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

A Boolean algebra is an algebra (set, operations, elements) consisting of a set B with ≥ 2 elements, together with three operations—the AND operation $\cdot$ (Boolean product), the OR operation $+$ (Boolean sum), and the NOT operation$'$ (complement)—defined on the set, such that for any element $a, b, c, \ldots$ of set B, $a \cdot b$, $a + b$, and a' are in B. Consider the four-element Boolean algebra $B_4 = (\{0, x, y, 1\}; \cdot, +, '; 0, 1)$. The AND, OR, and NOT operations are described by the following tables:

$\cdot$	0	x	y	1
0	0	0	0	0
x	0	x	0	x
y	0	0	y	y
1	0	x	y	1

$+$	0	x	y	1
0	0	x	y	0
x	x	x	1	1
y	y	1	y	1
1	1	1	1	1

$'$	
0	1
x	y
y	x
1	0

Consider the two-element Boolean algebra $B_2 = (\{0, 1\}; \cdot, +, '; 0, 1)$. The three operations. (AND), + (OR), '(NOT) are defined as follows:

$\cdot$	0	1
0	0	0
1	0	1

$+$	0	1
0	0	1
1	1	1

$'$	
0	1
1	0

The two-element Boolean algebra B_2 among all other B_i, where $i > 2$, defined as switching algebra, is the most useful. Switching algebra consists of two elements represented by 1 and 0 as the largest number and the smallest number respectively.

Boolean function

*Boolean algebra is a switching 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

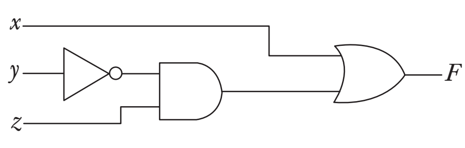
The function F is equal to 1 if x is 1 or if both y' and z are equal to 1; F is equal to 0 otherwise. But saying that $y' = 1$ is equivalent to saying that $y = 0$ since y' is the complement of y. Therefore, we may say that F is equal to 1 if $x = 1$ or if $yz = 01$. The relationship between a function 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

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

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

*Two Element

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.
2. Express in algebraic form the input–output relationship of logic diagrams.
3. Find simpler circuits for the same function.

A Boolean function specified by a truth table can be expressed algebraically in many different ways. Two ways of forming Boolean expressions are canonical and non-canonical forms. Canonical forms express all binary variables in every product (AND) or sum (OR) term of the Boolean function. To determine the canonical sum-of-products form for a Boolean function $F(A, B, C) = A'B + C' + ABC$, which is in non-canonical form, the following steps are used:

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

$$= A'B(C + C') + (A + A')(B + B')C' + ABC,$$

where $x + x' = 1$ is a basic identity of Boolean algebra

$$= A'BC + A'BC' + ABC' + AB'C' + A'BC' + A'B'C' + ABC$$

$$= A'BC + A'BC' + ABC' + AB'C' + A'B'C' + ABC$$

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

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) $x + yz = (x + y)(x + z)$ 9.
(15) $(x + y)' = x'y'$	(16) $(xy)' = x' + y'$
(17) $(x')' = x$	

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.

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:

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

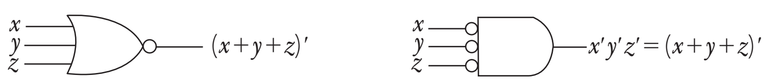
By letting $x = AB' + C'D$ the expression can be written as $x + x$. From identity 5 in Table 1-1 we find that $x + x = x$. Thus the expression can be reduced to only two terms:

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

DeMorgan's theorem

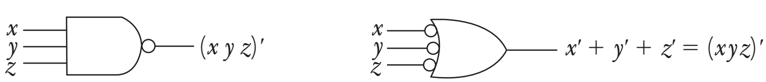
DeMorgan's theorem is very important in dealing with NOR and NAND gates. It states that a NOR gate that performs the $(x + y)'$ function is equivalent to the function $x'y'$. Similarly, a NAND function can be expressed by either $(xy)'$ or $(x' + y')$. For this reason the NOR and NAND gates have two 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. NAND and NOR gates can be used to implement any Boolean function, including basic logic gates such as AND, OR, and NOT. Hence, NAND and NOR gates are called as Universal gates.

Figure 1-4 Two graphic symbols for NOR gate.

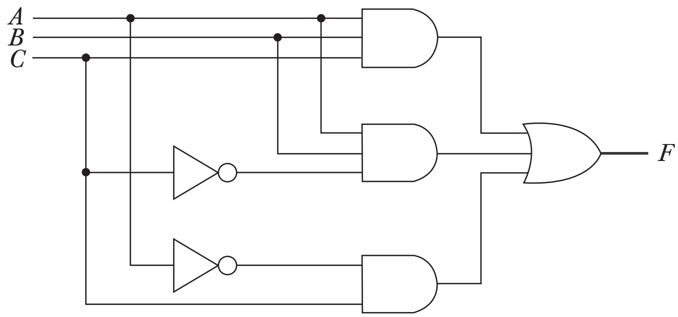


(a) OR-invert (b) invert-AND

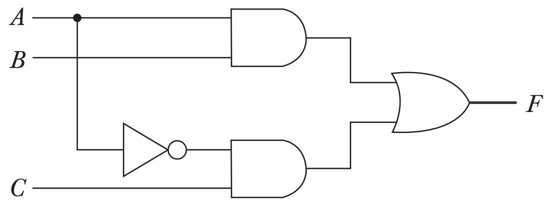
Figure 1-5 Two graphic symbols for NAND gate.



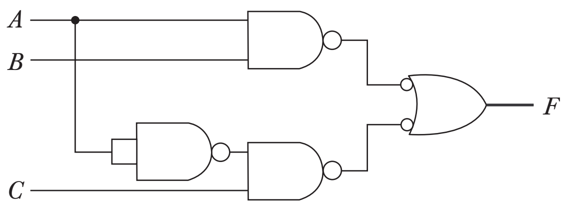
(a) AND-invert (b) invert-OR



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



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



(c) $F = AB + AC'$ using NAND gates

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

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.