Minterms, Maxterms, Karnaugh Map (K-Map), and Universal Gates

Objectives: After completing this chapter, you should be able to:

- Represent a Boolean function in the form of sum of minters and product of maxterm.
- Generate a truth table from a function that is represented by the sum of minterms.
- Generate a truth table from a function that is represented by the product of minterms.
- Developed a function from the truth table.
- Use K-map to simplify a function.
- Apply don't care condition in a K-map.
- Draw logic circuit using only NAND or NOR gates.

3.1 Introduction

Digital circuit can be represented by a truth table or Boolean function. In a digital circuit with multiple digital inputs and multiple digital outputs, the outputs depend on the current value of inputs. A Boolean function can be represented in the form of *sum of minterms* or the *product of maxterms*, which enable the designer to make a truth table more easily. Also, Boolean functions can be simplified using Karnaugh **map** (**K-map**) without using Boolean theorems, by transferring a function to K-map and reading simplified function from K-map. Most digital systems are designed by using universal gates (NAND or NOR).

3.2 Minterms

A *minterm* is associated with each combination of variables in a function. If a function has *n* variables, then it has 2^n minterms. Consider two Boolean variables, X and Y. There are four different possible combinations that can be generated from X AND Y, and they are $\bar{X}\bar{Y}$, $\overline{X}\ Y$, $X\bar{Y}$, and XY. These four combinations are called the *minterms* for X AND Y. Table 3.1 shows the minterms and their designations for F(X,Y) = (X AND Y).

In Table 3.1, $\bar{X}\bar{Y} = 1$, if X = 0 and Y = 0 then $\bar{X}\bar{Y}$ is represented by m_0 (decimal number of 00); $\overline{X}\ Y = 1$ if X = 0 and Y = 1 then $\overline{X}\ Y$ is represented by m_1; $X\bar{Y} = 1$ if X = 1 and Y = 0 and $X\bar{Y}$ is represented by m_2; XY = 1 if X = 1 and Y = 1 then XY is represented by m_3.

Application of Minterms. It is simple to generate a truth table from minterms and vice versa. Consider the function F(X,Y) = X $\bar{Y}$ + $\overline{X}\ Y$ and its truth table (Table 3.2); this function can be represented as F(X,Y) = m_1 + m_2 or each minterm that represents a *one* in the truth table. This may also be rewritten as F (X, Y) = $\sum(1,2)$.

Three-Variable Minterms The three variables X, Y, and Z generate eight minterms as shown in Table 3.3.

Example 3.1 Find the truth table for the following function:

$$F(X, Y, Z) = X'Y'Z + X'YZ + XYZ$$

The function F can be represented by a *sum of the minterms* (or where F = 1):

$$F(X, Y, Z) = m_1 + m_3 + m_7$$

or

$$F(X, Y, Z) = \sum (1, 3, 7)$$

Table 3.1 Minterms of F(X,Y)

X Y	Minterm	Designation
0 0	$\bar{X}\bar{Y}$	m_0
0 1	$\bar{X}$ Y	m_1
1 0	X $\bar{Y}$	m_2
1 1	X Y	m_3

Table 3.2 Truth table for function F(X,Y) = X $\bar{Y}$ + $\overline{X}\ Y$ with minterms

X	Y	F	
0	0	0	~~m0~~
0	1	1	m_1
1	0	1	m_2
1	1	0	~~m3~~

Table 3.3 Three-variable minterms

X Y Z	Minterms	Designation
0 0 0	$X'Y'Z'$	m_0
0 0 1	$X'Y'Z$	m_1
0 1 0	$X'YZ'$	m_2
0 1 1	$X'YZ$	m_3
1 0 0	$XY'Z'$	m_4
1 0 1	$XY'Z$	m_5
1 1 0	XYZ'	m_6
1 1 1	XYZ	m_7

Table 3.4 Truth table for function F(X,Y, Z) = X′Y′Z + X′YZ + XYZ

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	0
1 1 1	1

The truth table for this function contains a *one* in row 1, row 3, and row 7. The rest of the rows are *zeros* as shown in Table 3.4. The function for a truth table can also be determined from the sum of the minterms.

Example 3.2 The following truth table is given; find the function F.

In the following table, the output of function F is one when the input is $001 = m_1$, $011 = m_3$, $101 = m_5$, and $111 = m_7$; therefore $F(X,Y,Z) = m_1 + m_3 + m_5 + m_7$ or

$$F(X, Y, Z) = \sum (1, 3, 5, 7)$$

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	1
1 1 0	0
1 1 1	1

Substituting each designation of a minterm with its actual product term (e.g., $m_0 = X'Y'Z'$) results in the following function:

$$F(X, Y, Z) = X'Y'Z + X'YZ + XY'Z + XYZ$$

Example 3.3 For the following truth table:

(a) Find the function F.
(b) Simplify the function.
(c) Draw the logic circuit for the simplified function.

X Y Z	F
0 0 0	1
0 0 1	0
0 1 0	1
0 1 1	1
1 0 0	0
1 0 1	0
1 1 0	0
1 1 1	1

From the truth table, where F = 1, we select the minterms m_0, m_2, m_3, and m_7:

$$F(X, Y, Z) = m_0 + m_2 + m_3 + m_7$$

Or

$$F(X, Y, Z) = \underset{\bar{X}\bar{Z}}{\bar{X}\bar{Y}\bar{Z}} + \underset{\bar{Z}}{\bar{X}Y\bar{Z}} + \bar{X}YZ + XYZ$$

$$F(X, Y, Z) = \bar{X}\bar{Z}\left(\bar{Y} + Y\right) + YZ\left(\bar{X} + X\right)$$

$$F(X, Y, Z) = \bar{X}\bar{Z} + YZ$$

The logic circuit for the simplified function F is given in Fig. 3.1.

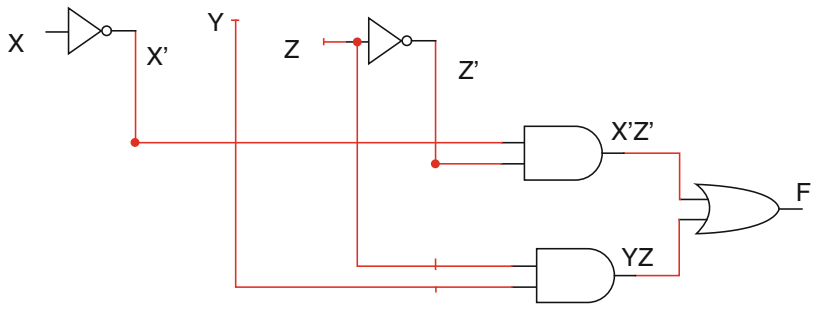


Fig. 3.1 Logic circuit for F(X,Y,Z) = $\overline{X}\ \bar{Z}$ + YZ

3.3 Maxterms

Maxterm is complement of a minterm. If the *minterm* $\mathbf{m_0}$ is $\left(\bar{X}\bar{Y}\bar{Z}\right)$, then the *maxterm* $\mathbf{M_0}$ is

$$\left(\overline{\bar{X}\bar{Y}\bar{Z}}\right) = \mathrm{X} + \mathrm{Y} + \mathrm{Z}$$

Table 3.5 shows the maxterms for three variables.

In a truth table, an output of *one* represents *minterms*, and an output of *zero* represent *maxterms*. Consider the truth Table 3.6, where the function F can be expressed as the *product of maxterms*. (Please note: the *product* of maxterms, as opposed to the *sum* of minterms.)

$F(X,Y,Z) = M_0M_2M_4M_5M_6$, or it can represented by

$$\mathrm{F}(\mathrm{X}, \mathrm{Y}, \mathrm{Z}) = \pi(0, 2, 4, 5, 6)$$

Substituting each designated maxterm with the corresponding maxterm results in:

$$\mathrm{F}(\mathrm{X}, \mathrm{Y}, \mathrm{Z}) = (\mathrm{X} + \mathrm{Y} + \mathrm{Z})(\mathrm{X} + \mathrm{Y}' + \mathrm{Z})(\mathrm{X}' + \mathrm{Y} + \mathrm{Z})(\mathrm{X}' + \mathrm{Y} + \mathrm{Z}')$$

Table 3.5 Maxterms of function F(X,Y,Z)

X Y Z	Maxterm	Designation
0 0 0	X + Y + Z	M_0
0 0 1	X + Y + Z′	M_1
0 1 0	X + Y′ + Z	M_2
0 1 1	X + Y′ + Z′	M_3
1 0 0	X′ + Y + Z	M_4
1 0 1	X′ + Y + Z′	M_5
1 1 0	X′ + Y′ + Z	M_6
1 1 1	X′ + Y′ + Z′	M_7

Table 3.6 Truth Table for $F(X,Y,Z) = M_0M_2M_4M_5M_6$

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	0
1 1 1	1

3.4 Karnaugh Map (K-Map)

Karnaugh maps are used to simplify a Boolean function without using Boolean algebra theorems. A K-map is also another way to represent the truth table of a function. K-maps are made of cells where each cell represents a minterm. Cells marked with a *one* will be the minterms used for the *sum of the minterms* representation of a function. Conversely, cells marked with a *zero* will be used for the *product of the maxterms* representation.

Two variables X and Y can have four minterms as shown in Table 3.7. Each minterm is represented by a cell in the K-map, so a two-variable K-map contains four cells as shown in Fig. 3.2.

In Fig. 3.2 each cell represents a minterm. The cell located at row 0 and column 0 represents m_0 (minterm zero) or $X'Y'$. The cell located at row 1 and column 1 is represented by m_3 or XY. As shown in Fig. 3.2, both cells in row zero contain X', so this row is labeled the X' row. Both cells in row 1 contain X so that row is labeled as the X row.

A K-map of a function is another way to represent the truth table of the function, as seen in Fig. 3.3.

Consider the function $F(X,Y) = XY' + X'Y' = m_2 + m_0$. The truth table for the function is given in Table 3.8, and the function is also mapped to a K-map as shown in Fig. 3.3.

Adjacent Cell Two cells are adjacent if they differ on only one variable. The cells $X'Y'$ and $X'Y$ are adjacent because their only difference is Y' and Y. Adjacent cells can be combined in order to simplify a K-map's function.

Table 3.7 Minterms for two variables

X Y	Minterms	Designation
0 0	$X'Y'$	m_0
0 1	$X'Y$	m_1
1 0	XY'	m_2
1 1	XY	m_3

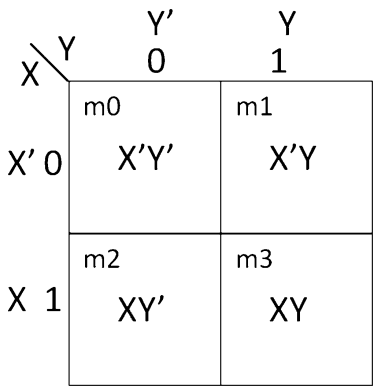


Fig. 3.2 Two-variable K-map

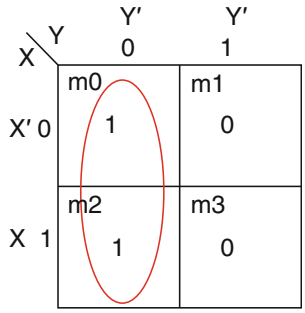


Fig. 3.3 The function in a K-map

Table 3.8 The truth table for the function F(X, Y) = XY′ + X′Y′

X Y	F
0 0	1
0 1	0
1 0	1
1 1	0

As shown in the K-map, the cells m_0 and m_2 contain ones and the other cells contain zeros. The cells m_0 and m_2 are adjacent to each other. Note that the adjacent cells take up the entire column of Y′, and all other cells are *zero*. Our simplified function is therefore F(X,Y) = Y′.

Example 3.4 Simplify the following function:

$$F(X, Y) = X'Y + XY' + XY$$

or

$$F(X, Y) = m_1 + m_2 + m_3$$

Transferring minterms into a K-map results in Fig. 3.4.

As shown in Fig. 3.4, the cells m_2 and m_3 are adjacent, so they can be combined. Likewise, the cells m_1 and m_3 can be combined. By reading the map, you will have the simplified function.

Cells m_2 and m_3 are the entire row X, and cells m_1 and m_3 are the entire column Y, with the other cell being *zero*. Therefore,

$$F(X, Y) = X + Y$$

3.4.1 Three-Variable Map

A three-variable K-map contains eight cells, and each cell represents a minterm as shown in Fig. 3.5.

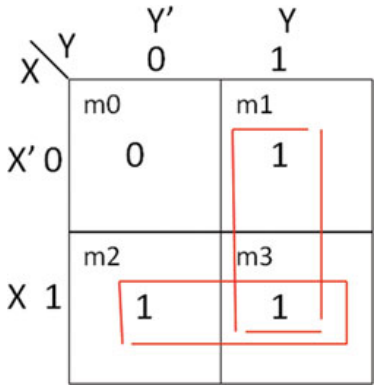


Fig. 3.4 K-map for F(X, Y) = X′Y + XY′ + XY

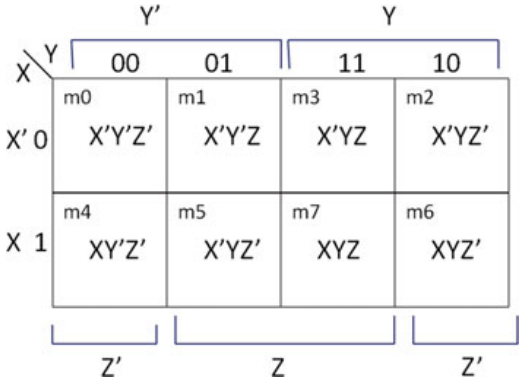


Fig. 3.5 Three-variable Map

Observe the following about the K-map in Fig. 3.5:

(a) At row 0, all four cells contain X′; therefore this row is labeled X′.
(b) At row 1, all four cells contain X; therefore this row is labeled X.
(c) At the columns 11 and 10, all four cells contain Y; therefore these columns are labeled Y.
(d) At the columns 00 and 01, all four cells contain Y′; therefore these columns are labeled Y′.
(e) At the columns 01 and 11, all four cells contain Z; therefore these columns are labeled Z.
(f) At the columns 00 and 10, all four cells contain Z′; therefore these columns are labeled Z′.

Adjacent cells can be grouped together in a K-map; in a K-map it can combine 2 cells, 4 cells, 8 cells, and 16 cells. Figure 3.6 shows how the *ones* could be grouped in a K-map.

In Fig. 3.6, all four cells can be combined. Folding the K-map horizontally twice will result in all of the *ones* overlapping, and row X′ covers all four *ones*.

For Fig. 3.7, consider folding a K-map once more. The four *ones* will overlap if the map is folded once horizontally, then vertically. Also note that Z′ covers all four *ones*.

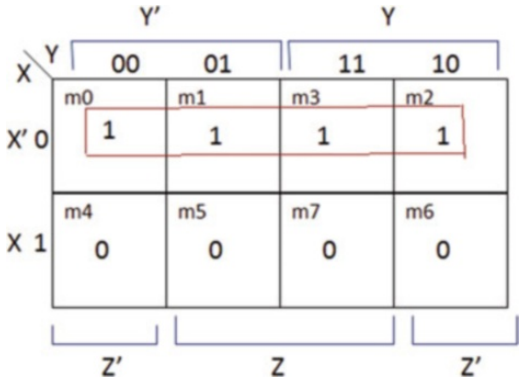


Fig. 3.6 The grouping of four cells in a K-map where the simplified F(X,Y, Z) = X′

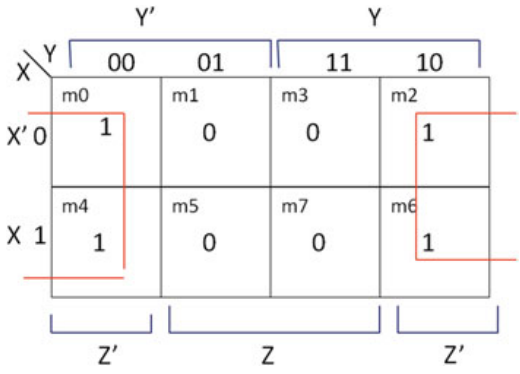


Fig. 3.7 Combing the four Z′ cells together in the K-map

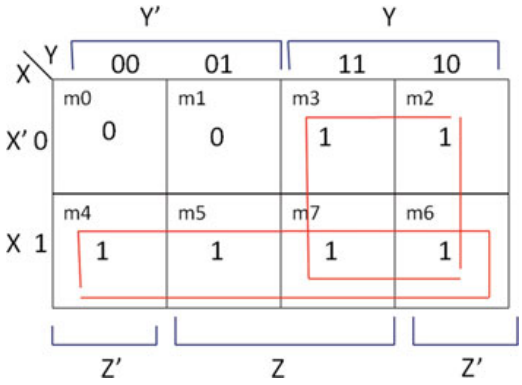


Fig. 3.8 The combination of the cells of the example function in a K-map

Example 3.5 Simplify the following function:

$$F(X, Y, Z) = X'Y + XZ + XZ'$$

First, each term of the function must be transferred to the K-map.

(a) The first term being X′Y, place a *one* on each cell located at the intersection of the X′ row and the Y column as shown in Fig. 3.8. (m_3 and m_2)

(b) The second term is XZ, so place a *one* on each cell located at the intersection of the Y′ column and the X row. (m_5 and m_7)
(c) The third term being XZ′, place a *one* on each cell in the intersection of the X row and the Z column. (m_4 and m_6)

Since our adjacent cells include the entire row X and every cell in the columns Y, a simplified form of this function would be F(X,Y,Z) = X + Y.

Example 3.6 Read the K-maps of Fig. 3.9a–d to determine the simplified function.

(a) All adjacent cells in columns Z′ and columns Y are *one*.

$$F(X, Y, Z) = Z' + Y$$

(b) The cells in row X′, columns Y′ are adjacent, as are the cells in row X, columns Y.

$$F(X, Y, Z) = X'Y' + XY$$

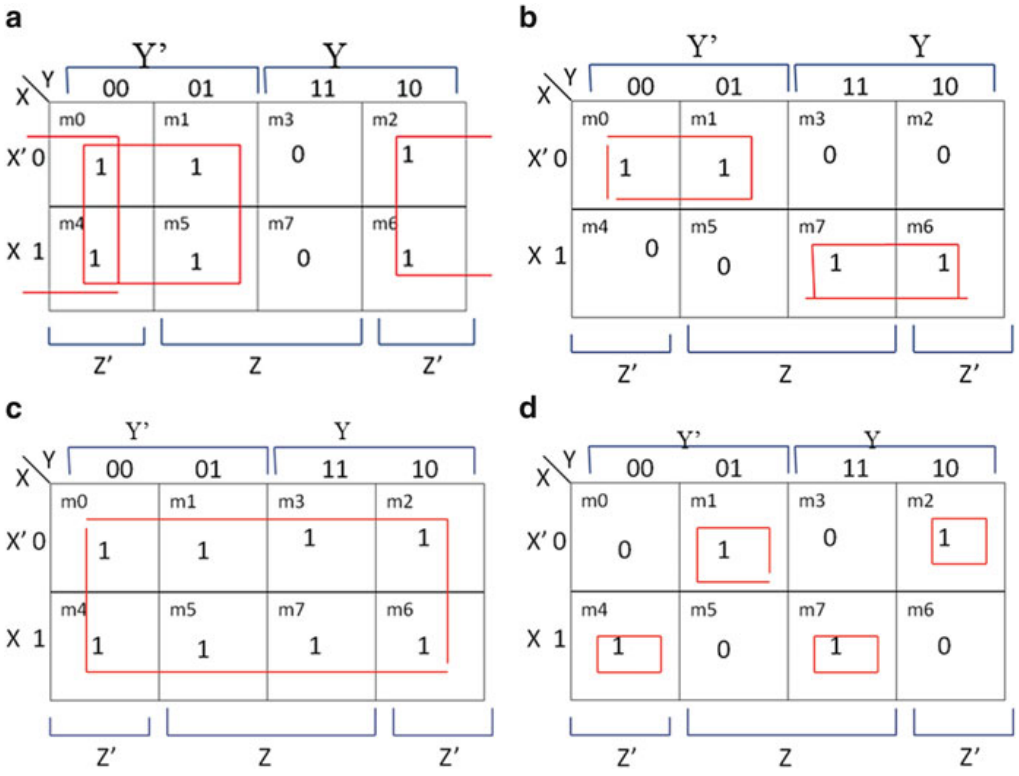


Fig. 3.9 K-Maps for Example 3.6

(c) All cells are *one*, so the function is always equal to *one*.

$$F(X, Y, Z) = 1$$

(d) Without adjacent cells to simplify the terms, the function equals the *ones*:

$$F(X, Y, Z) = X'Y'Z + X'YZ' + XY'Z' + XYZ$$

3.4.2 Four-Variable K-Map

Four-variable K-maps contain 16 cells as shown in Fig. 3.10. Please note the specific layout of the map.

The following describes the coverage of each variable by K-map:

- W covers rows 11 and 10.
- W′ covers rows 00 and 01.
- X covers rows 01 and 11.
- X′ covers rows 00 and 10.
- Y covers columns 11 and 10.
- Y′ covers columns 00 and 01.
- Z covers columns 01 and 11.
- Z′ covers columns 00 and 10.

Example 3.7 Simplify the following function:

$$F(W, X, Y, Z) = m_0 + m_2 + m_8 + m_{10}.$$

The function is transferred to the K-map as shown in Fig. 3.11. If the K-map is folded once vertically and horizontally from the middle, then all four cells

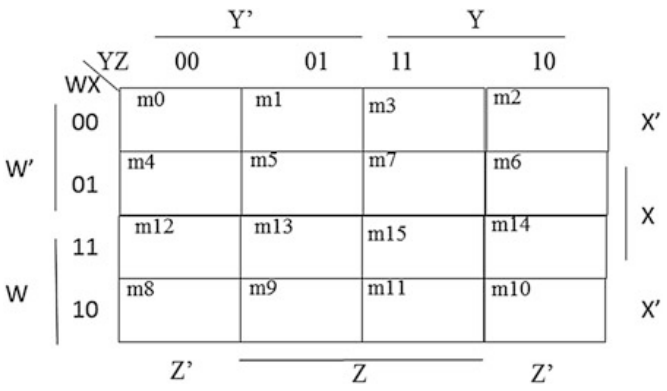


Fig. 3.10 Four-variable K-map

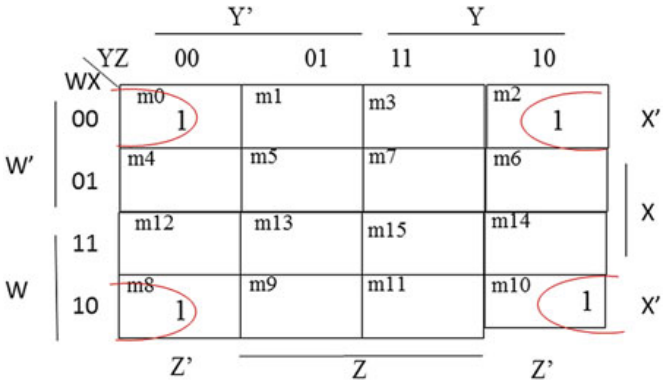
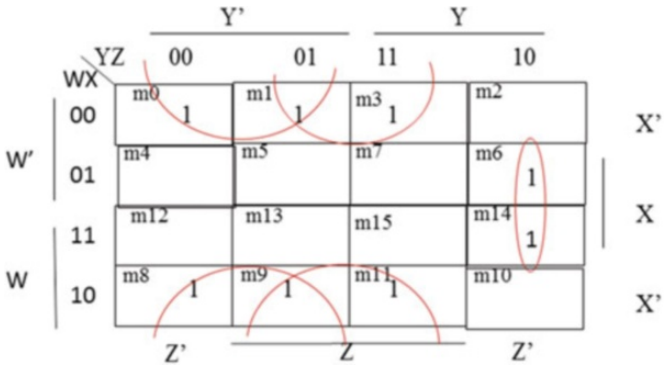


Fig. 3.11 K-map for F(W, X,Y, Z) = $m_0 + m_2 + m_8 + m_{10}$

containing *one* overlap each other. Note that each of these cells make up all intersections of X′ and Z′.

The simplified function is F(W,X,Y,Z) = X′Z′.

Example 3.8 Read the following K-map:



$$F(W, X, Y, Z) = X'Y' + X'Z + XYZ'$$

3.5 Sum of Products (SOP) and Product of Sums (POS)

The *sum of products* of a function is its simplified *sum of minterms*.

Observe the function F(X,Y) = XY′ + XY is in the form of SOP where the addition sign is used for OR logic, and XY is called product. Consider function F1 (X,Y,Z) = XY′ + YZ′ + XZ, and Fig. 3.12 shows logic circuit for function F1 which is made of AND-OR.

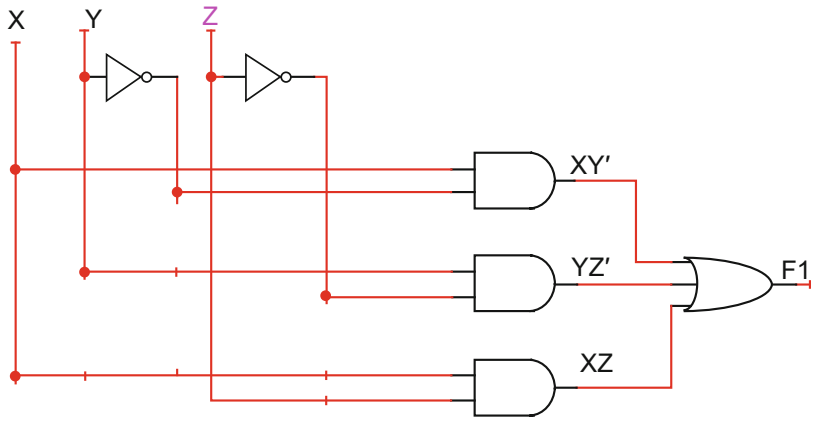


Fig. 3.12 Logic circuit for F1(X,Y,Z) = XY′ + YZ′ + XZ made of AND-OR gates

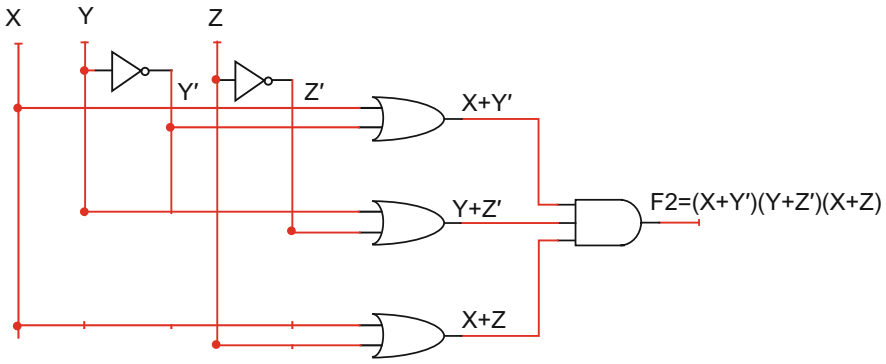


Fig. 3.13 Logic circuit for F2(X,Y,Z) = (X + Y′)(Y + Z′)(X + Z)

Consider function F2(X,Y,Z) = (X + Y′)(Y + Z′)(X + Z) which is represented by the product of sums, and Fig. 3.13 shows logic circuit for function F2 which is in the form OR-AND.

Example 3.9 Simplify the following function in the form of SOP and POS.

- **F(X,Y,Z) = ∑(0,1,6,7)**

Combining the *ones* (the minterms m_0, m_1, m_6, and m_7) in a K-map results in Fig. 3.14.

The *sum of products* is therefore: F(X,Y,Z) = X′Y′ + XY.

Combining the *zeros* in a K-map returns the significant *maxterms* as in Fig. 3.15. If F equals ∑(0,1,6,7), then F′ equals $\pi(M_2M_3M_4M_5)$.

Since the *product of maxterms* is equal to the *complement* of F, in order to find F, both sides of the function will be complemented:

$$F(X, Y, Z) = (XY' + X'Y)'$$

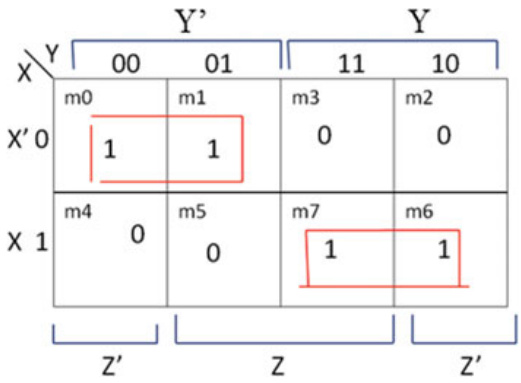


Fig. 3.14 K-map for F(X, Y,Z) = ∑(0,1,6,7)

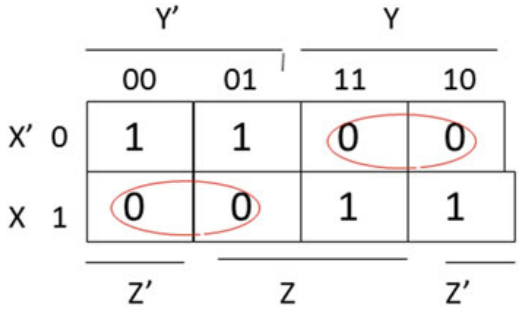


Fig. 3.15 K-map for F′(X, Y,Z) = π($M_2M_3M_4M_5$)

Using DeMorgan's theorem

$$F(X, Y, Z) = (XY')'(X'Y)'$$

or

F(X,Y,Z) = (X′ + Y)(X + Y′) in the final, *product of sums* form.

3.6 *Don't Care* Conditions

In a truth table, if certain combinations of the input variables are impossible, they are considered *don't care* conditions. These conditions are where the output of the function does not matter. For example, binary-coded decimal (BCD) is 4 bits and only 0000 to 1001 are used, so from 1010 to 1111 are not BCD; the truth table or K-map values are *don't care*s. A truth table or K-map cell marked with a "X" or "d" is a *don't care* term, and output will not be affected whether it is a *one* or *zero*. The *don't care* can be used to expand the adjacency of cells in a K-map to further simplify a function, since their output does not matter.

Example 3.10 Figure 3.16 shows a K-map with *don't care* minterms at m_1, m_{10}, and m_{13}. Since a *don't care* can output either a *zero* or *one*, we can assume it is a *one* in order to expand a grouping of adjacent cells.

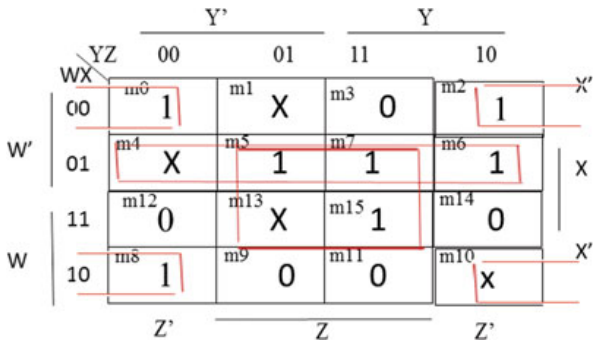


Fig. 3.16 K-map with *don't care* minterms

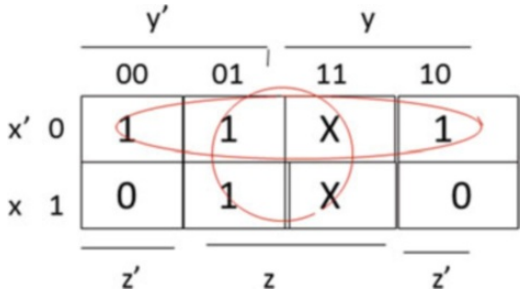


Fig. 3.17 K-map for F(X, Y,Z) = $m_0 + m_1 + m_2 + m_5$ and D(X,Y,Z) = $m_3 + m_7$

From Fig. 3.16, the function would be F(W,X,Y,Z) = XZ + X′Z′ + XW′.

When minterms for function F are *don't care* terms, the *don't care function* D is equal to the sum of the *don't care* minterm(s). If m_7 is the only *don't care*, then don't care function is represented by D(X,Y,Z) = m_7.

Example 3.11 Simplify the following function where D is a *don't care* function:

$$F(X, Y, Z) = m_0 + m_1 + m_2 + m_5$$
$$D(X, Y, Z) = m_3 + m_7$$

Using these values results in the K-map in Fig. 3.17. By grouping adjacent cells and using the *don't care* terms, F(X,Y,Z) = X′ + Z.

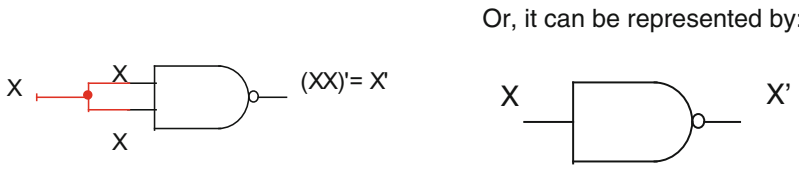
3.7 Universal Gates

The NAND and NOR gates are called universal gates. With NANDs or NORs, designers are able to construct other logic gates such as OR, AND, and NOT gates.

3.7.1 Using NAND Gates

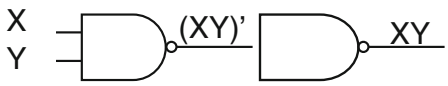
(a) **NOT from NAND**

A NOT gate is generated by connecting the inputs of a NAND gate together as shown in the following figure.



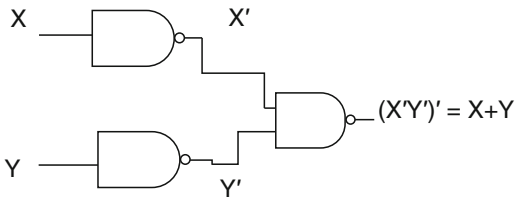
(b) **AND from NAND**

An AND gate is constructed by connecting the inputs to a NAND gate and putting another NAND on the output (to act as a NOT gate).



(c) **OR from NAND**

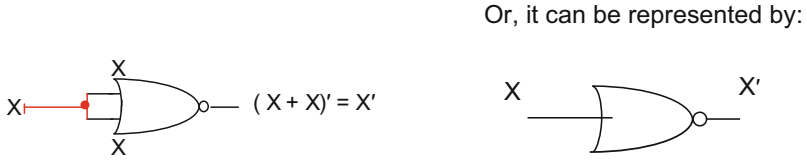
An OR gate is constructed by connecting each input to an individual NAND gate and putting every output into a single NAND which acts as a NOT gate.



3.7.2 Using NOR Gates

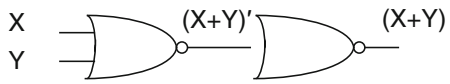
(a) **NOT from NOR**

A NOT gate is generated by connecting the inputs of a NOR gate together as shown in the following figure.



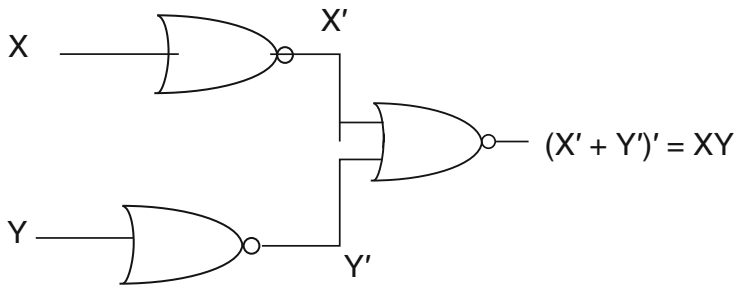
(b) **OR from NOR**

An OR gate is constructed by connecting the inputs to a NOR gate and putting another NOR on the output (to act as a NOT gate).



(c) **AND from NOR**

An AND gate is constructed by connecting each input to an individual NOR gate and putting every output into a single NOR which acts as a NOT gate.



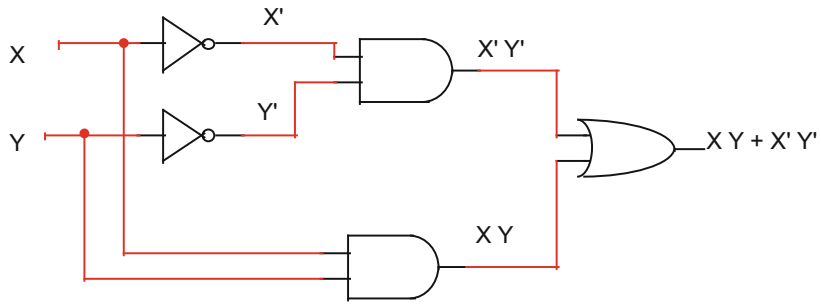


Fig. 3.18 Logic circuit for F(X,Y) = XY + X′Z′

3.7.3 *Implementation of Logic Functions Using NAND Gates or NOR Gates Only*

Many logic functions are implemented using only NAND or NOR gates, rather than a combination of various gates. Most logic gate ICs contain multiple gates of a single type, such as an IC containing eight AND gates. Using a single type of gate can reduce the number of ICs needed.

Consider the function F(X,Y) = X′Y′ + XY and its logic circuit diagram in Fig. 3.18.

This diagram would require one IC for AND gates, another for NOT gates, and one OR gate, or a total of three separate ICs.

By complementing the function twice, the right side of the equation may be easier to use with NAND and NOR gates.

Example 3.12 Create a logic circuit using only NAND and only NOR for the function F(X,Y) = X′Y′ + XY.

3.7.4 *Using NAND Gates*

- Complement the right side of the equation twice.
- F(X,Y) = X′Y′ + XY → F(X, Y) = [(X′Y′ + XY)′]′.
- Use Boolean theorems to make it NAND friendly:
  - F(X,Y) = [(X′Y′)′(XY)′]′

Consider the final function once more: F(X, Y) = [(X′Y′)′(XY)′]′, and substitute placeholders for the inner terms (Fig. 3.19).

- F = [(X′Y′)′(XY)′]′ = [AB]′ (A *NAND* B)
  - A = (X′Y′)′ = X′ *NAND* Y′
  - B = (XY)′ = X *NAND* Y
  - X′ = X *NAND* X
  - Y′ = Y *NAND* Y

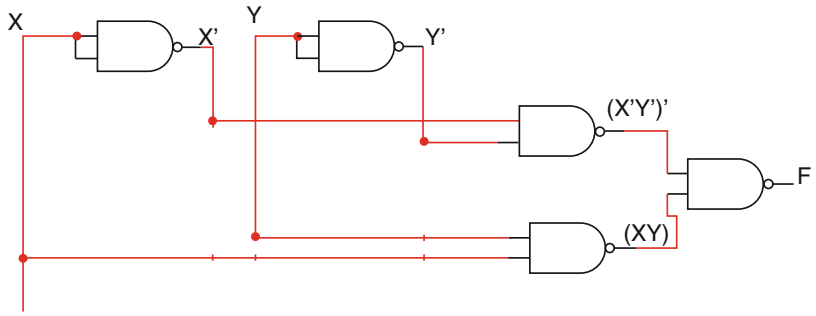


Fig. 3.19 Logic circuit of F(X,Y) = X′Y′ + XY using only NAND gates

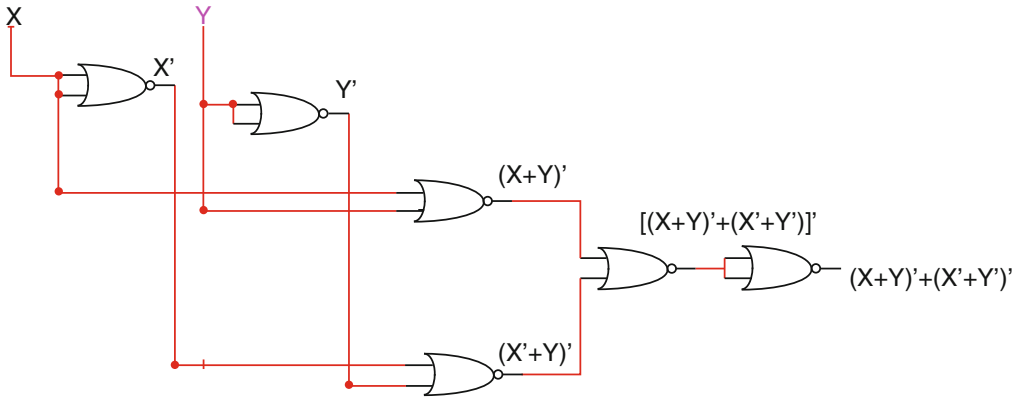


Fig. 3.20 Logic circuit of F(X,Y) = X′Y′ + XY using only NOR gates

3.7.5 Using NOR Gates

- Complement the right side of the equation twice:
  - F(X,Y) = X′Y′ + XY → F(X, Y) = [(X′Y′ + XY)′]′
- Use Boolean theorems to make it NOR friendly:
  - F(X,Y) = [(X′Y′)′(XY)′]′
  - F(X,Y) = [(X+Y)(X′+Y′)]′
  - F(X,Y) = [(X+Y)′ + (X′+Y′)′]′

Consider the final function once more: F(X, Y) = [(X+Y)′ + (X′+Y′)′]′, and substitute placeholders for the inner terms (Fig. 3.20).

- F = [(X′Y′)′(XY)′]′ = [A+B]′ (A *NOR* B)
  - A = (X+Y)′ = X *NOR* Y
  - B = (X′+Y′)′ = X′ *NOR* Y′
  - X′ = X *NOR* X
  - Y′ = Y *NOR* Y

3.8 Summary

- A digital circuit is made of the combination of the different gates with the multiple digital inputs and the multiple digital outputs; the outputs depend only to the current values of the inputs.
- A combinational logic circuit can be represented by a Boolean function or truth table.
- Boolean theorems or K-map can be used to simplify a Boolean function.
- Boolean function can be represented by the sum of the products (SOP) or the product of sums (POS).
- The NAND and NOR gates are called universal gates. It can generate other gates by using NAND or NOR gates.
- Don't care condition is the input value that never applied to a combinational circuit which results output with don't care condition (0 or 1).
- Chapters 1, 2, and 3 cover basic topics in order to be able to design digital system; Chap. 4 presents how to design a digital system and covers digital components that are used for designing digital system such as decoder, multiplexer, binary adder, binary subtractor, and arithmetic logic unit (ALU).