Unit 3 Techniques for Simplifying Boolean Expressions

Structure:

3.1 Introduction

In the last unit, you studied about rules and laws of Boolean algebra, basic gates, universal gates and realization of other gates using universal gates. In a digital system, there are two voltage levels of electrical signals i.e., 0v and 5v. If the supply to the system is maintained, the electrical devices can exist in one of the two voltage levels indefinitely. For example, a bipolar transistor that is non-conducting in a 5 volt system will have approximately 5 volts between collector and emitter. However, when the transistor is turned on and is conducting, the voltage between collector and emitter can be made zero by connecting a suitable load to the transistor. Among the two voltage levels, logic zero is assigned to 0v and logic one is assigned to 5v. The presence or absence of a particular condition can be indicated by the two states logic 0 and logic 1. An algebra developed in the nineteenth century by George Boole (1815-1864), an English mathematician, is well suited for representing the situation above. This branch of mathematics, called Boolean algebra, is a discrete algebra in which the variables can have one of two values, either 0 or 1. Associated with the algebra is a number of theorems which allow the manipulation and simplification of Boolean equations.

Shannon, who was the first to develop information theory, became aware that Boolean algebra was useful in the design of switching networks. Initially, the algebra was used in the design of relay networks. More recently switching circuits were implemented using discrete components but rapid technological advances have seen the introduction of MSI, LSI and VLSI devices and because of the sophisticated and versatile nature of these components there have been significant changes in the design techniques used by engineers. In spite of these changes it is still essential for engineers to have a good working knowledge of traditional switching theory.

In digital system designing, one of the main objectives to implement a Boolean function using minimum number of discrete gates. The cost of the circuit will be low when the number of gates used in implementing a Boolean function is smaller. Simplification of Boolean function can be performed using a purely algebraic process. Simplification using algebraic process can be tedious, and at the end of process designer is not always sure whether the simplest solution is obtained or not.

Instead of using algebraic process, Boolean function can be simplified easily by plotting the function to Karnaugh map and simple rules to reduce the Boolean function. Upto six variables simplification of Boolean function using karnaugh maps will be very straightforward. Tabulation method which has been developed by Quine and McCluskey will be better method to simplify the Boolean function which has more than six variables. So in this unit we will study about Boolean expressions and functions and simplification of Boolean expressions using Karnaugh map and Quine-McCluskey Methods.

Objectives:

By the end of Unit 3 the learners are able to

- define Boolean variable and Boolean function
- simplify the expressions using Boolean algebra
- explain Karnaugh maps
- explain Quine-McCluskey method

3.2 Boolean Algebra

3.2.1 Boolean Expressions and Functions

Boolean variable is a variable which can have either 0 or 1 as its value. A **Boolean function** is an expression formed with binary variables, the two binary operators AND and OR, one unary operator NOT, parentheses and equal sign.Truth tables are used to display the values of a given Boolean function. A **Boolean expression** is an expression using Boolean variables {X1, X2,….Xn} and the operations of a Boolean algebra.

Boolean functions in digital systems are defined by Boolean expression. The Boolean function value is evaluated by applying 0’s and 1’s to the variables in the Boolean expression. For example, the Boolean function F(A,B) is defined by the Boolean expression A’B’ + AB’ i.e., F(A,B) = A’B’ + AB’. The truth table for the function F(A,B) is given in the table 3.1.

Table 3.1: Truth table for F(A,B) = A’B’ + AB’

A	B	A'B'	AB'	F (A, B)
0	0	1	0	1
0	1	0	0	0
1	0	0	1	1
1	1	0	0	0

The domain in this function is the 2-tuple which represents the values of x and y. The range is {0, 1} in the last column. The possible value of the variables in the Boolean expression represents the n-tuple of a Boolean function. If two or more Boolean expressions represent the same function, then those Boolean expressions are said to be equivalent. (i.e., have the same truth table).

3.2.2 More on Boolean Functions

In order to use the above information to construct the digital circuits, the following basic problems need to solve:

1) how the Boolean expression is derived from the given truth table for a Boolean function?
2) Can the Boolean function be represented with an optimum set of operators?

First, question 1. How the Boolean expression can be derived from a given table (for example table 3.2) of values for a Boolean function?

Table 3.2: Deriving Boolean expression

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

From the above table 3.2, the function F is 1 when y = z = 1 and x =0. And hence we get the expression x’yz i.e., the value of the function F will be 1 if and only if x’ = y = z = 1. The function G will have value 1 in two cases: y = z = 1, x =0, and x = y = 0, z = 1. From the above information the expression for G can be represented in the sum of two product terms: x’yz + x’y’z

The above method explains the way to construct a Boolean expression from the values of a Boolean function. A Boolean product m1, m2, m3…mn is said to be a **minterm** of Boolean variables x1, x2, x3,…xn if mi = xi or mi = xi’. Only if all the variables of minterm are 1, the minterm product will result in 1. For the 1st scenario discussed above, x’yz will only be the minterm that has the value 1. In other words, a Boolean expression for a Boolean function can be quoted as the sum of the minterms. Minterms referred here are those with value 1 for a given combination.

Boolean sum discussed above is also referred to as sum of products (SOP) expansion or disjunctive normal form. **Maxterm** is the dual of minterm. It is also referred to as product of sums (POS) or conjunctive normal form. Duality can be observed from the table 3.3 below:

Table 3.3: Duality

x	y	z	minterm	maxterm
0	0	0	a'b'c'	a+b+c
0	0	1	a'b'c	a+b+c'
0	1	0	a'bc'	a+b'+c
0	1	1	a'bc	a+b'+c'
1	0	0	ab'c'	a'+b+c
1	0	1	ab'c	a'+b+c'
1	1	0	abc'	a'+b'+c
1	1	1	abc	a'+b'+c'

3.2.3 Functional Completeness

The other question posed in the previous section was: In order to express a Boolean, is it required to all the three operators? From the above discussion, using the operators {' + •} it is easy to construct any Boolean function. As these three operators used to express the every Boolean function, set of three operators is considered as **functionally complete**. Is there a smaller set of functionally complete operators? If one of three operators in a set can be expressed in terms of other two operators, then a smaller set which is functionally complete can be determined. One way to determine the smaller set is by using DeMorgan's law. For example:

x + y = (x'y')'

From the above expression we can observe that the + operator has been expressed in terms of {' •}. So, the set of {' •} is functionally complete. In a similar way Boolean products can be eliminated:

xy = (x' + y')'

Is the set {+ •} functionally complete? No, there is no way to represent the Boolean function F(x) = x' using these two operators, so this set is not functionally complete.

Finally, is there a set of one operator that is functionally complete? Only if we define a new operator. We could define a **NAND** operator "|" as shown in the table 3.4.

Table 3.4: NAND operator

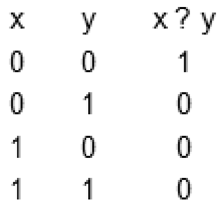
x	y	x | y
0	0	1
0	1	1
1	0	1
1	1	0

This set is functionally complete. In order to prove that operator | is functionally complete, the operators {• '} has to be expressed using |:

x' = x | x
xy = (x | y) | (x | y)

A similar operator that is functionally complete is the **NOR** operator. This is shown in table 3.5.

Table 3.5: NOR operator



x	y	x ? y
0	0	1
0	1	0
1	0	0
1	1	0

3.2.4 Simplification of Boolean Functions

From the above discussion, it can be observed that any Boolean function can be constructed using NOTs, ANDs, and ORs using minterm expansion. As minterm expansion requires more number of gates than necessary, a design engineer will rarely satisfied with it. A minterm expansion can be simplified using the identities and laws of Boolean algebra. For example, the minterm expansion for a Boolean function f of three variables might be represented as follows.

$f = x'y'z' + x'y'z + x'yz' + x'yz + xyz' + xyz$

The maximum gates required to implement above Boolean function are: 12 AND gates, 5 OR gates and 9 NOT gates. Above minterm expansion can be reduced by using the laws and identities of Boolean algebra:

$f = x'y'z' + x'y'z + x'yz' + x'yz + xyz' + xyz$

$= x'y'(z' + z) + x'y(z' + z) + xy(z' + z)$ distributive law

$= x'y' + x'y + xy$ complementarity & identity

$= x'(y' + y) + xy$ distributive law

$= x' + xy$ complementarity & identity

$= x' + y$ redundancy

From the above example, it can be observed that the long minterm has been reduced down to expression x'+y, which can construct using single NOT gate and OR gate. But to reduce the given minterm expansion, Engineer should be able to choose the appropriate laws and identities of Boolean algebra and apply them in the right step. Therefore, we will look at a very simple technique that usually leads to a significant simplification of minterms. It won't always produce the simplest form, but it's close enough for most engineers considering the difficulty of the alternative method.

3.3 Karnaugh Maps

Maurice Karnaugh a telecommunication engineer has invented Karnaugh Maps. In 1953, the Karnaugh map has been developed in Bell Labs while studying the application of digital logic to design of telephone circuits. This method is also known as K-maps. A Karnaugh map (K-map) is a visual representation of a Boolean functions. The plan is to recognize patterns in the visual representation and thus find a minimized circuit for the Boolean function K-maps are generally used in simplification of two, three or four variables Boolean function. For the Boolean functions with 5 variables or more, K-maps get cumbersome and other techniques like Tabular Method is used. A Karnaugh map is a 2-dimensional representation of the truth table for a Boolean function. For example, consider the table 3.6.

Table 3.6: Truth Table for the function f (x,y,z)

x	y	z	f(x,y,z)
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

The figure 3.1 shows the template for 3 variable Karnaugh maps

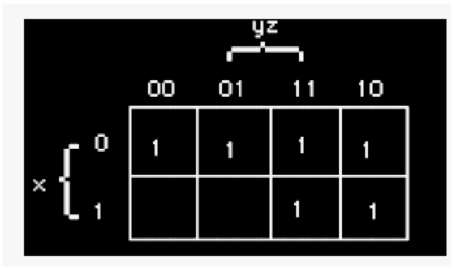
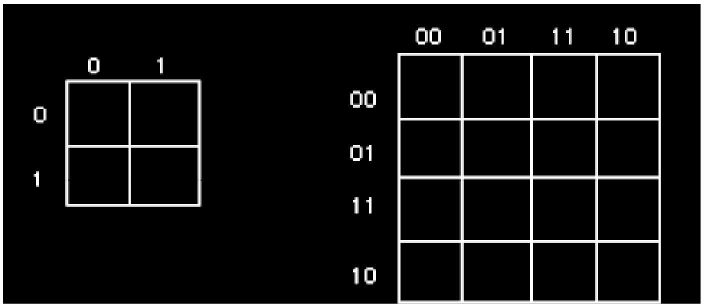


Figure 3.1: Karnaugh map for three variables

For three variables, there are eight possible combinations and each cell in K-map corresponds to one of the combination. The templates for 2 and 4 variable Karnaugh maps are given in figure 3.2(a) and 3.2(b) respectively.



(a) (b)

Figure 3.2: (a) Karnaugh map for 2 variables (b) Karnaugh map for 4 variables

Once we have placed the 1's in the map, there is a simple procedure that we apply.

Before analyzing the procedure, understanding the basis for the procedure is necessary. As there are many simple functions in Boolean functions, K-maps are useful. The simple functions in Boolean function are known as **product function**; product functions can be product of few variables or all variables. The product terms can have both normal variable and its complements. For example, A, A'B and ABC are all product functions but A + B' and AB + CD are not product functions.

Consider the Boolean function f(A,B,C) = AC. For inputs equal to 101 and 111, the value of the Boolean function will be 1. Its Karnaugh map is shown in figure 3.3.

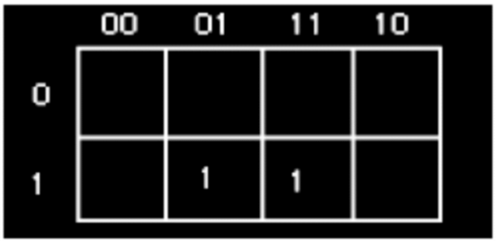


Figure 3.3: Karnaugh map for f(A,B,C)=AC

The minterms of the Boolean function lie in 1x2 rectangular block. Consider another Boolean function g(A,B,C) = B'. For input values equal to 000, 001, 100, 101, the Boolean function g will be 1. Its Karnaugh map is shown in figure 3.4.

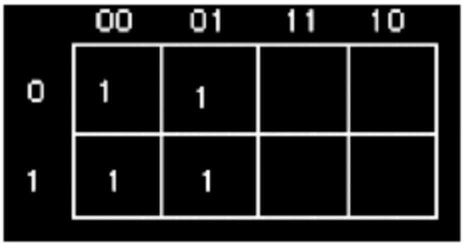


Figure 3.4: Karnaugh map for g(A,B,C)=B'

Notice that 1’s lie in a 2x2 rectangular block. From the above example, it can be observed that for every product function there will be corresponding K-map whose minterms can lie in a block of 1, 2, or 4 cells long. Thus, recognizing the product function which represented by a K-map has highest priority.

In order to recognize the product function represented by k-map, the truth set of the Boolean functions is written down. Truth set is the set of combinations for which Boolean function will be 1. For example consider the k-map shown in figure 3.5.

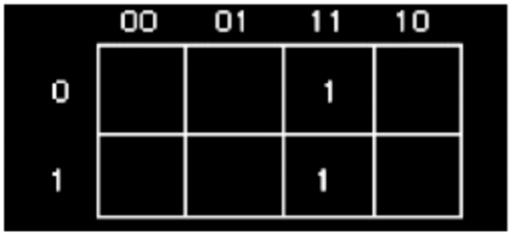


Figure 3.5: k-map for Boolean function with output 1

From the above k-map, it can be observed that values 011 and 111 will form the truth set. Next, the truth set values are analyzed. If the variables of the Boolean function are A, B, and C, it can be observed that value of A can be either 0 or 1; while the values of B and C has to be 1. The above analysis can be characterized as follows: {*11}. A particular value for x will only be accepted if A is involved in unknown product function. From the above truth set it can be observed that value of A can be either 0 or 1, thus it can be concluded that A is not involved. From the truth set it is observed that value of B is 1 for all inputs, thus B must be involved but not B’. The same is applicable for C. From the above discussion, we can say that BC is the product function. For the K-maps given in figure 3.6, derive the product functions associated with it.

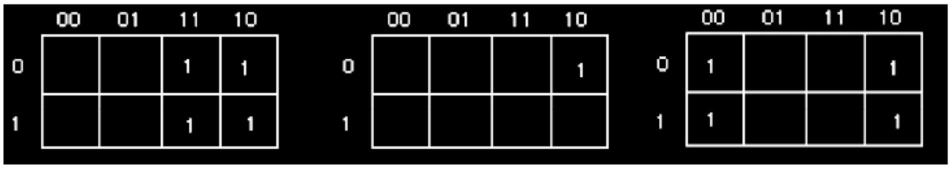


Figure 3.6: k-map Examples

1) Truth set for the first k-map is: {011, 010, 111, 110} = {*1*}. It can be observed that B should be involved in the product function, but A and C are not necessary to be involved. Thus the product function is B.
2) Truth set for the second k-map is: {010} as there is only one value in the truth set, the Boolean function for the k-map will be A'BC'.
3) Truth set for the third k-map is: {000, 100, 010, 110} = {**0} = z'. It can be observed that complement of C should be involved in the product function, but A and B are not necessary to be involved. From the k-map it can be noticed that there is a wraparound of rectangular block.

Finally, try some 2 and 4 variable maps as shown in figure 3.7.

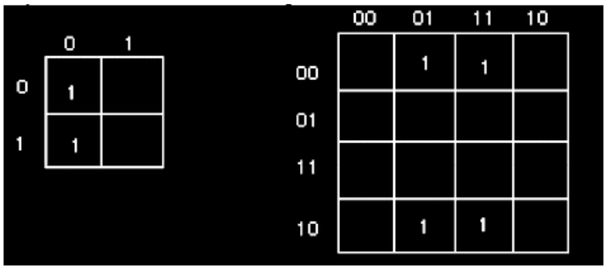


Figure 3.7: 2 and 4 variable k-maps

1) Truth set = {00, 10} = {*0} = y'
2) (variables are wxyz): truth set = {0001, 0011, 1001, 1011} = {*0*1} = x'z.

Using K-map, the same expression can be used to represent any Boolean function. Another expression with the same Karnaugh map as the two-variable one above: x'y' + xy' + x'yxy; therefore, we can represent this as y' and build a circuit as such.

But sometimes the Boolean expressions will not be equivalent to the k-maps product function. The first example we looked at did not break down into a 1, 2, or 4 cell rectangular blocks. Now consider the figure 3.8.

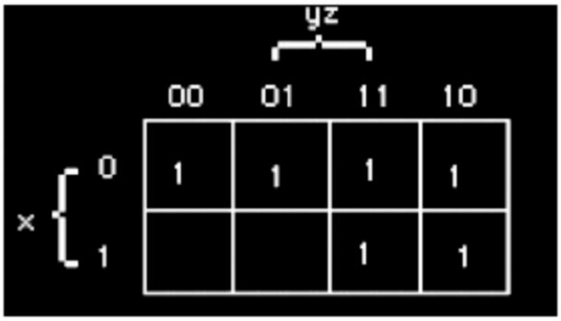


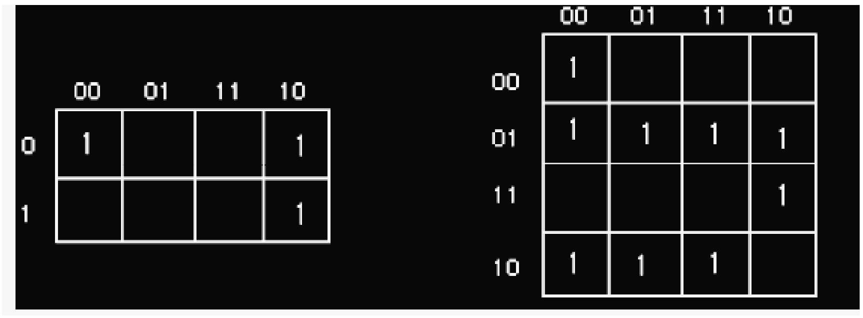
Figure 3.8.: K-maps for 3 variables with "x" representation

Blocks in the above k-map can be broken down into two separate product blocks. x' is represented by one product block and the other product represents y. The Boolean function which has been derived from the k-map is x' + y, in the present method, the two product functions are determined and ORed together. It can be observed that the blocks has been determined directly without using any truth sets and same can be applied to any Boolean function. The main idea is to use only minimum blocks to cover the terms and the Boolean function is derived by summing all the product blocks. The simple Boolean expression can be derived when the large sized blocks are used. For example, in the k-map above, one horizontal and two vertical blocks could be used: the first two ones form one horizontal block, ones in column 3 forms first vertical block, and ones in column 4 forms the other vertical block. By using above method the following Boolean expression is derived: x'y'+yz+yz'.

In the above k-map, if the first two ones form a block and the four ones forms another block, then the Boolean expression cab be reduced to x'y'+y. But the Boolean expression derived using second method is not that simple as the Boolean expression which is derived using the first method. By using Boolean identities and theorems, the Boolean expression x'y'+y can be further reduced to x'y. From the above discussion, it can be observed that a simple Boolean expression can be derived by choosing the blocks carefully. An **implicant** of a function is a product term that is included in the function.

In the Boolean expression above implicants are x'y', yz and yz'. A **prime implicant** of a function is an implicant of the function that is not included in any other implicant of the function. Therefore, only few prime implicants should be used to cover the minterms of the Boolean function.

As additional practice, simplify the following functions represented by Karnaugh maps shown in figure 3.9.

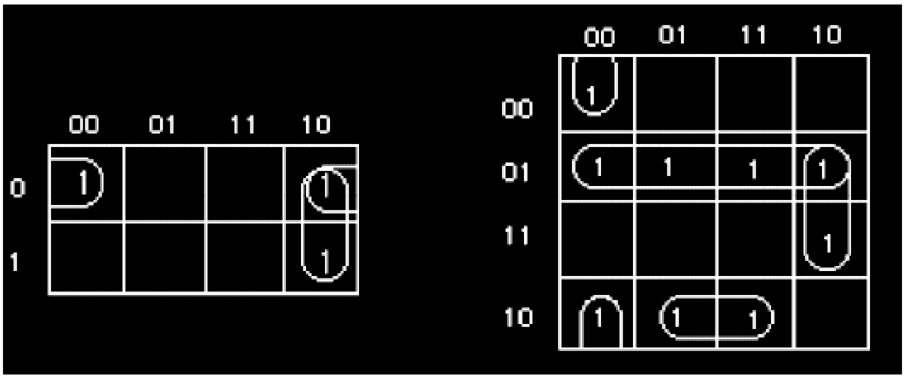


(a) **(b)**

Figure 3.9: K-maps for (a) 3 variable and (b) 4 variables

1) x'z' + yz'
2) w'x + xyz' + wx'z x'y'z'

Notice that you can overlap the blocks if necessary as shown in figure 3.10.



(a) **(b)**

Figure 3.10: Over lapping in K-maps for (a) 3 variable and (b) 4 variables

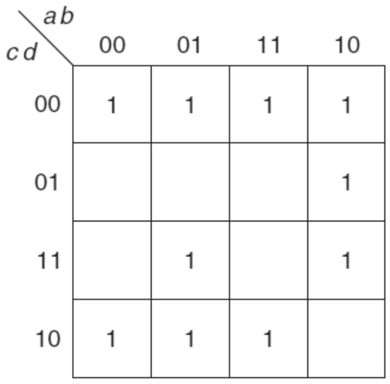
Instead of using identities for simplification of Boolean functions, it can be done easily by using k-maps. The main important thing is to determine the simple Boolean expression, a right set of blocks need to be chosen.

There are no rules to choose the right set of blocks to determine the simple expression, it comes with practice.

Now let us see some examples

Example 1: Simplify f (a, b, c, d)=∑m(0, 2, 4, 6, 7, 8, 9, 11, 12, 14).

Solution: Write the karnaugh Map and enter 1's as shown in figure 3.11 in the corresponding cells.



cd \ ab	00	01	11	10
00	1	1	1	1
01				1
11		1		1
10	1	1	1	

Figure 3.11: Karnaugh Map for example 1.

Now group the 1's staring from groups of eight 1's, then four 1's, two 1's as shown in figure 3.12.

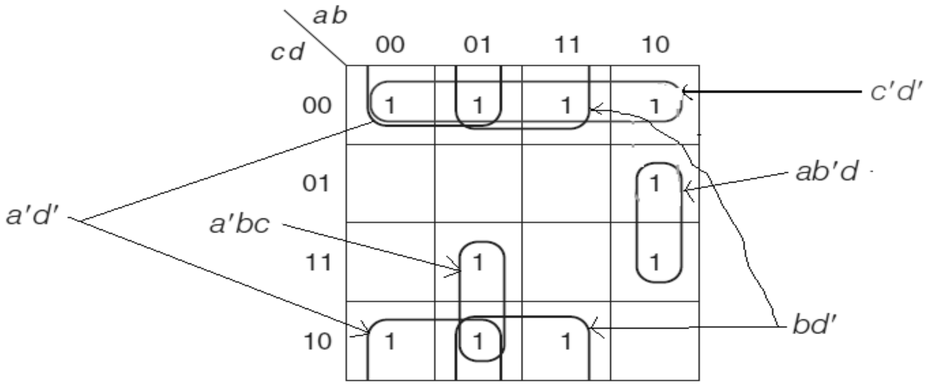


Figure 3.12: Karnaugh Map simplification for example 1.

Finally we get,

$f = a^{l}d^{l} + bd^{l} + a^{l}bc + ab^{l}d + c^{l}d^{l}$

Example 2: Simplify the following expression using K-Map.

$x'yz' + x'yz + xy'z' + xy'z + xyz$

Solution: The K-map and its simplification is shown in figure 3.13.

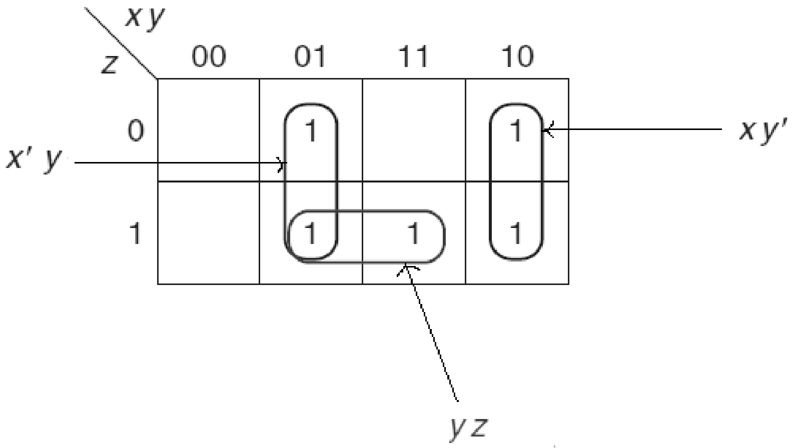


Figure 3.13: Karnaugh Map for example 2.

Therefore,

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

3.4 Quine-McCluskey Method

When a Boolean expression contains more than six variables, then it becomes very difficult to solve the expression using Karnaugh map method. In such cases the Quine–McCluskey method can be used to solve these types of Boolean equations. This method was developed by W.V. Quine and Edward J. McCluskey. This method is sometimes referred to as the method of prime implicants or the tabulation method.

The following steps are followed to simplify the given Boolean expression using Quine-McCluskey method.

Step 1: If the function is not given in minterm form, then translate the decimal values to minterms.

F(A,B,C,D) = ∑m(0, 2, 3, 6, 7, 8, 9, 10, 13)

Step 2: If the function is not given in binary minterm form, then translate the decimal values to binary notation of minterms.
F(A,B,C,D) = ∑m(0000, 0010, 0011, 0110, 0111, 1000, 1001, 1010, 1101)

Step 3: Minterms are grouped depending upon number of one's they have and entered in table form as shown in the table 3.7.

Table 3.7: Quine-McCluskey Method

1's	m	minterms	1-Cube	2-Cube	3-Cube
0	m0	0000			
1	m2	0010			
	m8	1000			
2	m3	0011			
	m6	0110			
	m9	1001			
	m10	1010			
3	m7	0111			
	m13	1101			

Step 4: The minterms in the adjacent blocks are compared to determine the minterms which are differed by only one bit. Replace the missing literal by – and place the minterms in the next column. The minterms in the present column which are combined are placed with a check mark (refer to the table 3.8).

Table 3.8: Quine-McCluskey Method for Step 4

1's	m	minterms	1-Cube	2-Cube	3-Cube
0	m0	0000 √	00-0		
1	m2	0010 √	-000		
	m8	1000 √	001-		
2	m3	0011 √	0-10		
	m6	0110 √	-010		
	m9	1001 √	100-		
	m10	1010 √	10-0		
3	m7	0111 √	0-11		
	m13	1101 √	011-		
			1-01		

Step 5: The minterms in the adjacent blocks are compared to determine the minterms which are differed by only one bit. Note -'s must line up. Replace the missing literal by – and place the minterms in the next column. The minterms in the present column which are combined are placed with a check mark. Table 3.9 shows completion of first phase of Q-M method.

Table 3.9: Quine-McCluskey Method for Step 5

1's	m	minterms	1-Cube	2-Cube	3-Cube
0	m0	0000 √	00-0 √	-0-0	NULL
1	m2	0010 √	-000 √	0-1-	
	m8	1000 √	001- √		
2	m3	0011 √	0-10 √		
	m6	0110 √	-010 √		
	m9	1001 √	100-		
	m10	1010 √	10-0 √		
3	m7	0111 √	0-11 √		
	m13	1101 √	011- √		
			1-01		

Step 6: The prime implicant chart is to be formed with minterms in a row and prime implicants in column. When there is intersection of minterm and the prime implicant, the √ placed (refer the table 3.10).

Table 3.10: Quine-McCluskey Method for Step 6

Minterms		0000	0010	0011	0110	0111	1000	1001	1010	1101
8,9	100-						√	√		
9, 13	1-01							√		√
0,2,8,10	-0-0	√	√				√		√	
2,3,6,7	0-1-		√	√	√	√				

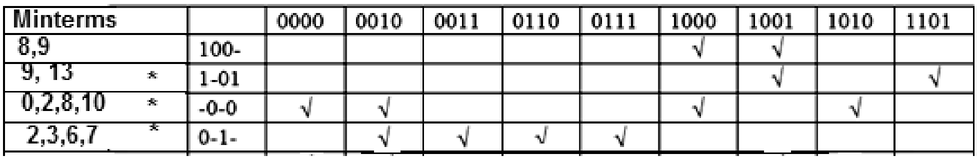
Step 7: Select prime implicants for minterms with only one √ in a column. (refer the table 3.11).

Table 3.11: Quine-McCluskey Method for Step 7

Minterms		0000	0010	0011	0110	0111	1000	1001	1010	1101
8,9	100-						√	√		
9, 13 *	1-01							√		√
0,2,8,10 *	-0-0	√	√				√		√	
2,3,6,7 *	0-1-		√	√	√	√				

Step 8: Repeat step 8 for minterms with only two √ in a column. (refer the table 3.12). If the minterms of those colums are already included, then not required to be considered (refer the table 3.12).

Table 3.12: Quine-McCluskey Method for Step 8



Minterms		0000	0010	0011	0110	0111	1000	1001	1010	1101
8,9	100-						√	√		
9, 13 *	1-01							√		√
0,2,8,10 *	-0-0	√	√				√		√	
2,3,6,7 *	0-1-		√	√	√	√				

In table 3.12, two √ correspond to colums 0010, 1000 and 1001. These can be dropped since corresponding minterms are already included.

Now the final expression is
F(A,B,C,D) = 1-01 + -0-0 + 0-1-

Step 9: Translate to literal notation
F(A,B,C,D) = AC'D + B'D' + A'C

Don't Cares

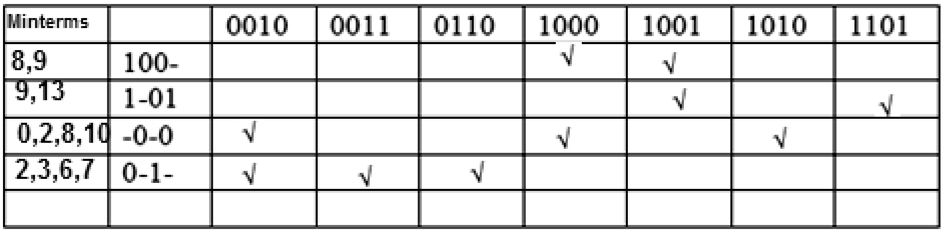
It is important to note that don't cares are used to find out prime implicant but not compulsory to include in the final expression.

Step A: Translate to canonical minterm representation
G(A,B,C,D) = Σm(2,3,6,8,9,10,13) +Σd(0,7)

Step B: Form table and find prime implicants. Include minterms and don't cares in table. Since G(A,B,C,D) is just F(A,B,C,D) with mintems 0 and 7 changed to don't cares; this process would be identical to steps 3, through 5 from above. See steps 3 through 5 from above.

Step C: Form table with don't cares missing. Only include required minterms. Note 0000 and 0111 columns missing. Check required minterms covered by each prime implicant (refer the table 3.13).

Table 3.13: Step C



Minterms		0010	0011	0110	1000	1001	1010	1101
8,9	100-				√	√		
9,13	1-01					√		√
0,2,8,10	-0-0	√			√		√	
2,3,6,7	0-1-	√	√	√				

Step D: Select column in which a specific minterm is only covered by one prime implicant (refer the table 3.14).

Table 3.14: Step D

Minterms		0010	0011	0110	1000	1001	1010	1101
8,9	100-				√	√		
9,13 *	1-01					√		√
*0,2,8,10	-0-0	√			√		√	
2,3,6,7*	0-1-	√	√	√				

Step E: Extract essential prime implicants (refer the table 3.15).

Table 3.15: Step E

		0010	0011	0110	1000	1001	1010	1101
	100-				√	√		
*	1-01					√		√
*	-0-0	√			√		√	
*	0-1-	√	√	√				

Step F: Translate back to literals.

G(A,B,C,D) = 1-01 + -0-0 + 0-1-

G((A,B,C,D) = AC’D + B’D’ +A’C

3.5 Exercises

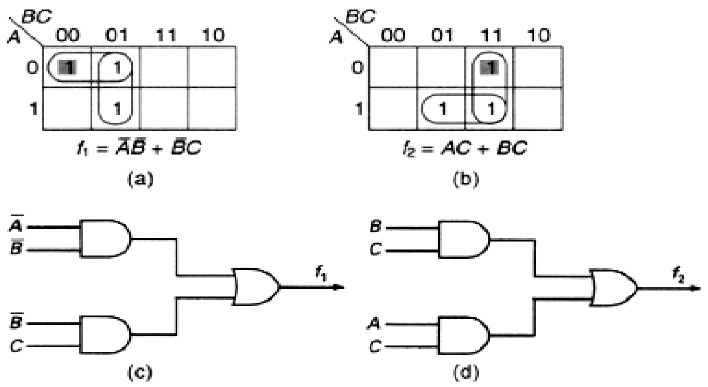
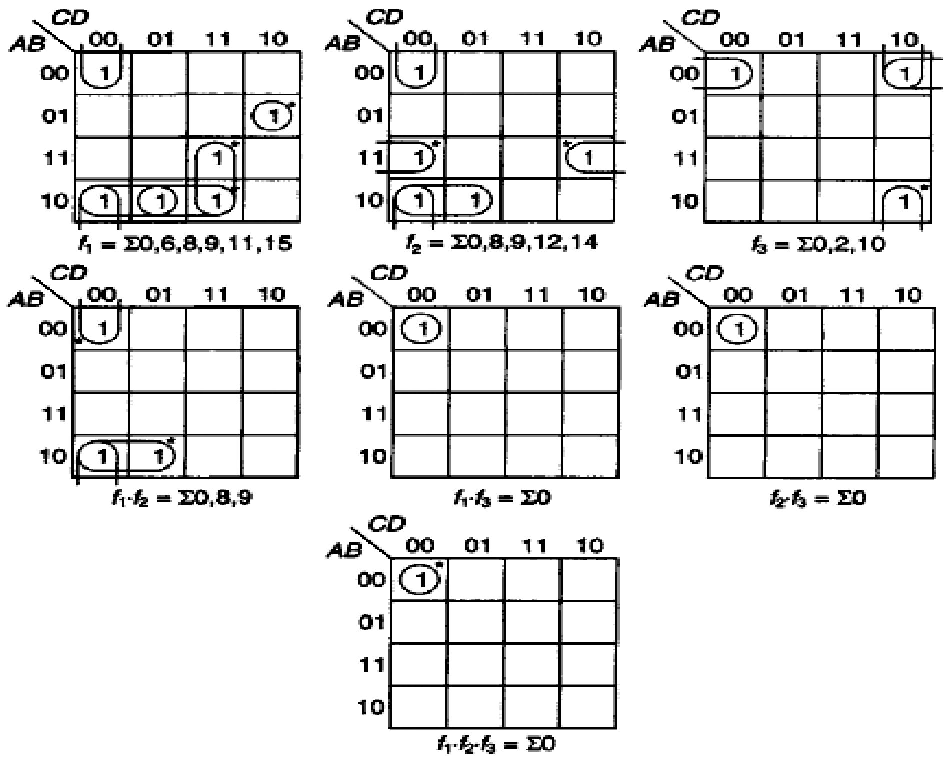


Figure 3.14: (c) and (d) independent implementation of f1 and f2 in (a) and (b)

Function and product maps for determining multiple output circuits



CD
AB
00 01 11 10
$f_1 = \Sigma 0,6,8,9,11,15$
$f_2 = \Sigma 0,8,9,12,14$
$f_3 = \Sigma 0,2,10$
$f_1 \cdot f_2 = \Sigma 0,8,9$
$f_1 \cdot f_3 = \Sigma 0$
$f_2 \cdot f_3 = \Sigma 0$
$f_1 \cdot f_2 \cdot f_3 = \Sigma 0$

Figure 3.15: K-maps for determining multiple output circuits

Self Assessment Questions

1. If x and y is 0 and 1, the Boolean function xy+xy' result is __________.
2. A minterm has a value ___ if and only if all the values of its variables are 1.
3. If x, y, z are 0, 0, 1 the minterm is _______ and maxterm is_________.
4. A Karnaugh map is a _______ representation of the truth table for a Boolean function.
5. Implicants that cover as many cells of the map as possible are called __________.
6. A prime implicant of a function is an implicant of the function that is not included in any other implicant of the function (True or false?).

3.6 Summary

Let us recapitulate the important concepts discussed in this unit.

- A Boolean function is an expression formed with binary variables, the two binary operators AND and OR, one unary operator NOT, parentheses and equal sign.
- A Boolean product m1, m2, m3…mn is said to be a **minterm** of Boolean variables x1, x2, x3,…xn if mi = xi or mi = xi'.
- K-maps are generally used in simplification of two, three or four variables Boolean function.
- A **prime implicant** of a function is an implicant of the function that is not included in any other implicant of the function.
- Quine-McCluskey method is sometimes referred to as the method of prime implicants or the tabulation method.

3.7 Terminal Questions

1. Simplify the expression algebraically x'y'z' + x'y'z + x'yz' + x'yz + xyz' + xyz
2. What is a Karnaugh map? Explain.
3. Simplify the following three-variable Boolean functions using Karnaugh map

   (a) $f_1 = \sum 1, 2, 5, 6$

   (b) $f_2 = \sum 0, 1, 2, 3, 7$
4. Find the minimized sum-of-products expressions using Karnaugh map

   (a) $f_1(A, B, C) = \sum 0, 1, 3, 4, 6, 7$

   (b) $f_2(A, B, C, D) = \sum 0, 1, 2, 3, 7, 8, 9, 11, 12, 15$
5. Minimize the following functions using the Quine-McCluskey tabular method:

   (a) $f(A, B, C, D) = \sum 0, 1, 3, 6, 9, 10, 11, 12, 14, 15$

   (b) $f(A, B, C, D, E) = \sum 0, 1, 5, 8, 11, 12, 14, 16, 20, 21, 25, 27, 28, 30, 31$

   with 'don't care' terms 2, 7, 13, 22, 23

3.8 Answers

Self Assessment Questions

1. 0
2. 1
3. Minterm → $x' y' z$ Maxterm → $x+y+z'$
4. 2-dimensional
5. Prime implicants
6. True

Terminal Questions

1. Refer to the sub-section 3.2.4
2. Refer to the sub-section 3.3
3. Refer to the section 3.3 for method
4. Refer to the section 3.3 for method
5. Refer to the section 3.4 for method