**Module -7**

**Karnaugh Maps**


1. **Introduction**
2. **Canonical and Standard forms**
   - **2.1 Minterms**
   - **2.2 Maxterms**
   - **2.3 Canonical Sum of Product or Sum-of-Minterms (SOM)**
   - **2.4 Canonical product of sum or Product-of-Maxterms(POM)**
   - **2.5 Standard Sum-of-Product(SOP) and Product-of-sum(POS)**
3. **Karnaugh Map**
   - **3.1 K-maps with 2, 3, 4 variables**
   - **3.2 Minimisation using K-Maps**
   - **3.3 K-maps with don't care conditions**
4. **Summary**


**Learning Objectives:**

1. Learn the techniques of simplifying logic expression
2. Understand standard and canonical forms of logic expressions
3. Study the k-map technique for mapping and simplification.

## 1. Introduction

Boolean algebra is the mathematical basis for logic design. Generating Boolean equations to implement a desired logic function is a necessary step before a circuit can be implemented. Truth tables provide a simple means of describing logical relationships between inputs and outputs. Once a truth table has been created, it is not always easy to convert truth table directly into Boolean equation. This conversion becomes more difficult as the number of variables in an equation increases.

The operation of any logic circuit can always be expressed as sum of products form of product of sum form. It is also known that these expressions have longer or unwieldy form to write as well as implement. It is for this reason that minimization of logic expression is so important. This results in simpler circuit and reduced Boolean expression.

When circuit with more than two or three variables is involved a better method of circuit reduction that works well is the Karnaugh maps. A graphical means of converting a truth table into a logic equation was invented by Karnaugh and is popularly known as Karnaugh map or K-map.

Digital circuits can be minimized using Boolean algebra or Karnaugh maps. K-maps are most commonly used manual method because K-map involves simple representation and inspection of truth table in different way. In this unit, we shall discuss about the minimization technique based on Karnaugh maps.

## 2. Canonical and Standard and forms

There are several forms of Boolean expressions used for representing the logical functions. A given logic function can be represented in terms different combinations of logical variables with complemented as well as un-complemented forms. Any logic function can be represented in terms either canonical form or a standard form. Let us begin the discussion by defining a function using a truth table as shown in fig. 1.

| Row | A | B | C | F |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 1 | 1 |
| 4 | 1 | 0 | 0 | 0 |
| 5 | 1 | 0 | 1 | 0 |
| 6 | 1 | 1 | 0 | 1 |
| 7 | 1 | 1 | 1 | 1 |

**Figure 1: A logic function defined by truth table.**

This truth table indicates the output (F) of logic circuit as a function of three variables (A,B,C). This logic function may be defined as F(A,B,C). Let us define **product** and **sum** terms as defined in normal algebra. These terms do not literally mean ñmultiplicationò or ñadditionò. In Boolean algebra a ñ**product**ò means a logical AND operation while ñ**sum**ò is a logical OR operation.

Before learning about the sum-of-product term and product-of-sum terms, it is essential to define two important terms namely minterm and maxterm.

**2.1 Minterm**:

A minterm is a product term in which all the variables appear exactly once, either complemented or un-complemented form. Example: (A.Bô.C.Dô). A minterm is TRUE for the minimum number of combinations of inputs i.e. exactly once. As it is required to minimize the coverage, all the variables are coupled through AND operation. Since AND term is TRUE only if all of the inputs are TRUE. The product term contains both complemented as well as un-complemented variable. To generate the output of product term as ó1ô, the un-complemented variable must be ó1ô and complemented variable must be ó0ô. Let us now show the minterms for the logic function defined in the fig.1.

| Row | A | B | C | Minterm | Designation | F |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | Aô Bô Cô | m0 | 1 |
| 1 | 0 | 0 | 1 | Aô Bô C | m1 | 0 |
| 2 | 0 | 1 | 0 | Aô B Cô | m2 | 0 |
| 3 | 0 | 1 | 1 | Aô B C | m3 | 1 |
| 4 | 1 | 0 | 0 | A Bô Cô | m4 | 0 |
| 5 | 1 | 0 | 1 | A Bô C | m5 | 0 |
| 6 | 1 | 1 | 0 | A B Cô | m6 | 1 |
| 7 | 1 | 1 | 1 | A B C | m7 | 1 |

**Figure 2: Table of Minterms of 3 variables**

Figure 2 indicates the assignment of Minterm to each term of the truth table. Each minterm is TRUE only for specific combination of inputs with which it is associated. In order to qualify as the minterm, the product must contain all input variables either in un-complemented form or complemented form.

**2.2 Maxterm**:

A maxterm is a sum term in which all the variables appear exactly once, either complemented or un-complemented form. Example: (Aô+B+C). A maxterm is TRUE for the maximum number of combinations of inputs i.e. more than once. As it is necessary to maximize the coverage, all the variables are coupled by an OR operation for Boolean operation. The OR operation is TRUE if any of its inputs are TRUE.

The sum term contains both complemented as well as un-complemented variable. To generate the output of sum term as ó1ô, the un-complemented variable must be ó1ô and complemented variable must be ó0ô. Let us now show the assignment of maxterms for the logic function.

| Row | A | B | C | Maxterm | Designation | F |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | A+B+C | M0 | 1 |
| 1 | 0 | 0 | 1 | A+B+ Cô | M1 | 0 |
| 2 | 0 | 1 | 0 | A+ Bô+ C | M2 | 0 |
| 3 | 0 | 1 | 1 | A+B ô+Cô | M3 | 1 |
| 4 | 1 | 0 | 0 | Aô+ B+ C | M4 | 0 |
| 5 | 1 | 0 | 1 | Aô+ B+Cô | M5 | 0 |
| 6 | 1 | 1 | 0 | Aô+Bô+ C | M6 | 1 |
| 7 | 1 | 1 | 1 | Aô+Bô+ Cô | M7 | 1 |

**Figure 3: Table of Maxterms of 3 variables**

Fig. 3 indicates the assignment of Maxterm to each term of the truth table. Each maxterm is TRUE only for specific combination of inputs with which it is associated. In order to qualify as the maxterm, the sum must contain all input variables either in un-complemented form or complemented form.

**Relation between Minterm and Maxterm**

In order to understand the relation between minterm and maxterm, consider the minterm and maxterm for a particular row. Following table gives the minterms and maxterms for three variable. Here number of minterms as well as maxterms are $2^3 = 8$.

| Variable | | | Minterm | Maxterm |
|---|---|---|---|---|
| A | B | C | $m_i$ | $M_i$ |
| 0 | 0 | 0 | AôBôCô = m0 | A+B+C =M0 |
| 0 | 0 | 1 | AôBôC = m1 | A+B+Cô=M1 |
| 0 | 1 | 0 | AôB Cô = m2 | A+Bô+C=M2 |
| 0 | 1 | 1 | AôB C = m3 | A+Bô+Cô=M3 |
| 1 | 0 | 0 | A BôCô = m4 | Aô+B+C=M4 |
| 1 | 0 | 1 | A BôC = m5 | Aô+B+Cô=M5 |
| 1 | 1 | 0 | A B Cô = m6 | Aô+Bô+C=M6 |
| 1 | 1 | 1 | A B C = m7 | Aô+Bô+Cô=M7 |

**Figure 4: Table for Minterm and Maxterm**

Let us consider the first minterm form table.

Minterm = AôBôCô

Let us take complement of Minterm i.e. (AôBôCô)ô

= (Aô)ô+(Bô)ô+(Cô)ô (Appling De Morgans theorm)

= A+B+C = Maxterm

It is clear that each minterm is the logical inverse of the corresponding maxterm and vice versa. This indicates when the minterm is TRUE only when that combination is present while the maxterm is FALSE only when that combination is present.

**2.3 Canonical Sum of Product or Sum-of-Minterms (SOM)**

When a Boolean function is expressed as the logical sum of all the minterms from the rows of a truth table, for which the value of function is 1 then it is known as the **canonical sum of minterm** form.

Let us find out Sum of minterm for Fig.1.

F(A,B,C) = × (0,3,6,7)

= m0+m3+m6+m7

= AôBôCô+ AôB C+ A B Cô+ A B C

**2.4 Canonical product of sum or Product-of-Maxterms (POM)**

When a Boolean function is expressed as the logical product of all the maxterms from the rows of a truth table, for which the value of function is 0 then it is known as the **canonical product of maxterm** form. Let us find out product of maxterm for Fig.1.

F(A,B,C) = Ø(1,2,4,5)

= M1 M2 M4 M5

= (A+B+ Cô) (A+ Bô+ C) (Aô+ B+ C) (Aô+ B+Cô)

Note, the relation between the sum of products and product of sum in canonical form of an expression is

$$m_i' = M_i$$

Here $m_i$ is the minterm for row i an $M_i$ is the maxterm for row i.

To convert from one canonical form to another, interchange × and Ø signs and replace the row numbers with those not included in the original form. For example

× m(0,3,6,7) = Ø M(1,2,4,5)

**2.5 Standard Sum-of-Product (SOP) and Product-of-sum(POS)**

Another important form of a Boolean expression is standard form. Standard form of Boolean expression facilitates simplification. The result after simplification provides more desirable implementations. The standard form of a Boolean expression has the same canonical sum-of-products and product-of-sum form except that the product and sum terms need not be minterms and maxterms respectively.

The standard form of Boolean expression is Product of Sums (POS). It contains two level implementation of the Boolean expression. First level is the Oring to provide sum term and then at second level ANDing results in Product term. Both SOP and POS are the two standard forms of Boolean expression. Both includes the sum and product terms for the two level implementations of Boolean expression.

A **sum of product** expression contains terms with product of literals or variable.

F(A,B,C) = Bô+ AôBCô+AB

Similarly, the **product of sum** expression contains terms with sum of literals.

F(A,B,C) = (A+C) . (A+B+C) . Cô

Both these expressions represent **standard SOP** and **POS** forms of Boolean expressions. From the above expressions it is also clear that all the individual terms do not involve all the three variables. If each term of SOP or POS form involves all the three variables in each term then these expressions are known as **Canonical SOP and POS forms**.

Each individual term in canonical SOP form is known as **Minterm** (e.g. ABC) and each individual term in canonical POS form is known as **Maxterm** (e.g. AõBõCõ).

SOP from can be converted to canonical SOP by ANDing the term with the ORing term of missing variable and its complement.

f = AB = AB.(C+Cõ) = ABC +ABCõ

Similarly, POS from can be converted to canonical POS by ORing the term with the ANDing term of missing variable and its complement.

f = (A+C) = (A+B.Bõ+C) = (A+B+C) (A+Bõ+C)

**3. Karnaugh Map**

A Karnaugh map (K-map) is a graphical method used to minimize Boolean expressions without using Boolean algebra rules, laws and theorems. A K-map can be thought of as a special version of a truth table. Using a K-map, it is possible to minimize expressions with two to four variables easily and preferred for many small design problems. Many larger digital designs prefer computer implementation of different algorithms. The Quine-McCluskey method is a technique suitable for reduction of Boolean expression using computer algorithms.

Karnaugh maps (K-maps) contain exactly the same information as truth tables. The difference is that the K-map uses a matrix format to hold the output values. The K-map is arranged so that logically adjacent terms are also adjacent in the matrix. These terms can be logically combined and hence minimized. More importantly, the minimization process is performed visually as compared to traditional minimization techniques.

### 3.1 : K-map of 2,3,and 4 variables:

The K-map consists of a number of squares called cells. Each square cell represents a minterm (or maxterm). Fig. 5 indicates graphical representation of k-maps for 2, 3 and 4 variables.

| Y / X | 0 | 1 |
|---|---|---|
| 0 | m0 | m1 |
| 1 | m2 | m3 |

| YZ / X | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | m0 | m1 | m3 | m2 |
| 1 | m4 | m5 | m7 | m6 |

| ZW / XY | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | m0 | m1 | m3 | m2 |
| 01 | m4 | m5 | m7 | m6 |
| 11 | m12 | m13 | m15 | m14 |
| 10 | m8 | m9 | m11 | m10 |

**Figure 5:Graphical representation for k-map of 2, 3 and 4 variables.**

In K-map, rows and columns are assigned the binary code (particularly Gray code) such that two adjacent rows or columns differ only in one bit. In addition, the column on extreme left is adjacent to column on the extreme right. Similarly, the top row is adjacent to the bottom row. Two cells are said to be adjacent if they are physically adjacent to each other or made adjacent by wrapping left to right column or to and bottom rows.

The K-map indicates representation of $2^n$ cells for n=2,3 and 4. Each cell represents the minterm and the minterm number is written in each cell. Any Boolean function can be represented by placing ó1ô in the cell corresponding to presence of minterm.

Some logical relationships, however, do not require that every possible result necessarily be a one or a zero. For example, out of 16 possible results from the combination of four variables, only 12 results may be used by the application. This may sound odd, but one explanation could be that the particular application simply cannot provide the full 16 combinations of inputs. The specific reasons for this are as numerous as the many different applications that exist. In such circumstances these so-called ***don't care*** results can be used to reduce the complexity of your logic. Because the application does not care what result is generated for these few combinations, you can arbitrarily set the results to 0s or 1s so that the logic is minimized.

### 3.2 Minimisation using K-Maps

Karnaugh maps contain *exactly* the same information as truth tables. The difference is that the Karnaugh map uses a 'matrix' format to hold the output values. The Karnaugh map is arranged so that, as far as possible, logically adjacent product terms are also adjacent in the 'matrix' and so can be logically combined hence minimised. Essentially the minimization process, described using Boolean algebra is performed visually.

The procedure for minimisation using K-map is

1. Write a minterm Boolean expression from a truth table. Each 1 in Output column of truth table produces ANDed term. These ANDed terms are ORed to form sum-of-products (minterms).
2. Draw the K-map with the desired number of cells.
3. Fill the cells by 1 where the product term (minterm) is present. Each ANDed set of variables from the minterm is placed in appropriate square of the map. The map is just a very special output column of the truth table.

4. Group/ Draw loops around neighbouring cells **in pairs or quads or octets** 1s together on the map.
5. Eliminate the variable. When a variable and its complement are within a loop, that variable is eliminated.
6. OR the terms that remain to generate simplified minterm Boolean expressions

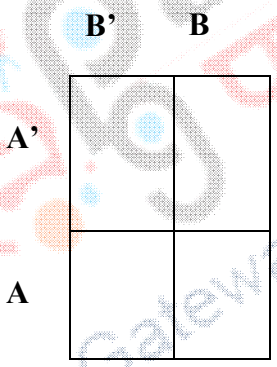
**Example 1:** Let us consider an example to understand the procedure of converting truth table into a simplified Boolean expression using K-map.

(a) Given Truth table

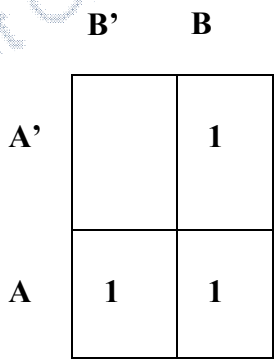
| Inputs | | Outputs | |
|---|---|---|---|
| A | B | Y | |
| 0 | 0 | 0 | |
| 0 | 1 | 1 | $A\hat{o}B$ |
| 1 | 0 | 1 | $A.B\hat{o}$ |
| 1 | 1 | 1 | A.B |

(b) Minterm Boolean expression $Y = A\hat{o}B + A.B\hat{o} + A.B$

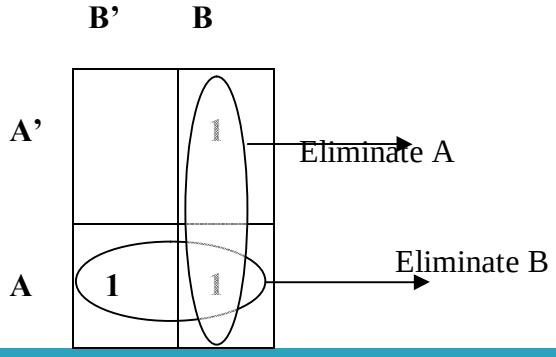
(c) Let us draw a K-map of two variables




(d) Plotting 1s on map




(e) Looping 1s



(f) Eliminating variables to form simplified Boolean expressions Y= A+ B

**Example-2:** Let us consider a truth table with three variables.

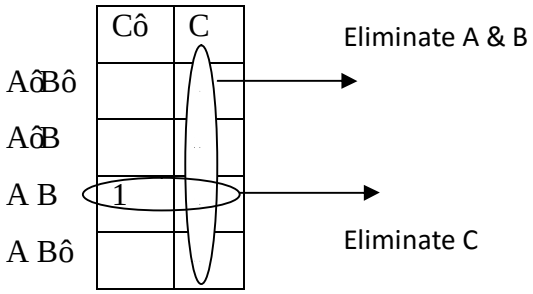
| Inputs | | | Output | |
|---|---|---|---|---|
| A | B | C | Y | |
| 0 | 0 | 0 | 0 | |
| 0 | 0 | 1 | 1 | $A\bar{B}\bar{C}$ |
| 0 | 1 | 0 | 0 | |
| 0 | 1 | 1 | 1 | $A\bar{B}C$ |
| 1 | 0 | 0 | 0 | |
| 1 | 0 | 1 | 1 | $AB\bar{C}$ |
| 1 | 1 | 0 | 1 | $ABC\bar{0}$ |
| 1 | 1 | 1 | 1 | ABC |

Unsimplified Boolean expression is $Y= A\bar{B}\bar{C}+A\bar{B}C+AB\bar{C}+ABC\bar{0}+ABC$

Let us draw K-map & map 1s

| | $C\bar{0}$ | C |
|---|---|---|
| $A\bar{B}\bar{0}$ | | 1 |
| $A\bar{B}$ | | 1 |
| A B | 1 | 1 |
| A $B\bar{0}$ | | 1 |

Grouping together



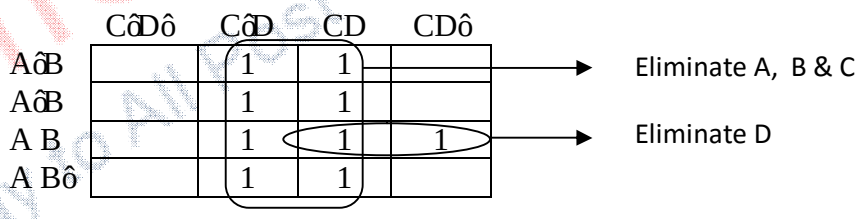

Simplified Boolean expression is

Y= C + A B

**Example -3:** Let us consider a truth table of 4 variables.

| Inputs | | | | Outputs |
|---|---|---|---|---|
| A | B | C | D | Y |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Plotting and grouping 1s on K-map




The simplified Boolean expression is Y = D + ABC

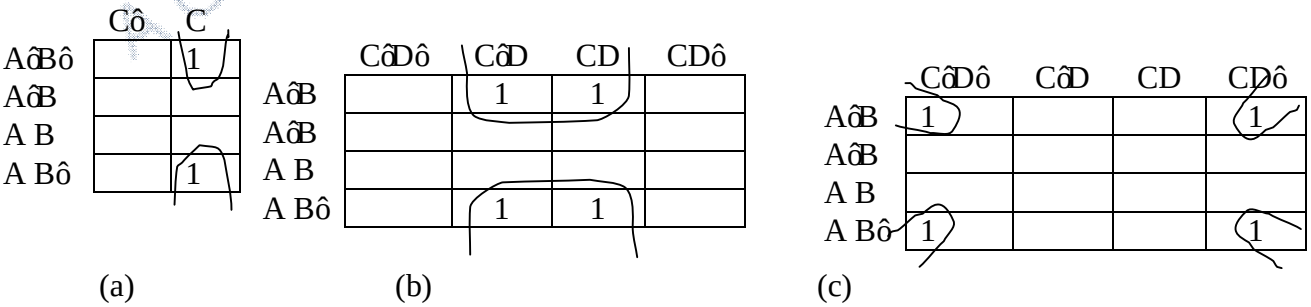



Fig. 6: Some unusual groups / loops

The K-map of Figure 6(a) contains two 1s. Let us think of the top and bottom edges of the map as being connected and as if rolled into a tube, The 1s can then be looped into a group of two and one variable can be eliminated.

Consider a K-map of four variables in figure 6 (b). The top and bottom edges of the map are considered as if tolled into a tube and connected for looping. In figure 6 (c) loop can be formed if the corners of the map are considered connected as if the map is wrapped around a ball. The four 1s in the corner of the map are then grouped into a single loop and thus two variables can be eliminated.

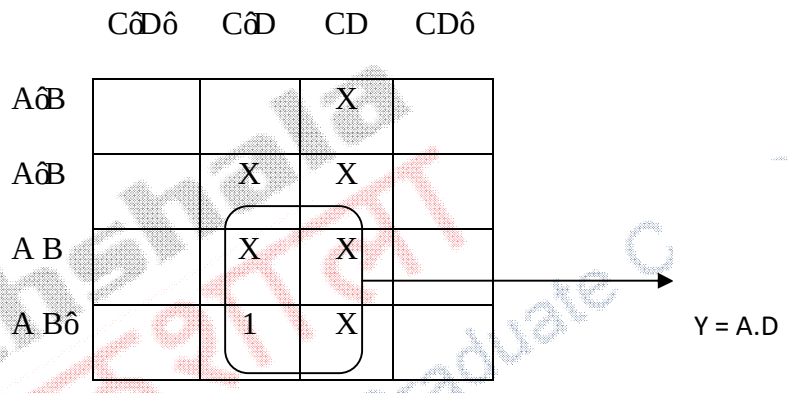
### 3.3 K-maps with don't care conditions

Consider a truth table which accepts inputs 4-bit BCD and generates one output. Note that the binary numbers 0000 to 1001 on the table are used to specify decimal numbers from 0 to 9

| BCD number | | | | Decimal equivalent | Output |
|---|---|---|---|---|---|
| A | B | C | D | number | Y |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 2 | 0 |
| 0 | 0 | 1 | 1 | 3 | 0 |
| 0 | 1 | 0 | 0 | 4 | 0 |
| 0 | 1 | 1 | 0 | 5 | 0 |
| 0 | 1 | 1 | 1 | 6 | 0 |
| 0 | 1 | 0 | 0 | 7 | 0 |
| 1 | 0 | 0 | 0 | 8 | 0 |
| 1 | 0 | 0 | 1 | 9 | 1 |
| 1 | 0 | 1 | 0 | Not used | X |
| 1 | 0 | 1 | 1 | Not used | X |
| 1 | 1 | 0 | 0 | Not used | X |
| 1 | 1 | 1 | 0 | Not used | X |
| 1 | 1 | 1 | 1 | Not used | X |
| 1 | 1 | 0 | 0 | Not used | X |

In 4 variable truth table, there are six combinations (1010 through 1111) are not used in the code. These combinations are called don't cares when plotted on K maps. The don't care may have some effect on simplifying any logic diagram that might be constructed.

Let us consider a problem that generates a warning light when BCD count reaches (1001) i.e. decimal 9. Let us represent the truth table for this example.




For unsimplified Boolean expression without don't care condition would have resulted in

Y = A.B'C'D. But using 6 don't care conditions in K-map, We can form a quad so that B and C can be eliminated so as to result in Y = A.D

**4. Summary**

Any Boolean function F( ) can be expressed as a *unique* sum of minterms and a unique product of maxterms. In other words, every function F() has two canonical forms:

- Canonical Sum-Of-Products (sum of minterms)
- Canonical Product-Of-Sums (product of maxterms)

Another important form of a Boolean expression is standard form. Standard form of Boolean expression facilitates simplification. The result after simplification provides more desirable

implementations. Sum of Products and product of sum are two standard forms of Boolean expression.

A Karnaugh map (K-map) is a graphical method used to minimize Boolean expressions without using Boolean algebra rules, laws and theorems. A K-map can be thought of as a special version of a truth table. The K-map consists of a number of squares called cells. Each square cell represents a minterm (or maxterm).

Using a K-map, it is possible to minimize expressions with two to four variables easily and preferred for many small design problems. Many larger digital designs prefer computer implementation of different algorithms.

**Neighbouring cells** in pairs or quads or octets containing ó1ô or óXô are used for term reduction.