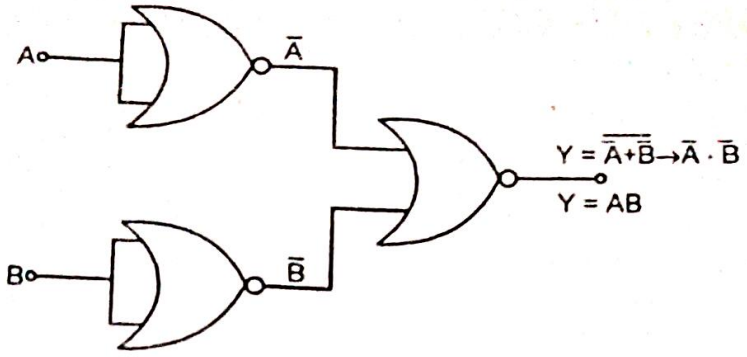
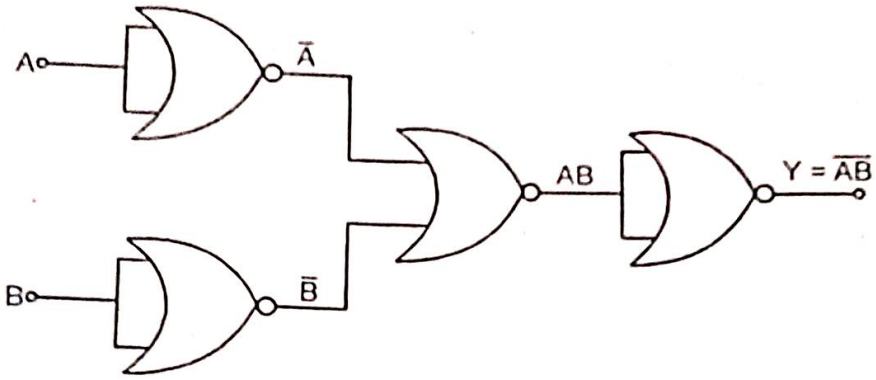
c) AND Gate



d) NAND Gate



6.4 Basic Laws of Boolean Algebra

a) Basic rules of Boolean addition are,

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 1$

Boolean addition is same as the logical OR.

b) Basic rules of Boolean multiplication are,

$0 . 0 = 0$

$0 . 1 = 0$

$1 . 0 = 0$

$1 . 1 = 1$

It is same as the logical AND.

c) Three of the basic laws of Boolean algebra are,

1) Commutative Laws

Law 1 $\Rightarrow A + B = B + A$

Law 2 $\Rightarrow A . B = B . A$

2) Associative Laws

Law 1 $\Rightarrow A + (B + C) = (A + B) + C$

Law 2 $\Rightarrow (AB) C = A (BC)$

3) Distributive Law

Law $\Rightarrow A (B + C) = AB + AC$

d) Absorption Laws

1) $A + AB = A$

2) $A . (A + B) = A$

3) $A + \bar{A} B = A + B$

4) $A . (\bar{A} + B) = A B$

e) Consensus Laws

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

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

f) Other basic laws are,

1) $A + 0 = A$
2) $A + 1 = 1$
3) $A + A = A$ (Idempotent)
4) $A + \bar{A} = 1$ (Complementary)

OR

5) $A . 0 = 0$
6) $A . 1 = A$
7) $A . A = A$ (Idempotent)
8) $A . \bar{A} = 0$ (Complementary)

AND

9) $\bar{\bar{A}} = A$ (Involution)
10) $A + AB = A$
11) $A + \bar{A} B = A + B$
12) $(A + B)(A + C) = A + BC$

g) Duality Theorem

The duality theorem says that, starting with a boolean relation, we can derive another boolean relation by

1) changing each OR sign to an AND sign.
2) changing each AND sign to an OR sign.
3) complementing any 0 or 1 appearing in the expression

Rule 1 say that $A + 0 = A$

Dual relation $A . 1 = A$

This is obtained by changing the OR sign to an AND sign and by complementing the 0 to get a 1.

(e.g) Distributive law states that

$$A (B + C) = AB + AC$$

By changing each OR and AND operation, we get the dual relation.

$$A + BC = (A + B)(A + C)$$

e Morgan's Theorem

1) $\overline{AB} = \bar{A} + \bar{B}$

2) $\overline{A + B} = \bar{A} \cdot \bar{B}$

blem: Verify all the laws with the help of truth table.

Introduction to half adder and full adder

Digital computers perform various arithmetic operations. The most basic operation he addition of two binary digits. The four elementary operations are,

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 10_2$

The first three operations produce a sum whose length is one digit, but when the last peration is performed sum is two digits.

The higher significant bit of this result is called CARRY and lower significant bit is called SUM.

The logic circuit which performs this operation is called a half-adder and the circuit which performs addition of three bits is a full-adder.

6.5.1 Half - Adder

The half-adder operation needs two binary inputs, augend and add end bits and two binary outputs sum and carry.

a) Block Schematic of Half Adder

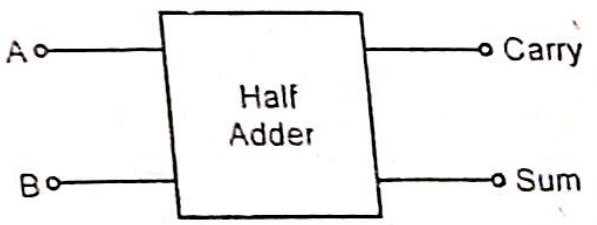


Figure 6.13

3.2 SUM-OF-PRODUCTS METHOD

Figure 3.4 shows the four possible ways to AND two input signals that are in complemented and uncomplemented form. These outputs are called *fundamental products*. Table 3.1 lists each fundamental product next to the input conditions producing a high output. For instance, $\bar{A}\bar{B}$ is high when A and B are low; $\bar{A}B$ is high when A is low and B is high; and so on. The fundamental products are also called *minterms*. Products $A'B'$, $A'B$, AB', AB are represented by m_0, m_1, m_2, and m_3 respectively. The suffix i of m_i comes from decimal equivalent of binary values (Table 3.1) that makes corresponding product term high.

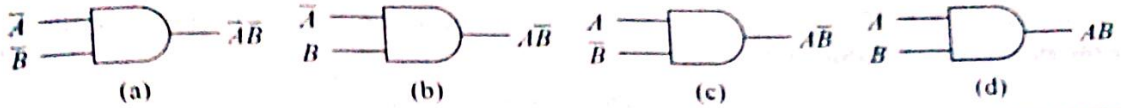


Fig. 3.4 *ANDing two variables and their complements.*

The idea of fundamental products applies to three or more input variables. For example, assume three input variables: A, B, C and their complements. There are eight ways to AND three input variables and their complements resulting in fundamental products of

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

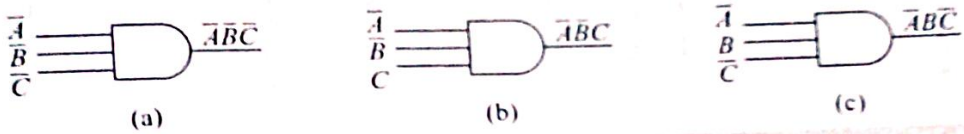


Fig. 3.5 *Examples of ANDing three variables and their complements.*

The above three variable minterms can alternatively be represented by m_0, m_1, m_2, m_3, m_4, m_5, m_6, and m_7 respectively. Note that, for n variable problem there can be 2^n number of minterms. Figure 3.5a shows the first fundamental product, Fig. 3.5b the second, and Fig. 3.5c the third. (For practice, draw the gates for the remaining fundamental products.) for twice variable case.

Table 3.2 summarizes the fundamental products by listing each one next to the input condition

Table 3.1 Fundamental Products for Two Inputs

A	B	Fundamental Product
0	0	$\bar{A}\bar{B}$
0	1	$\bar{A}B$
1	0	$A\bar{B}$
1	1	AB

Table 3.2 Fundamental Products for Three Inputs

A	B	C	Fundamental Products
0	0	0	$\bar{A}\bar{B}\bar{C}$
0	0	1	$\bar{A}\bar{B}C$
0	1	0	$\bar{A}B\bar{C}$
0	1	1	$\bar{A}BC$
1	0	0	$A\bar{B}\bar{C}$
1	0	1	$A\bar{B}C$
1	1	0	$AB\bar{C}$
1	1	1	ABC

that results in a high output. For instance, when $A = 1$, $B = 0$ and $C = 0$, the fundamental product results in an output of

$$Y = A\bar{B}\bar{C} = 1 \cdot \bar{0} \cdot \bar{0} = 1$$

Sum-of-Products Equation

Here is how to get the sum-of-products solution, given a truth table like Table 3.3. What you have to do is locate each output 1 in the truth table and write down the fundamental product. For instance, the first output 1 appears for an input of $A = 0$, $B = 1$, and $C = 1$. The corresponding fundamental product is $\bar{A}BC$. The next output 1 appears for $A = 1$, $B = 0$, and $C = 1$. The corresponding fundamental product is $A\bar{B}C$. Continuing like this, you can identify all the fundamental products, as shown in Table 3.4. To get the sum-of-products equation, all you have to do is OR the fundamental products of Table 3.4:

$$Y = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC \tag{3.17}$$

Alternate representation of Table 3.3,

$$Y = F(A, B, C) = \Sigma\, m\,(3, 5, 6, 7)$$

where 'Σ' symbolizes summation or logical OR operation that is performed on corresponding minterms and $Y = F(A, B, C)$ means Y is a function of three Boolean variables A, B and C.

Table 3.3 Design Truth Table

A	B	C	Y
0	0	0	0
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	1

Table 3.4 Fundamental Products for Table 3.3

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1 → $\bar{A}BC$
1	0	0	0
1	0	1	1 → $A\bar{B}C$
1	1	0	1 → $AB\bar{C}$
1	1	1	1 → ABC

Logic Circuit

After you have a sum-of-products equation, you can derive the corresponding logic circuit by drawing an AND-OR network, or what amounts to the same thing, a NAND-NAND network. In Eq. (3.17) each product is the output of a 3-input AND gate. Furthermore, the logical sum Y is the output of a 4-input OR gate. Therefore, we can draw the logic circuit as shown in Fig. 3.6. This AND-OR circuit is one solution to the design problem that we started with. In other words, the AND-OR circuit of Fig. 3.6 has the truth table given by Table 3.3.

We cannot build the circuit of Fig. 3.6 because a 4-input OR gate is not available as a TTL *chip* (a

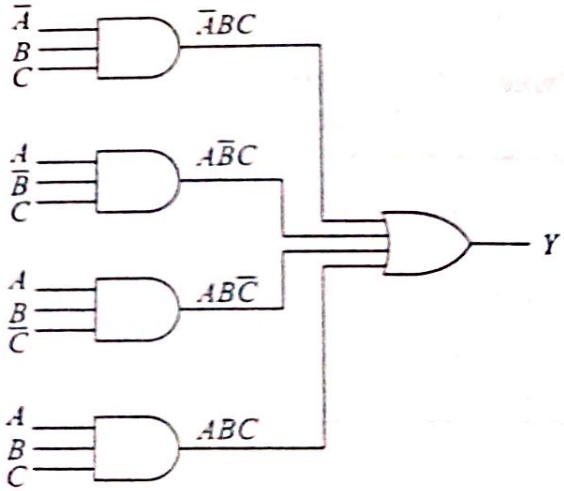


Fig. 3.6 *AND-OR solution.*

synonym for integrated circuit). But a 4-input NAND gate is. Figure 3.7 shows the logic circuit as a NAND-NAND circuit with TTL pin numbers. Also notice how the inputs come from a *bus*, a group of wires carrying logic signals. In Fig. 3.7, the bus has six wires with logic signals A, B, C, and their complements. Microcomputers are bus-organized, meaning that the input and output signals of the logic circuits are connected to buses.

EXAMPLE 3.4

Suppose a three-valuable truth table has a high output for these input conditions: 000, 010, 100, and 110. What is the sum-of-products circuit?

Solution

Here are the fundamental products:

$$000: \overline{A}\,\overline{B}\,\overline{C}$$
$$010: \overline{A}B\overline{C}$$
$$100: A\overline{B}\,\overline{C}$$
$$110: AB\overline{C}$$

When you OR these products, you get

$$Y = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + AB\overline{C}$$

The circuit of Fig. 3.7 will work if we reconnect the input lines to the bus as follows:

$\overline{A}$: pins 1 and 3
$\overline{B}$: pins 2 and 10
$\overline{C}$: pins 13, 5, 11, and 13
A : pins 9 and 1
B : pins 4 and 2

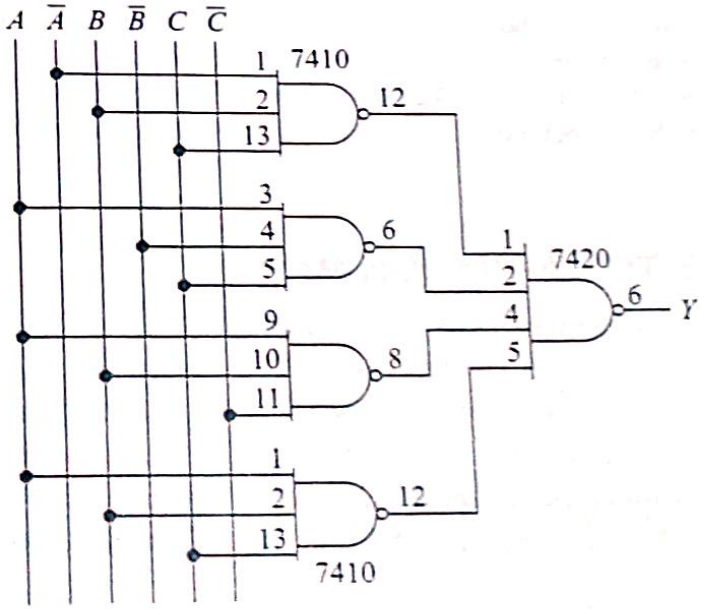


Fig. 3.7 *Combinational logic circuit.*

EXAMPLE 3.5

Simplify the Boolean equation in Example 3.4 and describe the logic circuit.

Solution

The Boolean equation is

$$Y = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + AB\overline{C}$$

Since $\overline{C}$ is common to each term, factor as follows:

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

Again, factor to get

$$Y = [\overline{A}(\overline{B} + B) + A(\overline{B} + B)]\overline{C}$$

Now, simplify the foregoing as follows:

$$Y = [\overline{A}(1) + A(1)]\overline{C} = (\overline{A} + A)\overline{C}$$

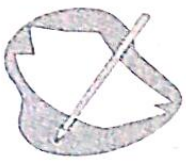
or

$$Y = \overline{C}$$

This final equation means that you don't even need a logic circuit. All you need is a wire connecting input $\overline{C}$ to output Y.

The lesson is clear. The AND-OR (NAND-NAND) circuit you get with the sum-of-products method is not necessarily as simple as possible. With algebra, you often can factor and reduce the sum-of-products equation to arrive at a simpler Boolean equation, which means a simpler logic circuit. A simpler logic circuit is preferred because it usually costs less to build and is more reliable.

SELF-TEST



4. How many fundamental products are there for two variables? How many for three variables?
5. The AND-OR or the NAND-NAND circuit obtained with the sum-of-products method is always the simplest possible circuit. (T or F)

3.3 TRUTH TABLE TO KARNAUGH MAP

A *Karnaugh map* is a visual display of the fundamental products needed for a sum-of-products solution. For instance, here is how to convert Table 3.5 into its Karnaugh map. Begin by drawing Fig. 3.8a. Note the variables and complements: the vertical column has $\overline{A}$ followed by A, and the horizontal row has $\overline{B}$ followed by B. The first output 1 appears for $A = 1$ and $B = 0$. The fundamental product for this input condition is $A\overline{B}$. Enter this fundamental product on the Karnaugh map as shown in Fig. 3.8b. This 1 represents the product $A\overline{B}$ because the 1 is in row A and column $\overline{B}$.

Similarly, Table 3.5 has an output 1 appearing for inputs of $A = 1$ and $B = 1$. The fundamental product is AB, which can be entered on the Karnaugh map as shown in Fig. 3.8c. The final step in drawing the Karnaugh map is to enter 0s in the remaining spaces (see Fig. 3.8d).

Table 3.5

A	B	Y
0	0	0
0	1	0
1	0	1
1	1	1

Table 3.6

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

In terms of decimal equivalence each position of Karnaugh map can be drawn as shown in Fig. 3.8b. Note that, Table 3.5 can be written using minterms as $Y = \Sigma\, m(2, 3)$ and Fig. 3.8e represents that.

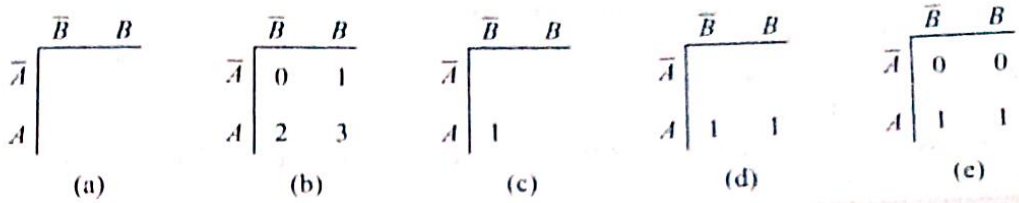


Fig. 3.8 *Constructing a Karnaugh map.*

Three-Variable Maps

Here is how to draw a Karnaugh map for Table 3.6 or for logic equation, $Y = F(A, B, C) = \Sigma m(2,6,7)$. First, draw the blank map of Fig. 3.9a. The vertical column is labeled $\overline{A}\,\overline{B}$, $\overline{A}B$, AB, and $A\overline{B}$. With this order, only one variable changes from complemented to uncomplemented form (or vice versa) as you move downward. In terms of decimal equivalence of each position the Karnaugh map is as shown in Fig. 3.9b. Note how minterms in the equation gets mapped into corresponding positions in the map.

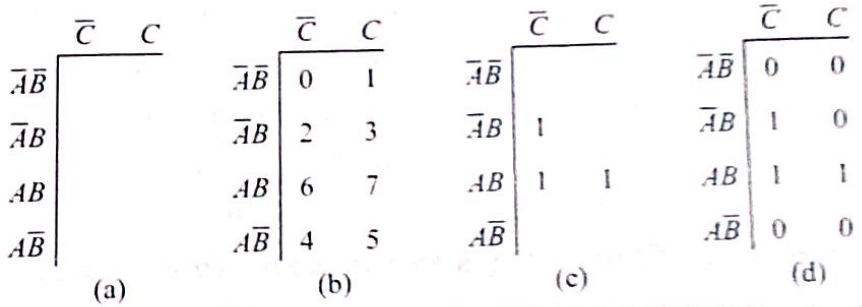


Fig. 3.9 *Three-variable Karnaugh map.*

Next, look for output 1s in Table 3.6. Output 1s appear for ABC inputs of 010, 110 and 111. The fundamental products for these input conditions are $\overline{A}B\overline{C}$, $AB\overline{C}$, and ABC Enter 1s for these products on the Karnaugh map (Fig. 3.9b).

The final step is to enter 0s in the remaining spaces (Fig. 3.9c).

Four-Variable Maps

Many digital computers and systems process 4-bit numbers. For instance, some digital chips will work with nibbles like 0000, 0001, 0010, and so on. For this reason, logic circuits are often designed to handle four input variables (or their complements). This is why you must know how to draw a four-variable Karnaugh map.

Here is an example. Suppose you have a truth table like Table 3.7. Start by drawing a blank map like Fig. 3.10a. Notice the order. The vertical column is $\overline{A}\,\overline{B}$, $\overline{A}B$, AB, and $A\overline{B}$. The horizontal row is $\overline{C}\,\overline{D}$, $\overline{C}D$, CD, and $C\overline{D}$. In terms of decimal equivalence of each position the Karnaugh map is as shown in Fig. 3.10b. In Table 3.7, you have output 1s appearing for $ABCD$ inputs of 0001, 0110, 0111, and 1110. The fundamental products for these input conditions are $\overline{A}\,\overline{B}\,\overline{C}D$, $\overline{A}BC\overline{D}$, $\overline{A}BCD$, and $ABC\overline{D}$. After entering 1s on the Karnaugh map, you have Fig. 3.10c. The final step of filling in 0s results in the complete map of Fig. 3.10d.

Table 3.7

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

SELF-TEST

6. What is a Karnaugh map?
7. How many entries are there on a four-variable Karnaugh map?

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

(a)

	$\overline{C}\,\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\,\overline{B}$	0	1	3	2
$\overline{A}B$	4	5	7	6
AB	12	13	15	14
$A\overline{B}$	8	9	11	10

(b)

	$\overline{C}\,\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\,\overline{B}$	0	1	3	2
$\overline{A}B$	4	5	7	6
AB	12	13	15	14
$A\overline{B}$	8	9	11	10

(c)

	$\overline{C}\,\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\,\overline{B}$	0	1	0	0
$\overline{A}B$	0	0	1	1
AB	0	0	0	1
$A\overline{B}$	0	0	0	0

(d)

Fig. 3.10 *Constructing a four-variable Karnaugh map.*

Entered Variable Map

As the name suggests, in *entered variable map* one of the input variable is placed inside Karnaugh map. This is done separately noting how it is related with output. This reduces the Karnaugh map size by one degree, i.e. a three variable problem that requires $2^3 = 8$ locations in Karnaugh map will

require $2^{(3-1)} = 4$ locations in entered variable map. This technique is particularly useful for mapping problems with more than four input variables.

We illustrate the technique by taking a three variable example, truth table of which is shown in Table 3.6. Let's choose C as *map entered variable* and see how output Y varies with C for different combinations of other two variables A and B. Fig. 3.11a shows the relation drawn from Table 3.6. For $AB = 00$ we find $Y = 0$ and is not dependent on C. For $AB = 01$ we find Y is complement of C thus we can write $Y = C'$. Similarly, for $AB = 10$, $Y = 0$ and for $AB = 11$, $Y = 1$. The corresponding entered variable map is shown in Fig. 3.11b. If we choose A as map entered variable we have table shown in Fig. 3.11c showing relation with Y for various combinations of BC; the corresponding entered variable map is shown in Fig. 3.11d.

A	B	Y
0	0	0
0	1	$\overline{C}$
1	0	0
1	1	1

(a)

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

(b)

B	C	Y
0	0	0
0	1	0
1	0	1
1	1	A

(c)

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

(d)

Fig. 3.11 *Entered variable map of truth table shown in Table 3.6.*

3.4 PAIRS, QUADS, AND OCTETS

Look at Fig. 3.112a. The map contains a pair of 1s that are horizontally adjacent (next to each other). The first 1 represents the product $ABCD$; the second 1 stands for the product $ABC\overline{D}$. As we move from the first 1 to the second 1, only one variable goes from uncomplemented to complemented form (D to $\overline{D}$); the other variables don't change form (A, B and C remain uncomplemented). Whenever this happens, you can *eliminate the variable that changes form.*

	$\overline{C}\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\overline{B}$	0	0	0	0
$\overline{A}B$	0	0	0	0
AB	0	0	1	1
$A\overline{B}$	0	0	0	0

(a)

	$\overline{C}\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\overline{B}$	0	0	0	0
$\overline{A}B$	0	0	0	0
AB	0	0	1	1
$A\overline{B}$	0	0	0	0

(b)

Fig. 3.12 *Horizontally adjacent 1s.*

Proof

The sum-of-products equation corresponding to Fig. 3.12a is

$$Y = ABCD + ABC\overline{D}$$

which factors into

$$Y = ABC(D + \overline{D})$$

Since D is ORed with its complement, the equation simplifies to

$$Y = ABC$$

In general, a pair of horizontally adjacent 1s like those of Fig. 3.12a means the sum-of-products equation will have a variable and a complement that drop out as shown above.

For easy identification, we will encircle two adjacent 1s as shown in Fig. 3.12b. Two adjacent 1s such as these are called a pair. In this way, we can tell at a glance that one variable and its complement will drop out of the corresponding Boolean equation. In other words, an encircled pair of 1s like those of Fig. 3.12b no longer stand for the ORing of two separate products, $ABCD$ and $ABC\overline{D}$. Rather, the encircled pair is visualized as representing a single reduced product ABC.

Here is another example. Figure 3.13a shows a pair of 1s that are vertically adjacent. These 1s correspond to $ABC\overline{D}$ and $AB\overline{C}\overline{D}$. Notice that only one variable changes from uncomplemented to complemented form (B to $\overline{B}$). Therefore, B and $\overline{B}$ can be factored and eliminated algebraically, leaving a reduced product of $AC\overline{D}$.

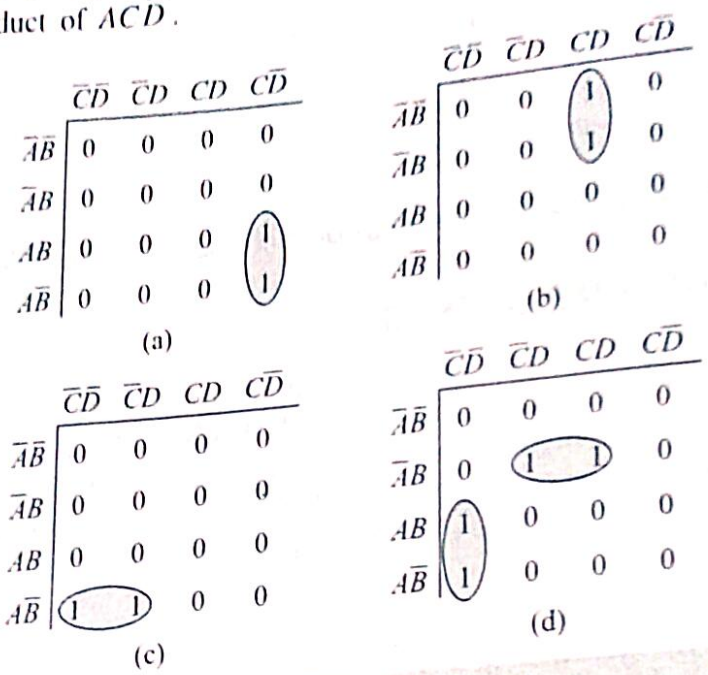


Fig. 3.13 *Examples of pairs.*

More Examples

Whenever you see a pair of horizontally or vertically adjacent 1s, you can eliminate the variable that appears in both complemented and uncomplemented form. The remaining variables (or their complements) will be the only ones appearing in the single-product term corresponding to the pair of 1s. For instance, a glance at Fig. 3.13b indicates that B goes from complemented to uncomplemented form when we move from the upper to the lower 1; the other variables remain the same. Therefore, the encircled pair of 1s in Fig. 3.13b, represents the product $\overline{A}CD$. Likewise, given the pair of 1s in Fig. 3.13c, the only change is from $\overline{D}$ to D. So the encircled pair of 1s stands for the product $A\overline{B}\overline{C}$.

If more than one pair exists on a Karnaugh map, you can OR the simplified products to get the Boolean equation. For instance, the lower pair of Fig. 3.13d represents the simplified product $A\overline{C}\,\overline{D}$; the upper pair stands for $\overline{A}BD$. The corresponding Boolean equation for this map is

$$Y = A\overline{C}\,\overline{D} + \overline{A}BD$$

Quad

A *quad* is a group of four 1s that are horizontally or vertically adjacent. The 1s may be end-to-end, as shown in Fig. 3.14a, or in the form of a square, as in Fig. 3.14b. When you see a quad, always encircle it because it leads to a simpler product. In fact, a quad eliminates *two variables and their complements.*

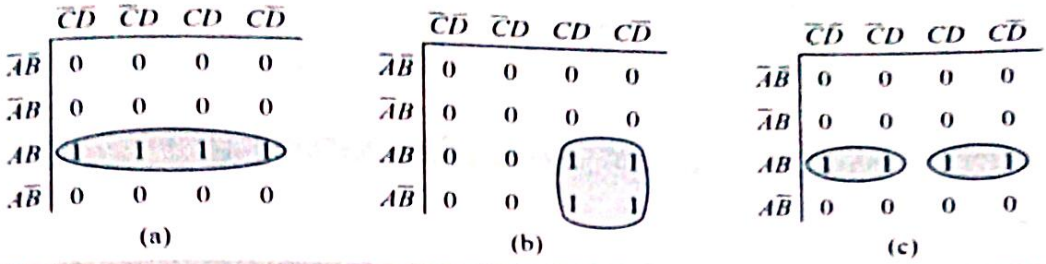


Fig. 3.14 *Examples of quads.*

Here is why a quad eliminates two variables and their complements. Visualize the four 1s of Fig. 3.14a as two pairs (see Fig. 3.14c). The first pair represents $AB\overline{C}$; the second pair stands for ABC. The Boolean equation for these two pairs is

$$Y = AB\overline{C} + ABC$$

This factors into

$$Y = AB(\overline{C} + C)$$

which reduces to

$$Y = AB$$

So, the quad of Fig. 3.14a represents a product whose two variables and their complements have dropped out.

A similar proof applies to any quad. You can visualize it as two pairs whose Boolean equation leads to a single product involving only two variables or their complements. There's no need to go through the algebra each time. Merely step through the different 1s in the quad and determine which two variables go from complemented to uncomplemented form (or vice versa); these are the variables that drop out.

For instance, look at the quad of Fig. 3.14b. Pick any 1 as a starting point. When you move horizontally, D is the variable that changes form. When you move vertically, B changes form. Therefore, the remaining variables (A and C) are the only ones appearing in the simplified product. In other words, the simplified equation for the quad of Fig. 3.14b is

$$Y = AC$$

The Octet

Besides pairs and quads, there is one more group to adjacent 1s to look for: the *octet*. This is a group of eight 1s like those of Fig. 3.15a on the next page. An octet like this eliminates *three variables and their complements*. Here's why. Visualize the octet as two quads (see Fig. 3.15b). The equation for these two quads is

$$Y = A\overline{C} + AC$$

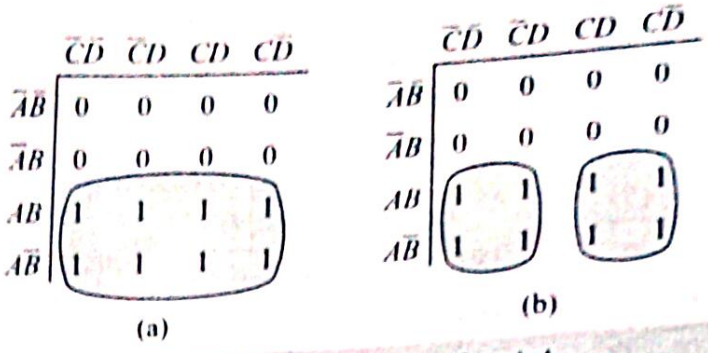


Fig. 3.15 *Example of octet.*

After factoring,

$$Y = A(\overline{C} + C)$$

But this reduces to

$$Y = A$$

So the octet of Fig. 3.15a means three variables and their complements drop out of the corresponding product.

A similar proof applies to any octet. From now on don't bother with the algebra. Merely step through the 1s of the octet and determine which three variables change form. These are the variables that drop out.

SELF-TEST

8. On a Karnaugh map, two adjacent 1s are called a ____.
9. On a Karnaugh map, an octet contains how many 1s?

3.5 KARNAUGH SIMPLIFICATIONS

As you know, a pair eliminates one variable and its complement, a quad eliminates two variables and their complements, and an octet eliminates three variables and their complements. Because of this, after you draw a Karnaugh map, encircle the octets first, the quads second, and the pairs last. In this way, the greatest simplification results.

An Example

Suppose you have translated a truth table into the Karnaugh map shown in Fig. 3.16a. First, look

for octets. There are none. Next, look for quads. When you find them, encircle them. Finally, look for and encircle pairs. If you do this correctly, you arrive at Fig. 3.16b.

The pair represents the simplified product $\overline{A}\,\overline{B}D$, the lower quad stands for $A\overline{C}$, and the quad on the right represents $C\overline{D}$. By ORing these simplified products, we get the Boolean equation corresponding to the entire Karnaugh map:

$$Y = \overline{A}\,\overline{B}D + A\overline{C} + C\overline{D} \tag{3.18}$$

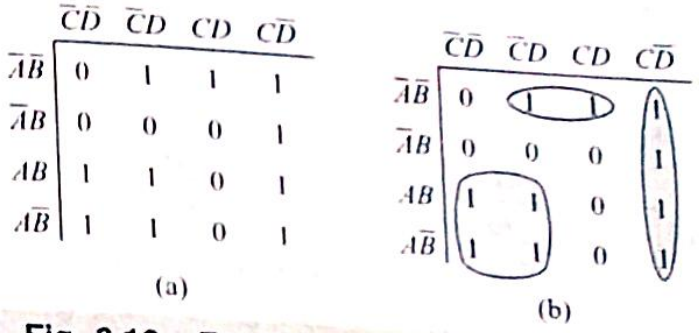


Fig. 3.16 *Encircling octets, quads and pairs.*

Overlapping Groups

You are allowed to use the same 1 more than once. Figure 3.17a illustrates this idea. The 1 representing the fundamental product $AB\overline{C}D$ is part of the pair and part of the octet. The simplified equation for the overlapping groups is

$$Y = A + B\overline{C}D \tag{3.19}$$

It is valid to encircle the 1s as shown in Fig. 3.17b, but then the isolated 1 results in a more complicated equation:

$$Y = A + \overline{A}B\overline{C}D$$

So, always overlap groups if possible. That is, use the 1s more than once to get the largest groups you can.

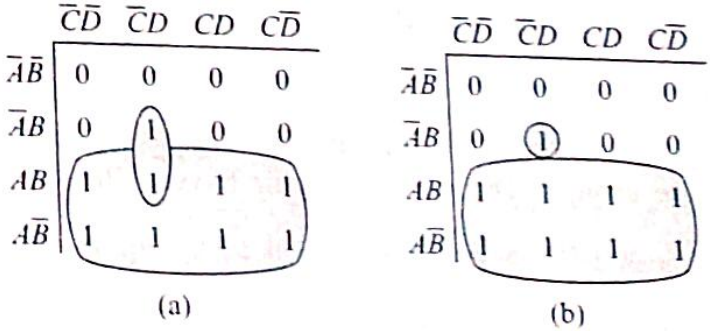


Fig. 3.17 *Overlapping groups.*

Rolling the Map

Another thing to know about is rolling. Look at Fig. 3.18a on the next page. The pairs result in this equation:

$$Y = B\overline{C}\,\overline{D} + BC\overline{D} \tag{3.20}$$

Visualize picking up the Karnaugh map and rolling it so that the left side touches the right side. If you are visualizing correctly, you will realize the two pairs actually form a quad. To indicate this, draw half circles around each pair, as shown in Fig. 3.18b. From this viewpoint, the quad of Fig. 3.18b has the equation

$$Y = B\overline{D} \tag{3.21}$$

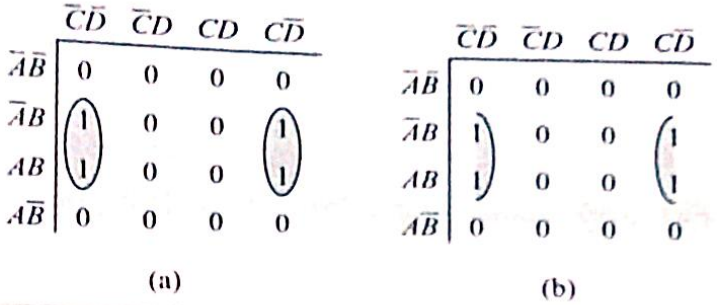


Fig. 3.18 *Rolling the Karnaugh map.*

Why is rolling valid? Because Eq. (3.20) can be algebraically simplified to Eq. (3.21). The proof starts with Eq. (3.20):

$$Y = B\overline{C}\,\overline{D} + BC\overline{D}$$

This factors into

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

which reduces to

$$Y = B\overline{D}$$

But this final equation is the one that represents a rolled quad like Fig. 3.18b. Therefore, 1s on the edges of a Karnaugh map can be grouped with 1s on opposite edges.

More Examples

If possible, roll and overlap to get the largest groups you can find. For instance, Fig. 3.19a shows an inefficient way to encircle groups. The octet and pair have a Boolean equation of

$$Y = \overline{C} + BC\overline{D}$$

You can do better by rolling and overlapping as shown in Fig. 3.19b; the Boolean equation now is

$$Y = \overline{C} + B\overline{D}$$

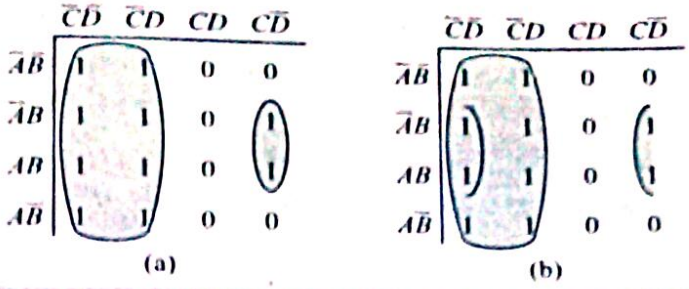


Fig. 3.19 *Rolling and overlapping.*

Here is another example. Figure 3.20a shows an inefficient grouping of 1s; the corresponding equation is

$$Y = \overline{C} + \overline{A}C\overline{D} + A\overline{B}C\overline{D}$$

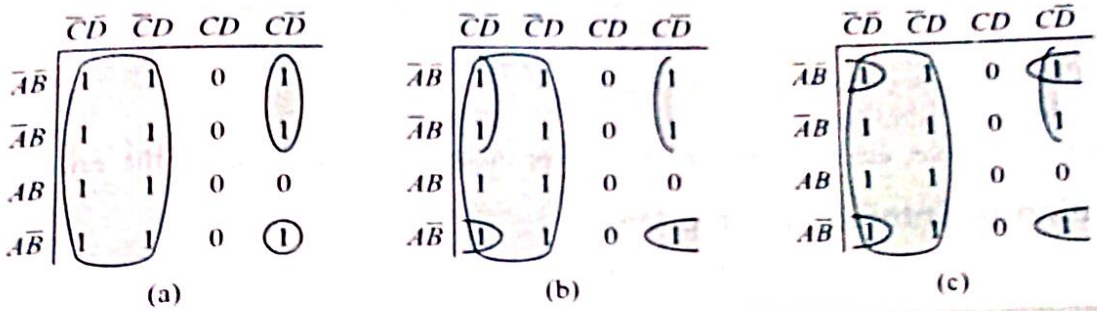


Fig. 3.20 *Different ways of encircling groups.*

If we roll and overlap as shown in Fig. 3.20b, the equation is simpler:

$$Y = \overline{C} + \overline{A}\overline{D} + A\overline{B}\overline{D}$$

It is possible to group the 1s as shown in Fig. 3.20c. The equation now becomes

$$Y = \overline{C} + \overline{A}\overline{D} + \overline{B}\overline{D} \tag{3.22}$$

Compare this with the preceding equation. As you can see, the equations are comparable in simplicity. Either grouping (Fig. 3.20b or c) is valid; therefore, you can use whichever you like.

Eliminating Redundant Groups

After you have finished encircling groups, eliminate any *redundant group*. This is a group whose 1s are already used by other groups. Here is an example. Given Fig. 3.21a, encircle the quad to get Fig. 3.21b. Next, group the remaining 1s into pairs by overlapping (Fig. 3.21c). In Fig. 3.21c, all the 1s of the quad are used by the pairs. Because of this, the quad is redundant and can be eliminated to get Fig. 3.21d. As you see, all the 1s are covered by the pairs. Figure 3.21d contains one less product than Fig. 3.21c; therefore, Fig. 3.21d is the most efficient way to group the 1s.

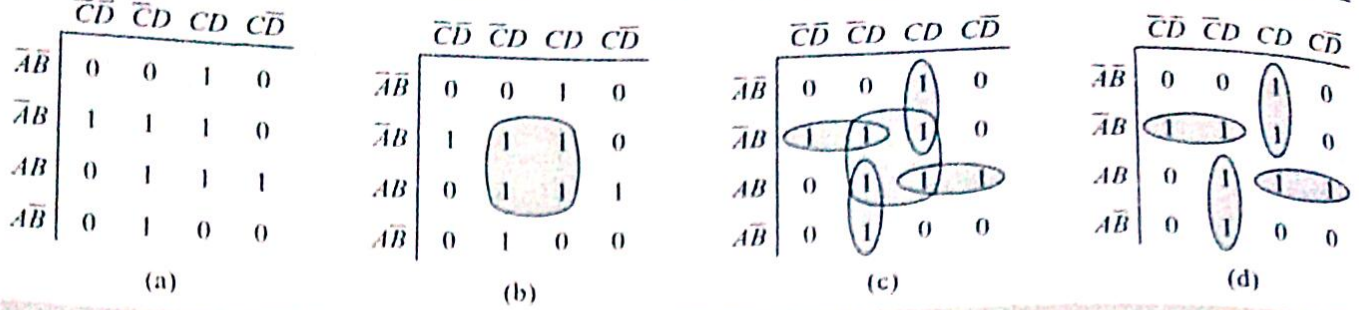


Fig. 3.21 *Eliminating an unnecessary group.*

Conclusion

Here is a summary of the Karnaugh-map method for simplifying Boolean equations:

1. Enter a 1 on the Karnaugh map for each fundamental product that produces a 1 output in the truth table. Enter 0s elsewhere.
2. Encircle the octets, quads, and pairs. Remember to roll and overlap to get the largest groups possible.
3. If any isolated 1s remain, encircle each.
4. Eliminate any redundant group.
5. Write the Boolean equation by ORing the products corresponding to the encircled groups.

Simplification of Entered Variable Map

This is similar to Karnaugh map method. Refer to entered variable maps shown in Fig. 3.11. The groupings for these are as shown in Fig. 3.22a and Fig. 3.22b. Note that in Fig. 3.22a C' is grouped with 1 to get a larger group as 1 can be written as $1 = 1 + C'$. Similarly A is grouped with 1 in Fig. 3.22b.

Next, the product term representing each group is obtained by including map entered variable in the group as an additional ANDed term. Thus, group 1 of Fig. 3.22a gives $B.(C') = BC'$ and group 2 gives $AB.(1) = AB$ resulting $Y = BC' + AB$.

In Fig. 3.22b, group 1 gives product term $B.(A) = AB$ and group 2 gives $BC'.(1) = BC'$ so that $Y = BC' + AB$. The final expression is same for both as they represent the same truth table (Table 3.6).

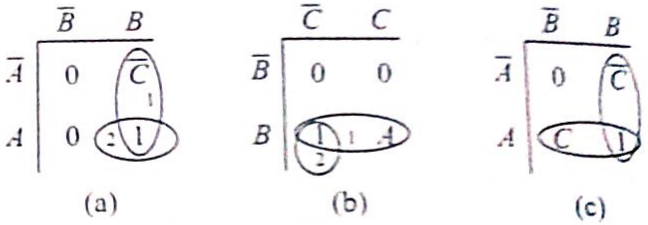


Fig. 3.22 *Simplification of entered variable map.*

Note that, entered variable map shown Fig. 3.22c for a different truth table (Take it as an exercise to prepare that truth table) has only two product terms and doesn't need a separate coverage of 1. This is because one can write $1 = C + C'$ and C is included in one group while C' in other. The output of this map can be written as $Y = AC + BC'$.

EXAMPLE 3.6

What is the simplified Boolean equation for the following logic equation expressed by minterms?

$$Y = F(A, B, C, D) = \Sigma m(7, 9, 10, 11, 12, 13, 14, 15)$$

Solution

We know, each minterm makes corresponding location in Karnaugh map 1 and thus Fig. 3.23a represents the given equation. There are no octets, but there is a quad as shown in Fig. 3.23b. By overlapping, we can find two more quads (see Fig. 3.23c). We can encircle the remaining 1 by making it part of an overlapped pair (Fig. 3.23d). Finally, there are no redundant groups.

The horizontal quad of Fig. 3.23d corresponds to a simplified product AB. The square quad on the right corresponds to AC, while the one on the left stands for AD. The pair represents BCD. By ORing these products, we get the simplified Boolean equation:

$$Y = AB + AC + AD + BCD \tag{3.23}$$

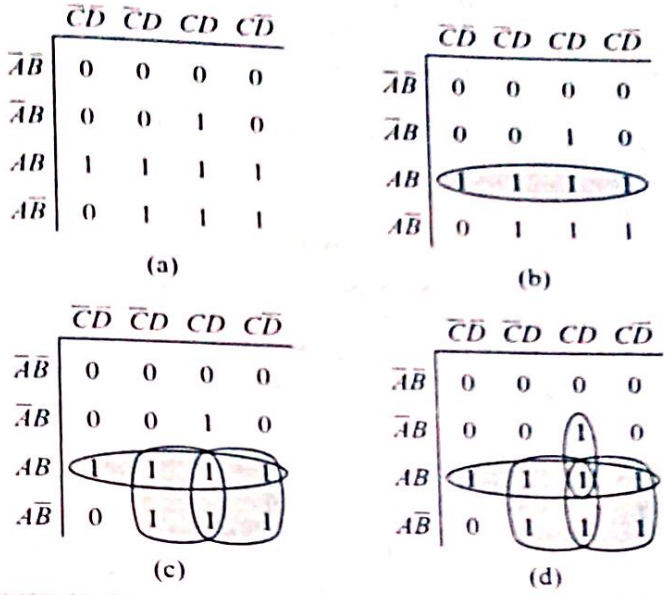


Fig. 3.23 *Using the Karnaugh map.*

SELF-TEST

10. Write the sum-of-product terms for the entries in Fig. 3.18. Use Boolean algebra to simplify the expression.

3.6 DON'T-CARE CONDITIONS

In some digital systems, certain input conditions never occur during normal operation; therefore, the corresponding output never appears. Since the output never appears, it is indicated by an X in the truth table. For instance, Table 3.8 on the next page shows a truth table where the output is low for all input entries from 0000 to 1000, high for input entry 1001, and an X for 1010 through 1111. The X is called a *don't-care condition*. Whenever you see an X in a truth table, you can let it equal either 0 or 1, whichever produces a simpler logic circuit.

Figure 3.24a shows the Karnaugh map of Table 3.8 with don't cares for all inputs from 1010 to 1111. These don't cares are like wild cards in poker because you can let them stand for whatever you like. Figure 3.24b shows the most efficient way to encircle the 1. Notice two crucial ideas. First, the 1 is included in a quad, the largest group you can find if you visualize all *X*'s as 1s. Second, after the 1 has been encircled, all *X*'s outside the quad are visualized as 0s. In this way, the *X*'s are used to the best possible advantage. As already mentioned, you are free to do this because don't cares correspond to input conditions that never appear.

The quad of Fig. 3.24b results in a Boolean equation of

$$Y = AD$$

The logic circuit for this is an AND gate with inputs of *A* and *D*, as shown in Fig. 3.24c. You can check this logic circuit by examining Table 3.8. The possible inputs are from 0000 to 1001; in this range a high *A* and a high *D* produce a high *Y* only for input condition 1001.

Table 3.8 Truth Table with Don't-Care Conditions

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

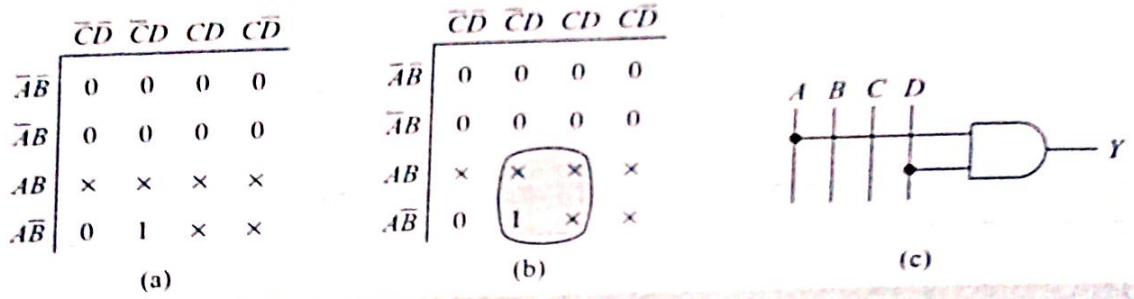


Fig. 3.24 *Don't care conditions.*

Remember these ideas about don't-care conditions:

1. Given the truth table, draw a Karnaugh map with 0s, 1s, and don't cares.
2. Encircle the actual 1s on the Karnaugh map in the largest groups you can find by treating the don't cares as 1s.
3. After the actual 1s have been included in groups, disregard the remaining don't cares by visualizing them as 0s.

EXAMPLE 3.7

Suppose Table 3.8 has high output for an input of 0000, low output, for 0001 to 1001, and don't cares for 1010 to 1111. What is the simplest logic circuit with this truth table?

Solution

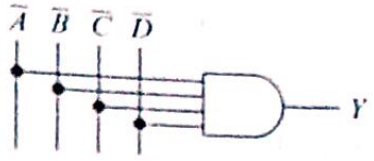
The truth table has a 1 output only for the input condition 0000. The corresponding fundamental product is $\bar{A}\bar{B}\bar{C}\bar{D}$. Figure 3.25a shows the Karnaugh map with a 1 for the fundamental product, 0s for inputs 0001 to 1001, and X's for inputs 1010 to 1111. In this case, the don't cares are of no help. The best we can do is to encircle the isolated 1, while treating the don't cares as 0s. So, the Boolean equation is

$$Y = \bar{A}\bar{B}\bar{C}\bar{D}$$

Figure 3.25b shows the logic circuit. The 4-input AND gate produces a high output only for the input condition $A = 0$, $B = 0$, $C = 0$, and $D = 0$.

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	(1)	0	0	0
$\bar{A}B$	0	0	0	0
AB	×	×	×	×
$A\bar{B}$	0	0	×	×

(a)



(b)

Fig. 3.25 *Decoding 0000.*

EXAMPLE 3.8

Give the simplest logic circuit for following logic equation where d represents don't care condition for following locations.

$$F(A, B, C, D) = \Sigma m(7) + d(10, 11, 12, 13, 14, 15)$$

Solution

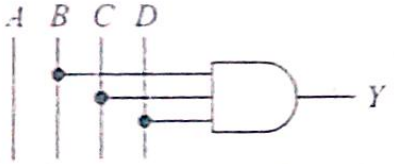
Figure 3.26a is the Karnaugh map. The most efficient encircling is to group the 1s into a pair using the don't care as shown. Since this is the largest group possible, all remaining don't cares are treated as 0s. The equation for the pair is

$$Y = BCD$$

and Fig. 3.26b is the logic circuit. This 3.input AND gate produces a high output only for an input of $A = 0$, $B = 1$, $C = 1$, and $D = 1$ because the input possibilities range only from 0000 to 1001.

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0
$\bar{A}B$	0	0	(1)	0
AB	×	×	(×)	×
$A\bar{B}$	0	0	×	×

(a)



(b)

Fig. 3.26 *Decoding 0111.*

SELF-TEST

11. What is meant by a don't-care condition on a Karnaugh map? How is it indicated?
12. How can using don't cares aid circuit simplification?

3.7 PRODUCT-OF-SUMS METHOD

With the sum-of-products method the design starts with a truth table that summarizes the desired input-output conditions. The next step is to convert the truth table into an equivalent sum-of-products equation. The final step is to draw the AND-OR network or its NAND-NAND equivalent.

The product-of-sums method is similar. Given a truth table, you identify the fundamental sums needed for a logic design. Then by ANDing these sums, you get the product-of-sums equation corresponding to the truth table. But there are some differences between the two approaches. With the sum-of-products method, the fundamental product produces an output 1 for the corresponding input condition. But with the product-of-sums method, the fundamental sum produces an output 0 for the corresponding input condition. The best way to understand this distinction is with an example.

Converting a Truth Table to an Equation

Suppose you are given a truth table like Table 3.9 and you want to get the product-of-sums equation. What you have to do is locate each output 0 in the truth table and write down its fundamental sum. In Table 3.9, the first output 0 appears for $A = 0$, $B = 0$, and $C = 0$. The fundamental sum for these inputs is $A + B + C$. Why? Because this produces an output zero for the corresponding input condition:

$$Y = A + B + C = 0 + 0 + 0 = 0$$

Table 3.9

A	B	C	Y	Maxterm
0	0	0	$0 \rightarrow A + B + C$	M_0
0	0	1	1	M_1
0	1	0	1	M_2
0	1	1	$0 \rightarrow A + \overline{B} + \overline{C}$	M_3
1	0	0	1	M_4
1	0	1	1	M_5
1	1	0	$0 \rightarrow \overline{A} + \overline{B} + C$	M_6
1	1	1	1	M_7

The second output 0 appears for the input condition of $A = 0$, $B = 1$, and $C = 1$. The fundamental sum for this is $A + \overline{B} + \overline{C}$. Notice that B and C are complemented because this is the only way to get a logical sum of 0 for the given input conditions:

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

Similarly, the third output 0 occurs for $A = 1$, $B = 1$, and $C = 0$; therefore, its fundamental sum is $\bar{A} + \bar{B} + C$:

$$Y = \bar{A} + \bar{B} + C = \bar{1} + \bar{1} + 0 = 0 + 0 + 0 = 0$$

Table 3.9 shows all the fundamental sums needed to implement the truth table. Notice that each variable is complemented when the corresponding input variable is a 1; the variable is uncomplemented when the corresponding input variable is 0. To get the product-of-sums equation, all you have to do is AND the fundamental sums:

$$Y = (A + B + C)(A + \bar{B} + \bar{C})(\bar{A} + \bar{B} + C) \tag{3.24}$$

This is the product-of-sums equation for Table 3.9.

As each product term was called minterm in SOP representation in POS each sum term is called *maxterm* and is designated by M_i as shown in Table 3.9. Equation 3.24 in terms of maxterm can be represented as

$$Y = F(A, B, C) = \Pi M(0, 3, 6)$$

where 'Π' symbolizes product, i.e. AND operation.

Logic Circuit

After you have a product-of-sums equation, you can get the logic circuit by drawing an OR-AND network, or if you prefer, a NOR-NOR network. In Eq. (3.24) each sum represents the output of a 3-input OR gate. Furthermore, the logical product Y is the output of a 3-input AND gate. Therefore, you can draw the logic circuit as shown in Fig. 3.27.

A 3-input OR gate is not available as a TTL chip. So, the circuit of Fig. 3.27 is not practical. With De Morgan's first theorem, however, you can replace the OR-AND circuit of Fig. 3.27 by the NOR-NOR circuit of Fig. 3.28.

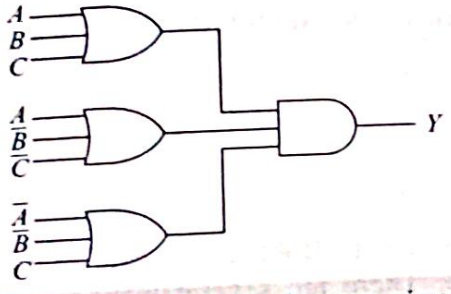


Fig. 3.27 *Product-of-sums circuit.*

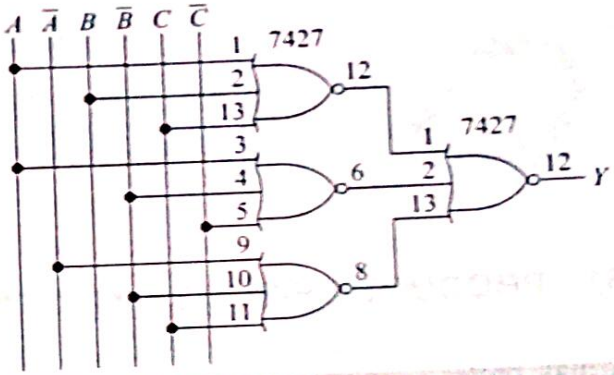


Fig. 3.28

Conversion between SOP and POS

We have seen that SOP representation is obtained by considering ones in a truth table while POS comes considering zeros. In SOP, each one at output gives one AND term which is finally ORed.

In POS, each zero gives one OR term which is finally ANDed. Thus SOP and POS occupy complementary locations in a truth table and one representation can be obtained from the other by
(i) identifying complementary locations,
(ii) changing minterm to maxterm or reverse, and finally
(iii) changing summation by product or reverse.

Thus Table 3.9 can be represented as

$$Y = F(A, B, C) = \Pi M(0, 3, 6) = \Sigma m(1, 2, 4, 5, 7)$$

Similarly Table 3.4 can be represented as

$$Y = F(A, B, C) = \Sigma m(3, 5, 6, 7) = \Pi M(0, 1, 2, 4)$$

EXAMPLE 3.9

Suppose a truth table has a low output for the first three input conditions: 000, 001, and 010. If all other outputs are high, what is the product-of-sums circuit?

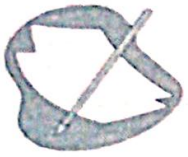
Solution

The product-of-sums equation is

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

The circuit of Fig. 3.28 will work if we reconnect the input lines as follows:
A : pins 1, 3, and 9
B : pins 2 and 4
C : pins 13 and 11
$\overline{B}$: pin 10
$\overline{C}$: pin 5

SELF-TEST



13. A product-of-sums expression leads to what kind of logic circuit?
14. Explain how to convert the complementary NAND-NAND circuit into its dual NOR-NOR circuit.

3.8 PRODUCT-OF-SUMS SIMPLIFICATION

After you write a product-of-sums equation, you can simplify it with Boolean algebra. Alternatively, you may prefer simplification based on the Karnaugh map. There are several ways of using the Karnaugh map. One can use a similar technique as followed in SOP representation but by forming largest group of zeros and then replacing each group by a sum term. The variable going in the formation of sum term is inverted if it remains constant with a value 1 in the group and it is not inverted if that value is 0. Finally, all the sum terms are ANDed to get simplest POS form. We illustrate this in Examples 3.11 and 3.12. In this section we also present an interesting alternative to above technique.

Sum-of-Products Circuit

Suppose the design starts with a truth table like Table 3.10. The first thing to do is to draw the Karnaugh map in the usual way to get Fig. 3.29a. The encircled groups allow us to write a sum-of-products equation:

$$Y = \overline{A}\,\overline{B} + AB + AC$$

Figure 3.29b shows the corresponding NAND-NAND circuit.

Table 3.10

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Complementary Circuit

To get a product-of-sums circuit, begin by complementing each 0 and 1 on the Karnaugh map of Fig. 3.29a. This results in the complemented map shown in Fig. 3.29c. The encircled 1s allow us to write the following sum-of-products equation:

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

Why is this $\overline{Y}$ instead of Y? Because complementing the Karnaugh map is the same as complementing the output of the truth table, which means the sum-of-products equation for Fig. 3.29c is for $\overline{Y}$ instead of Y.

Figure 3.29d shows the corresponding NAND-NAND circuit for $\overline{Y}$. This circuit does not produce the desired output; it produces the complement of the desired output.

Finding the NOR-NOR Circuit

What we want to do next is to get the product-of-sums solution, the NOR-NOR circuit that produces the original truth table of Table 3.10. De Morgan's first theorem tells us NAND gates can be replaced by bubbled OR gates; therefore, we can replace Fig. 3.29d by Fig. 3.30a. A bus with each variable

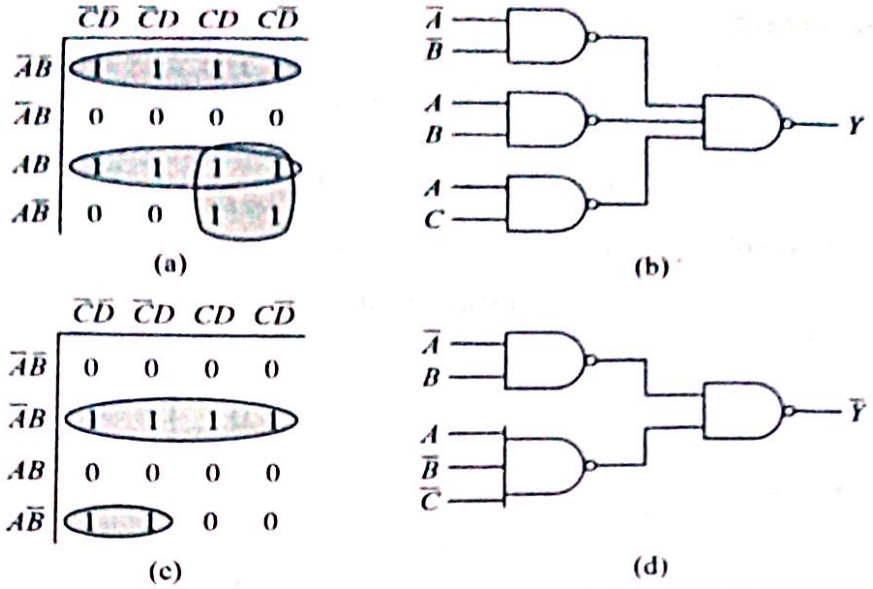


Fig. 3.29 *Deriving the sum-of-products circuit.*

and its complement is usually available in a digital system. So, instead of connecting $\overline{A}$ and B to a bubbled OR gate, as shown in Fig. 3.30a, we can connect A and $\overline{B}$ to an OR gate, as shown in Fig. 3.30b. In a similar way, instead of connecting A, $\overline{B}$, and $\overline{C}$ to a bubbled OR gate, we have connected $\overline{A}$, B, and C to an OR gate. In short, Fig. 3.30b is equivalent to Fig. 3.30a.

The next step toward a NOR-NOR circuit is to convert Fig. 3.30b into Fig. 3.30c, which is done by sliding the bubbles to the left from the output gate to the input gates. This changes the input OR gates to NOR gates. The final step is to use a NOR gate on the output to produce Y instead of $\overline{Y}$, as shown in the NOR-NOR circuit of Fig. 3.30d.

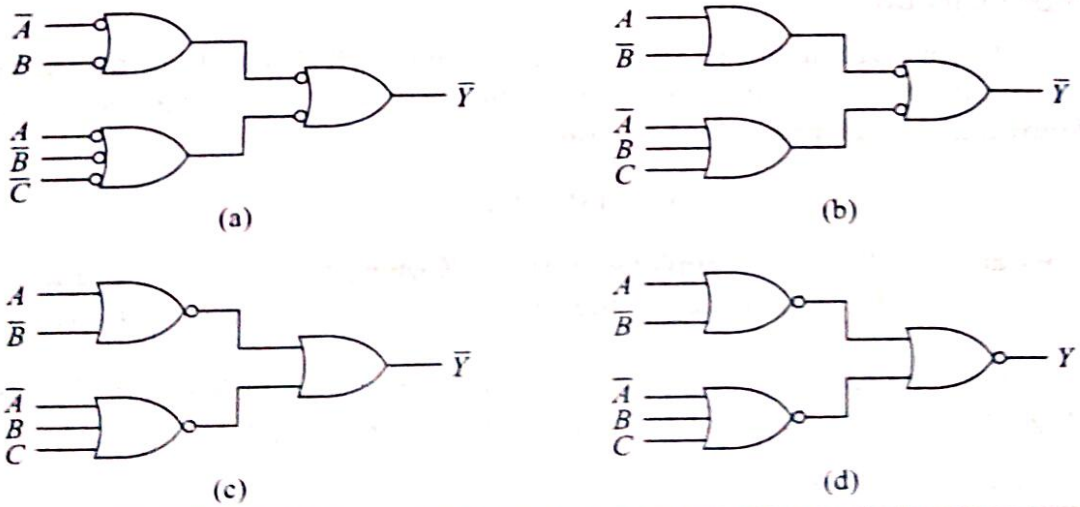


Fig. 3.30 *Deriving the product-of-sums circuit.*

From now on, you don't have to go through every step in changing a complementary NAND-NAND circuit to an equivalent NOR-NOR circuit. Instead, you can apply the duality theorem as described in the following.

Duality

An earlier section introduced the duality theorem of Boolean algebra. Now we are ready to apply this theorem to logic circuits. Given a logic circuit, we can find its dual circuit as follows: Change each AND gate to an OR gate, change each OR gate to an AND gate, and complement all input-output signals. An equivalent statement of duality is this: Change each NAND gate to a NOR gate, change each NOR gate to a NAND gate, and complement all input-output signals.

Compare the NOR-NOR circuit of Fig. 3.30d with the NAND-NAND circuit of Fig. 3.29d. NOR gates have replaced NAND gates. Furthermore, all input and output signals have been complemented. This is an application of the duality theorem. From now on, you can change a complementary NAND-NAND circuit (Fig. 3.29d) into its dual NOR-NOR circuit (Fig. 3.30d) by changing all NAND gates to NOR gates and complementing all signals.

Points to Remember

Here is a summary of the key ideas in the preceding discussion:

1. Convert the truth table into a Karnaugh map. After grouping the 1s, write the sum-of-products equation and draw the NAND-NAND circuit. This is the sum-of-products solution for Y.
2. Complement the Karnaugh map. Group the 1s, write the sum-of-products equation, and draw the NAND-NAND circuit for $\overline{Y}$. This is the complementary NAND-NAND circuit.
3. Convert the complementary NAND-NAND circuit to a dual NOR-NOR circuit by changing all NAND gates to NOR gates and complementing all signals. What remains is the product-of-sums solution for Y.
4. Compare the NAND-NAND circuit (Step 1) with the NOR-NOR circuit (Step 3). You can use whichever circuit you prefer, usually the one with fewer gates.

EXAMPLE 3.10

Show the sum-of-products and product-of-sums circuits for the Karnaugh map of Fig. 3.31a.

Solution

The Boolean equation for Fig. 3.31a on the next page is

$$Y = A + BC\overline{D}$$

Figure 3.31b is the sum-of-products circuit.

After complementing and simplifying the Karnaugh map, we get Fig. 3.31c. The Boolean equation for this is

$$\overline{Y} = \overline{A}\,\overline{B} + \overline{A}\,\overline{C} + \overline{A}D$$

Figure 3.31d is the sum-of-products circuit for the $\overline{Y}$. As shown earlier, we can convert the dual circuit into a NOR-NOR equivalent circuit to get Fig. 3.31e.

The two design choices are Fig. 3.31b and 3.31e. Figure 3.31b is simpler.