Unit 3

Simplification of Boolean Functions

Content:

- K-map
- Two and three Variable Maps
- Four variable Maps
- Product of Sums
- Sum of product simplification
- NAND and NOR implementation

Minterm

- A **minterm or a standard product** is a special product (ANDing of terms) of literals, in which each input variable appears exactly once.
- A function with n variables has 2^n minterms (since each variable can appear complemented or not)
- A three-variable function, such as f(x, y, z), has $2^3 = 8$ minterms:

  x′y′z′ x′y′z x′yz′ x′yz xy′z′ xy′z xyz′ xyz
- Each minterm is **true(i.e. produce 1)** for exactly one combination of inputs.

Maxterms

- A maxterms is a **sum** (or ORing of terms) of literals, in which each input variable appears exactly once.
- A function with n variables has 2^n maxterms
- The maxterms for a three-variable function f(x, y, z):

  $x' + y' + z'$ $x' + y' + z$ $x' + y + z'$ $x' + y + z$

  $x + y' + z'$ $x + y' + z$ $x + y + z'$ $x + y + z$

- Each maxterm is **false(i.e produce 0)** for exactly one combination of inputs.

Minterms and Maxterms for Three Binary Variables

			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

Note:

Each maxterm is the complement of its corresponding minterm and vice versa (viz. $m_0 = M_0'$, $M_4 = m_4'$ etc.).

Note:

A Boolean function may be expressed algebraically (SOP or POS form) from a given truth table by:

- Forming a minterm for each combination of the variables that produces a 1 in the function, and then taking the OR of all those terms. For example, the function f_1 in the table below is determined by expressing the combinations 001, 100 and 111 as x′y′z, xy′z′, and xyz respectively. Since each one of these minterms results in f_1 =1, we should have

$$f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$$

Similarly, it may be easily verified that

$$f_2 = x'yz + xy'z + xyz' + xyz = m_3 + m_5 + m_6 + m_7$$

TABLE 2-4
Functions of Three Variables

x	y	z	Function f_1	Function f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

- Forming a maxterm for each combination of the variables that produces a 0 in the function, and then taking the AND of all those maxterms.

Now consider the complement of a Boolean function. It may be read from the truth table by forming a minterm for each combination that produces a 0 in the function and then ORing those terms. The complement of f_1 is read as

$$f_1' = x'y'z' + x'yz' + x'yz + xy'z + xyz'$$

If we take the complement of f_1', we obtain the function f_1:

$$f_1 = (x + y + z)(x + y' + z)(x + y' + z')(x' + y + z')(x' + y' + z)$$
$$= M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6$$

Similarly, it is possible to read the expression for f_2 from the table:

$$f_2 = (x + y + z)(x + y + z')(x + y' + z)(x' + y + z)$$
$$= M_0M_1M_2M_4$$

Canonical forms

Boolean functions expressed as a sum of minterms or product of maxterms are said to be in *canonical form*. In an expression in canonical form, every variable appears in every term.

1. Sum of Minterms

To express the Boolean function as a product of minterms,

- it must be first brought into a form of AND terms.
- Each term is then inspected to see if it contains all the variables.

If it misses one or more variables, it is ANDed with an expression such as $x + x'$, where x is one of the missing variables.

Example: Express the Boolean function F = A + B'C in a sum of minterms.

Solution:

The function has three variables *A*, *B*, and C.

- The first term *A* is missing two variables; therefore:

  A = A (B + B ') = AB + AB' [B is missing variable]

  This is still missing one variable C, so

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

- The second term B'C is missing one variable:

  B'C = B'C(A + A') = AB'C + A'B'C

Combing all terms, we have

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

But AB'C appears twice, and according to THEOREM 1 of Boolean algebra $x + x = x$, it is possible remove one of them.

So,

Rearranging the minterms in ascending order, we finally obtain:

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

$$= m_1 + m_4 + m_5 + m_6 + m_7$$

Shorthand notation,

$$F(A, B, C) = \Sigma(1, 4, 5, 6, 7)$$

Note:

The summation symbol Σ stands for the ORing of terms: the numbers following it are the minterms of the function.

An **alternate procedure** for deriving the minterms of a Boolean function is to obtain the truth table of the function directly from the algebraic expression and then read the minterms from the truth table.

Truth Table for $F = A + B'C$

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

From the truth table, we can then read the five minterms of the function to be 1, 4, 5, 6, and 7.

So,

$$F(A, B, C) = \Sigma(1, 4, 5, 6, 7)$$

2. Product of Maxterms

To express the Boolean function as a product of maxterms,

- it must be first brought into a form of OR terms. This may be done by using the distributive law, $x + yz = (x + y)(x + z)$.
- Then any missing variable x in each OR term is ORed with xx'.

Example: Express the Boolean function $F = xy + x'z$ in a product of maxterm form.

Solution: First, convert the function into OR terms using the distributive law:

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

The function has three variables: *x*, *y*, and z. Each OR term is missing one variable; therefore:

$$x' + y = x' + y + zz' = (x' + y + z)(x' + y + z')$$
$$x + z = x + z + yy' = (x + y + z)(x + y' + z)$$
$$y + z = y + z + xx' = (x + y + z)(x' + y + z)$$

Combining all maxterms and removing repeated terms:

$$F = (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z')$$
$$= M_0 M_2 M_4 M_5$$

Shorthand notation:

$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

Note:

The product symbol Π denotes the ANDing of maxterms; the numbers are the maxterms of the function.

- **Conversion between canonical forms**

General Procedure:

To convert from one canonical form to another

- interchange the symbols
- list those numbers missing from the original form.

Note: In order to find the missing terms, one must realize that the total number of combination is 2^n (numbered as 0 to 2^n-1), where *n* is the number of binary variables in the function.

Consider a function, $F = xy + x'z$. First, we derive the truth table of the function

- The minterms of the function are 1, 3, 6, and 7.

The function expressed in sum of minterms is

$$F(x, y, z) = \Sigma(1, 3, 6, 7)$$

- Here, the missing terms are 0, 2, 4, and 5.

The function expressed in product of maxterm is

$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

TABLE 2-6
Truth Table for $F = xy + x'z$

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Standard Forms

- Another way to express Boolean functions in which the terms that form the function may contain one, two, or any number of literals.

There are two types of standard forms: the sum of products and product of sums.

- The *sum of products(SOP)* is a Boolean expression containing AND terms, called *product terms*, of one or more literals each. The *sum* denotes the ORing of these terms.

  Example: F1 = y' + xy + x’yz

- A *product of sums(POS)* is a Boolean expression containing OR terms, called *sum terms*. Each term may have any number of literals. The *product* denotes the ANDing of these terms.

  Example: F1 = x(y' + z)(x' + y + z' + w)

- Function can also be in non-standard form: *F3 = (AB + CD) (A'B' + CD')* is neither in SOP nor in POS forms. It can be changed to a standard form by simplifying as *F3 = A'B'CD + ABC'D'*.

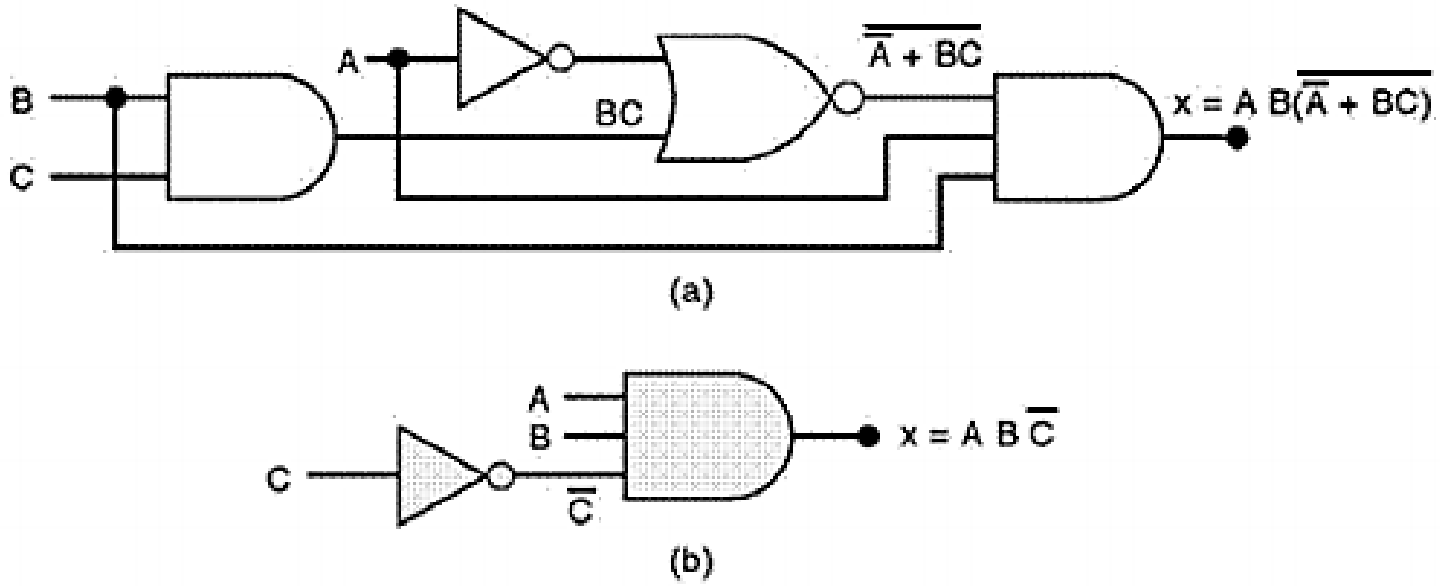
- **Simplifying Logic Circuits (Boolean functions):**

Two methods

First obtain one expression for the circuit, then try to simplify.

Example: In diagram below, (a) can be simplified to (b) using one of following two methods:

1. Algebraic method (use Boolean algebra theorems)
2. Karnaugh mapping method (systematic, step-by-step approach)



METHOD 1: Minimization by Boolean algebra

- Make use of relationships and theorems to simplify Boolean Expressions
- Perform algebraic manipulation resulting in a complexity reduction.
- This method relies on your algebraic skill
- 3 things to try:

➢ **Grouping**

A + AB + BC

A (1+ B) + BC

A + BC [since 1+ B =1]

➢ **Multiplication by redundant variables**

Multiplying by terms of the form A + A' does not alter the logic Such multiplications by a variable missing from a term may enable minimization.

Example:

$$AB + A\overline{C} + BC = AB(C + \overline{C}) + A\overline{C} + BC$$
$$= ABC + AB\overline{C} + A\overline{C} + BC$$
$$= BC(1 + A) + A\overline{C}(1 + B)$$
$$= BC + A\overline{C}$$

- **Application of DeMorgan's theorem**

Expressions containing several inversions stacked one upon the other often are simplified by using DeMorgan's law which **unwraps** multiple inversions.

Example:

$$\overline{\overline{A\overline{B}C} + \overline{ACD} + B\overline{C}} = \overline{(\overline{A} + B + \overline{C}) + (\overline{A} + \overline{C} + \overline{D}) + B\overline{C}}$$
$$= \overline{(\overline{A} + B + \overline{C} + \overline{D}) + B\overline{C}}$$
$$= \overline{(\overline{A} + B + \overline{C} + \overline{D})}$$
$$= A\overline{B}CD$$

❖ **(Logic Design):** Design a logic circuit having 3 inputs, A, B, C will have its output HIGH only when a majority of the inputs are HIGH.

Solution:

Step 1 Set up the truth table:

Step 2 Write minterm (AND term) for each case where the output is 1.

Step 3 Write the SOP from the output.

$$x = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$

Step 4 Simplify the output expression.

$$x = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$

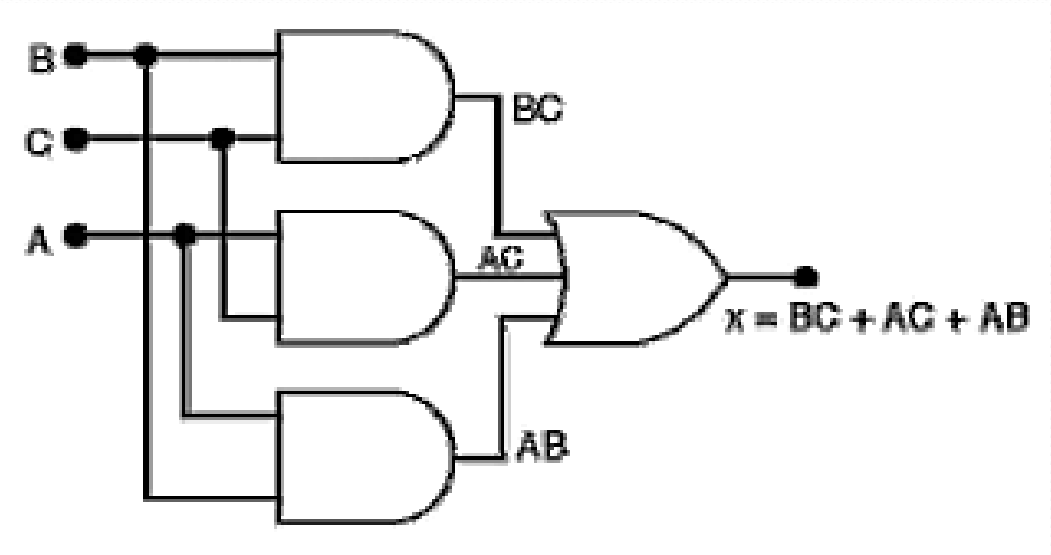
$$x = \bar{A}BC + ABC + A\bar{B}C + \boxed{ABC} + AB\bar{C} + \boxed{ABC}$$

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

$$= BC + AC + AB$$

A	B	C	x	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\rightarrow \bar{A}BC$
1	0	0	0	
1	0	1	1	$\rightarrow A\bar{B}C$
1	1	0	1	$\rightarrow AB\bar{C}$
1	1	1	1	$\rightarrow ABC$

Step 5 Implement the circuit.



METHOD 2: Minimization by K-map (Karnaugh map)

Algebraic minimization of Boolean functions is rather awkward because it lacks specific rules to predict each succeeding step in the manipulative process. The map method provides a simple straightforward procedure for minimizing Boolean functions. This method may be regarded as a pictorial form of a truth table. The map method, first proposed by Veitch and modified by Karnaugh, is also known as the "Veitch diagram" or the "Karnaugh map.“

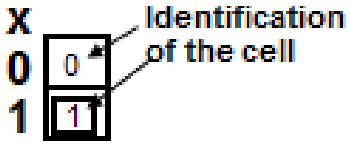
- The k-map is a diagram made up of grid of squares.
- Each square represents one minterm.
- The minterms are ordered according to Gray code (only one variable changes between adjacent squares).

In fact, the map presents a visual diagram of all possible ways a function may be expressed in a standard form. By recognizing various patterns, the user can derive alternative algebraic expressions for the same function, from which he can select the simplest one.

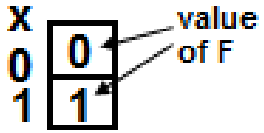
Note: Rectangle divided into 2^n cells where n is number of variables

Each Minterm is identified by a decimal number whose binary representation is identical to the binary interpretation of the input values of the minterm.

x	F
0	1
1	0



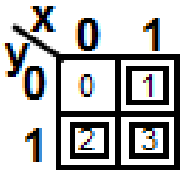
Karnaugh Map

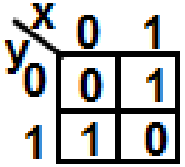


$F(x) = \Sigma(1)$

1-cell

x	y	F
0	0	0
0	1	1
1	0	1
1	1	1





$F(x,y) = \Sigma(1,2)$

- **Two variable maps**

There are (2^2 = 4) four minterms for a Boolean function with two variables. Hence, the two-variable map consists of (2^2 = 4) four squares, one for each minterm, as shown in Figure:

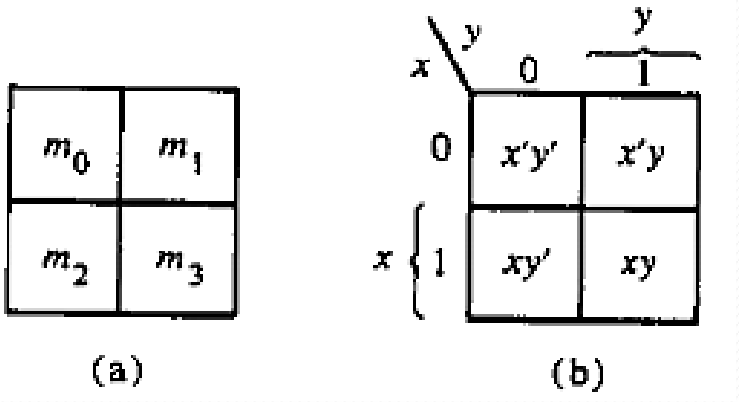


Fig: Two-variable map

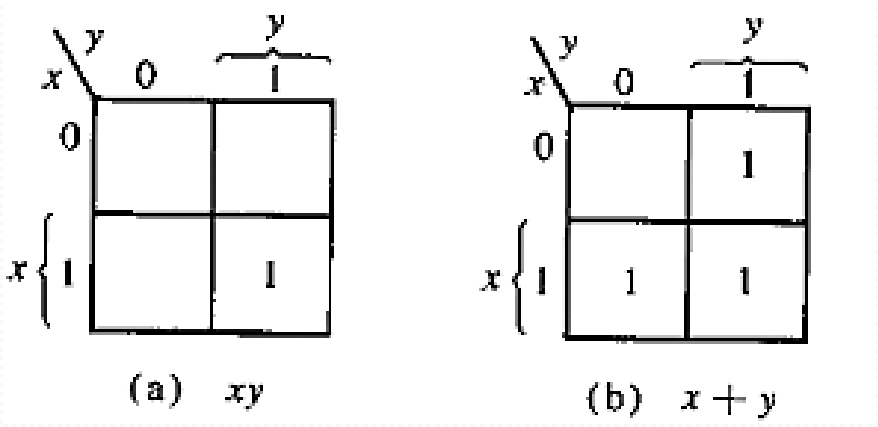


Fig: Representation of functions in the map

- **Three variable maps**

There are (2^3 = 8) eight minterms for three binary variables. Therefore, a three-variable map consists of (2^3 = 8) eight squares, as shown in Figure.

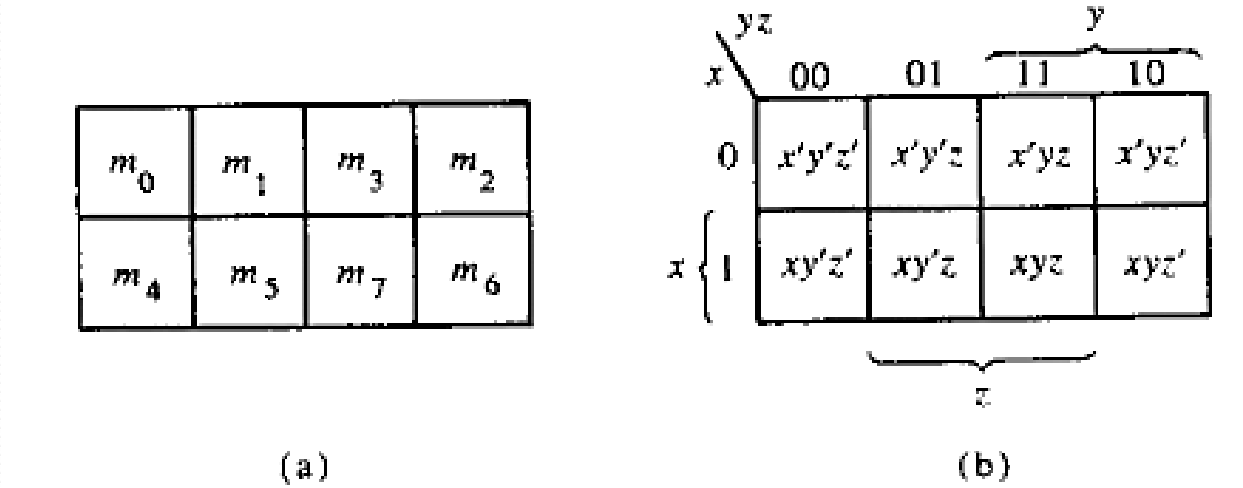


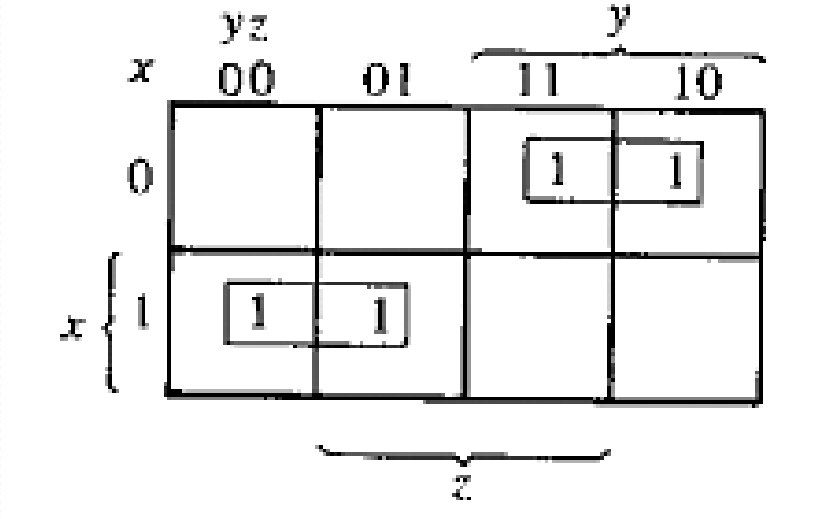
Fig: Three-variable map

Question: Simplify the Boolean function

$$F(X, Y, Z) = \Sigma(2, 3, 4, 5).$$

Solution:

Step 1: First, a 1 is marked in each minterm that represents the function. This is shown in Figure, where the squares for minterms 010, 011, 100, and 101 are marked with 1's. For convenience, all of the remaining squares for which the function has value 0 are left blank rather than entering the 0's.



Step 2: Explore collections of squares on the map representing product terms to be considered for the simplified expression. We call such objects *rectangles*. Rectangles that correspond to product terms are restricted to contain numbers of squares that are powers of 2, such as 1, 2(pair), 4(quad), 8(octet) ... Goal is to find the fewest such rectangles that include all of the minterms marked with 1's. This will give the fewest product terms.

Step 3: Sum up each rectangles (it may be pair, quad etc. representing term) eliminating the variable that changes in value (or keeping intact the variables which have same value) throughout the rectangle.

From figure, logical sum of the corresponding two product terms gives the optimized expression for *F:*

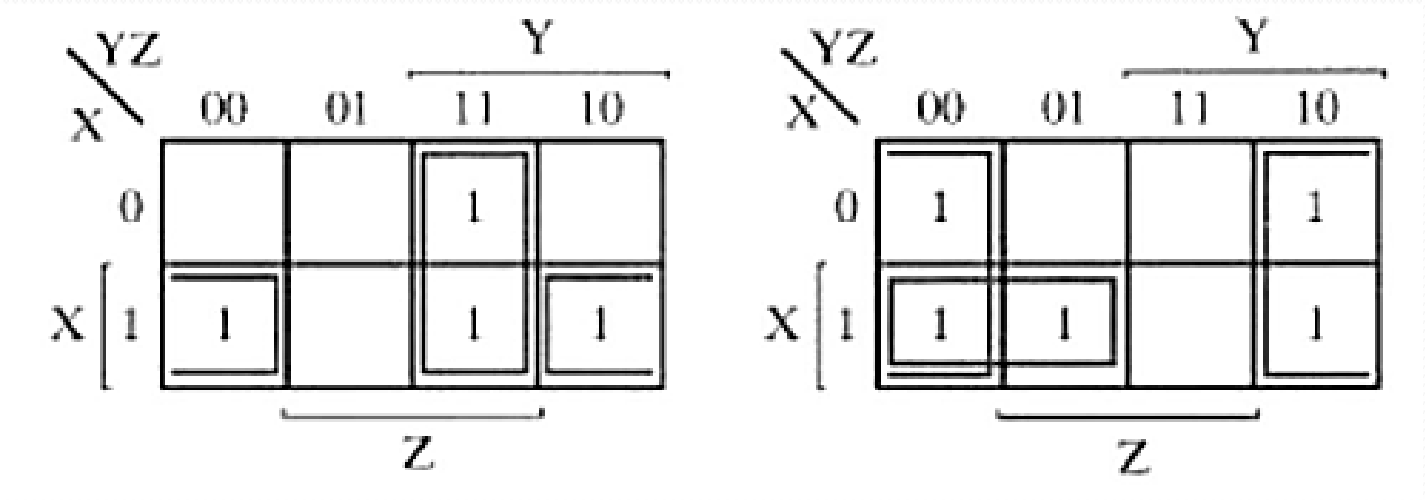
$$F = X'Y+XY'$$

Q. Simplify the following two Boolean functions:

$$F(X, Y, Z) = \Sigma(3, 4, 6, 7)$$

$$G(X, Y, Z) = \Sigma(0, 2, 4, 5, 6)$$

Solution: The map for F and G are given below:



Writing the simplified expression for both functions:

F = YZ +XZ' and G = Z' +XY'

Note:

- One square represents one minterm, giving a term of three literals.
- Two adjacent squares represent a term of two literals.
- Four adjacent squares represent a term of one literal.
- Eight adjacent squares encompass the entire map and produce a function that is always equal to 1.

Assignment:

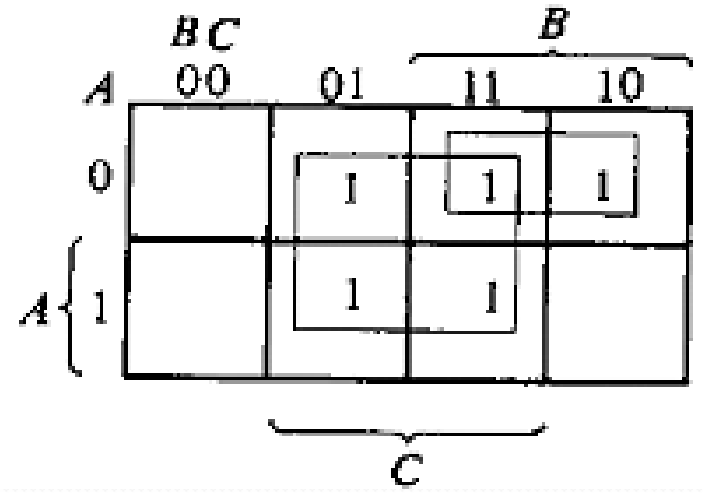
Q. Given the following Boolean function

F = A'C + A'B + AB'C + BC

a. Express it in sum of minterms.

b. Find the minimal sum of products expression.

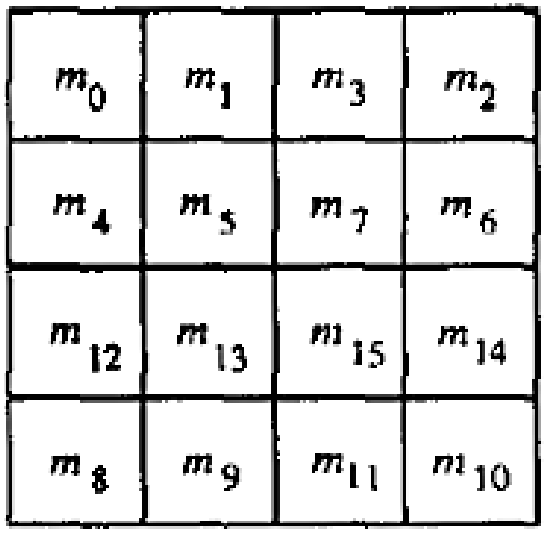
Note: Perform by using K-map



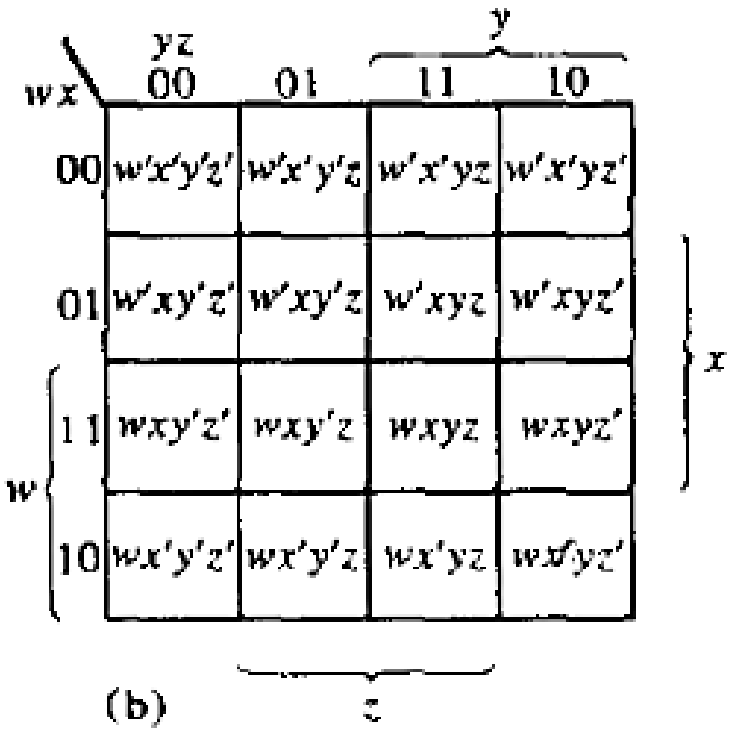
Answer: F(A, B, C) = Σ(1, 2, 3, 5, 7) = C + A'B

- **Four variable maps**

The map for Boolean functions of four binary variables is shown in Fig below. In (a) are listed the 16 minterms and the squares assigned to each. In (b) the map is redrawn to show the relationship with the four variables.



(a)



(b)

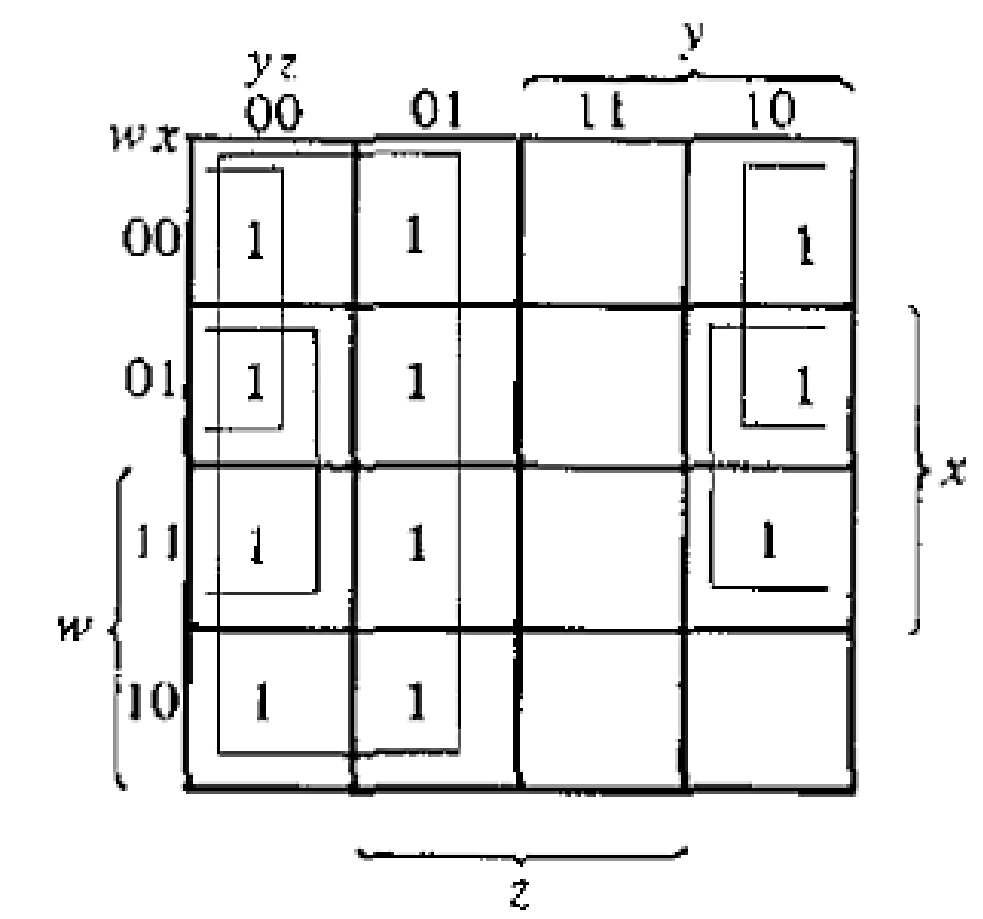
The map minimization of four-variable Boolean functions is similar to the method used to minimize three-variable functions. Adjacent squares are defined to be squares next to each other. In addition, the map is considered to lie on a surface with the top and bottom edges, as well as the right and left edges, touching each other to form adjacent squares. For example, m_0 and m_2 form adjacent squares, as do m_3 and m_{11}.

Question: Simplify the Boolean function

$$F(w, x, y, z) = \Sigma(0,1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$$

Solution:

Since the function has four variables, a four-variable map must be used. Map representation is shown below:



The simplified function is: F = y' + w'z' + xz'

Question: Simplify the Boolean function

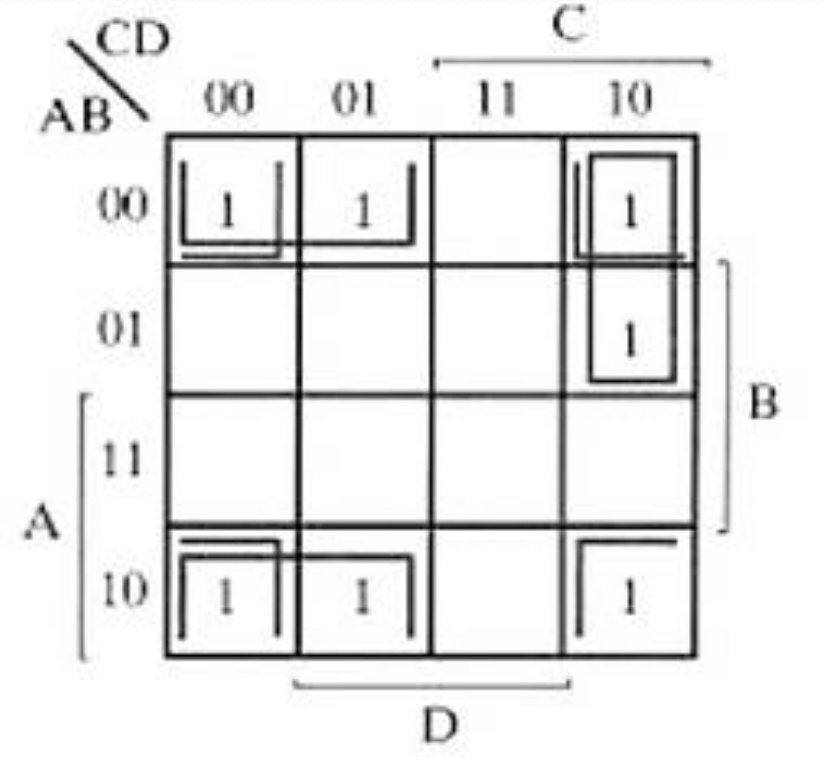
$$F = A'B'C' + B'CD' + A'BCD' + AB'C'$$

Solution:

First try just to reduce the standard form function into SOP form(sum of minterms) and then mark 1 for each minterm in the map.

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

This function also has 4 variables, so the area in the map covered by this function consists of the a squares marked with 1's in following Fig.



Optimized function thus is: F = B'D' + B'C' + A'CD'

❑ Don’t care Conditions

In practice, there are some applications where the function is not specified for certain combinations of the variables.

There are two cases in which this occur.

- In first case, the input combination never occur. As an example, four-bit binary code for the decimal digits has six combinations that are not used and not expected to occur.
- In the second case, the input combinations are expected to occur, but we do not care what the output are in response to these combinations.

In both cases, the output are said to be unspecified for the input combinations. Functions that have unspecified outputs for some input combinations are called *incompletely specified functions*. In most applications, we simply **don't care what value is assumed by the function** for the unspecified minterms.

For this reason, it is customary to call the unspecified minterms of a function ***don't-care conditions***. These don't-care conditions can be used on a map to provide further simplification of the Boolean expression.

Don't-care minterm is a combination of variables whose logical value is not specified. To distinguish the don't-care condition from 1's and 0's, an X(cross) is used. Thus, an X inside a square in the map indicates that we don't care whether the value of 0 or 1 is assigned to *F* for the particular minterm.

When choosing adjacent squares to simplify the function in a map, the don't-care minterms may be assumed to be either 0 or 1. When simplifying the function, we can choose to include each don't-care minterm with either the 1's or the 0's, depending on which combination gives the simplest expression.

Example: Simplify the Boolean function with Don't Care Conditions

$$F(w, x, y, z) = \Sigma(1, 3, 7, 11, 15)$$

that has the don't-care conditions $d(w, x, y, z) = \Sigma(0, 2, 5)$

Solution:

The map simplification is shown below. The minterms of *F* are marked by 1's, those of *d* are marked by X's, and the remaining squares are filled with 0's.

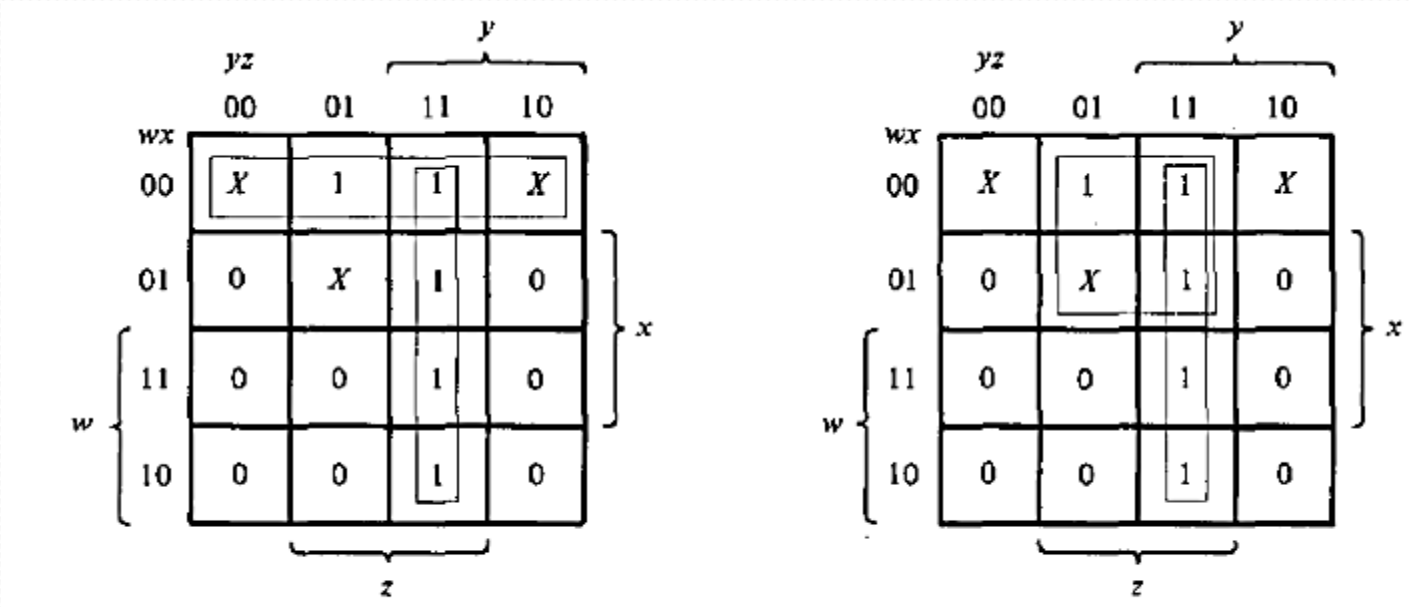


Fig: Map simplification with don't care conditions

In part (a) of the diagram, don't-care minterms 0 and 2 are included with the 1's, which results in the simplified function

$$F = yz + w'x'$$

In part (b), don't-care minterm 5 is included with the 1's and the simplified function now is

$$F = yz + w'z$$

Either one of the above expressions satisfies the conditions stated for this example.

- **Product of sum simplification**

The optimized Boolean functions derived from the maps in all of the previous examples were expressed in sum-of-products (SOP) form. With only minor modification, the product-of-sums form can be obtained.

Procedure:

The 1's placed in the squares of the map represent the minterms of the function. The minterms not included in the function belong to the complement of the function. From this, we see that the complement of a function is represented in the map by the squares not marked by 1's. If we mark the empty squares with 0's and combine them into valid rectangles, we obtain an optimized expression of the complement of the function (F'). We then take the complement of F to obtain the function *F* as a product of sums.

Question: Simplify the following Boolean function

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

in (a) Sum of products (SOP) and (b) Product of sums (POS).

Solution:

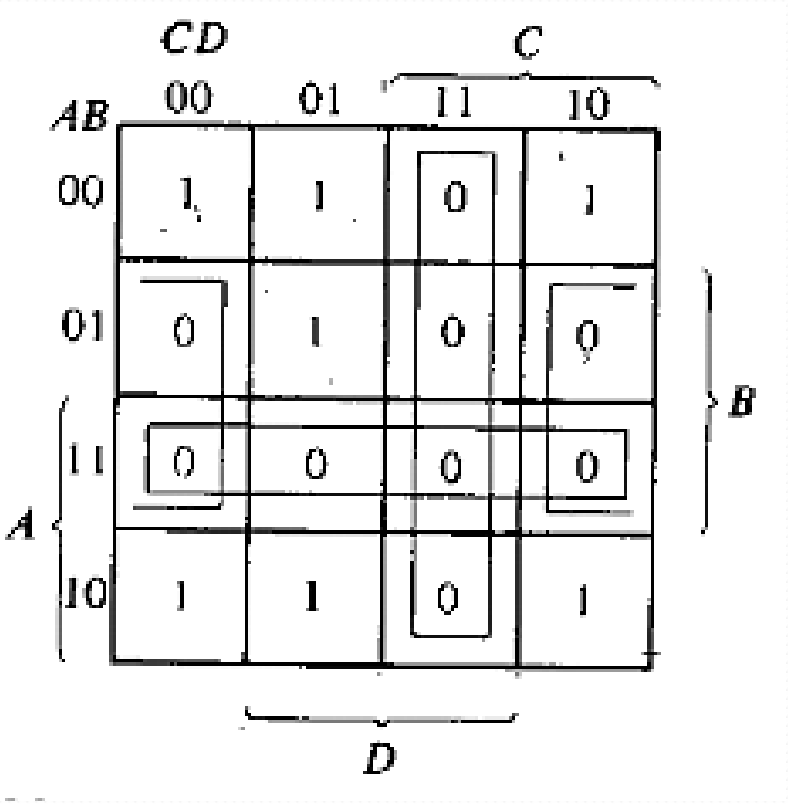
The 1's marked in the map below represent all the minterms of the function. The squares marked with 0's represent the minterms not included in F and, therefore, denote F '.

a) Combining the squares with 1's gives the simplified function in sum of products:

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

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

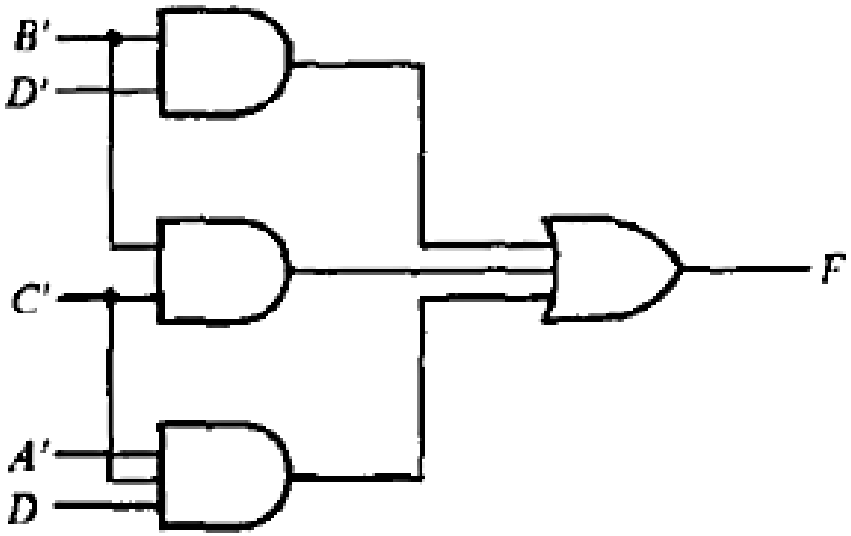
$$F' = AB + CD + BD'$$



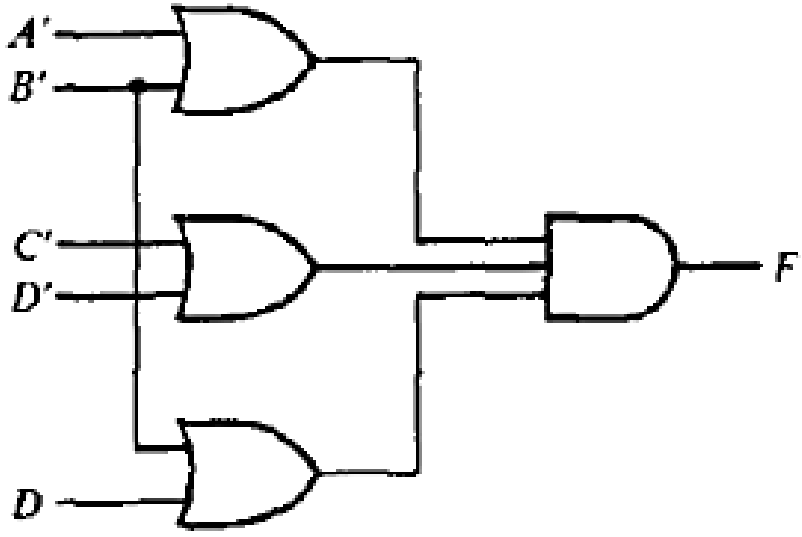
Applying DeMorgan's theorem, we obtain the simplified function in product of sums:

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

The Gate implementation of the simplified expressions obtained above in (a) and (b):



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



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

- **NAND and NOR implementation**

Digital circuits are more frequently constructed with NAND or NOR gates than with AND and OR gates. NAND and NOR gates are easier to fabricate with electronic components and are the basic gates used in all IC digital logic families. The procedure for **two-level implementation** is presented in this section.

- **NAND and NOR conversions (from AND, OR and NOT implemented Boolean functions)**

To facilitate the conversion to NAND and NOR logic, there are two other graphic symbols for these gates.

(a) NAND gate

Two equivalent symbols for the NAND gate are shown in diagram below:

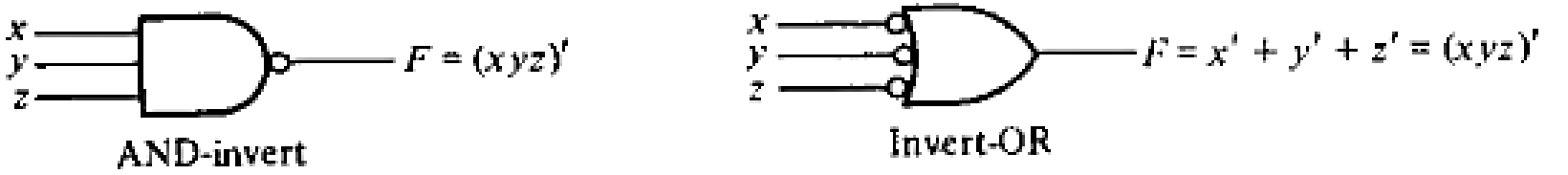


Fig: Two graphic symbols for NAND gate

(b) NOR gate

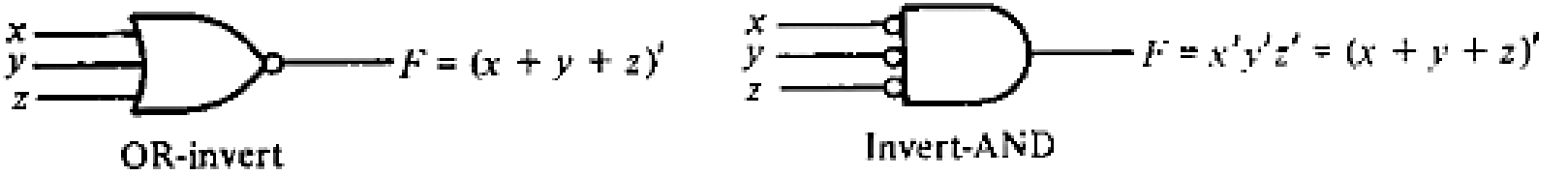


Fig: Two graphic symbols for NOR gate

(c) Inverter

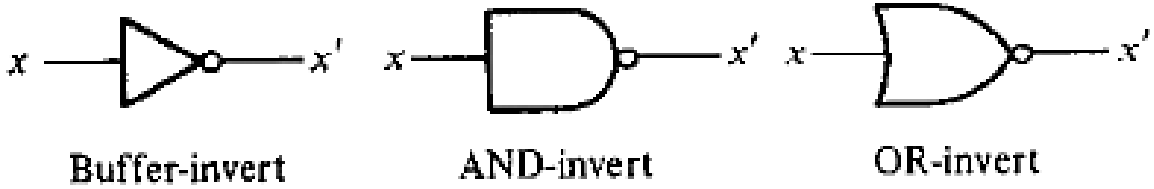


Fig: Two graphic symbols for NAND gate

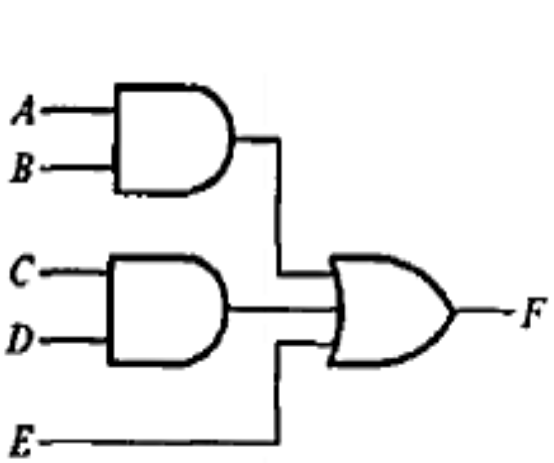
NAND implementation

The rule for obtaining the NAND logic diagram from a Boolean function is as follows:

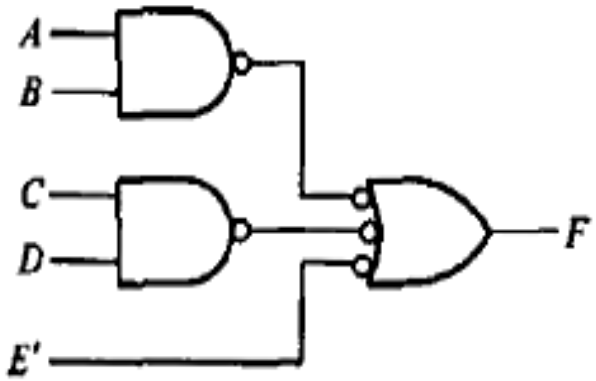
First method:

a) Simplify the function and express it in **sum of products**.
b) Draw a NAND gate for each product term of the function that has at least two literals. The inputs to each NAND gate are the literals of the term. This constitutes a group of **first-level gates.**
c) Draw a single NAND gate (using the AND-invert or invert-OR graphic symbol) in the second level, with inputs coming from outputs of first-level gates.
d) A term with a single literal requires an inverter in the first level or may be complemented and applied as an input to the **second-level NAND gate**.

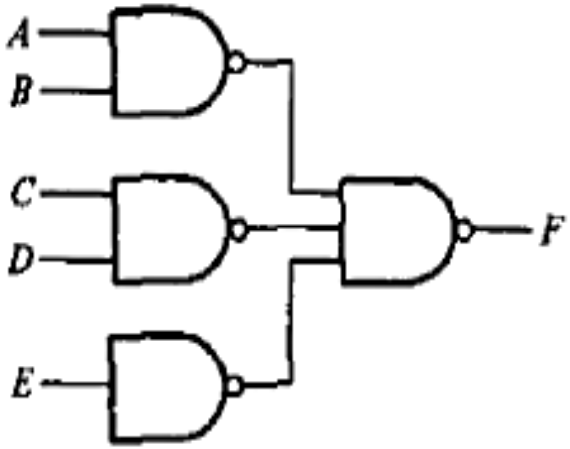
Example: $F=AB + CD + E =(AB + CD + E)'' =((AB)'(CD)'E')'$



(a) AND-OR



(b) NAND-NAND



(c) NAND-NAND

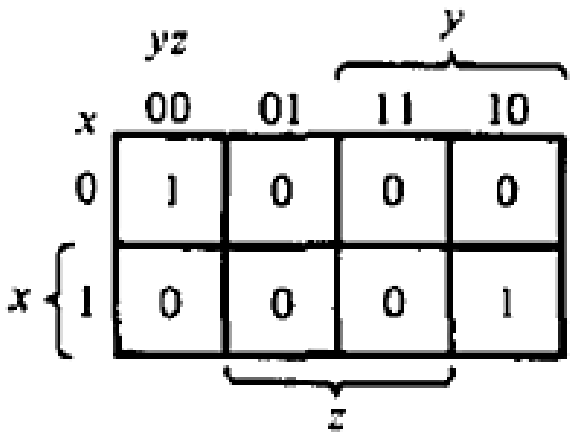
Second method:

If we combine the 0's in a map, we obtain the simplified expression of the *complement* of the function in sum of products. The complement of the function can then be implemented with two levels of NAND gates using the rules stated above. If the normal output is desired, it would be necessary to **insert a one input NAND or inverter gate**.

Example: Implement the following function with NAND gates:

$$F(x, y, z) = \Sigma\,(0, 6)$$

Solution: The first step is to simplify the function in sum of products form. This is attempted with the map. There are only two 1's in the map, and they can't be combined.



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

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

Fig: Map simplification in SOP

METHOD 1: Two-level NAND implementation is shown below:

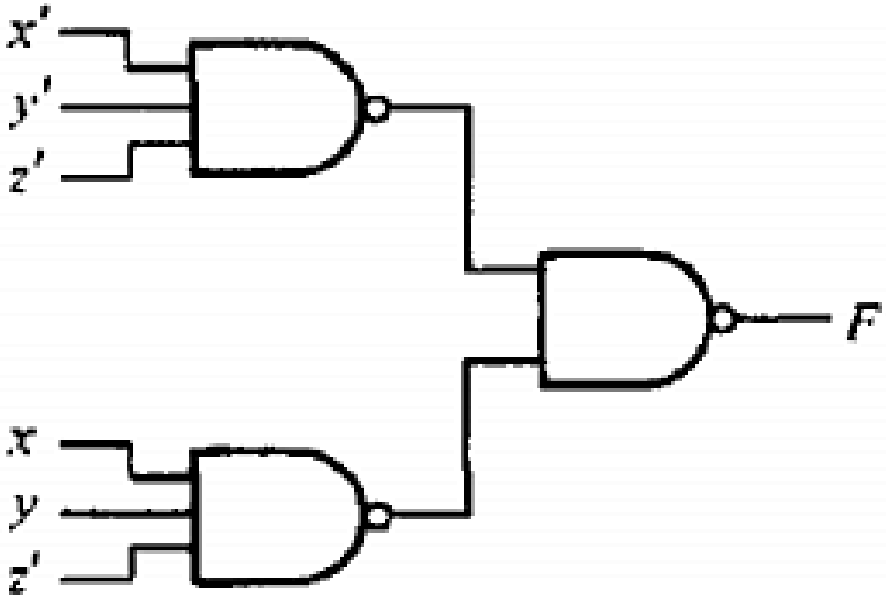


Fig: $F = x'y'z' + xyz'$

METHOD 2:

Next we try to simplify the complement of the function in sum of products. This is done by combining the 0's in the map:

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

The two-level NAND gate for generating F' is shown below:

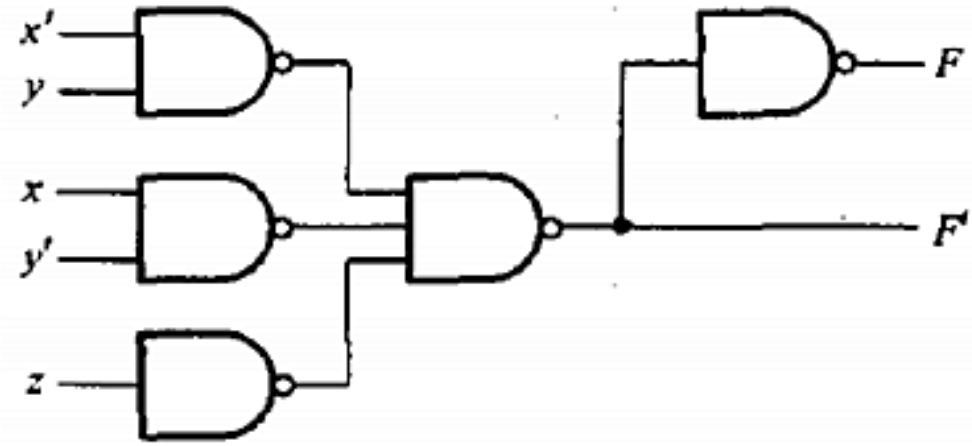


Fig: $F' = x'y + xy' + z$

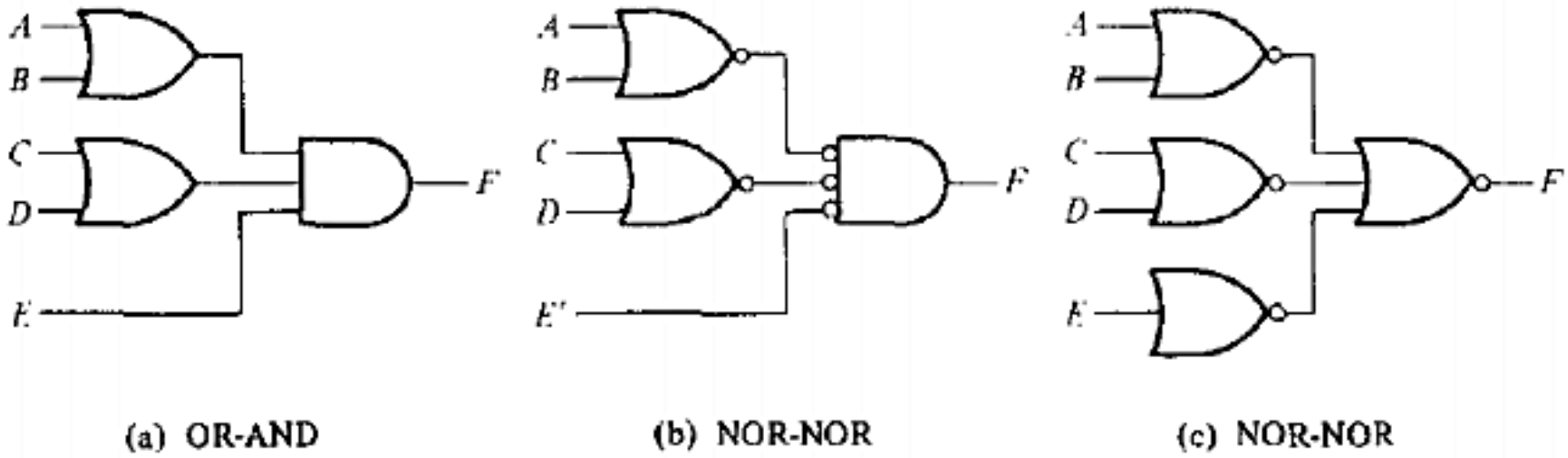
If output F is required, it is necessary to add a one· input NAND gate to invert the function. This gives a three-level implementation.

- **NOR Implementation**

The NOR function is the dual of the NAND function. For this reason, all procedures and rules for NOR logic are the duals of the corresponding procedures and rules developed for NAND logic. The implementation of a Boolean function with NOR gates requires that the function be simplified in product of sums form. A product of sums expression specifies a group of OR gates for the sum terms, followed by an AND gate to produce the product. The transformation from the OR-AND to the NOR-NOR diagram is depicted in Fig below. It is **similar to the NAND transformation discussed previously, except that now we use** the *product of sums* expression

Examples: F = (A+B)(C+D)E = ((A+B)(C+D)E)″

= ((A+B)'+(C+D)'+E′)′



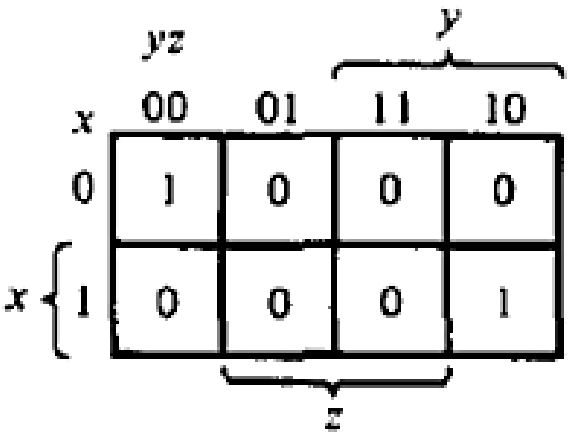
(a) OR-AND (b) NOR-NOR (c) NOR-NOR

Fig: Three ways to implement *F = (A + B) (C + D)E*

Question: Implement the following function with NOR gates:

$$F(x, y, z) = \Sigma(0, 6)$$

Solution:



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

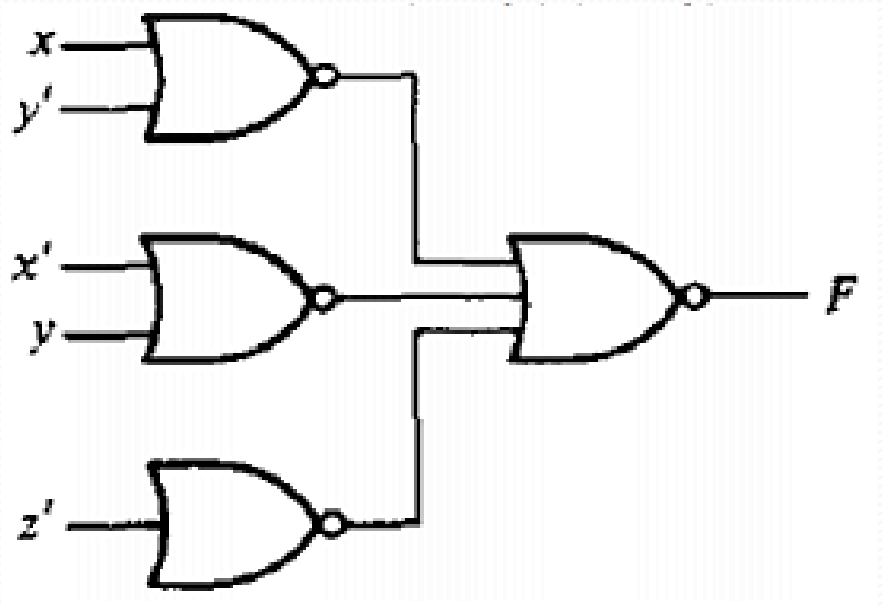
$$F' = x'y + xy' + z$$

Fig: Map simplification in SOP

METHOD 1

First, combine the 0's in the map to obtain $F' = x'y + xy' + z$ this is the complement of the function in sum of products. Complement F' to obtain the simplified function in product of sums as required for NOR implementation:

$$F = (x + y')(x' + y)z'$$



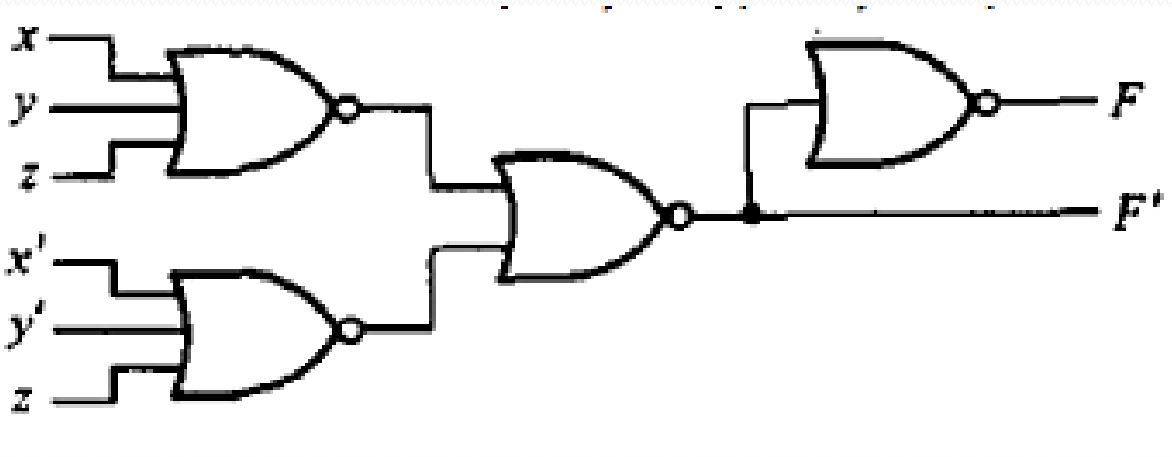
METHOD 2

A second implementation is possible from the complement of the function in product of sums. For this case, first combine the 1's in the map to obtain

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

Complement this function to obtain the complement of the function in product of sums as required for NOR implementation:

$$F' = (x + y + z)(x' + y' + z)$$



Summary of NAND and NOR implementation

Case	Function to simplify	Standard form to use	How to derive	Implement with	Number of levels to F
(a)	F	Sum of products	Combine 1's in map	NAND	2
(b)	F'	Sum of products	Combine 0's in map	NAND	3
(c)	F	Product of sums	Complement F' in (b)	NOR	2
(d)	F'	Product of sums	Complement F in (a)	NOR	3


Thank You!