Fundamentals of Digital Logic and Microcomputer Design. M. Rafiquzzaman

3

BOOLEAN ALGEBRA AND DIGITAL LOGIC GATES

This chapter describes fundamentals of logic operations, Boolean algebra, minimization techniques, and implementation of basic digital circuits.

Digital circuits contain hardware elements called "gates" that perform logic operations on binary numbers. Devices such as transistors can be used to perform the logic operations. Boolean algebra is a mathematical system that provides the basis for these logic operations. George Boole, an English mathematician, introduced this theory of digital logic. The term *Boolean variable* is used to mean the two-valued binary digit 1 or 0.

3.1 Basic Logic Operations

Boolean algebra uses three basic logic operations namely, NOT, OR, and AND. These operations are described next.

3.1.1 NOT Operation

The NOT operation inverts or provides the ones complement of a binary digit. This operation takes a single input and generates one output. The NOT operation of a binary digit provides the following result:

NOT 1 = 0
NOT 0 = 1

Therefore, NOT of a Boolean variable A, written as $\overline{A}$ (or A') is 1 if and only if A is 0. Similarly, $\overline{A}$ is 0 if and only if A is 1. This definition may also be specified in the form of a truth table:

Input	Output
A	$\overline{A}$
0	1
1	0

Note that a truth table contains the inputs and outputs of digital logic circuits. The symbolic representation of an electronic circuit that implements a NOT operation is shown

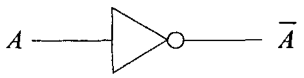


FIGURE 3.1 Symbol for a NOT gate

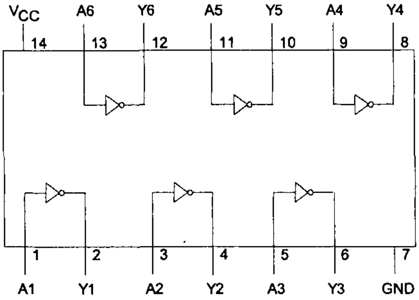


FIGURE 3.2 Pin diagram for the 74HC04 or 74LS04

in Figure 3.1.

A NOT gate is also referred to as an "inverter" because it inverts the voltage levels. As discussed in Chapter 1, a transistor acts as an inverter. A 0-volt at the input generates a 5-volt output; a 5-volt input provides a 0-volt output.

As an example, the 74HC04 (or 74LS04) is a hex inverter 14-pin chip containing six independent inverters in the same chip as shown in Figure 3.2.

Computers normally include a NOT instruction to perform the ones complement of a binary number on a bit-by-bit basis. An 8-bit computer can perform NOT operation on an 8-bit binary number. For example, the computer can execute a NOT instruction on an 8-bit binary number 01101111 to provide the result 10010000. The computer utilizes an internal electronic circuit consisting of eight inverters to invert the 8-bit data in parallel.

3.1.2 OR operation

The OR operation for two variables A and B generates a result of 1 if A or B, or both, are 1. However, if both A and B are zero, then the result is 0.

A plus sign + (logical sum) or $\vee$ symbol is normally used to represent OR. The four possible combinations of ORing two binary digits are

$$0+0=0$$
$$0+1=1$$
$$1+0=1$$
$$1+1=1$$

A truth table is usually used with logic operations to represent all possible combinations of inputs and the corresponding outputs. The truth table for the OR operation is

Inputs		
A	B	*Output* $= A + B$
0	0	0
0	1	1
1	0	1
1	1	1

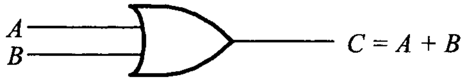


FIGURE 3.3 Symbol for an OR gate

Figure 3.3 shows the symbolic representation of an OR gate.
Logic gates using diodes provide good examples to understand how semiconductor devices are utilized in logic operations. Note that diodes are hardly used in designing logic gates. Figure 3.4 shows a two-input-diode OR gate. The diode (see Chapter 1) is a switch, and it closes when there is a voltage drop of 0.6 V between the anode and the cathode. Suppose that a voltage range of 0 to 2 V is considered as logic 0 and a voltage of 3 to 5 V is logic 1. If both *A* and *B* are at logic 0 (say 1.5 V) with a voltage drop across the diodes of 0.6 V to close the diode switches, a current flows from the inputs through *R* to ground, and the output *C* will be at 1.5 V - 0.6 V = 0.9 V (logic 0). On the other hand, if one or both inputs are at logic 1 (say 4.5 V) the output *C* will be at 4.5 - 0.6 V = 3.9 V (logic 1). Therefore, the circuit acts as an OR gate.

The 74HC32 (or 74LS32) is a commercially available quad 2-input 14-pin OR gate chip. This chip contains four 2-input/1-output independent OR gates as shown in Figure 3.5.

To understand the logic OR operation, consider Figure 3.6. *V* is a voltage source, *A* and *B* are switches, and *L* is an electrical lamp. *L* will be turned ON if either switch *A* or *B* or both are closed; otherwise, the lamp will be OFF. Hence, $L = A + B$. Computers normally contain an OR instruction to perform the OR operation between two binary numbers. For example, the computer can execute an OR instruction to OR $3A_{16}$ with 21_{16} on a bit by bit basis:

$$\begin{array}{rl} 3A_{16} & = 0\,0\,1\,1\quad 1\,0\,1\,0 \\ 21_{16} & = \underline{0\,0\,1\,0\quad 0\,0\,0\,1} \\ & \ \underbrace{0\,0\,1\,1}_{3}\quad \underbrace{1\,0\,1\,1}_{B}{}_{16} \end{array}$$

The computer typically utilizes eight two-input OR gates to accomplish this.

3.1.3 AND operation

The AND operation for two variables *A* and *B* generates a result of 1 if both *A* and *B* are 1.

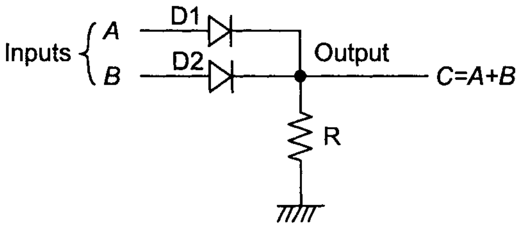


FIGURE 3.4 Diode OR gate

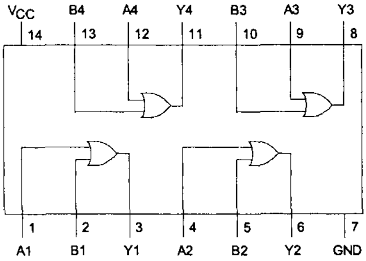


FIGURE 3.5 Pin diagram for 74HC32 or 74LS32

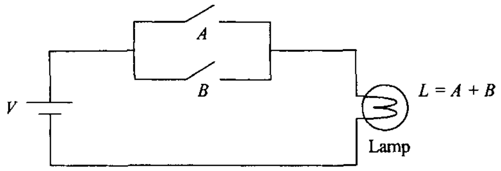


FIGURE 3.6 An example of the OR operation

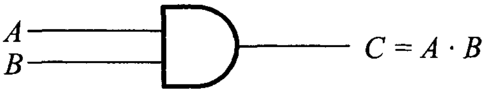


FIGURE 3.7 AND gate symbol

However, if either A or B, or both, are zero, then the result is 0.

The dot · and ∧ symbol are both used to represent the AND operation.

The AND operation between two binary digits is

$0 \cdot 0 = 0$

$0 \cdot 1 = 0$

$1 \cdot 0 = 0$

$1 \cdot 1 = 1$

The truth table for the AND operation is

Inputs		
A	*B*	*Output* = $A \cdot B = AB$
0	0	0
0	1	0
1	0	0
1	1	1

Figure 3.7 shows the symbolic representation of an AND gate. Figure 3.8 shows a two-input diode AND gate.

As we did for the OR gate, let us assume that the range 0 to +2 V represents logic

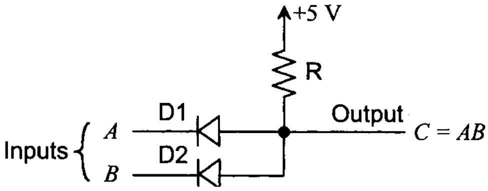


FIGURE 3.8 Diode AND gate

0 and the range 3 to 5 V is logic 1. Now, if *A* and *B* are both HIGH (say 3.3 V) and the anode of both diodes at 3.9 V, the switches in D_1 and D_2 close. A current flows from +5 V through resistor *R* to +3.3 V input to ground. The output *C* will be HIGH (3.9 V). On the other hand, if a low voltage (say 0.5 V) is applied at *A* and a high voltage (3.3V) is applied at B. The value of *R* is selected in such a way that 1.1 V appears at the anode side of D_1; at the same time 3.9 V appears at the anode side of D_2. The switches in both diodes will close because each has a voltage drop of 0.6 V between the anode and cathode. A current flows from the +5 V input through R and the diodes to ground. Output *C* will be low (1.1 V) because the output will be lower of the two voltages. Thus, it can be shown that when either one or both inputs are low, the output is low, so the circuit works as an AND gate. As mentioned before, diode logic gates are easier to understand, but they are not normally used these days.

Transistors are utilized in designing logic gates. Diode logic gates are provided as examples in order to illustrate how semiconductor devices are utilized in designing them.

The 74HC08 (or 74LS08) is a commercially available quad 2-input 14-pin AND gate chip. This chip contains four 2-input/1-output independent AND gates as shown in Figure 3.9. To illustrate the logic AND operation consider Figure 3.10. The lamp *L* will be on when both switches *A* and *B* are closed; otherwise, the lamp *L* will be turned OFF. Hence,

$$L = A \cdot B$$

Computers normally have an instruction to perform the AND operation between two binary numbers. For example, the computer can execute an AND instruction to perform ANDing

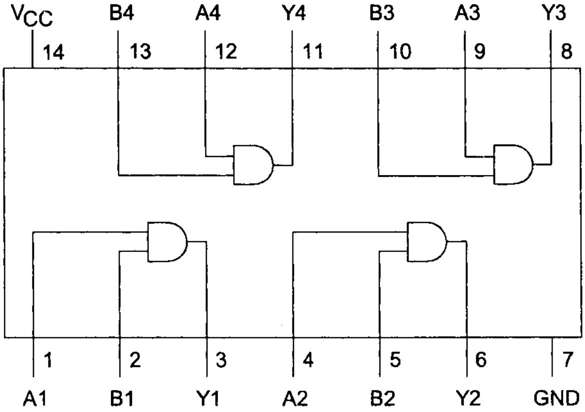


FIGURE 3.9 Pin Diagram for 74HC08 or 74LS08

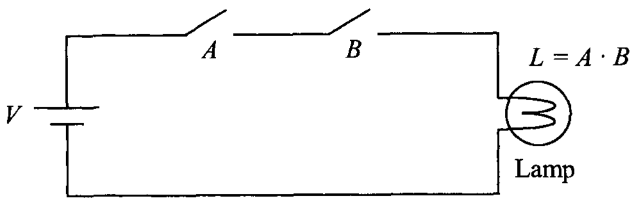


FIGURE 3.10 An example of the AND operation

31_{16} with $A1_{16}$ as follows:

$$\begin{array}{rl} 31_{16} = & 0\,0\,1\,1\ \ 0\,0\,0\,1 \\ A1_{16} = & \underline{1\,0\,1\,0\ \ 0\,0\,0\,1} \\ & \underbrace{0\,0\,1\,0}_{2}\ \ \underbrace{0\,0\,0\,1}_{1}{}_{16} \end{array}$$

The computer utilizes eight two-input AND gates to accomplish this.

3.2 Other Logic Operations

The four other important logic operations are NOR, NAND, Exclusive-OR (XOR) and Exclusive-NOR (XNOR).

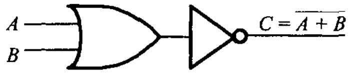
3.2.1 NOR operation

The NOR output is produced by inverting the output of an OR operation. Figure 3.11 shows a NOR gate along with its truth table. Figure 3.12 shows the symbolic representation of a NOR gate. In the figure, the small circle at the output of the NOR gate is called the inversion bubble. The 74HC02 (or 74LS02) is a commercially available quad 2-input 14-pin NOR gate chip. This chip contains four 2-input/1-output independent NOR gates as shown in Figure 3.13.

3.2.2 NAND operation

The NAND output is generated by inverting the output of an AND operation. Figure 3.14 shows a NAND gate and its truth table. Figure 3.15 shows the symbolic representation of a NAND gate.

The 74HC00 (or 74LS00) is a commercially available quad 2-input/1-output 14-pin NAND gate chip. This chip contains four 2-input/1-output independent NAND gates as shown in Figure 3.16.



NOR gate Truth Table

A	B	$C = \overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

FIGURE 3.11 A NOR gate with its truth table

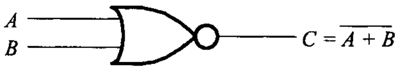


FIGURE 3.12 NOR gate symbol

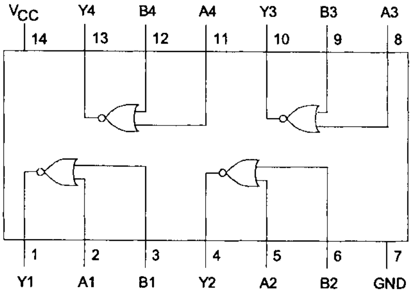
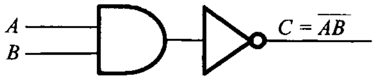


FIGURE 3.13 Pin diagram for 74HC02 or 74LS02



NAND gate Truth Table

A	B	$C = \overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

FIGURE 3.14 A NAND gate and its truth table

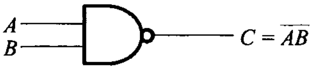


FIGURE 3.15 NAND gate symbol

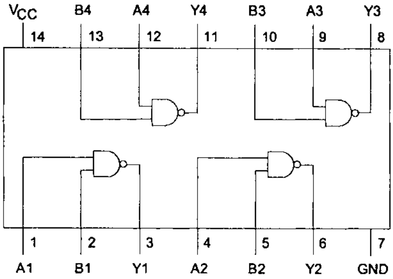


FIGURE 3.16 Pin diagram for 74HC00 or 74LS00

3.2.3 Exclusive-OR operation (XOR)

The Exclusive-OR operation (XOR) generates an output of 1 if the inputs are different and 0 if the inputs are the same. The ⊕ or ∀ symbol is used to represent the XOR operation. The XOR operation between binary digits is

$$\begin{aligned} 0 \oplus 0 &= 0 \\ 0 \oplus 1 &= 1 \\ 1 \oplus 0 &= 1 \\ 1 \oplus 1 &= 0 \end{aligned}$$

Most computers have an instruction to perform the XOR operation. Consider XORing $3A_{16}$ with 21_{16}.

$$\begin{aligned} 3A_{16} &= 0011\ \ 1010 \\ 21_{16} &= \underline{0010\ \ 0001} \\ & \ \ \underbrace{0001}_{1}\ \ \underbrace{1011}_{B}{}_{16} \end{aligned}$$

It is interesting to note that XORing any number with another number of the same length but with all 1's will generate the ones complement of the original number. For example, consider XORing 31_{16} with FF_{16}:

$$\begin{aligned} 31_{16} & \quad 0011\ \ 0001 \\ \text{1's complement of } 31_{16} & \quad \underbrace{1100}_{C}\ \ \underbrace{1110}_{E}{}_{16} \end{aligned}$$

$$\begin{aligned} 31_{16} \oplus FF_{16} & \quad 0011\ \ 0001 \\ & \quad \underline{1111\ \ 1111} \\ & \quad \underbrace{1100}_{C}\ \ \underbrace{1110}_{E}{}_{16} \end{aligned}$$

The truth table for Exclusive-OR operation is

Inputs		*Output*
A	*B*	$C = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

From the truth table, $A \oplus B$ is 1 only when $A = 0$ and $B = 1$ or $A = 1$ and $B = 0$. Therefore,

$$C = A \oplus B = A\bar{B} + \bar{A}B$$

Figure 3.17 shows an implementation of an XOR gate using AND and OR gates. Figure 3.18 shows the symbolic representation of the Exclusive-OR gate assuming that both true and complemented values of A and B are available.

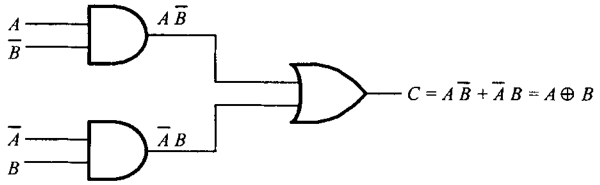


FIGURE 3.17 AND-OR Implementation of the Exclusive-OR gate

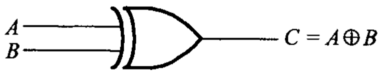


FIGURE 3.18 XOR symbol

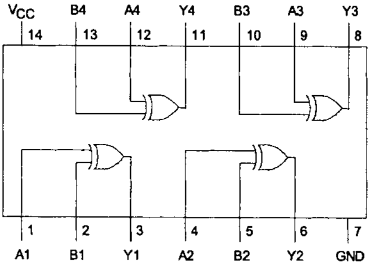
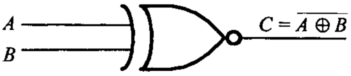


FIGURE 3.19 Pin diagram for 74HC86 or 74LS86



XNOR gate Truth Table

B	C
0	1
1	0
0	0
1	1

FIGURE 3.20 Exclusive-NOR symbol along with its truth table

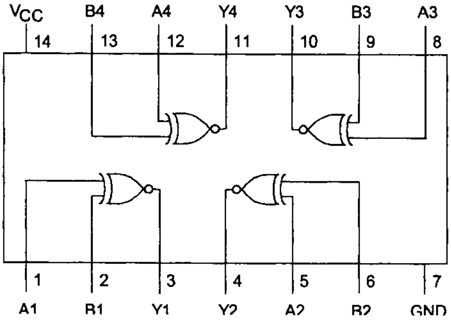


FIGURE 3.21 Pin Diagram for 74HC266 or 74LS266

The 74HC86 (or 74LS86) is a commercially available quad 2-input 14-pin Exclusive-OR gate chip. This chip contains four 2-input/1-output independent exclusive-OR gates as shown in Figure 3.19.

3.2.4 Exclusive-NOR Operation (XNOR)

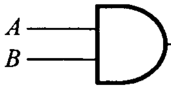
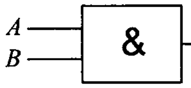
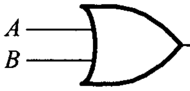
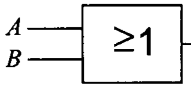
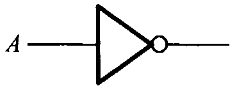
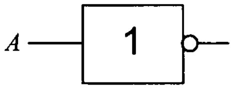
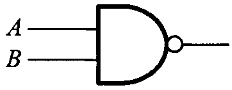
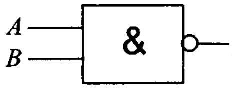
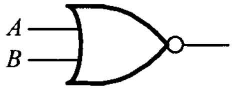
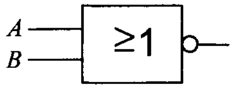
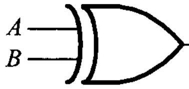
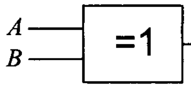
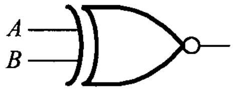
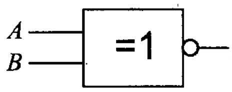
The one's complement of the Exclusive-OR operation is known as the Exclusive-NOR operation. Figure 3.20 shows its symbolic representation along with the truth table. The XNOR operation is represented by the symbol ⊙. Therefore, $C = \overline{A \oplus B} = A \odot B$. The XNOR operation is also called equivalence. From the truth table, output C is 1 if both A and B are 0's or both A and B are 1's; otherwise, C is 0. That is, $C = 1$, for $A = 0$ and $B = 0$ or $A = 1$ and $B = 1$. Hence, $C = A \odot B = \bar{A}\bar{B} + AB$

The 74HC266 (or 74LS266) is a quad 2-input/1-output 14-pin Exclusive-NOR gate chip. This chip contains four 2-input/1-output independent Exclusive-NOR gates shown in Figure 3.21.

Note that the symbol C is chosen arbitrarily in all the above logic operations to represent the output of each logic gate. Also, note that all logic gates (except NOT) can have at least two inputs with only one output. The NOT gate, on the other hand, has one input and one output.

3.3 IEEE Symbols for Logic Gates

The institute of Electrical and Electronics Engineers (IEEE) recommends rectangular shape symbols for logic gates: The original logic symbols have been utilized for years and will be retained in the rest of this book. IEEE symbols for gates are listed below:

Gate	Common Symbol	IEEE Symbol
AND	 $f = AB$	 $f = AB$
OR	 $f = A + B$	 $f = A + B$
NOT	 $f = \bar{A}$	 $f = \bar{A}$
NAND	 $f = \overline{AB}$	 $f = \overline{AB}$
NOR	 $f = \overline{A + B}$	 $f = \overline{A + B}$
Exclusive-OR	 $f = A \oplus B$	 $f = A \oplus B$
Exclusive-NOR	 $f = \overline{A \oplus B}$	 $f = \overline{A \oplus B}$

3.4 Positive and Negative Logic

The inputs and outputs of logic gates are represented by either logic 1 or logic 0. There are two ways of assigning voltage levels to the logic levels, positive logic and negative logic. The positive logic convention assigns a HIGH (*H*) voltage for logic 1 and LOW (*L*) voltage for logic 0. On the other hand, in the negative logic convention, a logic 1 = LOW (*L*) voltage and logic 0 = HIGH (*H*) voltage.

The IC data sheets typically define these levels in terms of voltage levels rather than logic levels. The designer decides on whether to use positive or negative logic. As an example, consider a gate with the following truth table:

A	B	f
L	L	H
L	H	H
H	L	H
H	H	L

Using positive logic, ($H = 1$ and $L = 0$) the following table is obtained:

A	B	f
0	0	1
0	1	1
1	0	1
1	1	0

This is the truth table for a NAND gate. However, negative logic, ($H = 0$ and $L = 1$) provides the following table:

A	B	f
1	1	0
1	0	0
0	1	0
0	0	1

This is the truth table for a NOR gate. Note that converting from positive to negative logic and vice versa for logic gates basically provides the *dual* (discussed later in this chapter) of a function. This means that changing 0's to 1's and 1's to 0's for both inputs and outputs of a logic gate, the logic gate is converted from a NOR gate to a NAND gate as shown in the example. In this book, the positive logic convention will be used.

Note that positive logic and active high logic are equivalent (HIGH = 1, LOW = 0). On the other hand, negative logic and active low logic are equivalent (HIGH = 0, LOW = 1). A signal is "active high" if it performs the required function when HIGH (H = 1). An "active low" signal, on the other hand, performs the required function when LOW (L = 0). A signal is said to be asserted when it is active. A signal is disasserted when it is not at its active level.

Active levels may be associated with inputs and outputs of logic gates. For example, an AND gate performs a logical AND operation on two active HIGH inputs and provides an active HIGH output. This also means that if both the inputs of the AND gate are asserted, the output is asserted.

3.5 Boolean Algebra

Boolean algebra provides basis for logic operations using binary variables. Alphabetic characters are used to represent the binary variables. A binary variable can have either true or complement value. For example, the binary variable A can be either A and/or $\bar{A}$ in a Boolean function.

A Boolean function is an operation expressing logical operations between binary variables. The Boolean function can have a value of 0 or 1. As an example of a Boolean function, consider the following:

$$f = \bar{A}\,\bar{B} + C$$

Here, the Boolean function f is 1 if both $\bar{A}$ and $\bar{B}$ are 1 or C is 1; otherwise, f is 0. Note that $\bar{A}$ means that if $A = 1$, then $\bar{A} = 0$. Thus, when $B = 1$, then $\bar{B} = 0$. It can therefore be concluded that f is one when $A = 0$ and $B = 0$ or $C = 1$.

A truth table can be used to represent a Boolean function. The truth table contains a combination of 1's and 0's for the binary variables. Furthermore, the truth table provides the value of the Boolean function as 1 or 0 for each combination of the input binary variables. Table 3.1 provides the truth table for the Boolean function $f = \bar{A}\,\bar{B} + C$. In the table, if $A = 1$, $B = 1$, and $C = 0$, $f = 0.0 + 0 = 0$. Note that table 3.1 contains three input variables (A, B, C) and one output variable (f). Also, by ORing ones in the truth table,

TABLE 3.1 Truth Table for $f = \bar{A}\,\bar{B} + C$

A	B	C	f
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

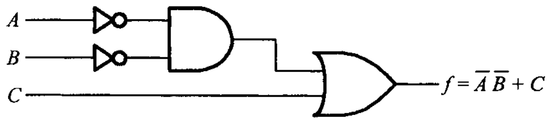


FIGURE 3.22 Logic diagram for $f = \bar{A}\,\bar{B} + C$

the function f contains several terms; however, the function can be simplified using the techniques to be discussed later.

A Boolean function can also be represented in terms of a logic diagram. Figure 3.22 shows the logic diagram for $f = \bar{A}\,\bar{B} + C$. The Boolean expression $f = \bar{A}\,\bar{B} + C$ contains two terms, $\bar{A}\,\bar{B}$ and C, which are inputs to logic gates. Each term may include a single or multiple variables, called "literals," which may or may not be complemented. For example, $f = \bar{A}\,\bar{B} + C$ contains three literals, $\bar{A}$, $\bar{B}$, and C. Note that a variable and its complement are both called literals. For two variables, the literals are A, B, $\bar{A}$, *and* $\bar{B}$.

Boolean functions can be simplified by using the rules (identities) of Boolean algebra. This allows one to minimize the number of gates in a logic diagram, which reduces the cost of implementing a logic circuit.

3.5.1 Boolean Identities

Here is a list of Boolean identities that are useful in simplifying Boolean expressions:

1. a) $A + 0 = A$ b) $A \cdot 1 = A$
2. a) $A + 1 = 1$ b) $A \cdot 0 = 0$
3. a) $A + A = A$ b) $A \cdot A = A$
4. a) $A + \bar{A} = 1$ b) $A \cdot \bar{A} = 0$
5. a) $(\bar{\bar{A}}) = A$
6. Commutative Law:
   a) $A + B = B + A$ b) $A \cdot B = B \cdot A$
7. Associative Law:
   a) $A + (B + C) = (A + B) + C$ b) $A \cdot (B \cdot C) = (A \cdot B) \cdot C$
8. Distributive Law:
   a) $A \cdot (B + C) = A \cdot B + A \cdot C$ b) $A + B \cdot C = (A + B) \cdot (A + C)$
9. DeMorgan's Theorem:
   a) $\overline{A + B} = \bar{A} \cdot \bar{B}$ b) $\overline{A \cdot B} = \bar{A} + \bar{B}$

In the list, each identity identified by b) on the right is the dual of the corresponding identity a) on the left. Note that the dual of a Boolean expression is obtained by changing 1's to 0's and 0's to 1's if they appear in the equation, and AND to OR and OR to AND on both sides of the equal sign.

For example, consider identity 4. Relation 4a is the dual of relation 4b because the AND in the expression is replaced by an OR and then, 0 by 1.

The Duality Principle of Boolean algebra states that a Boolean expression is unchanged if the dual of both sides of the equal sign is taken. Consider, for example, the Boolean function,

$f = \bar{B} + \bar{A}\,\bar{B}$ Therefore, $f = \bar{B} \cdot (1 + \bar{A})$

$= \bar{B}$

The dual of f,

$$f_D = \bar{B} \cdot (\bar{A} + \bar{B})$$

$$f_D = \bar{B} \cdot \bar{A} + \bar{B} \cdot \bar{B} = \bar{B}\,\bar{A} + \bar{B}$$

$$= \bar{B}\,(\bar{A} + 1) = \bar{B}$$

Hence, $f = f_D$. In order to verify some of the identities, consider the following examples:

i) Identity 2a) $A + 1 = 1$
   For $A = 0$, $A + 1 = 0 + 1 = 1$
   For $A = 1$, $A + 1 = 1 + 1 = 1$

ii) Identity 4b) $A \cdot \bar{A} = 0$. If $A = 1$, then $\bar{A} = 0$. Hence, $A \cdot \bar{A} = 1 \cdot 0 = 0$

iii) Identity 8b) $A + B \cdot C = (A + B) \cdot (A + C)$ is very useful in manipulating Boolean expressions. This identity can be verified by means of a truth table as follows:

A	B	C	$B \cdot C$	$A+B$	$A+C$	$A+B\cdot C$	$(A+B)\cdot(A+C)$
0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0
0	1	0	0	1	0	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

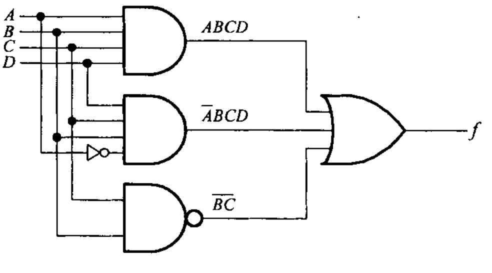
iv) Identities 9a) and 9b) (DeMorgan's Theorem) are useful in determining the one's complement of a Boolean expression. DeMorgan's theorem can be verified by means of a truth table as follows:

A	B	$\overline{A}$	$\overline{B}$	$\overline{A}\cdot\overline{B}$	$A+B$	$\overline{A+B}$	$A\cdot B$	$\overline{A\cdot B}$	$\overline{A}+\overline{B}$
0	0	1	1	1	0	1	0	1	1
0	1	1	0	0	1	0	0	1	1
1	0	0	1	0	1	0	0	1	1
1	1	0	0	0	1	0	1	0	0

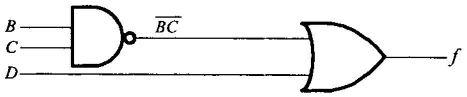
De Morgan's Theorem can be expressed in a general form for n variables as follows:

$$\overline{A+B+C+D+\ldots} = \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}\cdot\ldots$$
$$\overline{A\cdot B\cdot C\cdot D\cdot\ldots} = \overline{A}+\overline{B}+\overline{C}+\overline{D}+\ldots$$

The logic gates except for the inverter can have more than two inputs if the logic operation performed by the gate is commutative and associative (identities 6a and 7a). For example, the OR operation has these two properties as follows: A + B = B + A (commutative) and (A + B) + C = A+ (B + C) = A + B + C (associative). This means



(a) Implementation of $f = ABCD + \overline{A}BCD + \overline{BC}$



(b) implementation of the simplified function $f = \overline{BC} + D$

FIGURE 3.23 Implementation of Boolean function using logic gates

that the OR gate inputs can be interchanged. Thus, the OR gate can have more than two inputs . Similarly, using the identities 6b and 7b, it can be shown that the AND gate can also have more than two inputs. Note that the NOR and NAND operations, on the other hand, are commutative, but not associative. Therefore, it is not possible to have NOR and NAND gates with more than two inputs. However, NOR and NAND gates with more than two inputs can be obtained by using inverted OR and inverted AND respectively. The Exclusive-OR and Exclusive-NOR operations are both commutative and associative. Thus, these gates can have more than two inputs. However, Exclusive-OR and Exclusive-NOR gates with more than two inputs are uncommon from a hardware point of view.

3.5.2 Simplification Using Boolean Identities

Although there are no defined set of rules for minimizing a Boolean expression, appropriate identities can be used to accomplish this. Consider the Boolean function

$$f=ABCD+\bar{A}BCD+\overline{BC}$$

This equation can be implemented using logic gates as shown in Figure 3.23(a). The expression can be simplified by using identities as follows:

$f=BCD(A+\bar{A})+\overline{BC}$ By identity 4a)
$=BCD\bullet 1\,\overline{BC}$ By identity 1b)
$=BCD+\overline{BC}$

Assume $BC=E$, then $\overline{BC}=\bar{E}$ and,

$f=ED+\bar{E},$
$=(E+\bar{E})(\bar{E}+D)$ By identity 8b)
$=\bar{E}+D$ By identity 4a)

Substituting $\bar{E}=\overline{BC}$, $f=\overline{BC}+D$

The simplified form is implemented using logic gates in Figure 3.23(b). The logic diagram in Figure 3.23(b) requires only one NAND gate and an OR gate. This implementation is inexpensive compared to the circuit of Figure 3.23(a). Both logic circuits perform the same function. The following truth table can be used to show that the outputs produced by both circuits are equivalent:

A	B	C	D	$f=ABCD+\bar{A}BCD+\overline{BC}$	$f=\overline{BC}+D$
0	0	0	0	1	1
0	0	0	1	1	1
0	0	1	0	1	1
0	0	1	1	1	1
0	1	0	0	1	1
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	1	1	1
1	0	0	0	1	1
1	0	0	1	1	1
1	0	1	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	1	1	1

The following are some more examples for simplifying Boolean expressions using identities:

i) $f = \bar{x} + \bar{y} + \overline{xy} + \overline{xy}z = \overline{xy} + \overline{xy} + \overline{xy}z = \overline{xy} + \overline{xy}z = \overline{xy}\,(1 + z) = \overline{xy}$

ii) $f = \overline{ab}cd + \bar{a}cd + \bar{b}cd + (1 \oplus ab)\,cd = \overline{ab}cd + cd\,(\bar{a} + \bar{b}) + \overline{ab}cd = \overline{ab}cd + \overline{ab}cd + \overline{ab}cd$
$= \overline{ab}cd$

iii) $F = XY + X\bar{Z} + XZ = X\,(Y + \bar{Z} + Z) = X\,(Y + 1) = X \bullet 1 = X$

iv) $F = A\,\overline{BC} + AB + \bar{A}\,\bar{C} = A\,(\bar{B} + \bar{C}) + AB + \bar{A}\,\bar{C} = A\,\bar{B} + A\,\bar{C} + AB + \bar{A}\,\bar{C}$
$= A\,(B + \bar{B}) + \bar{C}(A + \bar{A}) = A + \bar{C}$

v) $f = x + \bar{x}y + x + \bar{y} = x + \bar{x}y + \bar{y} = (x + \bar{x})(x + y) + \bar{y} = x + y + \bar{y} = x + 1 = 1$

vi) $f = A\,(B \oplus 1)\,(\bar{A} + B) = A\,\bar{B}\,(\bar{A} + B) = A\,\bar{B}\,\bar{A} + A\,\bar{B}B = 0$

vii) $F = B\,(A + B) + A\,\bar{B} + \bar{B} = AB + BB + A\,\bar{B} + \bar{B} = AB + B + A\,\bar{B} + \bar{B}$
$= 1 + AB + A\,\bar{B} = 1$

viii) $f = (x + y + z)\,(\bar{x}\,y + \bar{y}\,z) = \bar{x}yx + \bar{x}yy + \bar{x}yz + \bar{y}zx + \bar{y}z\,y + \bar{y}z\,z$
$= \bar{x}y + \bar{x}yz + \bar{y}zx + \bar{y}z = \bar{x}y(1 + z) + \bar{y}z(x + 1) = \bar{x}y + \bar{y}z$

ix) $f = xy + xy\bar{z} + \bar{x}\,\bar{y} = xy\,(1 + \bar{z}) + \bar{x}\,\bar{y} = xy + \bar{x}\,\bar{y} = \overline{x \oplus y}$

x) $F = \bar{A}BC + ABC + B\bar{C} = BC(\bar{A} + A) + B\bar{C} = BC + B\bar{C} = B\,(C + \bar{C}) = B \bullet 1 = B$

xi) Show that $f = \overline{(a + \bar{b})(\bar{a} + b)}$ can be implemented using one Exclusive -OR gate.
Solution: $f = \overline{(a + \bar{b})(\bar{a} + b)}$ using DeMorgan's theorem,
$= \overline{(a + \bar{b})} + \overline{(\bar{a} + b)} = (\bar{a} \bullet \bar{\bar{b}}) + (\bar{\bar{a}} \bullet \bar{b}) = \bar{a}b + a\bar{b} = a \oplus b$

xii) Show that $f = \overline{(\bar{A} + \bar{B})(\bar{E} + F)}$ can be implemented using two AND and one OR gates.
Solution: $f = \overline{(\bar{A} + \bar{B})(\bar{E} + \bar{F})} = AB + EF$ using DeMorgan's theorem.

xiii) Express $f = (X + \bar{X}Z)\,(X + Z)$ using only one two-input OR gate.
Solution: $f = (X + \bar{X})\,(X + Z)(X + Z)$ using the distributive law. Hence, $f = X + Z$

xiv) Express f for $\bar{f} = (\bar{A} + \bar{B} + \bar{C}) + \overline{ABC}$ using only one three input AND gate.
Solution: Using DeMorgan's theorem, $f = \bar{\bar{f}} = \overline{(\bar{A} + \bar{B} + \bar{C}) + \overline{ABC}}$
$= (ABC) \bullet (ABC) = ABC$

3.5.3 Consensus Theorem

The Consensus Theorem is expressed as $AB + \bar{A}C + BC = AB + \bar{A}C$

The theorem states that the AND term BC can be eliminated from the expression if one of the literals such as B is ANDed with the true value of another literal (A) and the other term C is ANDed with its complement ($\bar{A}$). This theorem can sometimes be applied to simplify Boolean equations. The Consensus Theorem can be proved as follows:

$$\begin{aligned} AB+\bar{A}C+BC &= AB+\bar{A}C+BC(A+\bar{A}) \\ &= AB+\bar{A}C+ABC+\bar{A}BC \\ &= AB+ABC+\bar{A}C+\bar{A}BC \\ &= AB(1+C)+\bar{A}C(1+B) \\ &= AB+\bar{A}C \end{aligned}$$

The dual of the Consensus Theorem can be expressed as

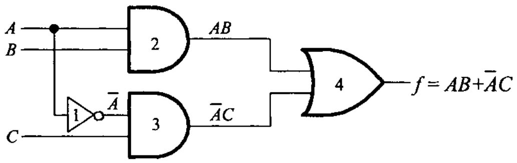
$$(A + B)(\bar{A} + C)(B + C) = (A + B)(\bar{A} + C)$$

To illustrate how a Boolean expression can be manipulated by applying the Consensus Theorem, consider the following:

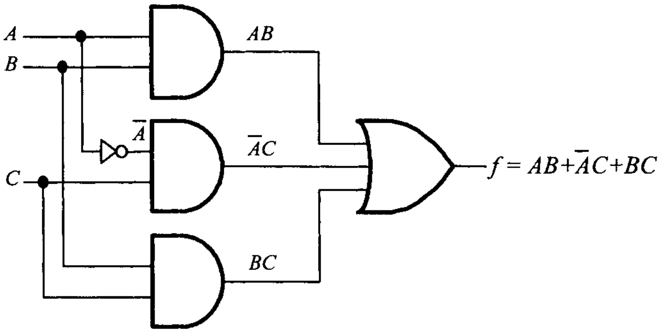
$$\begin{aligned} f &= (B+\bar{D})(\bar{B}+C) \\ &= B\bar{B}+BC+\bar{B}\bar{D}+C\bar{D} \\ &= BC+\bar{B}\bar{D}+C\bar{D}\text{, since } B\bar{B}=0 \end{aligned}$$

Because C is ANDed with B, and $\bar{D}$ is ANDed with its complement $\bar{B}$, by using the Consensus Theorem, $C\bar{D}$ can be eliminated. Thus, $f = BC + \bar{B}\,\bar{D}$.

The Consensus Theorem can be used in logic circuits for avoiding undesirable behavior. To illustrate this, consider the logic circuits in Figure 3.24. In Figure 3.24(a), the



(a) Logic circuit for $f = AB + \bar{A}C$



(b) Logic circuit for $f = AB + \bar{A}C + BC$

FIGURE 3.24 Logic circuit for the Consensus Theorem

output is one i) if B and C are 1 and $A = 0$ or ii) if B and C are 1 and $A = 1$.

Suppose that in Figure 3.24(a), $B = 1$, $C = 1$, and $A = 0$. Assume that the propagation delay time of each gate is 10 ns (nanoseconds). The circuit output f will be 1 after 30 ns (3 gate delays). Now, if input A changes from 0 to 1, the outputs of NOT gate 1 and AND gate 2 will be 0 and 1 respectively after 10 ns. This will make output $f = 1$ after 20 ns. The output of AND gate 3 will be low after 20 ns, which will not affect the output of f.

Now, assume that B and C stay at 1 while A changes from 1 to 0. The outputs of NOT gate 1 and AND gate 2 will be 1 and 0 respectively after 10 ns. Because the output of AND gate 3 is 0 from the previous case, this will change output of OR gate 4 to 0 for a brief period of time. After 10 ns, the output of AND gate 3 changes to 1, making the output of f HIGH (desired value). Note that, for $B = 1$, $C = 1$, and $A = 0$, the output f should have stayed at 1 from the equation $f = AB + \overline{A}C$. However, f changed to zero for a short period of time. This change is called a "glitch" or "hazard" and occurs from the gate delays in a circuit. Glitches can cause circuit malfunction and should be eliminated. Application of the Consensus theorem gets rid of the glitch. By adding the redundant term BC, the modified logic circuit for f is obtained. Figure 3.24(b) shows the logic circuit. Now, consider the case in which the glitch occurs in Figure 3.24(a) when B and C stay at 1 while A changes from 1 to 0. For the circuit in Figure 3.24(b) the glitch will disappear, because $BC = 1$ throughout any changes in values of A and $\overline{A}$. Thus, minimization of logic gates might not always be desirable; rather, a circuit without any hazards would bė the main objective of the designer.

There are two types of hazards: static and dynamic. Static hazard occurs when a signal should remain at one value, but instead it oscillates a few times before settling back to its original value. Dynamic hazard occurs, when a signal should make a clean transition to a new logic value, but instead it oscillates between the two logic values before making the transition to its final value. Both types of hazards occur because of *races* in the various paths of a circuit. A *race* is a situation in which signals traveling through two or more paths compete with each other to affect a common signal. It is, therefore, possible for the final signal value to be determined by the winner of the race. One way to eliminate races is by applying the Consensus theorem as illustrated in the preceding example.

3.5.4 Complement of a Boolean Function

The complement of a function f can be obtained algebraically by applying DeMorgan's Theorem. It follows from this theorem that the complement of a function can also be derived by taking the dual of the function and complementing each literal.

Example 3.1

Find the complement of the function $f = \overline{C}(AB + \overline{A}\,\overline{B}D + \overline{A}B\overline{D})$

i) Using DeMorgan's Theorem ii) By taking the dual and complementing each literal

Solution

Using DeMorgan's Theorem as many times as required, the complement of the function can be obtained:

$$\bar{f} = \overline{\bar{C}\left(AB + \bar{A}\,\bar{B}D + \bar{A}B\bar{D}\right)}$$

$$= \bar{\bar{C}} + \overline{\left(AB + \bar{A}\,\bar{B}D + \bar{A}B\bar{D}\right)}$$

$$= C + \left(\overline{AB} \cdot \overline{\bar{A}\,\bar{B}D} \cdot \overline{\bar{A}B\bar{D}}\right)$$

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

By taking the dual and complementing each literal, we have:

The dual of *f*: $\bar{C} + (A + B)(\bar{A} + \bar{B} + D)(\bar{A} + B + \bar{D})$

Complementing each literal: $C + (\bar{A} + \bar{B})(A + B + \bar{D})(A + \bar{B} + D) = \bar{f}$

3.6 Standard Representations

The standard representations of a Boolean function typically contain either logical product (AND) terms called "minterms" or logical sum (OR) terms called "maxterms." These standard representations make the minimization procedures easier. The standard representations are also called "Canonical forms."

A minterm is a product term of all variables in which each variable can be either complemented or uncomplemented. For example, there are four minterms for two variables, A and B. These minterms are $\bar{A}\,\bar{B}$, $\bar{A}B$, $A\bar{B}$, and AB. On the other hand, there are eight minterms for three variables, A, B, and C. These minterms are $\bar{A}\,\bar{B}\,\bar{C}$, $\bar{A}\,\bar{B}C$, $\bar{A}B\bar{C}$, $\bar{A}BC$, $A\bar{B}\,\bar{C}$, $A\bar{B}C$, $AB\bar{C}$, and ABC. These product terms represent numeric values from 0 through 7. In general, there are 2^n minterms for n variables.

A minterm is represented by the symbol m_j, where the subscript j is the decimal equivalent of the binary number of the minterm. For example, the decimal equivalents (j) of the binary numbers represented by the four minterms of two variables, A and B, are 0 ($\bar{A}\,\bar{B}$), 1($\bar{A}\,B$), 2($A\,\bar{B}$), and 3 (AB). Therefore, the symbolic representations of the four minterms of two variables are m_0, m_1, m_2, and m_3 as follows:

A	B	Minterm	Symbol
0	0	$\bar{A}\,\bar{B}$	m_0
0	1	$\bar{A}B$	m_1
1	0	$A\bar{B}$	m_2
1	1	AB	m_3

In general, the n minterms of p ($n = 2^p$) variables are: $m_0, m_1, m_2, \ldots, m_{n-1}$.

It has been shown that a Boolean function can be defined by a truth table. A Boolean function can be exressed in terms of minterms. For example, consider the following truth table:

A	B	f
0	0	1
0	1	0
1	0	1
1	1	1

One can determine the function f by logically summing (ORing) the product terms for which f is 1. Therefore,

$$f = \bar{A}\,\bar{B} + A\bar{B} + AB$$

This is called the Sum-of-Products expression. A logic diagram of a sum-of-products expression contains several AND gates followed by a single OR gate. In terms of minterms, f can be represented as:

$$f = \Sigma\, m(0, 2, 3)$$

The symbol Σ denotes the logical sum (OR) of the minterms.

A maxterm, on the other hand, can be defined as a logical sum (OR) term that contains all variables in complemented or uncomplemented form. The four maxterms of two variables are $A + B$, $\bar{A} + B$, $A + \bar{B}$, and $\bar{A} + \bar{B}$. A maxterm is obtained from the logical sum of all the variables by complementing each variable. Each maxterm is represented by the symbol M_j, where the subscript j is the decimal equivalent of the binary number of the maxterm. Therefore, the four maxterms of the two variables, A and B, can be represented as follows:

A	B	Maxterm	Symbol
0	0	$A+B$	M_0
0	1	$A+\bar{B}$	M_1
1	0	$\bar{A}+B$	M_2
1	1	$\bar{A}+\bar{B}$	M_3

In the preceding, consider maxterm M_2 as an example. Since $A = 1$ and $B = 0$, the maxterm M_2 is found as $\bar{A} + B$ by taking the logical sum of the complement of A (since $A = 1$) and true value of B (since $B = 0$). In general, there are n maxterms ($M_0, M_1, \ldots, M_{n-1}$) for p variables, where $n = 2^p$.

The relationship between minterm and maxterm can be established by using DeMorgan's theorem. Consider, for example, minterm m_1 and maxterm M_1 for two variables:

$$m_1 = \bar{A}\,B, \quad M_1 = A + \bar{B}$$

Taking the complement of m_1,

$$\begin{aligned} \overline{m_1} &= \overline{\bar{A}B} \\ &= \bar{\bar{A}} + \bar{B} \text{ by DeMorgan's Theorem} \\ &= A + \bar{B} \\ &= M_1 \end{aligned}$$

Therefore $m_1 = \overline{M_1}$, or $\overline{m_1} = M_1$. This implies that $m_j = \overline{M_j}$, or $\overline{m_j} = M_j$. That is, a minterm is the complement of its corresponding maxterm and vice versa.

In order to represent a Boolean function in terms of maxterms, consider the following truth table:

A	B	f	$\bar{f}$
0	0	1	0
0	1	0	1
1	0	0	1
1	1	0	1

Taking the logical sum of minterms of $\bar{f}$,

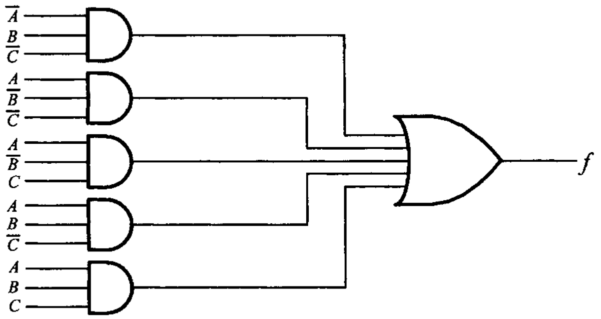


FIGURE 3.25 (a) Logic diagram of a sum of minterms

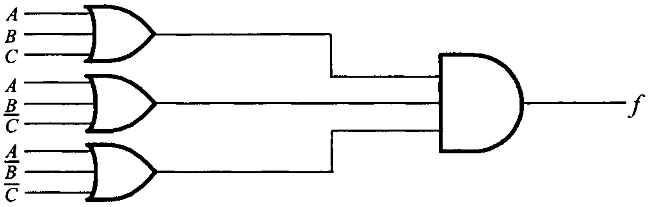


FIGURE 3.25 (b) Logic diagram of a product of maxterms

$$\begin{aligned}\bar{f} &= \bar{A}B + A\bar{B} + AB\\ &= m_1 + m_2 + m_3\\ &= \sum m(1,2,3)\end{aligned}$$

By taking complement of $\bar{f}$,

$$\begin{aligned}f = \bar{\bar{f}} &= \overline{m_1 + m_2 + m_3} = \overline{m_1} \cdot \overline{m_2} \cdot \overline{m_3}\\ &= M_1 \cdot M_2 \cdot M_3 \text{ (since } M_j = \overline{m_j})\\ &= (A + \bar{B})(\bar{A} + B)(\bar{A} + \bar{B})\end{aligned}$$

This is called the *product-of-sums* expression. The logic diagram of a *product-of-sums* expression contains several OR gates followed by a single AND gate. Hence, $f = \Pi M(1, 2, 3)$ where the symbol Π represents the logical product (AND) of maxterms M_1, M_2, and M_3 in this case. Note that one can express a Boolean function in terms of maxterms by inspecting a truth table and then logically ANDing the maxterms for which the Boolean function has a value of 0.

A Boolean function that is not expressed in terms of sums of minterms or product of maxterms can be represented by a truth table. The function can then be expressed in terms of minterms or maxterms. For example, consider $f = A + B\bar{C}$. The function f is not in a sum of minterms or product of maxterms form, since each term does not include all three variables A, B, and C. The truth table for f can be determined as follows:

A	B	C	$f = A + B\overline{C}$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

From the truth table, the sum of minterm form ($f = 1$) is:

$$f = \Sigma m(2, 4, 5, 6, 7) = \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + A\overline{B}C + AB\overline{C} + ABC$$

From the truth table, the product of maxterm form ($f = 0$) is:

$$f = \Pi M(0, 1, 3) = (A + B + C)(A + B + \overline{C})(A + \overline{B} + \overline{C})$$

The complement of f, $\bar{f} = \Sigma M(0, 1, 3)$, is obtained by the logical sum of minterms for f=0. Also, note that a function containing all minterms is 1. This means that in the above truth table, if f=1 for all eight combinations of A, B, and C, then $f = \Sigma m(0, 1, 2, 3, 4, 5, 6, 7) = 1$. As mentioned before, the logic diagram of a sum of minterm form contains several AND gates and a single OR gate. This is illustrated by the logic diagram for $f = \Sigma m(2, 4, 5, 6, 7) = \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + A\overline{B}C + AB\overline{C} + ABC$ as shown in figure 3.25(a). Similarly, the logic diagram of a product of maxterm expression form contains several OR gates and a single AND gate. This is illustrated by the logic diagram for $f = \Pi M(0, 1, 3) = (A + B + C)(A + B + \overline{C})(A + \overline{B} + \overline{C})$ as shown in figure 3.25(b).

Example 3.2

Using the following truth table, express the Boolean function f in terms of sum-of-products (minterms) and product-of-sums (maxterms):

A	B	C	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Solution

From the truth table, $f = 1$ for minterms m_1, m_2, m_3, and m_6. Therefore, the Boolean function f can be expressed by taking the logical sum (OR) of these minterms as follows:

$$f = \Sigma m(1, 2, 3, 6,) = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + \overline{A}BC + AB\overline{C}$$

Now, let us express f in terms of maxterms. By inspecting the truth table, $f = 0$ for maxterms

M_0, M_4, M_5, and M_7. Therefore, the function f can be obtained by logically ANDing these maxterms as follows:

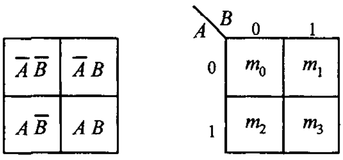
$$f = \Pi M(0, 4, 5, 7) = (A + B + C)(\bar{A} + B + C)(\bar{A} + B + \bar{C})(\bar{A} + \bar{B} + \bar{C})$$

3.7 Karnaugh Maps

A Karnaugh map or simply a K-map is a diagram showing the graphical form of a truth table. Since there is no specific set of rules for minimizing a Boolean function using identities, it is difficult to know whether the minimum expression is obtained. The K-map provides a systematic procedure for simplifying Boolean functions of typically up to five variables. K-maps for more than five variables are difficult to use. However, a computer program using a tabular method such as the Quine-McCluskey algorithm can be used to minimize Boolean functions.

The K-map is a diagram containing squares with each square representing one of the minterms of the Boolean function. For example, the K-map of two variables (A,B) contains four squares. The four minterms $\bar{A}\,\bar{B}$, $\bar{A}B$, $A\bar{B}$, and AB are represented by each square. Similarly, there are 8 squares for three variables, 16 squares for four variables, and 32 squares for five variables. Since any Boolean function can be expressed in terms of minterms, the K-map can be used to visually represent a Boolean function.

The K-map is drawn in such a way that there is only a 1-bit change from one square to the next (Gray code). Squares can be combined in groups of 2^n where n=0,1,2,3,4,5, and the Boolean function can be minimized by following certain rules. This minimum



(a) (b)

FIGURE 3.26 Two-variable K-map

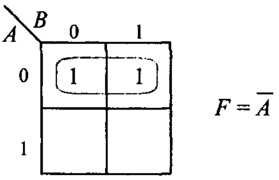


FIGURE 3.27 K-Map for $F = \Sigma m(0,1)$

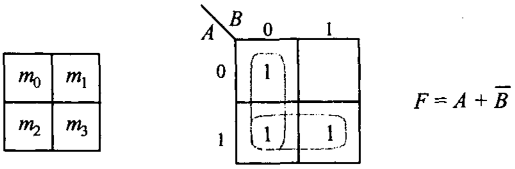


FIGURE 3.28 K-Map for $F = \Sigma m(0,2,3)$

expression will reduce the total number of gates for implementation. Thus, the cost of building the logic circuit is reduced.

3.7.1 Two-Variable K-map

Figure 3.26 shows the K-map for two variables. Since there are four minterms with two variables, four squares are required to represent them. This is depicted in the map of Figure 3.26(a). Each square represents a minterm. Figure 3.26(b) shows the K-map for two variables. Since each variable has a value of 0 or 1, in the K-map of Figure 3.26(b), the 0 and 1 shown on the left of the map corresponds to A while the 0 and 1 on the top are assigned to the variable B. The squares containing minterms with one variable change are called "adjacent" squares. A square is adjacent of another square placed horizontally or vertically next to it. For example, consider the minterms m_0 and m_1. Since $m_0 = \overline{A}\,\overline{B}$ and $m_1 = \overline{A}B$, there is a one variable change ($\overline{B}$ in m_0 and B in m_1, $\overline{A}$ is same in both squares). Therefore, m_0 and m_1 are adjacent squares. Similarly, other adjacent squares in the map include m_0 and m_2, or m_1 and m_3. $m_0(\overline{A}\,\overline{B})$ and $m_3(AB)$ are not adjacent squares since both variables change from 0's to 1's. The adjacent squares can be combined to eliminate one of the variables. This is based on the Boolean identities $A + \overline{A} = 1$ *or* $B + \overline{B} = 1$.

The adjacent squares can also be identified by considering the map as a book. By closing the book at the middle vertical line, m_0 and m_2 will respectively be placed on m_1 and m_3. Thus, m_0 and m_1 are adjacent; squares m_2 and m_3 are also adjacent. Similarly, by closing the map at the middle horizontal line, m_0 will fall on m_2 while m_1 will be placed on m_3. Thus, m_0 and m_2 or m_1 and m_3 are adjacent squares.

Now, let us consider a Boolean function, $F = \Sigma m(0,1)$. Figure 3.27 shows that the function F containing two minterms m_0 and m_1 are identified by placing 1's in the corresponding squares of the map. In order to minimize the function F, the two squares can be combined as shown since they are adjacent. The map is then inspected for common variables looking at the squares vertically and horizontally. Since $A = 0$ is common to both squares, $F = \overline{A}$. This can be proven analytically by using Boolean identities as follows:

$$F = \Sigma m(0,1) = \overline{A}\,\overline{B} + \overline{A}B$$

$$= \overline{A}(\overline{B} + B) = \overline{A} \text{ (since } \overline{B} + B = 1)$$

In a two-variable K-map, adjacent squares can be combined in groups of 2 or 4.

Next, consider $F = \Sigma m(0,2,3)$. The K-map is shown in Figure 3.28. Where 1's are placed in the squares defined by the minterms m_0, m_2, and m_3. By combining the adjacent squares m_0 with m_2 and m_2 with m_3, the common terms can be determined to simplify the function F. For example, by inspecting m_0 and m_2 vertically and horizontally, the term $\overline{B}$ is the common term. On the other hand, by looking at m_2 and m_3 horizontally and vertically, variable A is the common term. The minimized form of the function F can be obtained by logically ORing these common terms. Therefore,

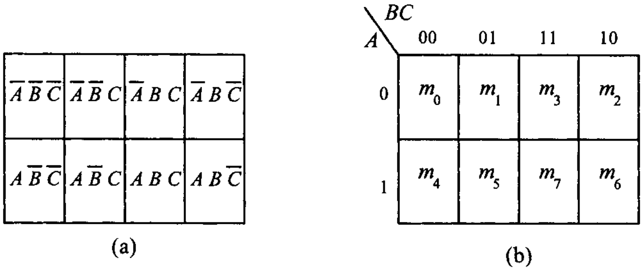
$$F = A + \overline{B}.$$

Note that the function $F = 1$ for $F = \Sigma m(0, 1, 2, 3)$ in which all squares in the K-map are 1.

3.7.2 Three-Variable K-map

Figure 3.29 shows the K-map for three variables. Figure 3.29(a) shows a map with three literals in each square. There are eight minterms ($m_0, m_1, \ldots, m_7$) for three variables. Figure 3.29(b) shows these minterms — one for each square in the K-map.

Like the two-variable K-map, a square in a three-variable K-map is adjacent to the squares placed horizontally or vertically next to it. Consider the minterms m_1, m_2, m_3, and m_7. For example, m_3 is adjacent *to* m_1, m_2, and m_7; m_1 is adjacent to m_3; m_2 is adjacent



$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$	$\bar{A}BC$	$\bar{A}B\bar{C}$
$A\bar{B}\bar{C}$	$A\bar{B}C$	ABC	$AB\bar{C}$

(a)

$A \backslash BC$	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

(b)

FIGURE 3.29 Three-variable K-map

to m_3; m_7 is adjacent to m_3. But, m_7 is adjacent neither to m_1 nor to m_2; m_1 is not adjacent to m_2 and vice versa.

Like the two-variable map, the K-map can be considered as a book. The adjacent squares can also be determined by closing the book at the middle horizontal and vertical lines. For example, closing the book at the middle horizontal line, the adjacent pair of squares are m_0 and m_4, m_1 and m_5, m_3 and m_7, m_2 and m_6. On the other hand, closing the book at the middle vertical line, the adjacent pair of squares are m_0 and m_2, m_1 and m_3, m_4 and m_6, m_5 and m_7.

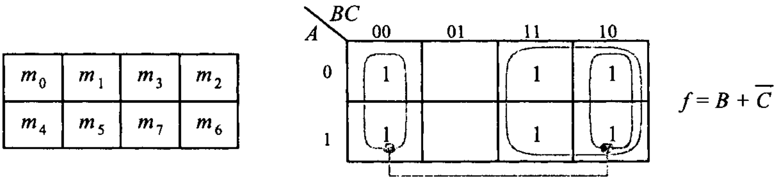
For a three variable K-map, adjacent squares can be combined in powers of 2: 1 (2^0), 2 (2^1), 4 (2^2) and 8 (2^3). The Boolean function is 1 when all eight squares are 1. It is desirable to combine as many squares as possible. For example, grouping two (2^1) adjacent squares will provide a product term of two literals and combining four (2^2) adjacent squares will provide a product term of one literal for a three-variable K-map. The following examples illustrate this.

Example 3.3

Simplify the Boolean function

$$f(A, B, C) = \Sigma\, m(0, 2, 3, 4, 6, 7)$$

using a K-map.

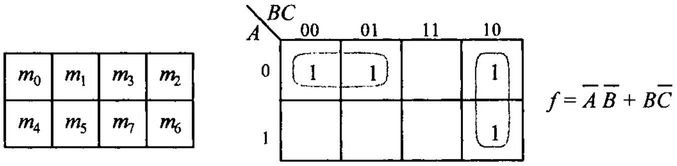


m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

$A \backslash BC$	00	01	11	10
0	1		1	1
1	1		1	1

$f = B + \bar{C}$

FIGURE 3.30 K-map for $f(A, B, C) = \Sigma\, m(0, 2, 3, 4, 6, 7)$



m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

$A \backslash BC$	00	01	11	10
0	1	1		1
1				1

$f = \bar{A}\bar{B} + B\bar{C}$

FIGURE 3.31 K-map for $f(A, B, C) = \Sigma\, m(0, 1, 2, 6)$

Solution

Figure 3.30 shows the K-map along with the grouping of adjacent squares. First, a 1 is placed in the K-map for each minterm that represents the function. Next, the adjacent squares are identified by squares next to each other. Therefore, m_2, m_3, m_6, and m_7 can be combined as a group of adjacent squares. The common term for this grouping is B. Note that combining four (2^2) squares provides the result with only one literal, B. Next, by folding the K-map at the middle vertical line, adjacent squares m_0, m_2, m_4, and m_6 can be identified. Combining them together will provide the single common term $\overline{C}$. Therefore,

$$f = B + \overline{C}$$

This result can be verified analytically by using the identities as follows:

$$\begin{aligned} f &= \Sigma\, m(0, 2, 3, 4, 6, 7) \\ &= \overline{A}\,\overline{B}\,\overline{C} + \overline{A}B\overline{C} + \overline{A}\,BC + A\,\overline{B}\,\overline{C} + A\,B\,\overline{C} + ABC \\ &= \overline{B}\,\overline{C}\,(A + \overline{A}) + B\overline{C}(\overline{A} + A) + BC(\overline{A} + A) \\ &= \overline{B}\,\overline{C} + B\overline{C} + BC \\ &= \overline{C}(\overline{B} + B) + BC \\ &= \overline{C} + BC \\ &= (B + \overline{C})(C + \overline{C}) = B + \overline{C} \qquad \text{(using the Distributive Law)} \end{aligned}$$

Example 3.4

Simplify the Boolean function

$$f(A, B, C) = \Sigma\, m(0, 1, 2, 6)$$

using a K-map.

Solution

Figure 3.31 shows the K-map along with the grouping of adjacent squares. From the K-map, grouping adjacent squares and logically ORing common product terms,

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

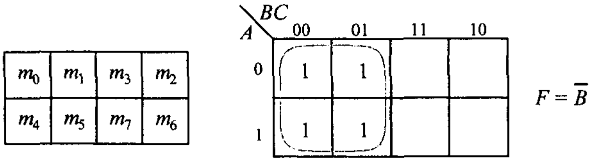
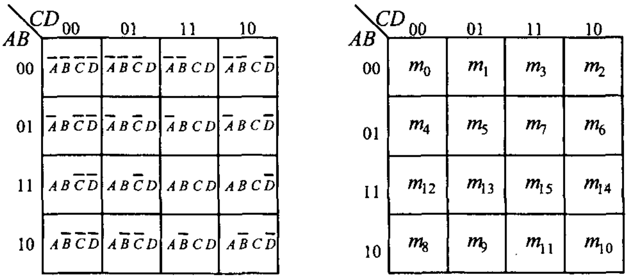


FIGURE 3.32 K-map for $F = \overline{A}\,\overline{B}\,\overline{C} + A\,\overline{B}\,\overline{C} + \overline{B}C$


CD
AB
00
01
11
10
$\overline{A}\overline{B}\overline{C}\overline{D}$
$\overline{A}\overline{B}\overline{C}D$
$\overline{A}\overline{B}CD$
$\overline{A}\overline{B}C\overline{D}$
$\overline{A}B\overline{C}\overline{D}$
$\overline{A}B\overline{C}D$
$\overline{A}BCD$
$\overline{A}BC\overline{D}$
$AB\overline{C}\overline{D}$
$AB\overline{C}D$
$ABCD$
$ABC\overline{D}$
$A\overline{B}\overline{C}\overline{D}$
$A\overline{B}\overline{C}D$
$A\overline{B}CD$
$A\overline{B}C\overline{D}$
m_0
m_1
m_3
m_2
m_4
m_5
m_7
m_6
m_{12}
m_{13}
m_{15}
m_{14}
m_8
m_9
m_{11}
m_{10}

(a) (b)

FIGURE 3.33 Four-variable K-map

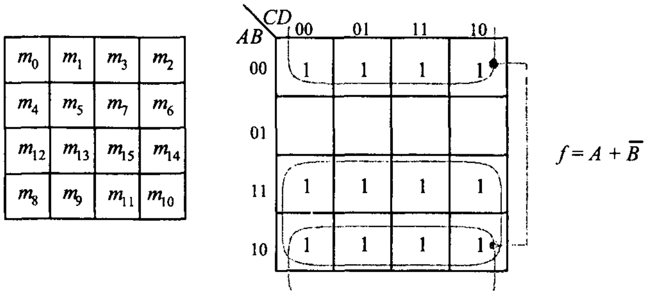


FIGURE 3.34 K-map for $f(A, B, C, D) = \Sigma m(0, 1, 2, 3, 8, 9, 10, 11, 12, 13, 14, 15)$

Example 3.5

Simplify the Boolean function

$$F(A, B, C) = \bar{A}\,\bar{B}\,\bar{C} + A\,\bar{B}\,\bar{C} + \bar{B}C$$

using a K-map.

Solution

The function contains three variables, A, B, and C, and is not expressed in minterm form. The first step is to express the function in terms of minterms as follows:

$$\begin{aligned} F &= \bar{A}\,\bar{B}\,\bar{C} + A\,\bar{B}\,\bar{C} + \bar{B}C(A + \bar{A}) \\ &= \bar{A}\,\bar{B}\,\bar{C} + A\,\bar{B}\,\bar{C} + A\bar{B}C + \bar{A}\,\bar{B}\,C \\ &= \Sigma\, m(0, 1, 4, 5) \end{aligned}$$

Figure 3.32 shows the K-map. Note that the four (2^2) adjacent squares are grouped to provide a single literal $\bar{B}$ by eliminating the other literals. Therefore, $F = \bar{B}$. Although F is not expressed in minterm form, one can usually identify the squares with 1's in the K-map for the function $F = \bar{A}\,\bar{B}\,\bar{C} + A\,\bar{B}\,\bar{C} + \bar{B}C$ by inspection. This will avoid the lengthy process of converting such functions into minterm form.

3.7.3 Four-Variable K-map

A four-variable K-map, depicted in Figure 3.33, contains 16 squares because there are 16 minterms. Figure 3.33(a) includes four literals in each square. Figure 3.33(b) lists each minterm in its respective square. As before, a square is adjacent to the squares placed horizontally or vertically next to it. For example, m_7 is adjacent to m_3, m_5, m_6, and m_{15}. Also, by closing the K-map at the middle vertical line, the adjacent pairs of squares are m_3 and m_1, m_2 and m_0, m_4 and m_6, m_{12} and m_{14}, m_8 and m_{10}, and so on. On the other hand, closing it at the middle horizontal line will provide the following adjacent squares: m_0 and m_8, m_1 and m_9, m_3 and m_{11}, m_2 and m_{10}, and so on.

For a four-variable K-map, adjacent squares can be grouped in powers of 2: 1 (2^0), 2 (2^1), 4 (2^2), 8 (2^3), and 16 (2^4). The Boolean function is 1 when all 16 minterms are 1. Combining two adjacent squares will provide a product term of three literals; four adjacent squares will provide a product term of two literals; eight adjacent squares will yield a product term of one literal.

Example 3.6

Simplify the Boolean function

$$f(A, B, C, D) = \Sigma\, m(0, 1, 2, 3, 8, 9, 10, 11, 12, 13, 14, 15)$$

using a K-map.

Solution

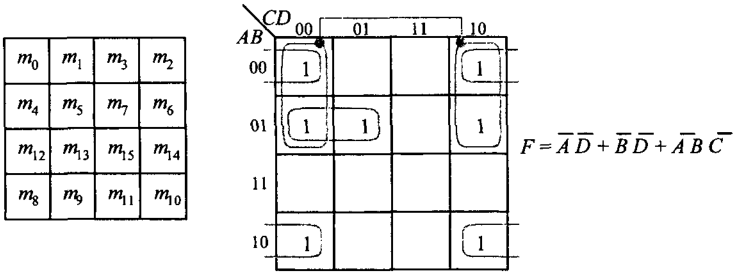

m_0 m_1 m_3 m_2
m_4 m_5 m_7 m_6
m_{12} m_{13} m_{15} m_{14}
m_8 m_9 m_{11} m_{10}
CD
AB
00
01
11
10
1
$F = \bar{A}\,\bar{D} + \bar{B}\,\bar{D} + \bar{A}\,B\,\bar{C}$

FIGURE 3.35 K-map for $F(A, B, C, D) = \Sigma\, m(0, 2, 4, 5, 6, 8, 10)$

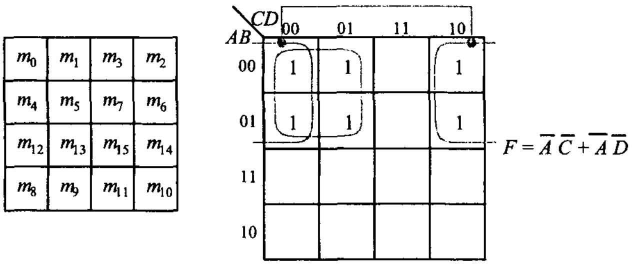

m_0 m_1 m_3 m_2
m_4 m_5 m_7 m_6
m_{12} m_{13} m_{15} m_{14}
m_8 m_9 m_{11} m_{10}
CD
AB
00
01
11
10
1
$F = \bar{A}\,\bar{C} + \bar{A}\,\bar{D}$

FIGURE 3.36 K-map for $F = \bar{A}\,\bar{B}\,\bar{C} + \bar{A}\,B\,\bar{C} + \bar{A}\,B\,\bar{D} + \bar{A}\,\bar{B}\,C\,\bar{D}$

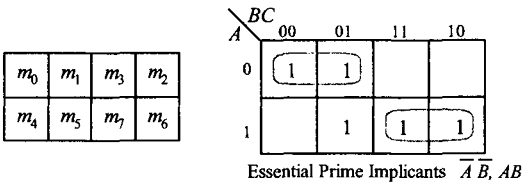


FIGURE 3.37 K-map for Example 3.9

Figure 3.34 shows the K-map. The 8 adjacent squares combined in the bottom two rows yield the common product term of one literal, A. Because the top row is adjacent to the bottom row, combining the minterms in these two rows will provide a common product term of a single literal, $\bar{B}$. Therefore, by ORing these two terms, the minimized form of the function, $F = A + \bar{B}$ is obtained.

Example 3.7

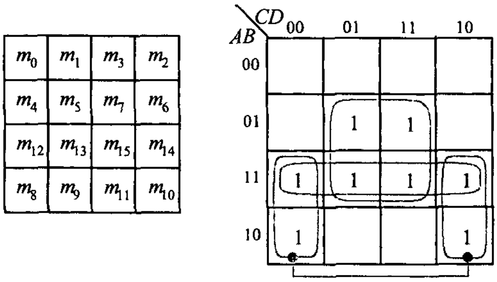
Simplify the Boolean function $f(A, B, C, D) = \Sigma\, m(0, 2, 4, 5, 6, 8, 10)$ using a K-map.

Solution

Figure 3.35 shows the K-map. The common product term obtained by grouping the adjacent squares m_0, m_2, m_4, and m_6 will contain $\bar{A}\,\bar{D}$. The common product term obtained by grouping the adjacent squares m_0, m_2, m_8, and m_{10} will be $\bar{B}\,\bar{D}$. Combining the adjacent squares m_4 and m_5 will provide the common term $\bar{A}\ B\ \bar{C}$. ORing these common product terms will yield the minimum function, $F(A, B, C, D) = \bar{A}\,\bar{D} + \bar{B}\,\bar{D} + \bar{A}\,B\,\bar{C}$.

Example 3.8

Simplify the Boolean Function, $F = \bar{A}\,\bar{B}\,\bar{C} + \bar{A}\,B\,\bar{C} + \bar{A}\,B\,\bar{D} + \bar{A}\,\bar{B}\,C\,\bar{D}$ using a K-map.


m_0 m_1 m_3 m_2
m_4 m_5 m_7 m_6
m_{12} m_{13} m_{15} m_{14}
m_8 m_9 m_{11} m_{10}
CD
AB
00
01
11
10
1

Prime Implicants BD, $A\overline{D}$, and AB

FIGURE 3.38 K-map for Example 3.10

Solution

Figure 3.36 shows the K-map. In the figure, the function F can be expressed in terms of minterms as follows:

$$F = \overline{A}\,\overline{B}\,\overline{C}(D+\overline{D}) + \overline{A}\,B\,\overline{C}(D+\overline{D}) + \overline{A}\,B\,\overline{D}(C+\overline{C}) + \overline{A}\,\overline{B}\,C\,\overline{D}$$
$$= \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,B\,\overline{C}\,D + \overline{A}\,B\,\overline{C}\,\overline{D} + \overline{A}\,B\,C\,\overline{D} + \overline{A}\,B\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,C\,\overline{D}$$
$$= m_1 + m_0 + m_5 + m_4 + m_6 + m_4 + m_2$$
$$= m_1 + m_0 + m_5 + m_4 + m_6 + m_2$$

because $m_4 + m_4 = m_4$

Rearranging the terms: $F = m_0 + m_1 + m_2 + m_4 + m_5 + m_6$

Therefore, $F = \Sigma\, m(0, 1, 2, 4, 5, 6)$

These minterms are marked as 1 in the K-map. The adjacent squares are grouped as shown. The minimum form of the function, $F = \overline{A}\,\overline{C} + \overline{A}\,\overline{D}$.

3.7.4 Prime Implicants

A prime implicant is the product term obtained as a result of grouping the maximum number of allowable adjacent squares in a K-map. The prime implicant is called "essential" if it is the only term covering the minterms. A prime implicant is called "nonessential" if another prime implicant covers the same minterms. The simplified expression for a function can be determined using the K-map as follows:

i) Determine all the essential prime implicants.

ii) Express the minimum form of the function by logically ORing the essential prime implicants obtained in i) along with other prime implicants that may be required to cover any remaining minterms not covered by the essential prime implicants.

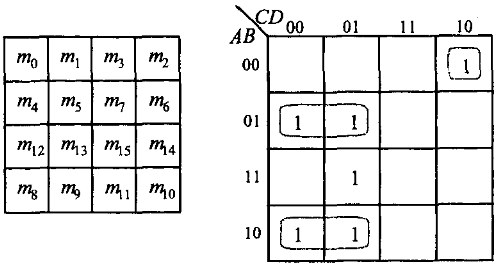

m_0 m_1 m_3 m_2
m_4 m_5 m_7 m_6
m_{12} m_{13} m_{15} m_{14}
m_8 m_9 m_{11} m_{10}
CD
AB
00
01
11
10
1

FIGURE 3.39 K-map for $f = \Sigma\, m(2, 4, 5, 8, 9, 13)$

Example 3.9

Find the prime implicants from the K-map of Figure 3.37 and then determine the simplified expression for the function.

Solution

The essential prime implicants are AB, $\bar{A}\,\bar{B}$ because minterms m_0 and m_1 can only be covered by the term $\bar{A}\,\bar{B}$ and minterms m_6 and m_7 can only be covered by the term AB.

The terms AC and $\bar{B}C$ are nonessential prime implicants because minterm m_5 can be combined with either m_1 or m_7. The term AC can be obtained by combining m_5 with m_7 whereas the term $\bar{B}C$ is obtained by combining m_5 with m_1. The function can be expressed in two simplified forms as follows:

$$f = \bar{A}\,\bar{B} + AB + AC$$

or

$$f = \bar{A}\,\bar{B} + AB + \bar{B}C$$

Example 3.10

Find the essential prime implicants from the K-map of Figure 3.38 and then find the simplified expression for the function.

Solution

The prime implicants can be obtained as follows:

1. By combining minterms m_5, m_7, m_{13}, and m_{15}, the prime implicant BD is obtained.
2. By combining minterms m_8, m_{10}, m_{12}, and m_{14}, the prime implicant $A\bar{D}$ is obtained.
3. By combining minterms m_{12}, m_{13}, m_{14}, and m_{15}, the prime implicant AB is obtained.

The terms BD and $A\bar{D}$ are essential prime implicants whereas AB is a nonessential prime implicant because minterms m_5 and m_7 can only be covered by the term BD and minterms m_8 and m_{10} can only be covered by the term $A\bar{D}$. However, minterms m_{12}, m_{13}, m_{14}, and m_{15} can be covered by these two prime implicants (BD and $A\bar{D}$). Therefore, the term AB is not an essential prime implicant. Because all minterms are covered by the essential prime implicants, BD and $A\bar{D}$, the term AB is not required to simplify the function. Therefore,

$$f = BD + A\bar{D}\,.$$

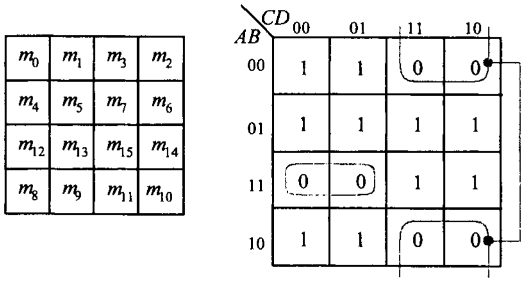
Example 3.11

Find the prime implicants and then simplify the function using a K-map.

$$f = \Sigma\, m(2, 4, 5, 8, 9, 13)$$

Solution

Figure 3.39 shows the K-map. The essential prime implicants are $\bar{A}\,\bar{B}\,C\,\bar{D}$, $\bar{A}\,B\,\bar{C}$, and $A\,\bar{B}\,\bar{C}$ because minterms m_4 and m_5 can only be covered by the term $\bar{A}\,B\,\bar{C}$, minterms m_8



m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}

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

FIGURE 3.40 K-map for $f(A, B, C, D) = \Sigma\, m(0, 1, 4, 5, 6, 7, 8, 9, 14, 15)$

and m_9 can only be covered by the term $A\,\bar{B}\,\bar{C}$, and minterm m_2 can only be covered by the term $\bar{A}\,\bar{B}\,C\,\bar{D}$.

Minterm m_{13} can be combined with either m_5 or m_9. Combining m_{13} with m_5 will yield the term $B\bar{C}D$; combining m_{13} with m_9 will provide the term $A\bar{C}D$. Therefore, minterm m_{13} can be covered by either $B\bar{C}D$ or $A\bar{C}D$. Therefore, $B\bar{C}D$ and $A\bar{C}D$ are nonessential prime implicants. Hence, the function has two simplified forms:

$$f = \bar{A}\,\bar{B}\,C\,\bar{D} + \bar{A}\,B\,\bar{C} + A\,\bar{B}\,\bar{C} + B\bar{C}D$$

or

$$f = \bar{A}\,\bar{B}\,C\,\bar{D} + \bar{A}\,B\,\bar{C} + A\,\bar{B}\,\bar{C} + A\bar{C}D$$

3.7.5 Expressing a Function in Product-of-sums Form Using a K-Map

So far, the simplified Boolean functions derived from the K-map were expressed in sum-of-products form. This section will describe the procedure for obtaining the simplified Boolean function in product-of-sums form.

In the K-map, the minterms of a function are represented by 1's. If the empty squares in the K-map are identified as 0's, combining the appropriate adjacent squares will provide the simplified expression of the complement of the function ($\bar{f}$). By taking the complement of $\bar{f}$, the simplified expression for the function, f, can be obtained.

Example 3.12

Simplify the Boolean function $f(A, B, C, D) = \Sigma\, m(0, 1, 4, 5, 6, 7, 8, 9, 14, 15)$ in product-of-sums form using a K-map.

Solution

Figure 3.40 shows the K-map. Combining the 0's, a simplified expression for the complement of the function can be obtained as follows:

$$\bar{f} = \bar{B}C + AB\bar{C}$$

By DeMorgan's Theorem,

$$\bar{\bar{f}} = f = \overline{(\bar{B}C + AB\bar{C})} = \overline{(\bar{B}C)} \bullet \overline{(AB\bar{C})} + (B + \bar{C}) \bullet (\bar{A} + \bar{B} + C)$$

The example illustrates the procedure for simplifying a function in product-of-sums form from its expression as a sum of minterms. The procedure is similar for simplifying a function expressed in product-of-sums (maxterms).

To represent a function expressed in product-of-sums in the K-map, the complement of the function must first be taken. The squares will then be identified as 1's for the minterms of the complement of the function. For example, consider the following function expressed in maxterm form:

$$f = (\bar{A} + B + C)(A + \bar{B} + \bar{C})(A + B + C)$$

This function can be represented in the K-map by taking its complement and representing in terms of minterms as follows:

$$\begin{aligned} \bar{f} &= A\bar{B}\,\bar{C} + \bar{A}BC + \bar{A}\,\bar{B}\,\bar{C} \\ &= \Sigma\, m(0, 3, 4) \end{aligned}$$

Placing 1's in the K-map for m_0, m_3, and m_4 will provide the minterms for $\bar{f}$. The simplified expression for the sum-of-products form of the function, $\bar{f}$ can be obtained by grouping 1's. Finally, the product-of-sums form of the function, f, can be obtained by complementing the function, $\bar{f}$.

3.7.6 Don't Care Conditions

The squares of a K-map are marked with 1's for the minterms of a function. The other squares are assumed to be 0's. This is not always true, because there may be situations

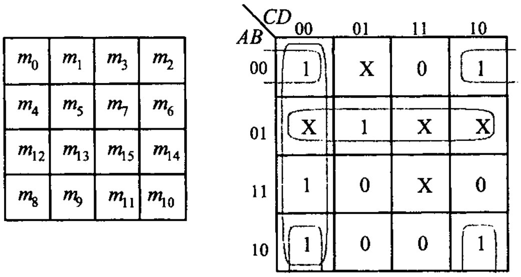


FIGURE 3.41 K-map for Example 3.13

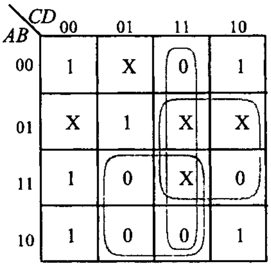


FIGURE 3.42 Determine $\bar{f}$ by combining 0's and don't care conditions for Example

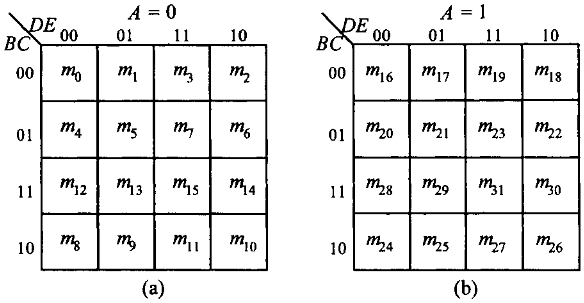

A = 0
DE
BC
00 01 11 10
00: m_0 m_1 m_3 m_2
01: m_4 m_5 m_7 m_6
11: m_{12} m_{13} m_{15} m_{14}
10: m_8 m_9 m_{11} m_{10}
(a)
A = 1
DE
BC
00 01 11 10
00: m_{16} m_{17} m_{19} m_{18}
01: m_{20} m_{21} m_{23} m_{22}
11: m_{28} m_{29} m_{31} m_{30}
10: m_{24} m_{25} m_{27} m_{26}
(b)

FIGURE 3.43 Five-Variable K-map

Five-Variable K-map

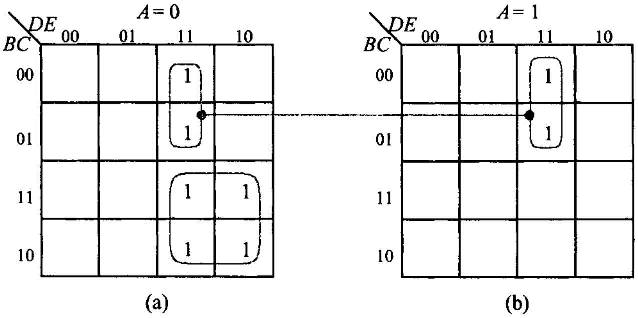


FIGURE 3.44 K-map for Example 3.14

in which the function is not defined for all combinations of the variables. Such functions having undefined outputs for certain combinations of literals are called "incompletely specified functions." One does not normally care about the value of the function for undefined minterms. Therefore, the undefined minterms of a function are called "don't care conditions." Simply put, the don't care conditions are situations in which one or more literals in a minterm can never happen, resulting in nonoccurence of the minterm.

As an example, BCD numbers include ten digits (0 through 9) and are defined by four bits (0000_2 through 1001_2). However, one can represent binary numbers from 0000_2 through 1111_2 using four bits. This means that the binary combinations 1010_2 through 1111_2 (10_{10} through 15_{10}) can never occur in BCD. Therefore, these six combinations (1010_2 through 1111_2) are don't care conditions in BCD. The functions for these six combinations of the four literals are unspecified. The don't care condition is represented by the symbol X. This means that the symbol X will be placed inside a square in the K-map for which the function is unspecified. The don't care minterms can be used to simplify a function. The function can be minimized by assigning 1's or 0's for X's in the K-map while determining adjacent squares. These assigned values of X's can then be grouped with 1's or 0's in the K-map, depending on the combination that provides the minimum expression. Note that a don't care condition may not be required if it does not help in minimizing the function. To help in understanding the concept of don't care conditions, the following example is provided.

Example 3.13

Simplify the function $f(A, B, C, D) = \Sigma\, m(0, 2, 5, 8, 10, 12)$ using a K-map. Assume that the minterms m_1, m_4, m_6, m_7, and m_{15} can never occur.

Solution

The don't care conditions are

$$d(A, B, C, D) = \Sigma\, m(1, 4, 6, 7, 15)$$

Figure 3.41 shows the K-map. By assigning X = 1 and combining 1's as shown, f can be expressed in sum-of-products form as follows:

$$f = \bar{C}\,\bar{D} + \bar{A}\,B + \bar{B}\,\bar{D}$$

On the other hand, by assigning X = 0 and combining 0's as shown in Figure 3.42, $\bar{f}$ can be obtained as a product-of-sums. Thus,

$$\begin{aligned} \bar{f} &= CD + AD + BC \\ f &= \bar{\bar{f}} = \overline{CD + AD + BC} \\ &= (\overline{CD})(\overline{AD})(\overline{BC}) \\ &= (\bar{C} + \bar{D})(\bar{A} + \bar{D})(\bar{B} + \bar{C}) \end{aligned}$$

3.7.7 Five-Variable K-map

Figure 3.43 shows a five-variable K-map. The five-variable K-map contains 32 squares. It contains two four-variable maps for $BCDE$ with $A = 0$ in one of the two maps and $A = 1$ in the other. The value of a minterm in each map can be determined by the decimal value of the five literals. For example, minterm m_{14} from Figure 3.43(a) can be expressed in terms of the five literals as $\bar{A}BCD\bar{E}$. On the other hand, minterm m_{26} can be expressed in terms of the five literals from Figure 3.43(b) as $AB\bar{C}D\bar{E}$.

When simplifying a function, each K-map can first be considered as an individual four-variable map with $A = 0$ or $A = 1$. Combining of adjacent squares will be identical to typical four-variable maps. Next, the adjacent squares between the two K-maps can be determined by placing the map in Figure 3.43(a) on top of the map in Figure 3.43(b).

Two squares are adjacent when a square in Figure 3.43(a) falls on the square in Figure 3.43(b) and vice versa. For example, minterm m_0 is adjacent to minterm m_{16}, minterm m_1 is adjacent to minterm m_{17}, and so on.

Example 3.14

Simplify the function

$$f(A, B, C, D, E) = \Sigma\, m(3, 7, 10, 11, 14, 15, 19, 23)$$

using a K-map.

Solution

Figure 3.44 shows the K-map.

$$f = \bar{A}BD + \bar{B}DE$$

To find the adjacent squares, the K-maps are first considered individually. From Figure 3.44(a), combining minterms m_{10}, m_{11}, m_{14}, and m_{15} will yield the product term $\bar{A}BD$.

Minterms m_{19} and m_{23} are in the K-map of Figure 3.44(b). However, they are adjacent to minterms m_3 and m_7 in Figure 3.44(a). Combining m_3, m_7, m_{19}, and m_{23} together, the product term $\bar{B}DE$ can be obtained. Literals A or $\bar{A}$ are not included here because adjacent squares belong to both $A = 0$ and $A = 1$. Therefore, the minimum form of f is

$$f = \bar{A}BD + \bar{B}DE$$

3.8 Quine–McCluskey Method

When the number of variables in a K-map is more than five, it becomes impractical to use K-maps in order to minimize a function. A tabular method known as Quine–McCluskey can be used. A computer program is usually written for the Quine–McCluskey method. One uses this program to simplify a function with more than five variables.

Like the K-map, the Quine–McCluskey method first finds all prime implicants of the function. A minimum number of prime implicants is then selected that defines the function. In order to understand the Quine–McCluskey method, an example will be provided using tables and manual check-off procedures. Although a computer program rather than manual approach is normally used by logic designers, a simple manual example is presented here so that the method can be easily understood.

The Quine–McCluskey method first tabulates the minterms that define the function. The following example illustrates how a Boolean function is minimized using the Quine–McCluskey method.

TABLE 3.2 Simplifying $F = \Sigma\, m(0, 2, 4, 5, 6, 8, 10)$ Using the Quine–McCluskey Method

		(i)					(ii)						(iii)			
Minterm	*A*	*B*	*C*	*D*			*A*	*B*	*C*	*D*			*A*	*B*	*C*	*D*
0	0	0	0	0	✓	0,2	0	0	–	0	✓	0,2,4,6	0	–	–	0
2	0	0	1	0	✓	0,4	0	–	0	0	✓	0,2,8,10	–	0	–	0
4	0	1	0	0	✓	0,8	–	0	0	0	✓	0,4,2,6	0	–	–	0
8	1	0	0	0	✓	2,6	0	–	1	0	✓	0,8,2,10	–	0	–	0
5	0	1	0	1	✓	2,10	–	0	1	0	✓					
6	0	1	1	0	✓	4,5	0	1	0	–						
10	1	0	1	0	✓	4,6	0	1	–	0	✓					
						8,10	1	0	–	0	✓					

Example 3.15

In Example 3.7, $F(A, B, C, D) = \Sigma m(0, 2, 4, 5, 6, 8, 10)$ is simplified using a K-map. The minimum form is $F = \bar{A}\,\bar{D} + \bar{B}\,\bar{D} + \bar{A}B\bar{C}$. Verify this result using the Quine–McCluskey method.

Solution

First arrange the binary representation of the minterms as shown in Table 3.2. In the table, the minterms are grouped according to the number of 1's contained in their binary representations. For example, consider column (i). Because minterms m_2, m_4, and m_8 contain one 1, they are grouped together. On the other hand, minterms m_5, m_6, and m_{10} contain two 1's, so they are grouped together.

Next, consider column (ii). Any two minterms that vary by one bit in column (i) are grouped together in column (ii). Starting from the top row, proceeding to the bottom row, and comparing the binary representation of each minterm in column (i), pairs of minterms having only a one-variable change are grouped together in column (ii) with the variable bit replaced by the symbol –. For example, comparing $m_0 = 0000$ with $m_2 = 0010$, there is a one-variable change in bit position 1. This is shown in column (ii) by placing – in bit position 1 with the other three bits unchanged. Therefore, the top row of column (ii) contains 00–0. The procedure is repeated until all minterms are compared from top to bottom for one unmatched bit and are represented by replacing this bit position with – and other bits unchanged. A ✓ is placed on the right-hand side to indicate that this minterm is compared with all others and its pair with one bit change is found. If a minterm does not have another minterm with one bit change, no check mark is placed on its right. This means that the prime implicant will contain four literals and will be included in the simplified of the function F. In column (i), for each minterm, a corresponding pair with one bit change is identified. These pairs are listed in column (ii).

Finally, consider column (iii). Each minterm pair in column (ii) is compared to the next, starting from the top, to find another pair with one bit change; for example m_0, m_2 = 00–0 and m_4, m_6 = 01–0. For this case, bit position 2 does not match. This bit position is replaced by – in the top row of column (iii). Therefore, in column (iii), the top row groups these four minterms 0, 2, 4, 6 with $ABCD$ as 0 – – 0. Similarly, all other pairs in column (ii) are compared from top to bottom for one bit change and are listed accordingly in column (iii) if an unmatched bit is found. A check mark is placed in the right of column (ii) if an unmatched bit is found between two pairs. Note that minterms 4 and 5 do not have any other pair in the list of column (ii) having one unmatched bit. Therefore, this pair is not checked on the right and must be included in the simplified form of F as a prime implicant containing three variables. The two rows of column (iii) (0,2,4,6 and 0,4,2,6) are the same and contain 0 – – 0. Therefore, this term should be considered once. Similarly, the groups 0,2,8,10 and 0,8,2,10 containing -0-0 should be considered once. In column (iii), there are no more groups that exist with one unmatched bit.

The comparison process stops. The prime implicants will be the unchecked terms $\bar{A}B\bar{C}$ (from column (ii)) along with, $\bar{A}\,\bar{D}$ and $\bar{B}\,\bar{D}$ [from column (iii)]. Thus, the simplified form for F is

$$F = \bar{A}\,\bar{D} + \bar{B}\,\bar{D} + \bar{A}B\bar{C}$$

This agrees with the result of Example 3.7.

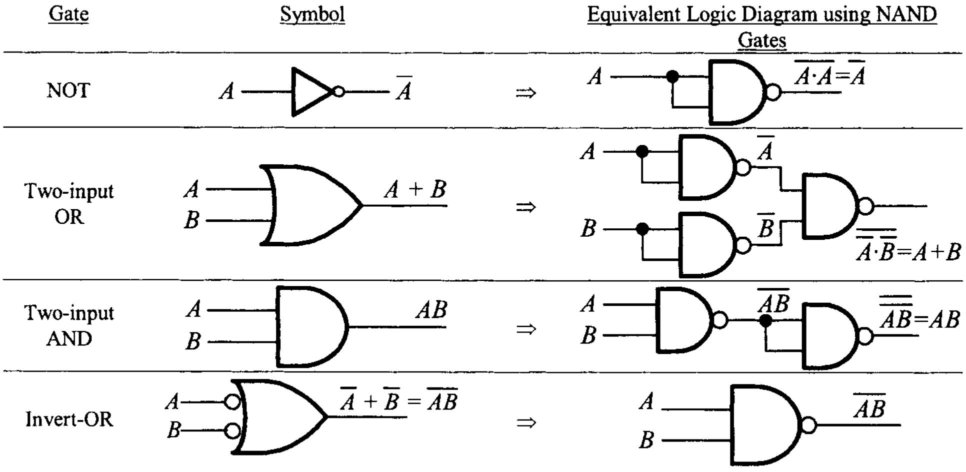

Gate
Symbol
Equivalent Logic Diagram using NAND Gates
NOT
A
$\overline{A}$
⇒
A
$\overline{A \cdot A} = \overline{A}$
Two-input OR
A
B
A + B
⇒
A
$\overline{A}$
B
$\overline{B}$
$\overline{\overline{A} \cdot \overline{B}} = A + B$
Two-input AND
A
B
AB
⇒
A
B
$\overline{AB}$
$\overline{\overline{AB}} = AB$
Invert-OR
A
B
$\overline{A} + \overline{B} = \overline{AB}$
⇒
A
B
$\overline{AB}$

FIGURE 3.45 Logic equivalents using NAND gates

3.9 Implementation of Digital Circuits with NAND, NOR, and Exclusive-OR/ Exclusive-NOR Gates

This section first covers implementation of logic circuits using NAND and NOR gates. These gates are extensively used for designing digital circuits. The NAND and NOR gates are called "universal gates" because any digital circuit can be implemented with them. These gates are, therefore, more commonly used than AND and OR gates. Finally, Exclusive-NOR gates are used to design parity generation and checking circuits.

3.9.1 NAND Gate Implementation

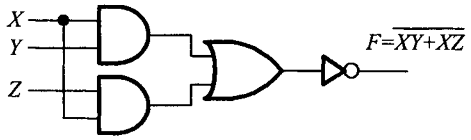
Any logic operation can be implemented by NAND gates. Figure 3.45 shows how NOT, AND, OR, and AND-invert operations can be implemented with NAND gates. A Boolean function can be implemented using NAND gates by first obtaining the simplified expression of the function in terms of AND-OR- NOT logic operations. The function can then be converted to NAND logic. A function expressed in sum-of-products form can be readily implemented using NAND gates.

Example 3.16

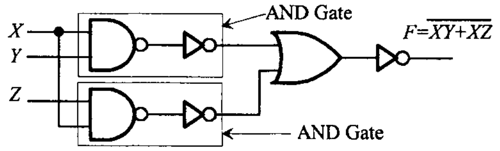
Implement the simplified function $F = \overline{XY + XZ}$ using NAND gates.

Solution

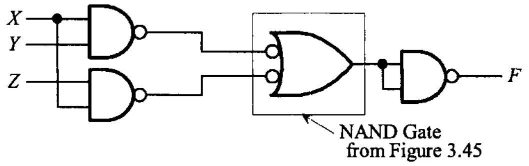
First implement the function using AND, OR, and NOT gates as follows:



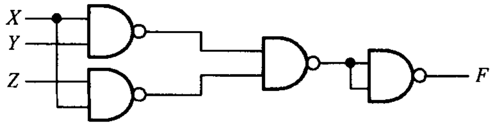
Now convert the AND, OR, and NOT gates to NAND gates as follows:



The NOT gates can be represented as bubbles at the inputs of the OR gate as follows:



Therefore, the function $F = \overline{XY + XZ}$ can be implemented using only NAND gates as follows:



This is a three-level implementation since 3 gate delays are required to obtain the output F.

Example 3.17

Implement the following Boolean function using NAND gates:

$$f(A, B, C, D) = \Sigma\, m(0, 3, 4, 8, 11, 12, 15)$$

Assume both true and complemented inputs are available.

Solution

From the K-map of Figure 3.46,

$$f(A, B, C, D) = \bar{C}\bar{D} + \bar{B}CD + ACD$$

Figure 3.47 shows the logic diagram using AND and OR gates. Note that the logic circuit of Figure 3.48 (c) has four gate delays. Figure 3.48 shows the various steps for implementing this circuit using NAND gates. In Figure 3.48(a), each AND gate of Figure 3.47 is represented by an AND gate with two inverters at the output. For example, consider AND gate 1 of Figure 3.47. The AND gate and an inverter are used to form the NAND gate shown in the top row of Figure 3.48(b) with an inverter (indicated by a bubble at the OR gate input). AND gates 3 and 4 are represented in the same way as AND gate 1 in Figure 3.48(b).

Finally, in Figure 3.48(c), the OR gate with the bubbles at the input in Figure 3.48(b) is replaced by a NAND gate. Thus, the NAND gate implementation in Figure 3.48(c) is obtained.

Example 3.18

Implement the following functions with NAND gates:

$$f = (CD + \bar{D})(AB)$$

Assume both true and complemented inputs are available.

Solution

Figure 3.49 shows the AND-OR implementation of the function. The AND-OR implementation in the figure can be converted to the NAND implementation as shown in Figure 3.50.

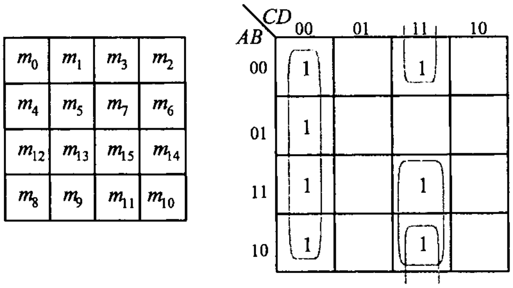


FIGURE 3.46 K-map for Example 3.17

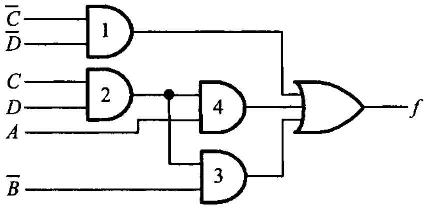


FIGURE 3.47 Logic diagram for $f = \bar{C}\,\bar{D} + \bar{B}CD + ACD$

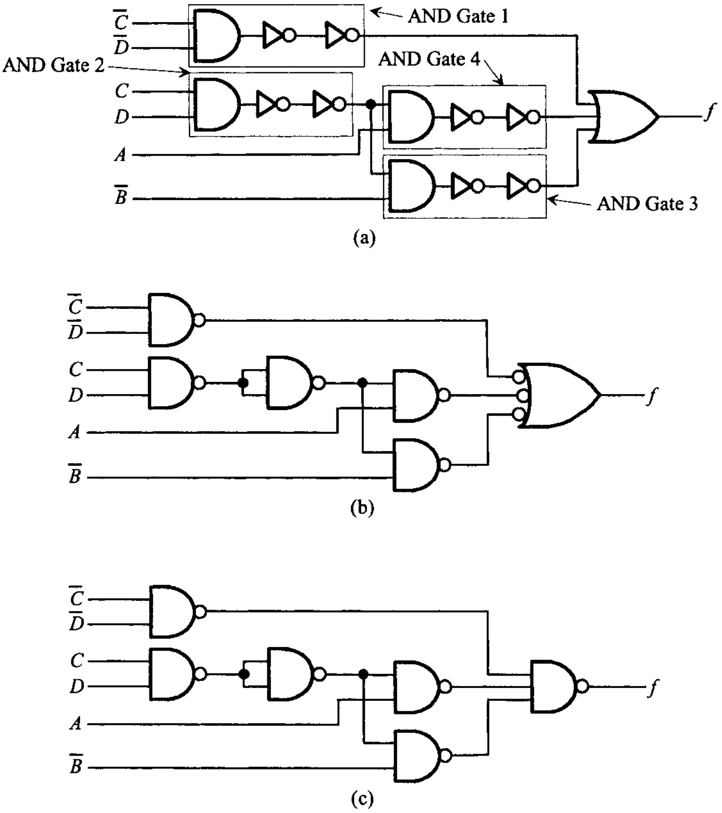


FIGURE 3.48 Steps for NAND gate implementation of Figure 3.47

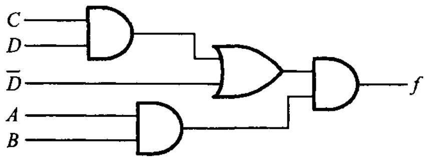


FIGURE 3.49 AND-OR implementation of Example 3.18

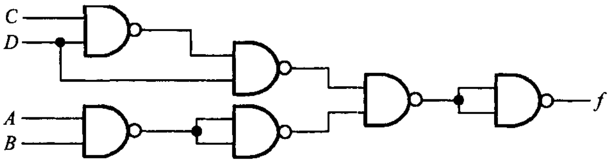


FIGURE 3.50 NAND gate implementation of Figure 3.49

3.9.2 NOR Gate Implementation

Figure 3.51 shows the NOR gate equivalent logic diagrams for NOT, OR, AND, and OR-invert logic operations. A Boolean function can be implemented using NOR gates by first obtaining the simplified expression of the function in terms of AND and OR gates. The function can then be converted to NOR logic. A function expressed in product-of-sums can be implemented using NOR gates.

Example 3.19

Implement the following function using NOR gates:

$$f = w(x + \bar{y})(x + z)$$

Assume both true and complemented inputs are available.

Solution

Figure 3.52 shows the AND-OR implementation of the logic equation. Figure 3.53 shows the NOR implementation.

Example 3.20

Implement the following function using NOR gates:

$$f = \bar{a}\,(b+c)\,(a + d)$$

Note that both true and complemented inputs are not available.

Solution

Figure 3.54 shows the AND-OR implementation of the logic equation. Figure 3.55 shows the NOR implementation.

3.9.3 XOR / XNOR Implementations

As mentioned before, the Exclusive-OR operation between two variables A and B can be expressed as

$$A \oplus B = A\bar{B} + \bar{A}B.$$

The Exclusive-NOR or equivalence operation between A and B can be expressed as

$$A \odot B = \overline{A \oplus B} = AB + \bar{A}\,\bar{B}.$$

The following identities are applicable to the Exclusive-OR operation:

i) $A \oplus 0 = A \bullet 1 + \bar{A} \bullet 0 = A$
ii) $A \oplus 1 = A \bullet 0 + \bar{A} \bullet 1 = \bar{A}$
iii) $A \oplus A = A \bullet \bar{A} + \bar{A} \bullet A = 0$
iv) $A \oplus \bar{A} = A \bullet A + \bar{A} \bullet \bar{A} = A + \bar{A} = 1$

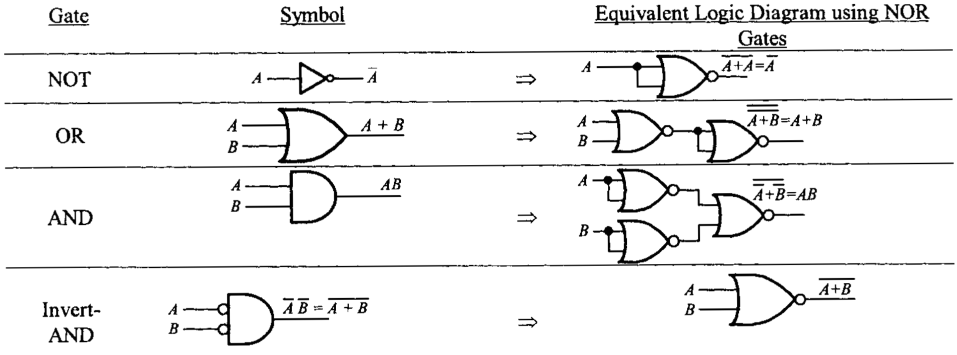

Gate
Symbol
Equivalent Logic Diagram using NOR Gates
NOT
A
$\overline{A}$
⇒
$\overline{A+A}=\overline{A}$
OR
A
B
$A + B$
$\overline{\overline{A+B}}=A+B$
AND
AB
$\overline{\overline{A}+\overline{B}}=AB$
Invert-AND
$\overline{A}\,\overline{B}=\overline{A+B}$
$\overline{A+B}$

FIGURE 3.51 Logic equivalents using NOR gates

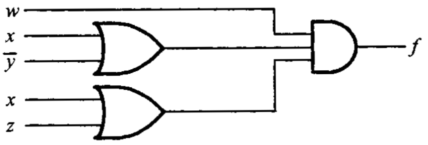
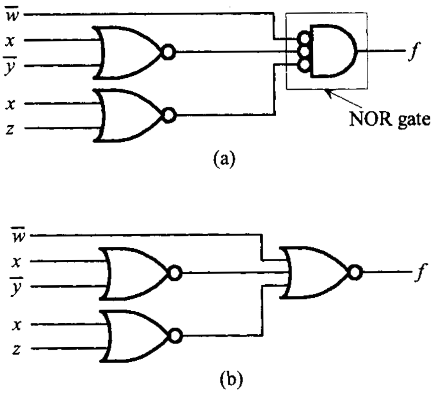


FIGURE 3.52 AND-OR implementation of Example 3.19



(a)

(b)

FIGURE 3.53 NOR implementation of Example 3.19

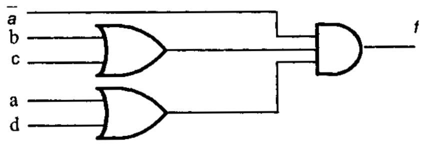
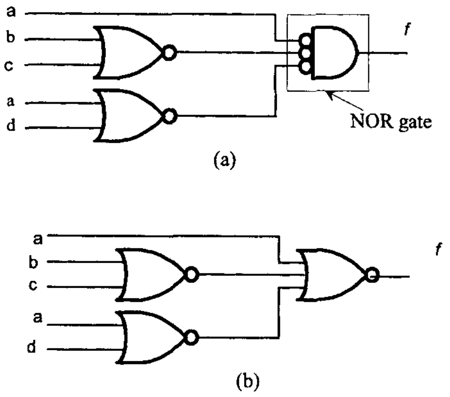


FIGURE 3.54 AND-OR implementation of Example 3.20



(a)

(b)

FIGURE 3.55 NOR implementation of Example 3.20

Finally, Exclusive-OR is commutative and associative:

$$\begin{aligned} A \oplus B &= B \oplus A \\ (A \oplus A) \oplus C &= A \oplus (B \oplus C) \\ &= A \oplus B \oplus C \end{aligned}$$

The Exclusive-NOR operation among three or more variables is called an "even function" because the Exclusive-NOR operation among three or more variables includes product terms in which each term contains an even number of 1's. For example, consider Exclusive-NORing three variables as follows:

$$f = \overline{A \oplus B \oplus C} = \overline{(A\bar{B} + \bar{A}B) \oplus C}$$

Let $D = A\bar{B} + \bar{A}B$. Then $\bar{D} = A\bar{B} + \bar{A}B = AB + \bar{A}\,\bar{B}$. Hence,

$$\begin{aligned} f &= \overline{D \oplus C} \\ &= DC + \bar{D}\,\bar{C} \\ &= (A\bar{B} + \bar{A}B)C + \overline{(A\bar{B} + \bar{A}B)}\bar{C} \\ &= (A\bar{B} + \bar{A}B)C + (AB + \bar{A}\,\bar{B})\bar{C} \end{aligned}$$

Hence,

$$f = A\bar{B}C + \bar{A}\,BC + AB\bar{C} + \bar{A}\,\bar{B}\,\bar{C}$$

Note that in this equation, $f = 1$ when one or more product terms in the equation are 1. However, by inspection, the binary equivalents of the right-hand side of the equation are 101, 011, 110, and 000. That is, the function is expressed as the logical sum (OR) of product terms containing even numbers of ones. Therefore, the function is called an even function. Similarly, it can be shown that Exclusive-OR operation among three or more variables is an odd function.

Exclusive-OR or Exclusive-NOR operation can be used for error detection and correction using parity during data transmission. Note that parity can be classified as either odd or even. The parity is defined by the number of 1's contained in a string of data bits. When the data contains an odd number of 1's, the data is said to have "odd parity"; On the other hand, the data has "even parity" when the number of 1's is even. To illustrate how parity is used as an error check bit during data transmission, consider Figure 3.56.

Suppose that Computer *X* is required to transmit a 3-bit message to Computer *Y*. To ensure that data is transmitted properly, an extra bit called the parity bit can be added by the transmitting Computer *X* before sending the data. In other words, Computer *X* generates the parity bit depending on whether odd or even parity is used during the transmission. Suppose that odd parity is used. The odd parity bit for the three-bit message

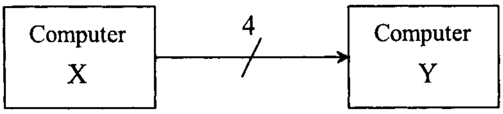
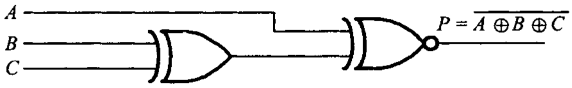
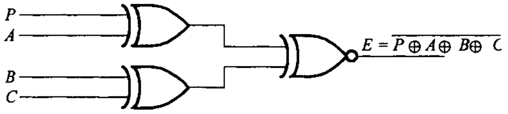


FIGURE 3.56 Parity generation and checking



(a) $P = \overline{A \oplus B \oplus C}$



(b) $E = \overline{P \oplus A \oplus B \oplus C}$

FIGURE 3.57 Implementation of parity generation and checking using XOR / XNOR gates

will be as follows:

Message			Odd Parity Bit
A	*B*	*C*	*P*
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Here $P = 1$ when the 3-bit message ABC contains an even number of 1's. Thus, the parity bit will ensure that the 3-bit message contains an odd number of 1's before transmission. $P = 1$ when the message contains an even number of 1's. Therefore, P is an even function. Thus,

$$P = \overline{A \oplus B \oplus C}.$$

The transmitting Computer X generates this parity bit. Computer X then transmits 4-bit information (a 3-bit message along with the parity bit) to Computer Y. Computer Y receives this 4-bit information and checks to see whether each 4-bit data item contains an odd number of 1's (odd parity). If the parity is odd, Computer Y accepts the 3-bit message; otherwise the computer sends the 4-bit information back to Computer X for retransmission. Note that Computer Y checks the parity of the transmitted data using the equation

$$E = \overline{P \oplus A \oplus B \oplus C}$$

Here the error $E = 1$ if the four bits have an even number of ones (even parity). That is, at least one of the four bits is changed during transmission. On the other hand, the error bit, $E = 0$ if the 4-bit data has an odd number of ones. Figure 3.57 shows the implementation of the parity bit, $P = \overline{A \oplus B \oplus C}$, and the error bit, $E = \overline{P \oplus A \oplus B \oplus C}$.

QUESTIONS AND PROBLEMS

3.1 Perform the following operations. Include your answers in hexadecimal.
$A6_{16}$ OR 31_{16}; $F7A_{16}$ AND $D80_{16}$; $36_{16} \oplus 2A_{16}$

3.2 Given $A = 1001_2$, $B = 1101_2$, find: A OR B; $B \wedge A$; $\overline{A}$; $A \oplus A$.

3.3 Perform the following operation: $A7_{16} \oplus FF_{16}$. What is the relationship of the result to $A7_{16}$?

3.4 Prove the following identities algebraically and by means of truth tables:
(a) $(A + B)\overline{(A + B)} = 0$
(b) $A + \overline{A}B = A + B$
(c) $XY + \overline{X}\,\overline{Y} + X\overline{Y} + \overline{X}Y = 1$
(d) $\overline{(A + \overline{A}B)} = \overline{A}\,\overline{B}$
(e) $(\overline{X} + Y)(X + \overline{Y}) = \overline{X \oplus Y}$
(f) $\overline{B}\,\overline{C} + ABC = \overline{A}\,\overline{C} = \overline{C \oplus (AB)}$

3.5 Simplify each of the following Boolean expressions as much as possible using identities:
(a) $XY + (1 \oplus X) + X\overline{Z} + X\overline{Y} + XZ$
(b) $AB\overline{C} + AB\overline{CD} + AB\overline{D}$
(c) $BC + ABC\overline{D} + \overline{A}BCD + ABCD$
(d) $(\overline{X} + \overline{Y})(\overline{XY}) + ZXY + X\overline{Z}Y$

3.6 Using DeMorgan's theorem, draw logic diagrams for $F = AB\overline{C} + \overline{A}\,\overline{B} + BC$
(a) Using only AND gates and inverters.
(b) Using only OR gates and inverters.
You may use two-input and three-input AND and OR gates for (a) and (b).

3.7 Using truth tables, express each one of the following functions and their complements in terms of sum of minterms and product of maxterms:
(a) $F = ABC + \overline{A}BD + \overline{A}\,\overline{B}\,\overline{C} + AC\overline{D}$
(b) $F = (W + X + Y)(W\overline{X} + Y)$

3.8 Express each of the following expressions in terms of minterms and maxterms.
(a) $F = B\overline{C} + \overline{A}B + B(A + C)$
(b) $F = (A + \overline{B} + C)(\overline{A} + B)$

3.9 Minimize each of the following functions using a K-map:
(a) $F(A, B, C) = \Sigma\, m(0, 1, 4, 5)$
(b) $F(A, B, C) = \Sigma\, m(0, 1, 2, 3, 6)$
(c) $F(X, Y, Z) = \Sigma\, m(0, 2, 4, 6)$

3.10 Minimize each of the following expressions for F using a K-map.
(a) $F(A, B, C) = \overline{B}\,\overline{C} + ABC + AB\overline{C}$
(b) $F(A, B, C) = \overline{A}B\overline{C} + BC$
(c) $F(A, B, C) = \overline{A}\,\overline{C} + A(\overline{B}\,\overline{C} + B\overline{C})$

3.11 Simplify each of the following functions for F using a K-map.

(a) $F(W, X, Y, Z) = \Sigma\, m(0, 1, 4, 5, 8, 9)$
(b) $F(A, B, C, D) = \Sigma\, m(0, 2, 8, 10, 12, 14)$
(c) $\overline{F}(A, B, C, D) = \Sigma\, m(2, 4, 5, 6, 7, 10, 14)$
(d) $F(W, X, Y, Z) = \Sigma\, m(2, 3, 6, 7, 8, 9, 12, 13)$
(e) $\overline{F}(W, X, Y, Z) = \Sigma\, m(0, 2, 4, 6, 8, 10, 12, 14)$
(f) $F(W, X, Y, Z) = \Sigma\, m(1, 3, 5, 7, 9, 11, 13, 15)$

3.12 Minimize each of the following expressions for $\overline{F}$ using a K-map in sums-of-product form:

(a) $F(W, X, Y, Z) = \overline{W}\,\overline{X}\,YZ + WYZ$
(b) $F = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}CD + ABCD$
(c) $F = (\overline{A} + \overline{B} + C + \overline{D})(\overline{A} + B + C + \overline{D})(A + \overline{B} + C + \overline{D})$

3.13 Find essential prime implicants and then minimize each of the following functions for F using a K-map:

(a) $F(A, B, C, D) = \Sigma\, m(3, 4, 5, 7, 11, 12, 15)$
(b) $F(W, X, Y, Z) = \Sigma\, m(2, 3, 6, 7, 8, 9, 12, 13, 15)$

3.14 Minimize each of the following functions for f using a K-map and don't care conditions, d.

(a) $f(A, B, C) = \Sigma\, m(1, 2, 4, 7)$
$d(A, B, C) = \Sigma\, m(5, 6)$
(b) $f(X, Y, Z) = \Sigma\, m(2, 6)$
$d(X, Y, Z) = \Sigma\, m(0, 1, 3, 4, 5, 7)$
(c) $f(A, B, C, D) = \Sigma\, m(0, 2, 3, 11)$
$d(A, B, C, D) = \Sigma\, m(1, 8, 9, 10)$
(d) $f(A, B, C, D) = \Sigma\, m(4, 5, 10, 11)$
$d(A, B, C, D) = \Sigma\, m(12, 13, 14, 15)$

3.15 Minimize the following expression using the Quine–McCluskey method. Verify the results using a K-map. Draw logic diagrams using NAND gates. Assume true and complemented inputs. $F(A, B, C, D) = \Sigma\, m(0, 1, 4, 5, 8, 12)$

3.16 Minimize the following expression using a K-map:
$F = AB + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + C\overline{D} + \overline{A}\,\overline{B}\,\overline{C}\,D$
and then draw schematics using:

(a) NAND gates.
(b) NOR gates.

3.17 Minimize the following function $F(A, B, C, D) = \Sigma\, m(6, 7, 8, 9)$ assuming that the condition $AB = 11_2$ can never occur. Draw schematics using:

(a) NAND gates.
(b) NOR gates.

3.18 It is desired to compare two 4-bit numbers for equality. If the two numbers are equal, the circuit will generate an output of 1. Draw a logic circuit using a minimum number of gates of your choice.

3.19 Show analytically that $A \oplus (A \oplus B) = B$.

3.20 Show that the Boolean function, $f = A \oplus B \oplus AB$ between two variables, *A and B,* can be implemented using a single two-input gate.

3.21 Design a parity generation circuit for a 5-bit data (4-bit message with an even parity bit) to be transmitted by computer X. The receiving computer Y will generate an error bit, $E = 1$, if the 5-bit data received has an odd parity; otherwise, $E = 0$. Draw logic diagrams for both parity generation and checking using XOR gates.

3.22 Draw a logic diagram for a two-input (A,B) Exclusive-OR operation using only four two-input (A,B) NAND gates. Assume that complemented inputs $\overline{A}$ and $\overline{B}$ are not available.

3.23 Determine by inspection whether the function, F in each of the following is odd or even, and comment on the result:

(a) $\overline{F} = A \oplus B \oplus C$ (b) $F = A \oplus B \oplus C$