.

Boolean expression

A Boolean function specified by a truth table can be expressed algebraically in many different ways. By manipulating a Boolean expression according to Boolean algebra rules, one may obtain a simpler expression that will require fewer gates. To see how this is done, we must first study the manipulative capabilities of Boolean algebra.

Table 1-1 lists the most basic identities of Boolean algebra. All the identities in the table can be proven by means of truth tables. The first eight identities show the basic relationship between a single variable and itself, or in conjunction with the binary constants 1 and 0. The next five identities (9 through 13) are similar to ordinary algebra. Identity 14 does not apply in ordinary algebra but is very useful in manipulating Boolean expressions. Identities 15 and 16 are called DeMorgan's theorems and are discussed below. The last identity states that if a variable is complemented twice, one obtains the original value of the variable.

TABLE 1-1 Basic Identities of Boolean Algebra

(1) $x + 0 = x$	(2) $x \cdot 0 = 0$
(3) $x + 1 = 1$	(4) $x \cdot 1 = x$
(5) $x + x = x$	(6) $x \cdot x = x$
(7) $x + x' = 1$	(8) $x \cdot x' = 0$
(9) $x + y = y + x$	(10) $xy = yx$
(11) $x + (y + z) = (x + y) + z$	(12) $x(yz) = (xy)z$
(13) $x(y + z) = xy + xz$	(14) $x + yx = (x + y)(x + z)$
(15) $(x + y)' = x'y'$	(16) $(xy)' = x' + y'$
(17) $(x')' = x$	

The identities listed in the table apply to single variables or to Boolean functions expressed in terms of binary variables. For example, consider the following Boolean algebra expression:

$$AB' + C'D + AB' + C'D$$

By letting $x = AB' + C'D$ the expression can be written as $x + x$. From identity 5 in Table 1-1 we find that $x + x = x$. Thus the expression can be reduced to only two terms:

$$AB' + C'D + A'B + C'D = AB' + C'D$$

DeMorgan's theorem

DeMorgan's theorem is very important in dealing with NOR and NAND gates. It states that a NOR gate that performs the $(x + y)'$ function is equivalent

to the function $x'y'$. Similarly, a NAND function can be expressed by either $(xy)'$ or $(x' + y')$. For this reason the NOR and NAND gates have two distinct graphic symbols, as shown in Figs. 1-4 and 1-5. Instead of representing a NOR gate with an OR graphic symbol followed by a circle, we can represent it by an AND graphic symbol preceded by circles in all inputs. The invert-AND symbol for the NOR gate follows from DeMorgan's theorem and from the convention that small circles denote complementation. Similarly, the NAND gate has two distinct symbols, as shown in Fig. 1-5.

To see how Boolean algebra manipulation is used to simplify digital circuits, consider the logic diagram of Fig. 1-6(a). The output of the circuit can be expressed algebraically as follows:

$$F = ABC + ABC' + A'C$$

Each term corresponds to one AND gate, and the OR gate forms the logical sum of the three terms. Two inverters are needed to complement A' and C'. The expression can be simplified using Boolean algebra.

$$F = ABC + ABC' + A'C = AB(C + C') + A'C$$
$$= AB + A'C$$

Note that $(C + C') = 1$ by identity 7 and $AB \cdot 1 = AB$ by identity 4 in Table 1-1.

The logic diagram of the simplified expression is drawn in Fig. 1-6(b). It requires only four gates rather than the six gates used in the circuit of Fig. 1-6(a). The two circuits are equivalent and produce the same truth table relationship between inputs A, B, C and output F.

Figure 1-4   Two graphic symbols for NOR gate.

(a) OR-invert   (b) invert-AND
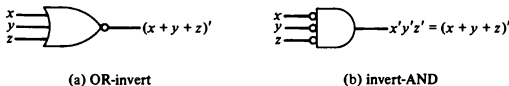
Figure 1-5   Two graphic symbols for NAND gate.

(a) AND-invert   (b) invert-OR
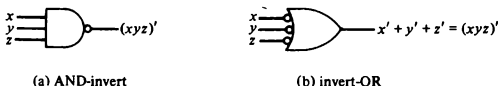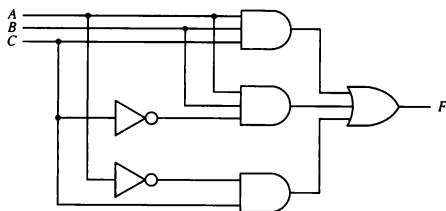
(a) $F = ABC + ABC' + A'C$
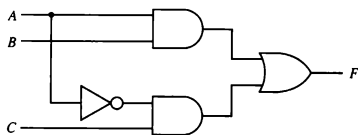
(B) $F = AB + A'C$

Figure 1-6   Two logic diagrams for the same Boolean function.

Complement of a Function

The complement of a function F when expressed in a truth table is obtained by interchanging 1's and 0's in the values of F in the truth table. When the function is expressed in algebraic form, the complement of the function can be derived by means of DeMorgan's theorem. The general form of DeMorgan's theorem can be expressed as follows:

$$(x_1 + x_2 + x_3 + \cdots + x_n)' = x_1' x_2' x_3' \cdots x_n'$$
$$(x_1 x_2 x_3 \cdots x_n)' = x_1' + x_2' + x_3' + \cdots + x_n'$$

From the general DeMorgan's theorem we can derive a simple procedure for obtaining the complement of an algebraic expression. This is done by changing all OR operations to AND operations and all AND operations to OR operations and then complementing each individual letter variable. As an example, consider the following expression and its complement:

$$F = AB + C'D' + B'D$$
$$F' = (A' + B')(C + D)(B + D')$$

The complement expression is obtained by interchanging AND and OR operations and complementing each individual variable. Note that the complement of C' is C.

1-4 Map Simplification

The complexity of the logic diagram that implements a Boolean function is related directly to the complexity of the algebraic expression from which the function is implemented. The truth table representation of a function is unique, but the function can appear in many different forms when expressed algebraically. The expression may be simplified using the basic relations of Boolean algebra. However, this procedure is sometimes difficult because it lacks specific rules for predicting each succeeding step in the manipulative process. The map method provides a simple, straightforward procedure for simplifying Boolean expressions. This method may be regarded as a pictorial arrangement of the truth table which allows an easy interpretation for choosing the minimum number of terms needed to express the function algebraically. The map method is also known as the Karnaugh map or K-map.

minterm   Each combination of the variables in a truth table is called a minterm. For example, the truth table of Fig. 1-3 contains eight minterms. When expressed in a truth table a function of n variables will have 2^n minterms, equivalent to the 2^n binary numbers obtained from n bits. A Boolean function is equal to 1 for some minterms and to 0 for others. The information contained in a truth table may be expressed in compact form by listing the decimal equivalent of those minterms that produce a 1 for the function. For example, the truth table of Fig. 1-3 can be expressed as follows:

$$F(x, y, z) = \Sigma\ (1, 4, 5, 6, 7)$$

The letters in parentheses list the binary variables in the order that they appear in the truth table. The symbol Σ stands for the sum of the minterms that follow in parentheses. The minterms that produce 1 for the function are listed in their decimal equivalent. The minterms missing from the list are the ones that produce 0 for the function.

The map is a diagram made up of squares, with each square representing one minterm. The squares corresponding to minterms that produce 1 for the function are marked by a 1 and the others are marked by a 0 or are left empty. By recognizing various patterns and combining squares marked by 1's in the map, it is possible to derive alternative algebraic expressions for the function, from which the most convenient may be selected.

The maps for functions of two, three, and four variables are shown in Fig. 1-7. The number of squares in a map of n variables is 2^n. The 2^n minterms are listed by an equivalent decimal number for easy reference. The minterm

(a) Two-variable map

(b) Three-variable map
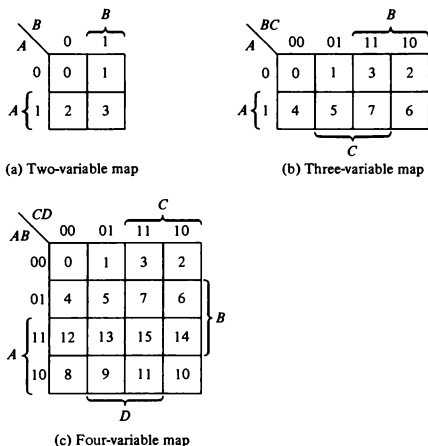
(c) Four-variable map

Figure 1-7 Maps for two-, three-, and four-variable functions.

numbers are assigned in an orderly arrangement such that adjacent squares represent minterms that differ by only one variable. The variable names are listed across both sides of the diagonal line in the corner of the map. The 0's and 1's marked along each row and each column designate the value of the variables. Each variable under brackets contains half of the squares in the map where that variable appears unprimed. The variable appears with a prime (complemented) in the remaining half of the squares.

The minterm represented by a square is determined from the binary assignments of the variables along the left and top edges in the map. For example, minterm 5 in the three-variable map is 101 in binary, which may be obtained from the 1 in the second row concatenated with the 01 of the second column. This minterm represents a value for the binary variables A, B, and C, with A and C being unprimed and B being primed (i.e., $AB'C$). On the other hand, minterm 5 in the four-variable map represents a minterm for four variables. The binary number contains the four bits 0101, and the corresponding term it represents is $A'BC'D$.

adjacent squares   Minterms of adjacent squares in the map are identical except for one variable, which appears complemented in one square and uncomplemented in the adjacent square. According to this definition of adjacency, the squares at the extreme ends of the same horizontal row are also to be considered

adjacent. The same applies to the top and bottom squares of a column. As a result, the four corner squares of a map must also be considered to be adjacent.

A Boolean function represented by a truth table is plotted into the map by inserting 1's in those squares where the function is 1. The squares containing 1's are combined in groups of adjacent squares. These groups must contain a number of squares that is an integral power of 2. Groups of combined adjacent squares may share one or more squares with one or more groups. Each group of squares represents an algebraic term, and the OR of those terms gives the simplified algebraic expression for the function. The following examples show the use of the map for simplifying Boolean functions.

In the first example we will simplify the Boolean function

$$F(A, B, C) = \Sigma \ (3, 4, 6, 7)$$

The three-variable map for this function is shown in Fig. 1-8. There are four squares marked with 1's, one for each minterm that produces 1 for the function. These squares belong to minterms 3, 4, 6, and 7 and are recognized from Fig. 1-7(b). Two adjacent squares are combined in the third column. This column belongs to both B and C and produces the term BC. The remaining two squares with 1's in the two corners of the second row are adjacent and belong to row A and the two columns of C', so they produce the term AC'. The simplified algebraic expression for the function is the OR of the two terms:

$$F = BC + AC'$$

The second example simplifies the following Boolean function:

$$F(A, B, C) = \Sigma \ (0, 2, 4, 5, 6)$$

The five minterms are marked with 1's in the corresponding squares of the three-variable map shown in Fig. 1-9. The four squares in the first and fourth columns are adjacent and represent the term C'. The remaining square marked with a 1 belongs to minterm 5 and can be combined with the square of minterm 4 to produce the term AB'. The simplified function is

$$F = C' + AB'$$

Figure 1-8 Map for $F(A, B, C) = \Sigma \ (3, 4, 6, 7)$.
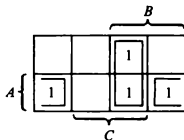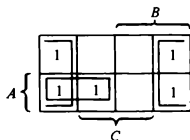
Figure 1-9 Map for $F(A, B, C) = \Sigma(0,2,4,5,6)$.

The third example needs a four-variable map.

$$F(A, B, C, D) = \Sigma(0, 1, 2, 6, 8, 9, 10)$$

The area in the map covered by this four-variable function consists of the squares marked with 1's in Fig. 1-10. The function contains 1's in the four corners that, when taken as a group, give the term $B'D'$. This is possible because these four squares are adjacent when the map is considered with top and bottom or left and right edges touching. The two 1's on the left of the top row are combined with the two 1's on the left of the bottom row to give the term $B'C'$. The remaining 1 in the square of minterm 6 is combined with minterm 2 to give the term $A'CD'$. The simplified function is

$$F = B'D' + B'C' + A'CD'$$

Product-of-Sums Simplification

The Boolean expressions derived from the maps in the preceding examples were expressed in sum-of-products form. The product terms are AND terms and the sum denotes the ORing of these terms. It is sometimes convenient to obtain the algebraic expression for the function in a product-of-sums form. The

Figure 1-10 Map for $F(A, B, C, D) = \Sigma(0,1,2,6,8,9,10)$.
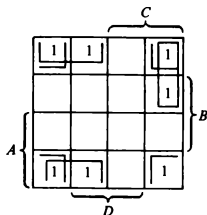
sums are OR terms and the product denotes the ANDing of these terms. With a minor modification, a product-of-sums form can be obtained from a map.

The procedure for obtaining a product-of-sums expression follows from the basic properties of Boolean algebra. The 1's in the map represent the minterms that produce 1 for the function. The squares not marked by 1 represent the minterms that produce 0 for the function. If we mark the empty squares with 0's and combine them into groups of adjacent squares, we obtain the complement of the function, F'. Taking the complement of F' produces an expression for F in product-of-sums form. The best way to show this is by example.

We wish to simplify the following Boolean function in both sum-of-products form and product-of-sums form:

$$F(A, B, C, D) = \Sigma \ (0, 1, 2, 5, 8, 9, 10)$$

The 1's marked in the map of Fig. 1-11 represent the minterms that produce a 1 for the function. The squares marked with 0's represent the minterms not included in F and therefore denote the complement of F. Combining the squares with 1's gives the simplified function in sum-of-products form:
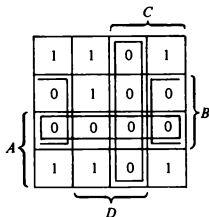
$$F = B'D' + B'C' + A'C'D$$

If the squares marked with 0's are combined, as shown in the diagram, we obtain the simplified complemented function:

$$F' = AB + CD + BD'$$

Taking the complement of F', we obtain the simplified function in product-of-sums form:

$$F = (A' + B')(C' + D')(B' + D)$$

Figure 1-11   Map for $F(A, B, C, D) = \Sigma \ (0,1,2,5,8,9,10)$.

The logic diagrams of the two simplified expressions are shown in Fig. 1-12. The sum-of-products expression is implemented in Fig. 1-12(a) with a group of AND gates, one for each AND term. The outputs of the AND gates are connected to the inputs of a single OR gate. The same function is implemented in Fig. 1-12(b) in product-of-sums form with a group of OR gates, one for each OR term. The outputs of the OR gates are connected to the inputs of a single AND gate. In each case it is assumed that the input variables are directly available in their complement, so inverters are not included. The pattern established in Fig. 1-12 is the general form by which any Boolean function is implemented when expressed in one of the standard forms. AND gates are connected to a single OR gate when in sum-of-products form. OR gates are connected to a single AND gate when in product-of-sums form.

NAND
implementation
A sum-of-products expression can be implemented with NAND gates as shown in Fig. 1-13(a). Note that the second NAND gate is drawn with the graphic symbol of Fig. 1-5(b). There are three lines in the diagram with small circles at both ends. Two circles in the same line designate double complementation, and since $(x')' = x$, the two circles can be removed and the resulting diagram is equivalent to the one shown in Fig. 1-12(a). Similarly, a product-of-sums expression can be implemented with NOR gates as shown in Fig. 1-13(b).

NOR
implementation
The second NOR gate is drawn with the graphic symbol of Fig. 1-4(b). Again the two circles on both sides of each line may be removed, and the diagram so obtained is equivalent to the one shown in Fig. 1-12(b).

Don't-Care Conditions

The 1's and 0's in the map represent the minterms that make the function equal to 1 or 0. There are occasions when it does not matter if the function produces 0 or 1 for a given minterm. Since the function may be either 0 or 1, we say that we don't care what the function output is to be for this minterm. Minterms that
don't-care conditions
may produce either 0 or 1 for the function are said to be don't-care conditions and are marked with an × in the map. These don't-care conditions can be used to provide further simplification of the algebraic expression.

Figure 1-12   Logic diagrams with AND and OR gates.

(a) Sum of products:
$F = B'D' + B'C' + A'C'D$

(b) Product of sums:
$F = (A' + B')(C' + D')(B' + D)$

(a) With NAND gates   (b) With NOR gates
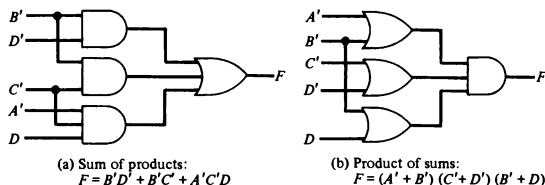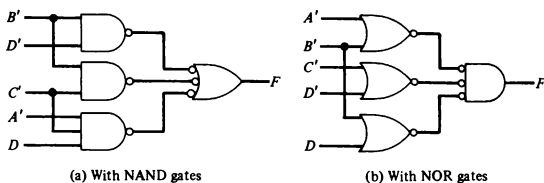
Figure 1-13   Logic diagrams with NAND or NOR gates.

When choosing adjacent squares for the function in the map, the ×'s may be assumed to be either 0 or 1, whichever gives the simplest expression. In addition, an × need not be used at all if it does not contribute to the simplification of the function. In each case, the choice depends only on the simplification that can be achieved. As an example, consider the following Boolean function together with the don't-care minterms:

$$F(A, B, C) = \Sigma \ (0, 2, 6)$$
$$d(A, B, C) = \Sigma \ (1, 3, 5)$$

The minterms listed with F produce a 1 for the function. The don't-care minterms listed with d may produce either a 0 or a 1 for the function. The remaining minterms, 4 and 7, produce a 0 for the function. The map is shown in Fig. 1-14. The minterms of F are marked with 1's, those of d are marked with ×'s, and the remaining squares are marked with 0's. The 1's and ×'s are combined in any convenient manner so as to enclose the maximum number of adjacent squares. It is not necessary to include all or any of the ×'s, but all the 1's must be included. By including the don't-care minterms 1 and 3 with the 1's in the first row we obtain the term A'. The remaining 1 for minterm 6 is combined with minterm 2 to obtain the term BC'. The simplified expression is

$$F = A' + BC'$$

Note that don't-care minterm 5 was not included because it does not contribute to the simplification of the expression. Note also that if don't-care minterms 1 and 3 were not included with the 1's, the simplified expression for F would have been

$$F = A'C' + BC'$$

This would require two AND gates and an OR gate, as compared to the expression obtained previously, which requires only one AND and one OR gate.
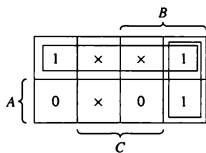
Figure 1-14   Example of map with don't-care conditions.

The function is determined completely once the ×'s are assigned to the 1's or 0's in the map. Thus the expression

$$F = A' + BC'$$

represents the Boolean function

$$F(A, B, C) = \sum (0, 1, 2, 3, 6)$$

It consists of the original minterms 0, 2, and 6 and the don't-care minterms 1 and 3. Minterm 5 is not included in the function. Since minterms 1, 3, and 5 were specified as being don't-care conditions, we have chosen minterms 1 and 3 to produce a 1 and minterm 5 to produce a 0. This was chosen because this assignment produces the simplest Boolean expression.

1-5   Combinational Circuits

block diagram

A combinational circuit is a connected arrangement of logic gates with a set of inputs and outputs. At any given time, the binary values of the outputs are a function of the binary combination of the inputs. A block diagram of a combinational circuit is shown in Fig. 1-15. The n binary input variables come from an external source, the m binary output variables go to an external destination, and in between there is an interconnection of logic gates. A combinational circuit transforms binary information from the given input data to the required output data. Combinational circuits are employed in digital computers for generating binary control decisions and for providing digital components required for data processing.

A combinational circuit can be described by a truth table showing the binary relationship between the n input variables and the m output variables. The truth table lists the corresponding output binary values for each of the 2^n input combinations. A combinational circuit can also be specified with m Boolean functions, one for each output variable. Each output function is expressed in terms of the n input variables.

Figure 1-15   Block diagram of a combinational circuit.

analysis

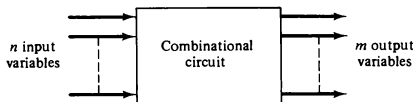The analysis of a combinational circuit starts with a given logic circuit diagram and culminates with a set of Boolean functions or a truth table. If the digital circuit is accompanied by a verbal explanation of its function, the Boolean functions or the truth table is sufficient for verification. If the function of the circuit is under investigation, it is necessary to interpret the operation of the circuit from the derived Boolean functions or the truth table. The success of such investigation is enhanced if one has experience and familiarity with digital circuits. The ability to correlate a truth table or a set of Boolean functions with an information-processing task is an art that one acquires with experience.

design

The design of combinational circuits starts from the verbal outline of the problem and ends in a logic circuit diagram. The procedure involves the following steps:

1. The problem is stated.
2. The input and output variables are assigned letter symbols.
3. The truth table that defines the relationship between inputs and outputs is derived.
4. The simplified Boolean functions for each output are obtained.
5. The logic diagram is drawn.

To demonstrate the design of combinational circuits, we present two examples of simple arithmetic circuits. These circuits serve as basic building blocks for the construction of more complicated arithmetic circuits.

Half-Adder

The most basic digital arithmetic circuit is the addition of two binary digits. A combinational circuit that performs the arithmetic addition of two bits is called a half-adder. One that performs the addition of three bits (two significant bits and a previous carry) is called a full-adder. The name of the former stems from the fact that two half-adders are needed to implement a full-adder.

The input variables of a half-adder are called the augend and addend bits. The output variables the sum and carry. It is necessary to specify two output variables because the sum of $1 + 1$ is binary 10, which has two digits. We assign symbols x and y to the two input variables, and S (for sum) and C

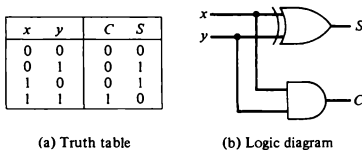| x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(a) Truth table   (b) Logic diagram

Figure 1-16   Half-adder.

(for carry) to the two output variables. The truth table for the half-adder is shown in Fig. 1-16(a). The C output is 0 unless both inputs are 1. The S output represents the least significant bit of the sum. The Boolean functions for the two outputs can be obtained directly from the truth table:

$$S = x'y + xy' = x \oplus y$$
$$C = xy$$

The logic diagram is shown in Fig. 1-16(b). It consists of an exclusive-OR gate and an AND gate.

Full-Adder

A full-adder is a combinational circuit that forms the arithmetic sum of three input bits. It consists of three inputs and two outputs. Two of the input variables, denoted by x and y, represent the two significant bits to be added. The third input, z, represents the carry from the previous lower significant position. Two outputs are necessary because the arithmetic sum of three binary digits ranges in value from 0 to 3, and binary 2 or 3 needs two digits. The two outputs are designated by the symbols S (for sum) and C (for carry). The binary variable S gives the value of the least significant bit of the sum. The binary variable C gives the output carry. The truth table of the full-adder is shown in Table 1-2. The eight rows under the input variables designate all possible combinations that the binary variables may have. The value of the output variables are determined from the arithmetic sum of the input bits. When all input bits are 0, the output is 0. The S output is equal to 1 when only one input is equal to 1 or when all three inputs are equal to 1. The C output has a carry of 1 if two or three inputs are equal to 1.

The maps of Fig. 1-17 are used to find algebraic expressions for the two output variables. The 1's in the squares for the maps of S and C are determined directly from the minterms in the truth table. The squares with 1's for the S output do not combine in groups of adjacent squares. But since the output is 1 when an odd number of inputs are 1, S is an odd function and represents

TABLE 1-2 Truth Table for Full-Adder

Inputs			Outputs	
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

the exclusive-OR relation of the variables (see the discussion at the end of Sec. 1-2). The squares with 1's for the C output may be combined in a variety of ways. One possible expression for C is

$$C = xy + (x'y + xy')z$$

Realizing that $x'y + xy' = x \oplus y$ and including the expression for output S, we obtain the two Boolean expressions for the full-adder:

$$S = x \oplus y \oplus z$$
$$C = xy + (x \oplus y)z$$

The logic diagram of the full-adder is drawn in Fig. 1-18. Note that the full-adder circuit consists of two half-adders and an OR gate. When used in subsequent chapters, the full-adder (FA) will be designated by a block diagram as shown in Fig. 1-18(b).

Figure 1-17   Maps for full-adder.

$$S = x'y'z + x'yz' + xy'z' + xyz$$
$$= x \oplus y \oplus z$$

$$C = xy + xz + yz$$
$$= xy + (x'y + xy')z$$

(a) Logic diagram

(b) Block diagram
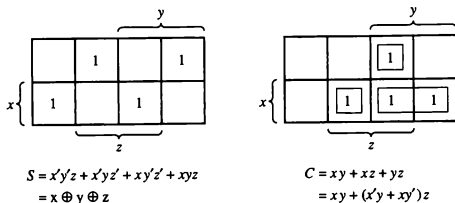
Figure 1-18 Full-adder circuit.

1-6 Flip-Flops

The digital circuits considered thus far have been combinational, where the outputs at any given time are entirely dependent on the inputs that are present at that time. Although every digital system is likely to have a combinational circuit, most systems encountered in practice also include storage elements, which require that the system be described in terms of sequential circuits. The most common type of sequential circuit is the synchronous type. Synchronous sequential circuits employ signals that affect the storage elements only at discrete instants of time. Synchronization is achieved by a timing device called a clock pulse generator that produces a periodic train of *clock pulses*. The clock pulses are distributed throughout the system in such a way that storage elements are affected only with the arrival of the synchronization pulse. Clocked synchronous sequential circuits are the type most frequently encountered in practice. They seldom manifest instability problems and their timing is easily broken down into independent discrete steps, each of which may be considered separately.
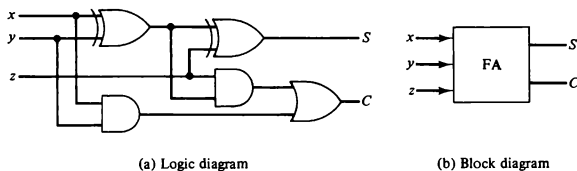
clocked sequential circuit

The storage elements employed in clocked sequential circuits are called flip-flops. A flip-flop is a binary cell capable of storing one bit of information. It has two outputs, one for the normal value and one for the complement value of the bit stored in it. A flip-flop maintains a binary state until directed by a clock pulse to switch states. The difference among various types of flip-flops is in the number of inputs they possess and in the manner in which the inputs affect the binary state. The most common types of flip-flops are presented below.

SR Flip-Flop

The graphic symbol of the SR flip-flop is shown in Fig. 1-19(a). It has three inputs, labeled S (for set), R (for reset), and C (for clock). It has an output Q and sometimes the flip-flop has a complemented output, which is indicated with a small circle at the other output terminal. There is an arrowhead-shaped symbol in front of the letter C to designate a *dynamic input*. The dynamic
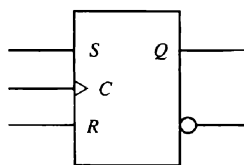
	S	R	Q (t + 1)	
	0	0	Q (t)	No change
	0	1	0	Clear to 0
	1	0	1	Set to 1
	1	1	?	Indeterminate

(a) Graphic symbol   (b) Characteristic table

Figure 1-19 SR flip-flop.

indicator symbol denotes the fact that the flip-flop responds to a positive transition (from 0 to 1) of the input clock signal.

The operation of the SR flip-flop is as follows. If there is no signal at the clock input C, the output of the circuit cannot change irrespective of the values at inputs S and R. Only when the clock signal changes from 0 to 1 can the output be affected according to the values in inputs S and R. If $S = 1$ and $R = 0$ when C changes from 0 to 1, output Q is set to 1. If $S = 0$ and $R = 1$ when C changes from 0 to 1, output Q is cleared to 0. If both S and R are 0 during the clock transition, the output does not change. When both S and R are equal to 1, the output is unpredictable and may go to either 0 or 1, depending on internal timing delays that occur within the circuit.

The characteristic table shown in Fig. 1-19(b) summarizes the operation of the SR flip-flop in tabular form. The S and R columns give the binary values of the two inputs. $Q(t)$ is the binary state of the Q output at a given time (referred to as *present state*). $Q(t + 1)$ is the binary state of the Q output after the occurrence of a clock transition (referred to as *next state*). If $S = R = 0$, a clock transition produces no change of state [i.e., $Q(t + 1) = Q(t)$]. If $S = 0$ and $R = 1$, the flip-flop goes to the 0 (clear) state. If $S = 1$ and $R = 0$, the flip-flop goes to the 1 (set) state. The SR flip-flop should not be pulsed when $S = R = 1$ since it produces an indeterminate next state. This indeterminate condition makes the SR flip-flop difficult to manage and therefore it is seldom used in practice.

D Flip-Flop

The D (data) flip-flop is a slight modification of the SR flip-flop. An SR flip-flop is converted to a D flip-flop by inserting an inverter between S and R and assigning the symbol D to the single input. The D input is sampled during the occurrence of a clock transition from 0 to 1. If $D = 1$, the output of the flip-flop goes to the 1 state, but if $D = 0$, the output of the flip-flop goes to the 0 state.

The graphic symbol and characteristic table of the D flip-flop are shown in Fig. 1-20. From the characteristic table we note that the next state $Q(t + 1)$

D	Q(t + 1)	
0	0	Clear to 0
1	1	Set to 1

(a) Graphic symbol   (b) Characteristic table
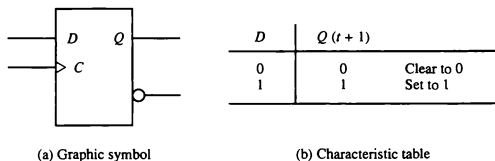
Figure 1-20   D flip-flop.

is determined from the D input. The relationship can be expressed by a characteristic equation:

$$Q(t + 1) = D$$

This means that the Q output of the flip-flop receives its value from the D input every time that the clock signal goes through a transition from 0 to 1.

Note that no input condition exists that will leave the state of the D flip-flop unchanged. Although a D flip-flop has the advantage of having only one input (excluding C), it has the disadvantage that its characteristic table does not have a "no change" condition $Q(t + 1) = Q(t)$. The "no change" condition can be accomplished either by disabling the clock signal or by feeding the output back into the input, so that clock pulses keep the state of the flip-flop unchanged.

JK Flip-Flop

A JK flip-flop is a refinement of the SR flip-flop in that the indeterminate condition of the SR type is defined in the JK type. Inputs J and K behave like inputs S and R to set and clear the flip-flop, respectively. When inputs J and K are both equal to 1, a clock transition switches the outputs of the flip-flop to their complement state.

The graphic symbol and characteristic table of the JK flip-flop are shown in Fig. 1-21. The J input is equivalent to the S (set) input of the SR flip-flop, and the K input is equivalent to the R (clear) input. Instead of the indeterminate condition, the JK flip-flop has a complement condition $Q(t + 1) = Q'(t)$ when both J and K are equal to 1.

T Flip-Flop

Another type of flip-flop found in textbooks is the T (toggle) flip-flop. This flip-flop, shown in Fig. 1-22, is obtained from a JK type when inputs J and K are connected to provide a single input designated by T. The T flip-flop
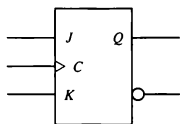
J	K	$Q(t+1)$	
0	0	$Q(t)$	No change
0	1	0	Clear to 0
1	0	1	Set to 1
1	1	$Q'(t)$	Complement

(a) Graphic symbol   (b) Characteristic table

Figure 1-21   *JK* flip-flop.

T	$Q(t+1)$	
0	$Q(t)$	No change
1	$Q'(t)$	Complement

(a) Graphic symbol   (b) Characteristic table

Figure 1-22   *T* flip-flop.

therefore has only two conditions. When $T = 0$ ($J = K = 0$) a clock transition does not change the state of the flip-flop. When $T = 1$ ($J = K = 1$) a clock transition complements the state of the flip-flop. These conditions can be expressed by a characteristic equation:
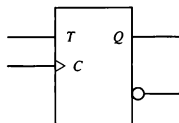
$$Q(t + 1) = Q(t) \oplus T$$

Edge-Triggered Flip-Flops

The most common type of flip-flop used to synchronize the state change during a clock pulse transition is the edge-triggered flip-flop. In this type of flip-flop, output transitions occur at a specific level of the clock signal (positive-edge transition), and input level exceeds this threshold level, the inputs are locked out so that the flip-flop is unresponsive to further changes in inputs until the clock pulse returns to 0 and another pulse occurs. Some edge-triggered flip-flops cause a transition on the rising edge of the clock signal (positive-edge transition), and others cause a transition on the falling edge (negative-edge transition).

Figure 1-23(a) shows the clock pulse signal in a positive-edge-triggered D flip-flop. The value in the D input is transferred to the Q output when the clock makes a positive transition. The output cannot change when the clock is in the 1 level, in the 0 level, or in a transition from the 1 level to the 0 level.
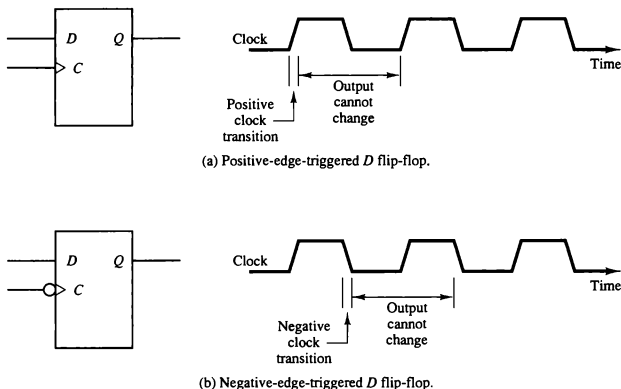
clock pulses

(a) Positive-edge-triggered D flip-flop.

(b) Negative-edge-triggered D flip-flop.

Figure 1-23   Edge-triggered flip-flop.

The effective positive clock transition includes a minimum time called the *setup time* in which the D input must remain at a constant value before the transition, and a definite time called the *hold time* in which the D input must not change after the positive transition. The effective positive transition is usually a very small fraction of the total period of the clock pulse.

Figure 1-23(b) shows the corresponding graphic symbol and timing diagram for a negative-edge-triggered D flip-flop. The graphic symbol includes a negation small circle in front of the dynamic indicator at the C input. This denotes a negative-edge-triggered behavior. In this case the flip-flop responds to a transition from the 1 level to the 0 level of the clock signal.

master-slave flip-flop   Another type of flip-flop used in some systems is the master-slave flip-flop. This type of circuit consists of two flip-flops. The first is the master, which responds to the positive level of the clock, and the second is the slave, which responds to the negative level of the clock. The result is that the output changes during the 1-to-0 transition of the clock signal. The trend is away from the use of master-slave flip-flops and toward edge-triggered flip-flops.

Flip-flops available in integrated circuit packages will sometimes provide special input terminals for setting or clearing the flip-flop asynchronously. These inputs are usually called "preset" and "clear." They affect the flip-flop on a negative level of the input signal without the need of a clock pulse. These inputs are useful for bringing the flip-flops to an initial state prior to its clocked operation.

Excitation Tables

The characteristic tables of flip-flops specify the next state when the inputs and the present state are known. During the design of sequential circuits we usually know the required transition from present state to next state and wish to find the flip-flop input conditions that will cause the required transition. For this reason we need a table that lists the required input combinations for a given change of state. Such a table is called a flip-flop excitation table.

Table 1-3 lists the excitation tables for the four types of flip-flops. Each table consists of two columns, $Q(t)$ and $Q(t + 1)$, and a column for each input to show how the required transition is achieved. There are four possible transitions from present state $Q(t)$ to next state $Q(t + 1)$. The required input conditions for each of these transitions are derived from the information available in the characteristic tables. The symbol × in the tables represents a don't-care condition; that is, it does not matter whether the input to the flip-flop is 0 or 1.

TABLE 1-3 Excitation Table for Four Flip-Flops

SR flip-flop				D flip-flop		
$Q(t)$	$Q(t + 1)$	S	R	$Q(t)$	$Q(t + 1)$	D
0	0	0	×	0	0	0
0	1	1	0	0	1	1
1	0	0	1	1	0	0
1	1	×	0	1	1	1

JK flip-flop				T flip-flop		
$Q(t)$	$Q(t + 1)$	J	K	$Q(t)$	$Q(t + 1)$	T
0	0	0	×	0	0	0
0	1	1	×	0	1	1
1	0	×	1	1	0	1
1	1	×	0	1	1	0

The reason for the don't-care conditions in the excitation tables is that there are two ways of achieving the required transition. For example, in a JK flip-flop, a transition from present state of 0 to a next state of 0 can be achieved by having inputs J and K equal to 0 (to obtain no change) or by letting $J = 0$ and $K = 1$ to clear the flip-flop (although it is already cleared). In both cases J must be 0, but K is 0 in the first case and 1 in the second. Since the required transition will occur in either case, we mark the K input with a don't-care ×

and let the designer choose either 0 or 1 for the K input, whichever is more convenient.
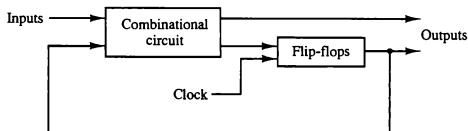
1-7 Sequential Circuits

A sequential circuit is an interconnection of flip-flops and gates. The gates by themselves constitute a combinational circuit, but when included with the flip-flops, the overall circuit is classified as a sequential circuit. The block diagram of a clocked sequential circuit is shown in Fig. 1-24. It consists of a combinational circuit and a number of clocked flip-flops. In general, any number or type of flip-flops may be included. As shown in the diagram, the combinational circuit block receives binary signals from external inputs and from the outputs of flip-flops. The outputs of the combinational circuit go to external outputs and to inputs of flip-flops. The gates in the combinational circuit determine the binary value to be stored in the flip-flops after each clock transition. The outputs of flip-flops, in turn, are applied to the combinational circuit inputs and determine the circuit's behavior. This process demonstrates that the external outputs of a sequential circuit are functions of both external inputs and the present state of the flip-flops. Moreover, the next state of flip-flops is also a function of their present state and external inputs. Thus a sequential circuit is specified by a time sequence of external inputs, external outputs, and internal flip-flop binary states.

Flip-Flop Input Equations

An example of a sequential circuit is shown in Fig. 1-25. It has one input variable x, one output variable y, and two clocked D flip-flops. The AND gates, OR gates, and inverter form the combinational logic part of the circuit. The interconnections among the gates in the combinational circuit can be specified by a set of Boolean expressions. The part of the combinational circuit that generates the inputs to flip-flops are described by a set of Boolean expressions *input equation* called flip-flop input equations. We adopt the convention of using the flip-flop input symbol to denote the input equation variable name and a subscript to

Figure 1-24 Block diagram of a clocked synchronous sequential circuit.
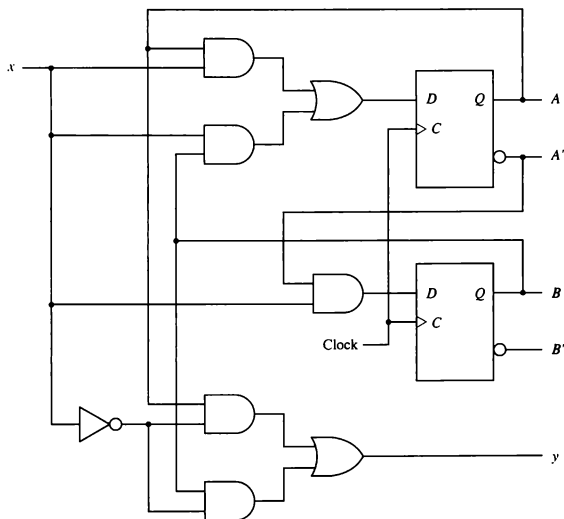
Figure 1-25   Example of a sequential circuit.

designate the symbol chosen for the output of the flip-flop. Thus, in Fig. 1-25, we have two input equations, designated D_A and D_B. The first letter in each symbol denotes the D input of a D flip-flop. The subscript letter is the symbol name of the flip-flop. The input equations are Boolean functions for flip-flop input variables and can be derived by inspection of the circuit. Since the output of the OR gate is connected to the D input of flip-flop A, we write the first input equation as

$$D_A = Ax + Bx$$

where A and B are the outputs of the two flip-flops and x is the external input. The second input equation is derived from the single AND gate whose output is connected to the D input of flip-flop B:

$$D_B = A'x$$

The sequential circuit also has an external output, which is a function of the input variable and the state of the flip-flops. This output can be specified algebraically by the expression

$$y = Ax' + Bx'$$

From this example we note that a flip-flop input equation is a Boolean expression for a combinational circuit. The subscripted variable is a binary variable name for the output of a combinational circuit. This output is always connected to a flip-flop input.

State Table

The behavior of a sequential circuit is determined from the inputs, the outputs, and the state of its flip-flops. Both the outputs and the next state are a function of the inputs and the present state. A sequential circuit is specified by a state table that relates outputs and next states as a function of inputs and present states. In clocked sequential circuits, the transition from present state to next state is activated by the presence of a clock signal.

The state table for the circuit of Fig. 1-25 is shown in Table 1-4. The table consists of four sections, labeled *present state, input, next state,* and *output*. The *present state* section shows the states of flip-flops A and B at any given time t. The input section gives a value of x for each possible present state. The *next state* section shows the states of the flip-flops one clock period later at time $t + 1$. The output section gives the value of y for each present state and input condition.

The derivation of a state table consists of first listing all possible binary combinations of present state and inputs. In this case we have eight binary combinations from 000 to 111. The next-state values are then determined from the logic diagram or from the input equations. The input equation for flip-flop A is

$$D_A = Ax + Bx$$

The next-state value of a each flip-flop is equal to its D input value in the present state. The transition from present state to next state occurs after application of a clock signal. Therefore, the next state of A is equal to 1 when the present state and input values satisfy the conditions $Ax = 1$ or $Bx = 1$, which makes D_A equal 1. This is shown in the state table with three 1's under the column for next state of A. Similarly, the input equation for flip-flop B is

$$D_B = A'x$$

The next state of B in the state table is equal to 1 when the present state of A is 0 and input x is equal to 1. The output column is derived from the output equation

$$y = Ax' + Bx'$$

TABLE 1-4 State Table for Circuit of Fig. 1-25

Present state		Input	Next state		Output
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

state table

The state table of any sequential circuit is obtained by the procedure used in this example. In general, a sequential circuit with m flip-flops, n input variables, and p output variables will contain m columns for present state, n columns for inputs, m columns for next state, and p columns for outputs. The present state and input columns are combined and under them we list the 2^{m+n} binary combinations from 0 through $2^{m+n} - 1$. The next-state and output columns are functions of the present state and input values and are derived directly from the circuit or the Boolean equations that describe the circuit.

State Diagram

state diagram

The information available in a state table can be represented graphically in a state diagram. In this type of diagram, a state is represented by a circle, and the transition between states is indicated by directed lines connecting the circles. The state diagram of the sequential circuit of Fig. 1-25 is shown in Fig. 1-26. The state diagram provides the same information as the state table and is obtained directly from Table 1-4. The binary number inside each circle identifies the state of the flip-flops. The directed lines are labeled with two binary numbers separated by a slash. The input value during the present state is labeled first and the number after the slash gives the output during the present state. For example, the directed line from state 00 to 01 is labeled 1/0, meaning that when the sequential circuit is in the present state 00 and the input
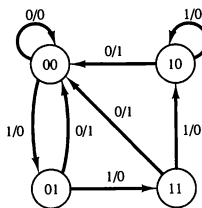
Figure 1-26   State diagrams of sequential circuit.

is 1, the output is 0. After a clock transition, the circuit goes to the next state 01. The same clock transition may change the input value. If the input changes to 0, the output becomes 1, but if the input remains at 1, the output stays at 0. This information is obtained from the state diagram along the two directed lines emanating from the circle representing state 01. A directed line connecting a circle with itself indicates that no change of state occurs.

There is no difference between a state table and a state diagram except in the manner of representation. The state table is easier to derive from a given logic diagram and the state diagram follows directly from the state table. The state diagram gives a pictorial view of state transitions and is the form suitable for human interpretation of the circuit operation. For example, the state diagram of Fig. 1-26 clearly shows that starting from state 00, the output is 0 as long as the input stays at 1. The first 0 input after a string of 1's gives an output of 1 and transfers the circuit back to the initial state 00.

Design Example

The procedure for designing sequential circuits will be demonstrated by a specific example. The design procedure consists of first translating the circuit specifications into a state diagram. The state diagram is then converted into a state table. From the state table we obtain the information for obtaining the logic circuit diagram.

We wish to design a clocked sequential circuit that goes through a sequence of repeated binary states 00, 01, 10, and 11 when an external input x is equal to 1. The state of the circuit remains unchanged when $x = 0$. This type *binary counter* of circuit is called a 2-bit binary counter because the state sequence is identical to the count sequence of two binary digits. Input x is the control variable that specifies when the count should proceed.

The binary counter needs two flip-flops to represent the two bits. The state diagram for the sequential circuit is shown in Fig. 1-27. The diagram is drawn to show that the states of the circuit follow the binary count as long as
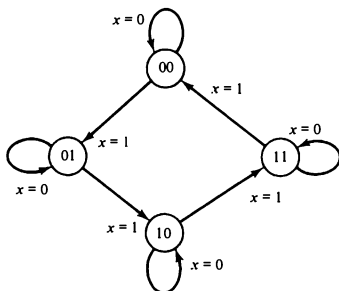
Figure 1-27   State diagram for binary counter.

$x = 1$. The state following 11 is 00, which causes the count to be repeated. If $x = 0$, the state of the circuit remains unchanged. This sequential circuit has no external outputs, and therefore only the input value is labeled in the diagram. The state of the flip-flops is considered as the outputs of the counter.

We have already assigned the symbol x to the input variable. We now assign the symbols A and B to the two flip-flop outputs. The next state of A and B, as a function of the present state and input x, can be transferred from the state diagram into a state table. The first five columns of Table 1-5 constitute the state table. The entries for this table are obtained directly from the state diagram.

excitation table   The excitation table of a sequential circuit is an extension of the state table. This extension consists of a list of flip-flop input excitations that will cause the

TABLE 1-5 Excitation Table for Binary Counter

Present state		Input	Next state		Flip-flop inputs			
A	B	x	A	B	J_A	K_A	J_B	K_B
0	0	0	0	0	0	×	0	×
0	0	1	0	1	0	×	1	×
0	1	0	0	1	0	×	×	0
0	1	1	1	0	1	×	×	1
1	0	0	1	0	×	0	0	×
1	0	1	1	1	×	0	1	×
1	1	0	1	1	×	0	×	0
1	1	1	0	0	×	1	×	1

required state transitions. The flip-flop input conditions are a function of the type of flip-flop used. If we employ JK flip-flops, we need columns for the J and K inputs of each flip-flop. We denote the inputs of flip-flop A by J_A and K_A, and those of flip-flop B by J_B and K_B.

The excitation table for the JK flip-flop specified in Table 1-3 is now used to derive the excitation table of the sequential circuit. For example, in the first row of Table 1-5, we have a transition for flip-flop A from 0 in the present state to 0 in the next state. In Table 1-3 we find that a transition of states from $Q(t) = 0$ to $Q(t + 1) = 0$ in a JK flip-flop requires that input $J = 0$ and input $K = \times$. So 0 and $\times$ are copied in the first row under J_A and K_A, respectively. Since the first row also shows a transition for flip-flop B from 0 in the present state to 0 in the next state, 0 and $\times$ are copied in the first row under J_B and K_B. The second row of Table 1-5 shows a transition for flip-flop B from 0 in the present state to 1 in the next state. From Table 1-3 we find that a transition from $Q(t) = 0$ to $Q(t + 1) = 1$ requires that input $J = 1$ and input $K = \times$. So 1 and $\times$ are copied in the second row under J_B and K_B, respectively. This process is continued for each row of the table and for each flip-flop, with the input conditions as specified in Table 1-3 being copied into the proper row of the particular flip-flop being considered.

Let us now consider the information available in an excitation table such as Table 1-5. We know that a sequential circuit consists of a number of flip-flops and a combinational circuit. From the block diagram of Fig. 1-24, we note that the outputs of the combinational circuit must go to the four flip-flop inputs J_A, K_A, J_B, and K_B. The inputs to the combinational circuit are the external input x and the present-state values of flip-flops A and B. Moreover, the Boolean functions that specify a combinational circuit are derived from a truth table that shows the input–output relationship of the circuit. The entries that list the combinational circuit inputs are specified under the "present state" and "input" columns in the excitation table. The combinational circuit outputs are specified under the "flip-flop inputs" columns. Thus an excitation table transforms a state diagram to a truth table needed for the design of the combinational circuit part of the sequential circuit.

The simplified Boolean functions for the combinational circuit can now be derived. The inputs are the variables A, B, and x. The outputs are the variables J_A, K_A, J_B, and K_B. The information from the excitation table is transferred into the maps of Fig. 1-28, where the four simplified flip-flop input equations are derived:

$$J_A = Bx \qquad K_A = Bx$$
$$J_B = x \qquad K_B = x$$

The logic diagram is drawn in Fig. 1-29 and consists of two JK flip-flops and an AND gate. Note that inputs J and K determine the next state of the counter when a clock signal occurs. If both J and K are equal to 0, a clock signal will
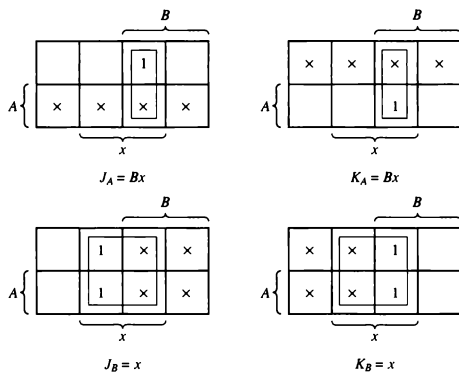
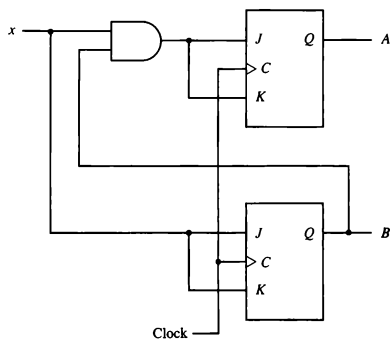Figure 1-28   Maps for combinational circuit of counter.

Figure 1-29   Logic diagram of a 2-bit binary counter.

have no effect; that is, the state of the flip-flops will not change. Thus when $x = 0$, all four inputs of the flip-flops are equal to 0 and the state of the flip-flops remains unchanged even though clock pulses are applied continuously.

Design Procedure

The design of sequential circuits follows the outline described in the preceding example. The behavior of the circuit is first formulated in a state diagram. The number of flip-flops needed for the circuit is determined from the number of bits listed within the circles of the state diagram. The number of inputs for the circuit is specified along the directed lines between the circles. We then assign letters to designate all flip-flops and input and output variables and proceed to obtain the state table.

For m flip-flops and n inputs, the state table will consist of m columns for the present state, n columns for the inputs, and m columns for the next state. The number of rows in the table will be up to 2^{m+n}, one row for each binary combination of present state and inputs. For each row we list the next state as specified by the state diagram. Next, the flip-flop type to be used in the circuit is chosen. The state table is then extended into an excitation table by including columns for each input of each flip-flop. The excitation table for the type of flip-flop in use can be found in Table 1-3. From the information available in this table and by inspecting present state-to-next state transitions in the state table, we obtain the information for the flop-flop input conditions in the excitation table.

The truth table for the combinational circuit part of the sequential circuit is available in the excitation table. The present-state and input columns constitute the inputs in the truth table. The flip-flop input conditions constitute the outputs in the truth table. By means of map simplification we obtain a set of flip-flop input equations for the combinational circuit. Each flip-flop input equation specifies a logic diagram whose output must be connected to one of the flip-flop inputs. The combinational circuit so obtained, together with the flip-flops, constitutes the sequential circuit.

The outputs of flip-flops are often considered to be part of the outputs of the sequential circuit. However, the combinational circuit may also contain external outputs. In such a case the Boolean functions for the external outputs are derived from the state table by combinational circuit design techniques.

A set of flip-flop input equations specifies a sequential circuit in algebraic form. The procedure for obtaining the logic diagram from a set of flip-flop input equations is a straightforward process. First draw the flip-flops and label all their inputs and outputs. Then draw the combinational circuit from the Boolean expressions given by the flip-flop input equations. Finally, connect outputs of flip-flops to inputs in the combinational circuit and outputs of the combinational circuit to flip-flop inputs.

$$\boxed{\text{PROBLEMS}}$$

1-1. Determine by means of a truth table the validity of DeMorgan's theorem for three variables: $(ABC)' = A' + B' + C'$.

1-2. List the truth table of a three-variable exclusive-OR (odd) function: $x = A \oplus B \oplus C$.

1-3. Simplify the following expressions using Boolean algebra.
   a. $A + AB$
   b. $AB + AB'$
   c. $A'BC + AC$
   d. $A'B + ABC' + ABC$

1-4. Simplify the following expressions using Boolean algebra.
   a. $AB + A(CD + CD')$
   b. $(BC' + A'D)(AB' + CD')$

1-5. Using DeMorgan's theorem, show that:
   a. $(A + B)'(A' + B')' = 0$
   b. $A + A'B + A'B' = 1$

1-6. Given the Boolean expression $F = x'y + xyz'$:
   a. Derive an algebraic expression for the complement F'.
   b. Show that $F \cdot F' = 0$.
   c. Show that $F + F' = 1$.

1-7. Given the Boolean function

$$F = xy'z + x'y'z + xyz$$

   a. List the truth table of the function.
   b. Draw the logic diagram using the original Boolean expression.
   c. Simplify the algebraic expression using Boolean algebra.
   d. List the truth table of the function from the simplified expression and show that it is the same as the truth table in part (a).
   e. Draw the logic diagram from the simplified expression and compare the total number of gates with the diagram of part (b).

1-8. Simplify the following Boolean functions using three-variable maps.
   a. $F(x, y, z) = \Sigma (0, 1, 5, 7)$
   b. $F(x, y, z) = \Sigma (1, 2, 3, 6, 7)$
   c. $F(x, y, z) = \Sigma (3, 5, 6, 7)$
   d. $F(A, B, C) = \Sigma (0, 2, 3, 4. 6)$

1-9. Simplify the following Boolean functions using four-variable maps.
   a. $F(A, B, C, D) = \Sigma (4, 6, 7, 15)$
   b. $F(A, B, C, D) = \Sigma (3, 7, 11, 13, 14, 15)$
   c. $F(A, B, C, D) = \Sigma (0, 1, 2, 4, 5, 7, 11, 15)$
   d. $F(A, B, C, D) = \Sigma (0, 2, 4, 5, 6, 7, 8, 10, 13, 15)$

1-10.   Simplify the following expressions in (1) sum-of-products form and (2) product-of-sums form.
a. $x'z' + y'z' + yz' + xy$
b. $AC' + B'D + A'CD + ABCD$

1-11.   Simplify the following Boolean function in sum-of-products form by means of a four-variable map. Draw the logic diagram with (a) AND-OR gates; (b) NAND gates.

$$F(A, B, C, D) = \Sigma\ (0, 2, 8, 9, 10, 11, 14, 15)$$

1-12.   Simplify the following Boolean function in product-of-sums form by means of a four-variable map. Draw the logic diagram with (a) OR-AND gates; (b) NOR gates.

$$F(w, x, y, z) = \Sigma\ (2, 3, 4, 5, 6, 7, 11, 14, 15)$$

1-13.   Simplify the Boolean function F together with the don't-care conditions d in (1) sum-of-products form and (2) product-of-sums form.

$$F(w, x, y, z) = \Sigma\ (0, 1, 2, 3, 7, 8, 10)$$
$$d(w, x, y, z) = \Sigma\ (5, 6, 11, 15)$$

1-14.   Using Table 1-2, derive the Boolean expression for the S (sum) output of the full-adder in sum-of-products form. Then by algebraic manipulation show that S can be expressed as the exclusive-OR of the three input variables.

$$S = x \oplus y \oplus z$$

1-15.   A majority function is generated in a combinational circuit when the output is equal to 1 if the input variables have more 1's than 0's. The output is 0 otherwise. Design a three-input majority function.

1-16.   Design a combinational circuit with three inputs x, y, z and three outputs A, B, C. When the binary input is 0, 1, 2, or 3, the binary output is one greater than the input. When the binary input is 4, 5, 6, or 7, the binary output is one less than the input.

1-17.   Show that a JK flip-flop can be converted to a D flip-flop with an inverter between the J and K inputs.

1-18.   Using the information from the characteristic table of the JK flip-flop listed in Fig. 1-21(b), derive the excitation table for the JK flip-flop and compare your answer with Table 1-3.

1-19.   A sequential circuit has two D flip-flops A and B, two inputs x and y, and one output z. The flip-flop input equations and the circuit output are as follows:

$$D_A = x'y + xA$$
$$D_B = x'B + xA$$
$$z = B$$

    a. Draw the logic diagram of the circuit.

    b. Tabulate the state table.

1-20. Design a 2-bit count-down counter. This is a sequential circuit with two flip-flops and one input x. When $x = 0$, the state of the flip-flops does not change. When $x = 1$, the state sequence is 11, 10, 01, 00, 11, and repeat.

1-21. Design a sequential circuit with two JK flip-flops A and B and two inputs E and x. If $E = 0$, the circuit remains in the same state regardless of the value of x. When $E = 1$ and $x = 1$, the circuit goes through the state transitions from 00 to 01 to 10 to 11 back to 00, and repeat. When $E = 1$ and $x = 0$, the circuit goes through the state transitions from 00 to 11 to 10 to 01 back to 00, and repeat.

REFERENCES

1. Hill, F. J., and G. R. Peterson, *Introduction to Switching Theory and Logical Design*, 3rd ed. New York: John Wiley, 1981.

2. Mano, M. M., *Digital Design*, 2nd ed. Englewood Cliffs, NJ: Prentice Hall, 1991.

3. Roth, C. H., *Fundamentals of Logic Design*, 3rd ed. St. Paul, MN: West Publishing, 1985.

4. Sandige, R. S., *Modern Digital Design*. New York: McGraw-Hill, 1990.

5. Shiva, S. G., *Introduction to Logic Design*. Glenview, IL: Scott, Foresman, 1988.

6. Wakerly, J. F., *Digital Design Principles and Practices*. Englewood Cliffs, NJ: Prentice Hall, 1990.

7. Ward, S. A., and R. H. Halstead, Jr., *Computation Structures*. Cambridge, MA: MIT Press, 1990.