Chapter 3: Rules and Law of Boolean algebra

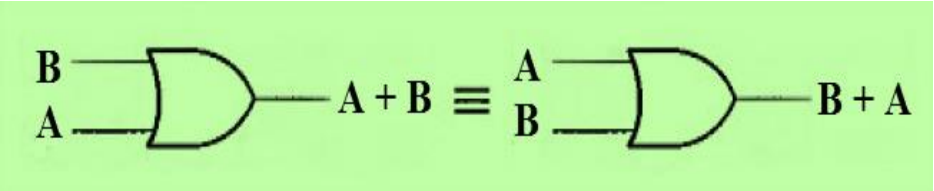
3.1 Law of Boolean Algebra

Three basic laws of Boolean algebra are: the commutative laws, the associative laws and the distributive law.

3.1.1 Commutative Laws

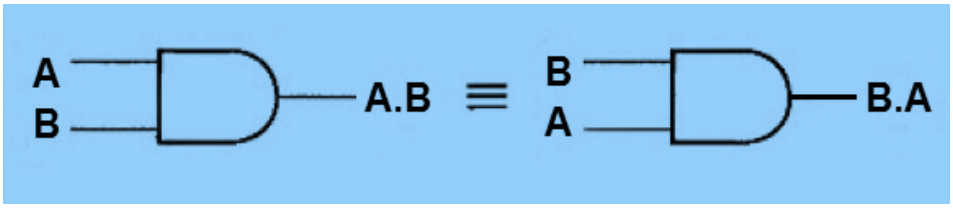
Commutative laws for **addition**

$$A + B = B + A$$



Commutative laws for **multiplication**

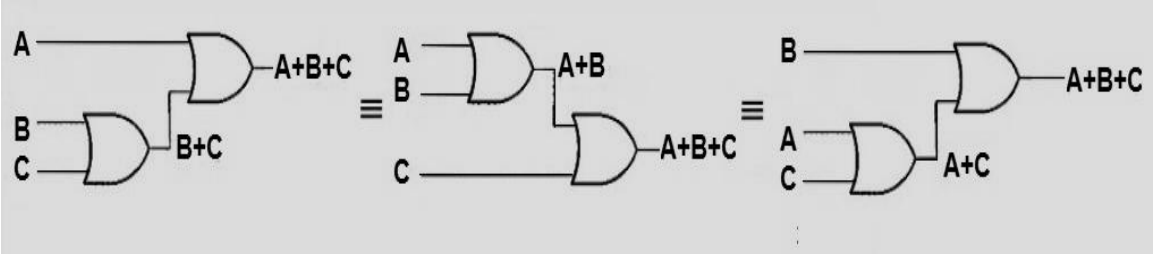
$$AB = BA$$



3.1.2 Associative Laws

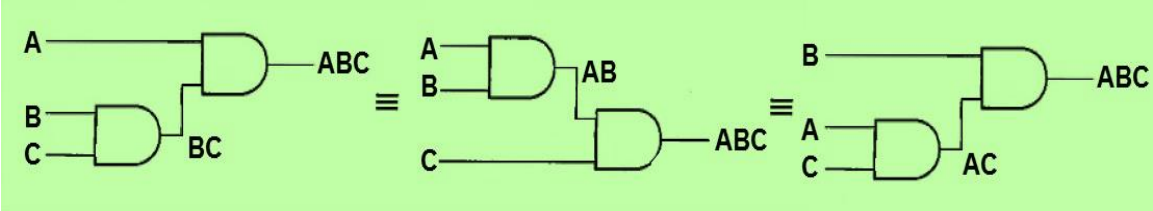
The associative law of **addition** is stated as follows for three variables

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



The associative law of **multiplication** is stated as follows for three variables

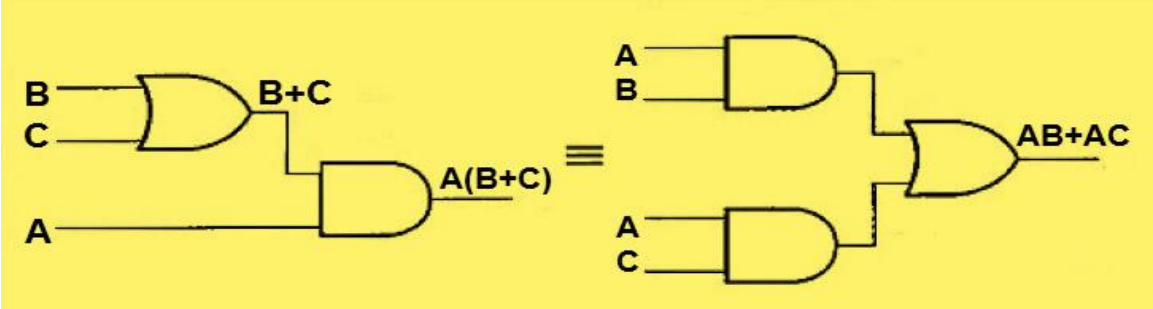
$$A\,(BC) = (AB)\,C = (AC)\,B$$



3.1.3 Distributive Laws

The distributive law is written for three variables as follows

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



3.2 Rules of Boolean Algebra

Table 5.1 lists 12 basic rules that are useful in manipulating and simplifying Boolean expressions. Rules 1 through 9 will be viewed in terms of their application to logic gates. Rules 10 through 12 will be derived in terms of the simpler rules and the laws previously discussed.

Table 5.1 Basic rules of Boolean algebra.

1. $A + 0 = A$	2. $A + 1 = 1$
3. $A.0 = 0$	4. $A.1 = A$
5. $A + A = A$	6. $A + \overline{A} = 1$
7. $A.A = A$	8. $A.\overline{A} = 0$
9. $\overline{\overline{A}} = A$	10. $A + AB = A$
11. $A + \overline{A}B = A + B$	12. $(A + B)(A + C) = A + BC$

Rule 1. $A + 0 = A$

A variable OR with 0 is always equal to the variable. If the input variable A is 1, the output variable X is 1, which is equal to A. If A is 0, the output is 0, which is also equal to A. This rule is illustrated in Fig. (5.1), where the lower input is fixed at 0.

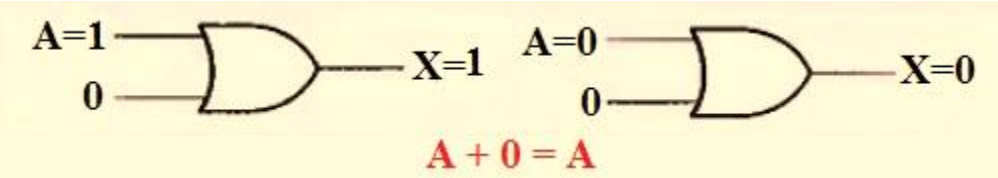


Fig. (5.1).

Rule 2. $A + 1 = 1$

A variable OR with 1 is always equal to 1. A 1 on an input to an OR gate produces a 1 on the output, regardless of the value of the variable on the other input. This rule is illustrated in Fig. (5.2), where the lower input is fixed at 1.

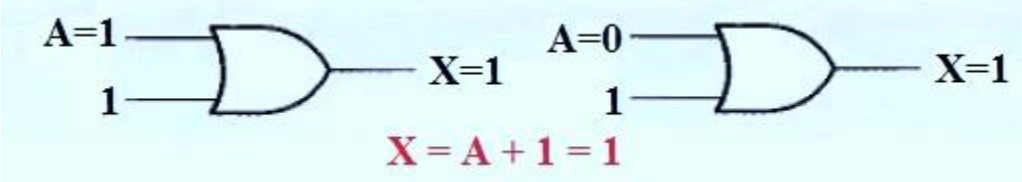


Fig. (5.2).

Rule 3. $A.0 = 0$

A variable AND with 0 is always equal to 0. Any time one input to an AND gate is 0, the output is 0, regardless of the value of the variable on the other input. This rule is illustrated in Fig. (5.3), where the lower input is fixed at 0.

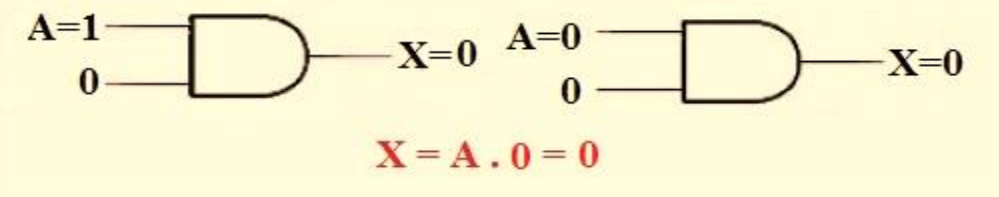


Fig. (5.3).

Rule 4. $A.1 = A$

A variable AND with 1 is always equal to the variable. If A is 0 the output of the AND gate is 0. If A is 1, the output of the AND gate is 1 because both inputs are now 1s. This rule is shown in Fig. (5.4), where the lower input is fixed at 1.

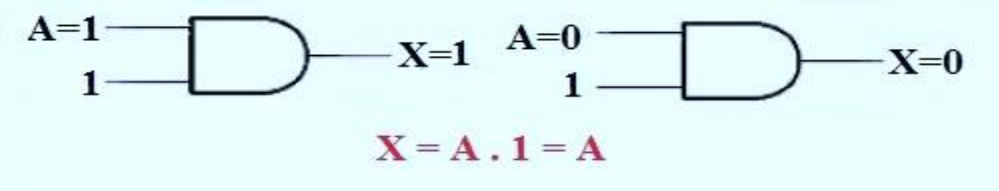


Fig. (5.4).

Rule 5. $A + A = A$

A variable OR with itself is always equal to the variable. If A is 0, then 0 + 0 = 0; and if A is 1, then 1 + 1 = 1. This is shown in Fig. (5.5), where both inputs are the same variable.

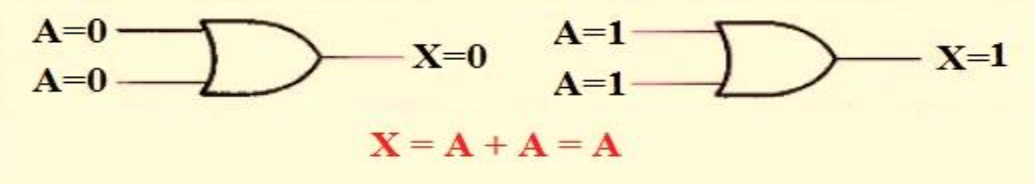


Fig. (5.5).

Rule 6. $A + \bar{A} = 1$

A variable OR with its complement is always equal to 1. If A is 0, then $0 + \bar{0} = 0 + 1 = 1$. If A is l, then $1 + \bar{1} = 1 + 0 = 1$. See Fig. (5.6), where one input is the complement of the other.

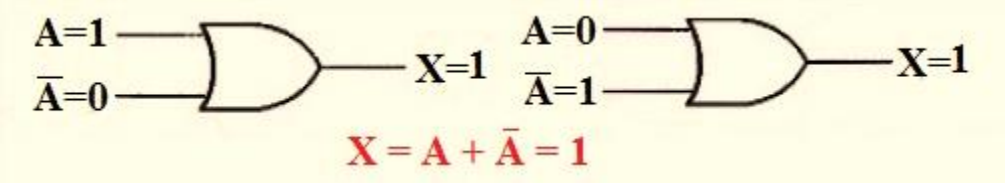


Fig. (5.6).

Rule 7. $A.A = A$

A variable AND with itself is always equal to the variable. If A = 0, then 0.0 = 0; and if A = 1. then 1.1 = 1. Fig. (5.7) illustrates this rule.

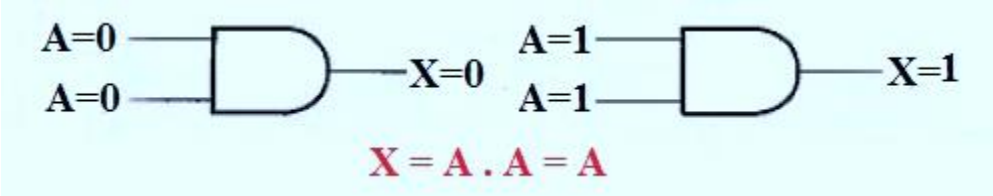


Fig. (5.7).

Rule 8. $A.\overline{A} = 0$

A variable AND with its complement is always equal to 0. Either A or $\overline{A}$ will always be 0: and when a 0 is applied to the input of an AND gate. The output will be 0 also. Fig. (5.8) illustrates this rule.

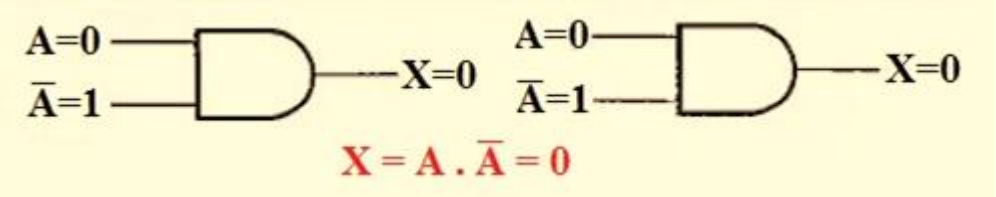


Fig. (5.8).

Rule 9. $\overline{\overline{A}} = A$

The double complement of a variable is always equal to the variable. If you start with the variable A and complement (invert) it once, you get $\overline{A}$. If you then take $\overline{A}$ and complement (invert) it, you get A, which is the original variable. This rule is shown in Fig. (5.9) using inverters.

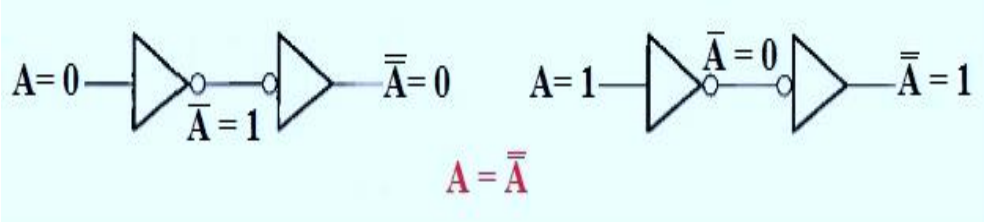


Fig. (5.9).

Rule 10. $A + AB = A$

This rule can be proved by applying the distributive law, rule 2, and rule 4 as follows:

$A + AB = A(1 + B)$ Factoring (distributive law)

$= A.1$ Rule 2: (1 + B) = 1

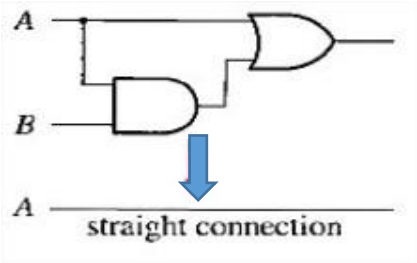
$= A$ Rule 4: A. 1 = A

The proof is shown in Table 5.2, which shows the truth table and the resulting logic circuit simplification.

Table 5.2

A	B	AB	A + AB
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

equal



Rule 11. $A + \bar{A}B = A + B$

This rule can be proved as follows:

$A + \bar{A}B = (A + AB) + \bar{A}B$ Rule 10: $A = A + AB$

$= A + AB + \bar{A}B$ Rule 7: $A = AA$

$= A + B(A + \bar{A})$ Rule 6: $(A + \bar{A}) = 1$

$= A + B.1$ Rule 4: $B.1 = B$

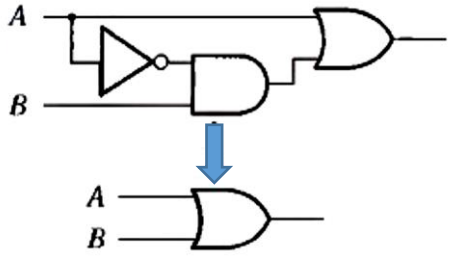
$= A + B$

The proof is shown in Table 5.3, which shows the truth table and the resulting logic circuit simplification.

Table (5.3).

A	B	$\bar{A}B$	$A + \bar{A}B$	A + B
0	0	0	0	0
0	1	1	1	1
1	0	0	1	1
1	1	0	1	1

equal



Rule 12. $(A + B)(A + C) = A + BC$

This rule can be proved as follows:

$(A + B)(A + C) = AA + AC + AB + BC$ Distributive law

$= A + AC + AB + BC$ Rule 7: $AA = A$

$= A(1 + C) + AB + BC$ Rule 2: $1 + C = 1$

$= A.1 + AB + BC$ Factoring (distributive law)

$= A(1 + B) + BC$ Rule 2: $1 + B = 1$

$= A.1 + BC$ Rule 4: $A.1 = A$

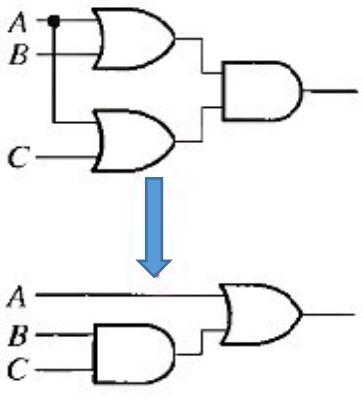
$= A + BC$

The proof is shown in Table 5.4, which shows the truth table and the resulting logic circuit simplification.

Table (5.4).

A	B	C	A+B	A+C	(A+B)(A+C)	BC	A+BC
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	1	1	1	1
1	0	0	1	1	1	0	1
1	0	1	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

equal



3.3 De Morgan's Theorems

De Morgan, proposed two theorems that are an important part of Boolean algebra. The two theorems show in equation below:

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

$$\overline{A + B} = \bar{A}\bar{B} \ldots\ldots\ldots\ldots 2$$

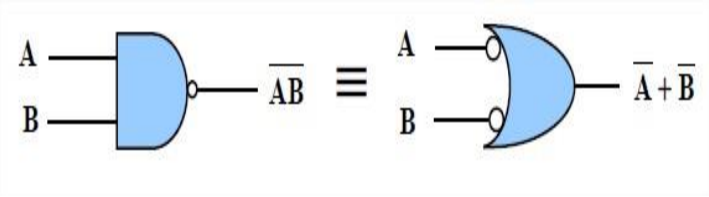
Eq1: The complement of two or more AND variables is equivalent to the OR of the complements of the individual variables.

Eq2: The complement of two or more OR variables is equivalent to the AND of the complements of the individual variables.

$\overline{AB} = \overline{A} + \overline{B}$

A	B	$\overline{AB}$	$\overline{A} + \overline{B}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

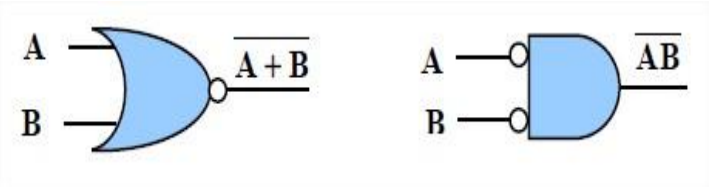
equal



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

A	B	$\overline{AB}$	$\overline{A} + \overline{B}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

equal



Example: Apply De Morgan's theorems to the expressions $\overline{XYZ}$, $\overline{X+Y+Z}$, $\overline{WXYZ}$ and $\overline{W+X+Y+Z}$

Solution:

$\overline{XYZ} = \bar{X} + \bar{Y} + \bar{Z}$

$\overline{X + Y + Z} = \bar{X}\,\bar{Y}\,\bar{Z}$

$\overline{WXYZ} = \overline{W} + \bar{X} + \bar{Y} + \bar{Z}$

$\overline{W + X + Y + Z} = \overline{W}\,\bar{X}\,\bar{Y}\,\bar{Z}$

Example: Apply De Morgan's theorems to the expressions $\overline{\overline{A + B\bar{C}} + D\,(\overline{E + \bar{F}})}$

Solution:

Step l. Identify the terms to which you can apply De Morgan's theorems, and think of each term as a single variable. Let $\overline{A + B\bar{C}}$ = X and $D\,(\overline{E + \bar{F}})$ = Y.

Step 2. Since $\overline{X + Y} = \bar{X}\,\bar{Y}$,

$\overline{\overline{A + B\bar{C}} + D\,(\overline{E + \bar{F}})} = (\overline{\overline{A + B\bar{C}}}).(\overline{D\,(\overline{E + \bar{F}})})$

Step 3. Use rule 9 ($\bar{\bar{A}} = A$) to cancel the double bars over the left term (this is not part of De Morgan's theorem).

$= (\overline{\overline{A + B\bar{C}}}).(\overline{D\,(\overline{E + \bar{F}})}) = (A + B\bar{C}).(\overline{D\,(\overline{E + \bar{F}})})$

Step 4. Applying De Morgan's theorem to the second term,

$(A + B\bar{C}).\left(\overline{D\,(\overline{E + \bar{F}})}\right) = (A + B\bar{C}).(\bar{D} + \left(\overline{\overline{E + \bar{F}}}\right))$

Step 5. Use rule 9 ($\bar{\bar{A}} = A$) to cancel the double bars over the $E + \bar{F}$ part of the term.

$(A + B\bar{C}).(\bar{D} + \left(\overline{\overline{E + \bar{F}}}\right)) = (A + B\bar{C}).(\bar{D} + (E + \bar{F}))$

Homework: Apply De Morgan's theorems to the expressions:

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

2- $Y = \overline{(\overline{A} + B) + CD}$

3.4 Simplification of Boolean Expressions

Many times in the application of Boolean algebra, we have to reduce a particular expression to its simplest form or change its form to a more convenient one in order to implement the expression most efficiently, the approach taken is to use the basic manipulation and simplify an expression.

Example: Using Boolean algebra techniques, simplify this expression:

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

Solution:

Step 1: Apply the distributive law to the second and third terms in the expression, as follows:

$$AB + AB + AC + BB + BC$$

Step 2: Apply rule 7 ($BB = B$) to the fourth term.

$$AB + AB + AC + B + BC$$

Step 3: Apply rule 5 ($AB + AB = AB$) to the first two terms.

$$AB + AC + B + BC$$

Step 4: Apply rule 10 ($B + BC = B$) to the last two terms.

$$AB + AC + B$$

Step 5: Apply rule 10 ($AB + B = B$) to the first and third terms.

$$B + AC$$

At this point the expression is simplified as much as possible.

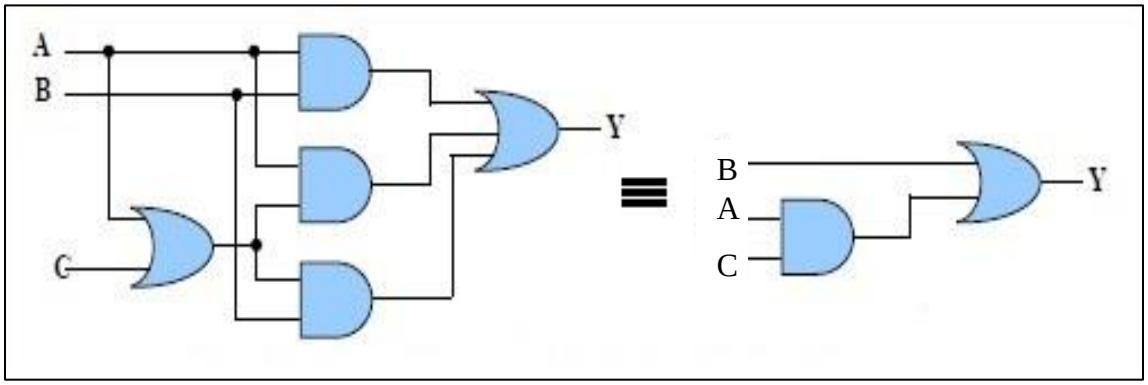


Fig. (5.10) Gate circuits for example above.

Homework: Simplify the Boolean expressions:

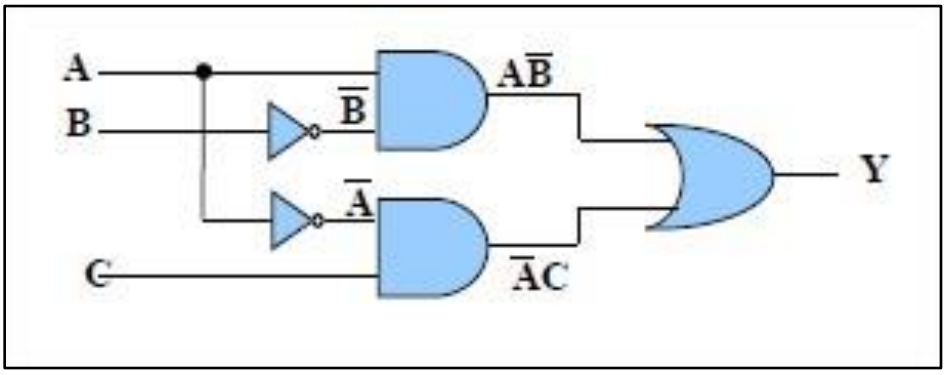
a) $(A + \bar{B})(A + C)$

b) $(A + \bar{A})(AB + AB\bar{C})$

c) $AB + (\bar{A} + \bar{B})C + AB$

3.5 Boolean Expressions for Gate Networks

Boolean expressions can be computed by implementing them in hardware using logic gates. This is most easily seen with an example. To find the Boolean expression for any logical circle, start from the input to the left heading to the final output of the circle by typing the output of each gate. For example take the logic circuit below:

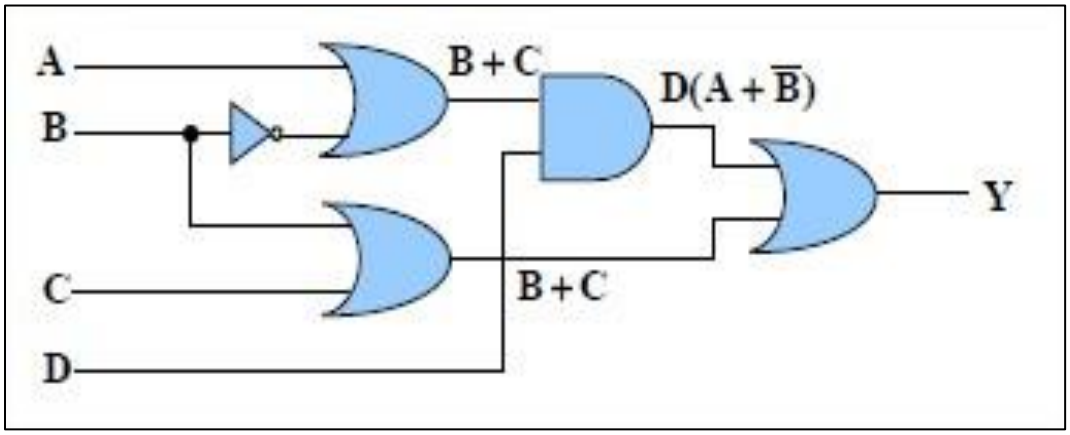


We can calculate the Boolean expression as follows:

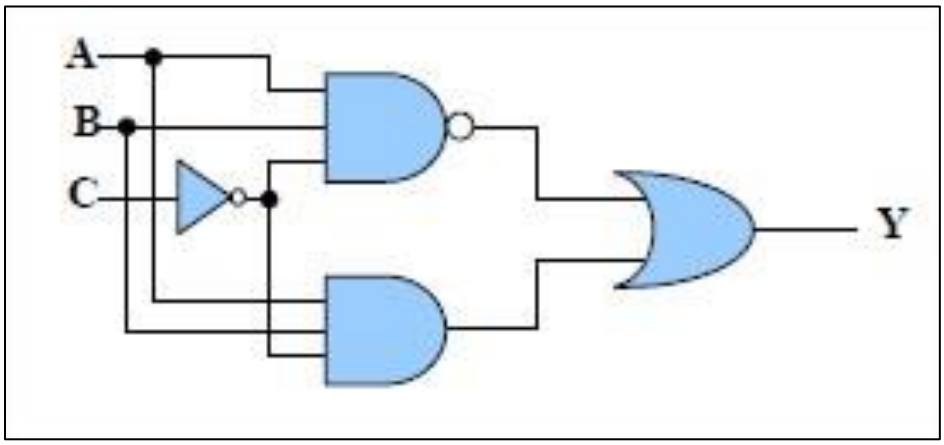
1. Boolean expression for AND gate with two input $A, \bar{B}$ is $A\bar{B}$.
2. Boolean expression for AND gate with two input $A, \bar{C}$ is $A\bar{C}$.
3. Boolean expression for OR gate with two input $A\bar{B}, A\bar{C}$ is $A\bar{B} + A\bar{C}$

The final out of the circle is: $Y = A\bar{B} + A\bar{C}$.

Example: Write the Boolean expression for logic circuit below:



Homework: Write the Boolean expression for logic circuit below:

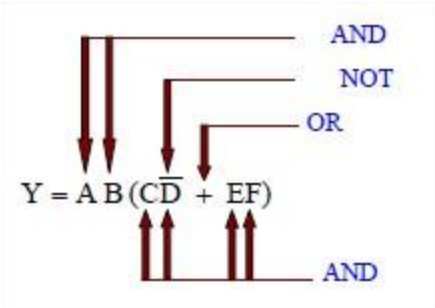


3.6 Gate Networks for Boolean Expressions

Now will discuss how to find logic circuit by using Boolean expression. Using some example will be easier to understand. To find the circuit for example below:

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

When dividing this Boolean expression, we find that A, B and $(C\overline{D} + EF)$ are three input for AND gate, the expression $(C\overline{D} + EF)$ can take C, $\overline{D}$ to AND gate and take E, F to another AND gate then take the output for the tow AND gate to OR gate.



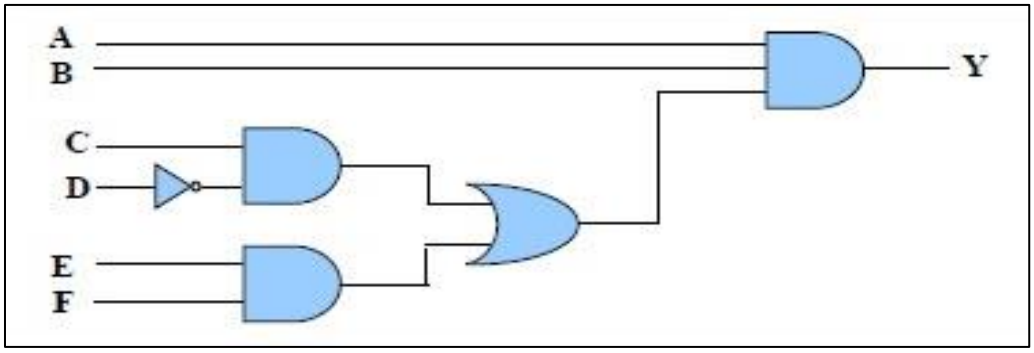
So the gate that we will used to find the circuit $AB\,(C\overline{D} + EF)$:

1- NOT gate to represent $\overline{D}$.

2- AND to represent each $C\overline{D}, EF$.

3- OR gate to represent $(C\overline{D} + EF)$.

4- AND to represent the output Y.



Homework: Draw the logic circuit for the Boolean expression below:

1. $A\overline{B} + \bar{A}B$
2. $AB + \overline{A\overline{B}} + \bar{A}BC$
3. $\bar{A}B\,(C + D)$

3.7 Boolean expression via a Truth Table

The truth table can be used to write a Boolean equation. To design and implement the problem, Boolean logical expressions (equation) are derived for the output logical function in terms of the binary variables representing the inputs. The logic expressions are given either in form of Sum of Product (SoP), or in the form of Product of Sum (PoS).

1. Sum – of – Product (SoP)

This form sometimes called 'Minterm'. A product term which contain each of the n-variable factors in either complemented or un-complemented form for output digits '1' only, is called SoP. For example the truth table below:

Input			Output	
A	B	C	F	
0	0	0	1	$\bar{A}\bar{B}\bar{C}$
0	0	1	0	
0	1	0	1	$\bar{A}B\bar{C}$
0	1	1	1	$\bar{A}BC$
1	0	0	0	
1	0	1	0	
1	1	0	1	$AB\bar{C}$
1	1	1	1	ABC

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

The logical SoP expression for the output digit '1' is written as,

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

This function can be put in another form such as

$F = \sum 0, 2, 3, 6, 7$

Since F = 1 in rows 0, 2, 3, 6, 7 only.

2. Product – of – Sum (PoS)

A logical equation can also be expressed as a Product of Sum (PoS) form (sometimes this method is called 'Maxterm'). This is done by considering the combination for F=0 (output = 0). So (for above example) form truth table F = 0 is in rows 1, 4 and 5 hence:

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

The PoS can be expressed as:

$$F = \Pi\, 1, 4, 5$$

Example: - Put F in SoP and PoS form and simplifying it:

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

Solution: -

SoP:

$$F = \sum 0, 1, 3$$
$$= \bar{A}\bar{B} + \bar{A}B + AB$$
$$= \bar{A}\,(\bar{B} + B) + AB$$
$$= \bar{A} + AB$$
$$F = \bar{A} + B$$

PoS:

$$F = \Pi\, 2$$
$$= \bar{A} + B$$

Example: - Put in SoP form: - $F = A\bar{B}C + \bar{A}BC + ABC$

Solution: - $F = A\bar{B}C + \bar{A}BC + ABC$

$A\bar{B}C \rightarrow 101$, $\bar{A}BC \rightarrow 011$, $ABC \rightarrow 111$

$$F = \sum 3, 5, 7$$

Example: - Put in Pos form and draw the truth table then find SoP: -

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

Solution: -

$$F = (A + \underset{\downarrow}{B} + \bar{C})\,(A + \underset{\downarrow}{\bar{B}} + C)\,(\bar{A} + \underset{\downarrow}{\bar{B}} + \bar{C})\,(\bar{A} + \underset{\downarrow}{\bar{B}} + C)$$

001 010 111 110

F = Π 1, 2, 6, 7

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

$$F = \sum 0, 3, 4, 5$$

$$F = \bar{A}\bar{B}\bar{C} + \bar{A}BC + A\bar{B}\bar{C} + A\bar{B}C$$

Example: - Represent F1, F2 in SoP and PoS form then simplified F1 and F2 using Boolean algebra and implement the function.

A	B	C	F1	F2
0	0	0	0	1
0	0	1	1	0
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

Solution:

In SoP

$F1 = \sum 1,2,3,5,6,7$

$F1 = \bar{A}\bar{B}C + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$

$F1 = \bar{A}(\bar{B}C + B\bar{C} + BC) + A(\bar{B}C + B\bar{C} + BC)$

$F1 = \bar{A}[\bar{B}C + B(\bar{C} + C)] + A[\bar{B}C + B(\bar{C} + C)]$

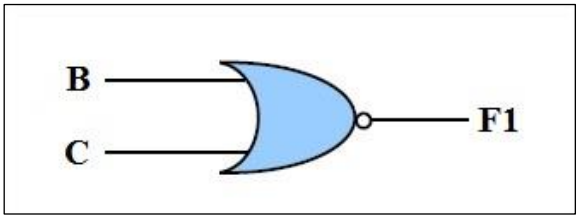
$F1 = \bar{A}[\bar{B}C + B] + A[\bar{B}C + B]$

$F1 = \bar{A}[C + B] + A[C + B]$

$F1 = \bar{A}C + \bar{A}B + AC + AB$

$F1 = B(\bar{A} + A) + C(\bar{A} + A)$

$F1 = B + C$



In PoS

$F1 = \Pi\ 0,4$

$F1 = (A + B + C)(\bar{A} + B + C)$

$F1 = A\bar{A} + AB + AC + \bar{A}B + BB + BC + \bar{A}C + BC + CC$

$F1 = AB + AC + \bar{A}B + B + BC + \bar{A}C + BC + C$

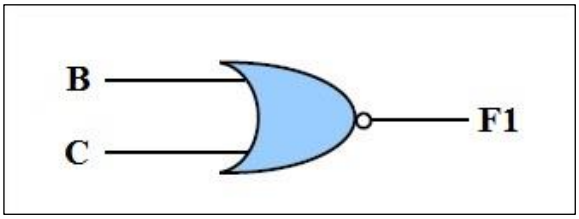
$F1 = AB + AC + \bar{A}B + B(1 + C) + \bar{A}C + B(1 + C)$

$F1 = AB + AC + \bar{A}B + B + \bar{A}C + B$

$F1 = B(A + \bar{A}) + C(\bar{A} + A) + B + C$

$F1 = B + C + B + C$

$F1 = B + C$



Homework solution for F2

Converting SoP to PoS and Vice Versa

To convert from SoP to PoS and Vice Versa we must follow three steps:

1. Evaluate each product term in the SoP (or PoS) expression that determine the binary numbers which represent the product term.
2. Determine all the binary numbers not included in the evaluation in step1.
3. Write the equivalent sum term for each binary number form step2 and express in PoS (or SoP) form

SoP to PoS: If any variable is missing from any term, we must add these missing variable to that term, by multiplying the term by the missing variable.

For example if the variable B is missing from the form AC, we must multiplying the term AC by $(B + \bar{B})$: $AC\ (B + \bar{B})$

PoS to SoP: If any variable is missing from any term, we must add these missing variable to that term, by adding the term by the missing variable.

For example if the variable A is missing from the form $(B + C)$, we must adding $A\bar{A}$ to the term $(B + \bar{C})$:

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

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

Example: - Convert the SoP expression to PoS: $F = B + AC$

Solution:

1th method

$F = B + AC$

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

$F = B\ (AC + A\bar{C} + \bar{A}C + \bar{A}\bar{C}) + ABC + A\bar{B}C$

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

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

$F = \sum 2, 3, 5, 6, 7$

$F = \Pi\ 0, 1, 4$

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

2th method

$F = B + AC$

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

$F = \sum 2, 3, 5, 6, 7$

$F = \Pi\, 0, 1, 4$

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

Example: - Convert the PoS expression to SoP:

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

Solution:

1th method

$F = [(A + B) + C\bar{C}]\,[(A + C) + B\bar{B}]\,[(A + B + \bar{C})]$

$F = (A + B + C)\,(A + B + C)\,(A + B + C)\,(A + B + C)\,(A + B + C)$

000 001 100 110 001

$F = \Pi\, 0, 1, 4, 6$

$F = \sum 2, 3, 5, 7$

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

$\bar{F} = \overline{(\bar{A} + \bar{B} + \bar{C})\,(A + B + \bar{C})\,(\bar{A} + B + C)\,(\bar{A} + \bar{B} + C)}$

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

2th method

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

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

$F = \Pi\, 0, 1, 4, 6$

$F = \sum 2, 3, 5, 7$

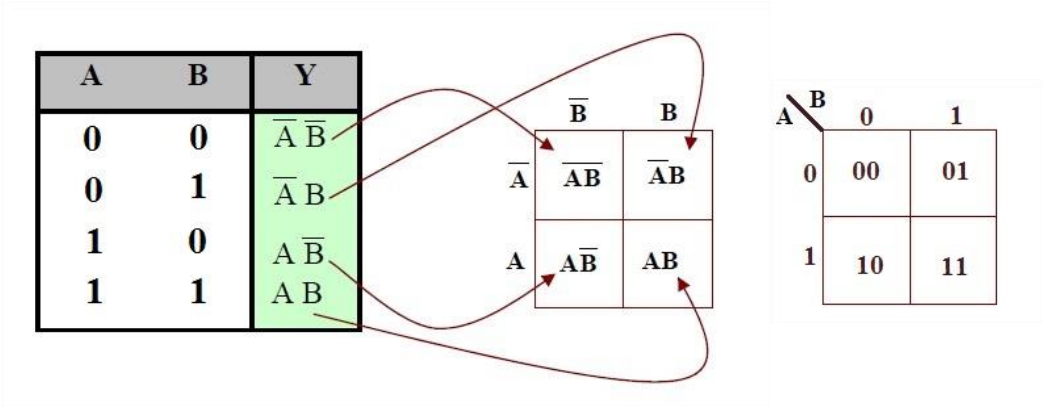
$F = \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + ABC$

3.8 Simplification using Karnaugh-map

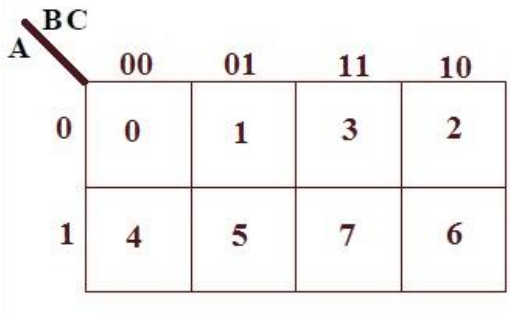
A Karnaugh map provides a systematic method for simplifying Boolean expressions and, if properly used, will produce the simplest SOP or POS expression. As you have seen, the effectiveness of algebraic simplification depends on your familiarity with all the laws, rules, and theorems of Boolean algebra and on your ability to apply them. The Karnaugh map, on the other hand, provides a "cookbook" method for simplification. The Karnaugh map is an array of cells in which each cell represents a binary value of the input variables. The cells are arranged in a way so that simplification of a given expression is simply grouping the cells.

Karnaugh maps can be used for expressions with two, three, four, and five variables. The number of cells in a Karnaugh map is equal to the total number of possible input variable combinations as is the number of rows in a truth table.

For two input variables, the number of cells is equal to $2^2 = 4$ cells



For three input variables, the number of cells is equal to $2^3 = 8$ cells



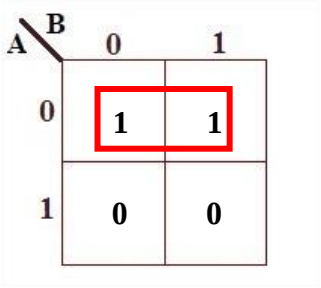
For 4 input variables, the number of cells is equal to $2^4 = 16$ cells

AB \ CD	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

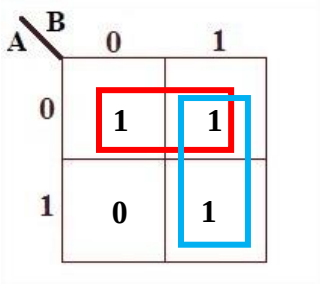
Not:

1. Number of 1's or 0's in one group must be 1, 2, 4, 8, and 16.
2. We must take maximum number of 1's or 0's in one group.
3. We must take at least one-time the 1's or 0's.

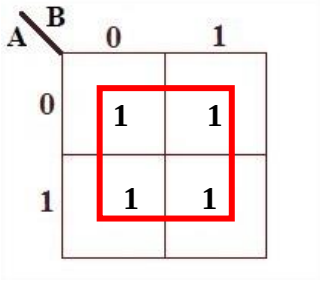
<u>The 2- variable K-map:</u>



$F = \overline{A}$	Sop
$F = \overline{A}$	PoS

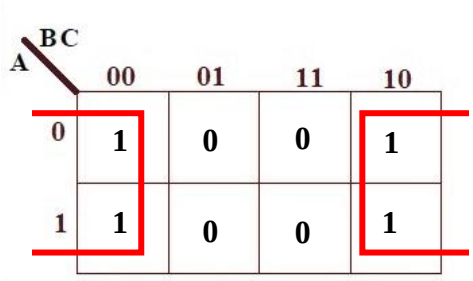


$F = \overline{A} + B$	Sop
$F = \overline{A} + B$	PoS

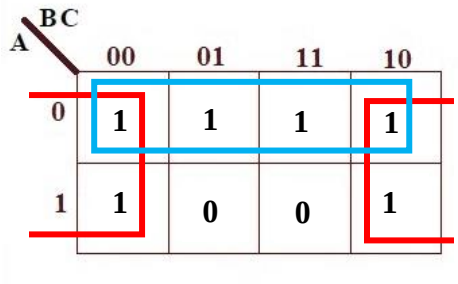


F = 1 SoP

The 3- variable K-map:

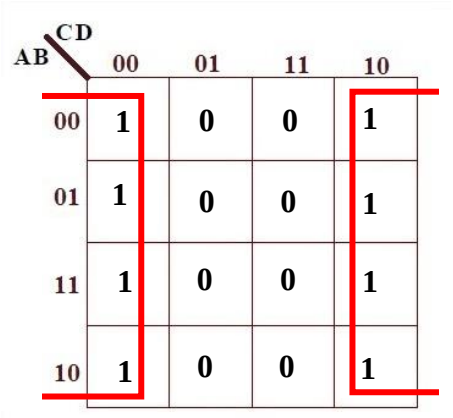


$F = \bar{C}$	Sop
$F = \bar{C}$	PoS

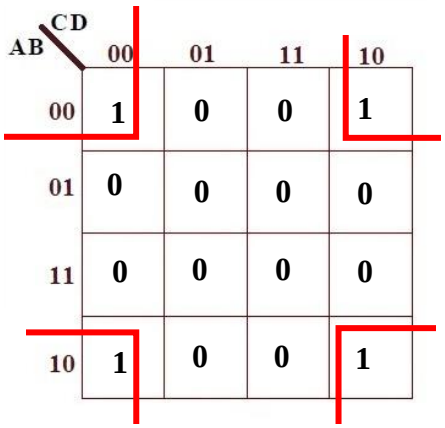


$F = \bar{A} + \bar{C}$	Sop
$F = \bar{A} + \bar{C}$	PoS

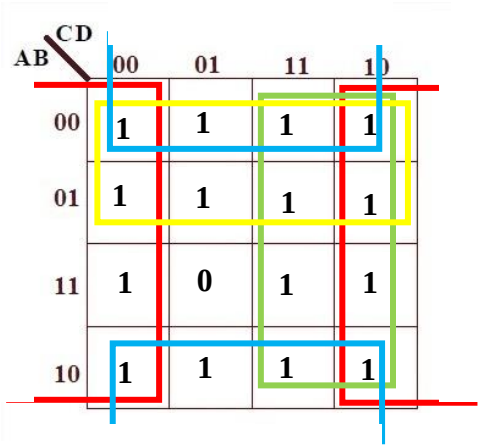
The 4- variable K-map:



$F = \bar{D}$	Sop
$F = \bar{D}$	PoS

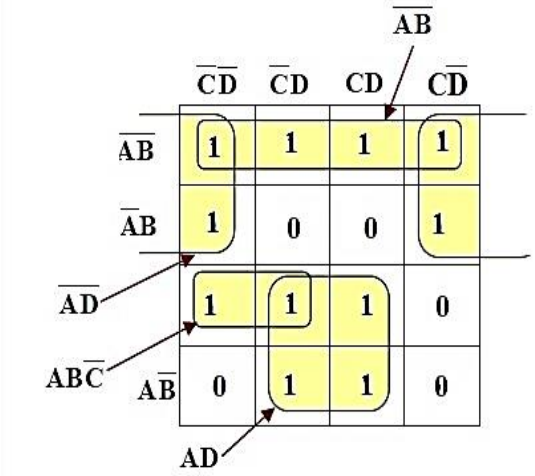


$F = \overline{B}\overline{D}$	Sop
$F = \overline{B}\overline{D}$	PoS

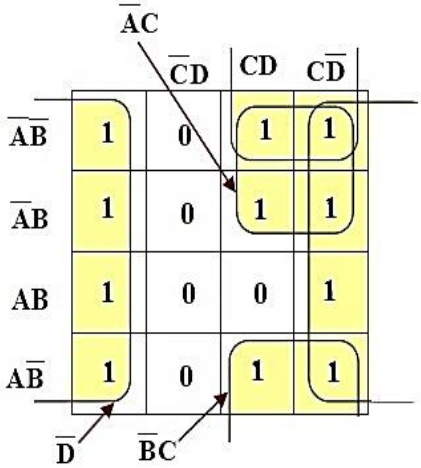


$F = \overline{A} + \overline{B} + C + \overline{D}$	Sop
$F = \overline{A} + \overline{B} + C + \overline{D}$	PoS

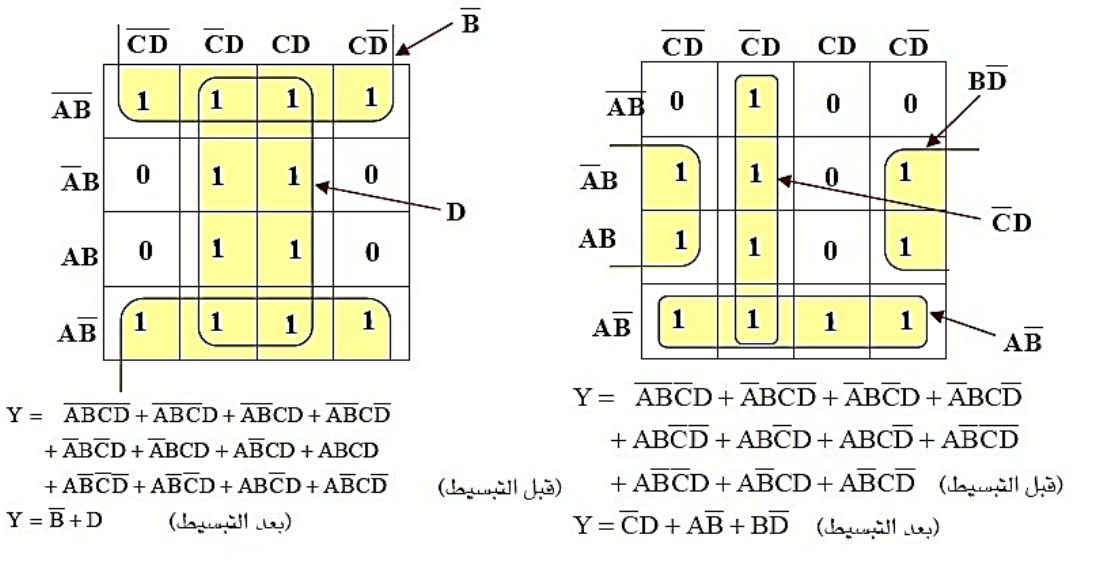
Different examples in K-map



$Y = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}CD + \overline{A}\overline{B}C\overline{D}$
$+ \overline{A}B\overline{C}\overline{D} + \overline{A}B\overline{C}D + \overline{A}BC\overline{D} + \overline{A}BCD$
$+ AB\overline{C}D + A\overline{B}\overline{C}D + A\overline{B}CD$ (قبل التبسيط)
$Y = AB\overline{C} + AD + \overline{A}B\overline{D} + \overline{A}\overline{B}$ (بعد التبسيط)



$Y = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}CD + \overline{A}\overline{B}C\overline{D} + \overline{A}\overline{B}C\overline{D}$
$+ \overline{A}BCD + \overline{A}BC\overline{D} + AB\overline{C}\overline{D} + AB\overline{C}D$
$+ A\overline{B}C\overline{D} + A\overline{B}CD + A\overline{B}C\overline{D}$ (قبل التبسيط)
$Y = \overline{A}C + \overline{B}C + \overline{D}$ (بعد التبسيط)



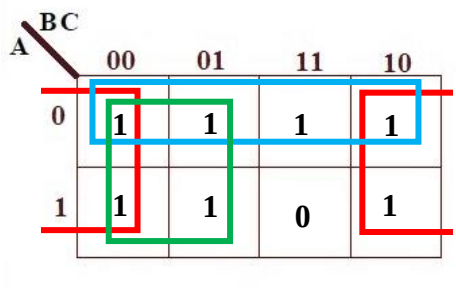
Example: - Simplify the following Boolean function using K-map:

$$F = \overline{A} + A\overline{B} + AB\overline{C}$$

Solution: The equation is SoP and the domain (A, B, C) there are missing variable is term 1&2 so:

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

Then the K-map must have 2^3=8 cells

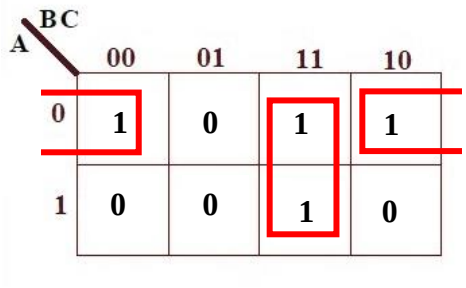


$$F = \overline{A} + \overline{B} + \overline{C}$$

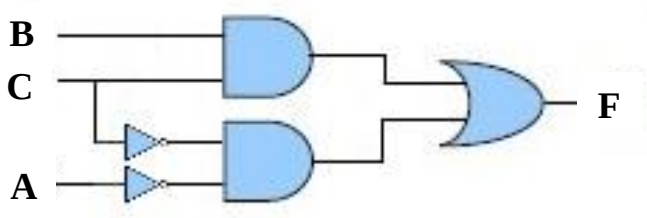
Example: - Implement the logic function by truth table below using K-map.

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

Solution:



$$F = \overline{A}\overline{C} + BC$$



"Don't Care" Conditions

Sometimes a situation arises in which some input variable combinations are not allowed. For example, recall that in the BCD code there are six invalid combinations: 1010, 1011, 1100, 1101, 1110, and 1111. Since these un-allowed states will never occur in an application involving the BCD code, they can be treated as "don't care" terms with respect to their effect on the output. That is, for these "don't care" terms either a 1 or a 0 may be assigned to the output: it really does not matter since they will never occur. The "don't care" terms can be used to advantage on the Karnaugh map. Fig. (5-9) shows that for each "don't care" term, an X is placed in the cell. When grouping the 1 s, the Xs can be treated as 1s to make a larger grouping or as 0s if they cannot be used to advantage. The larger a group, the simpler the resulting term will be.

The truth table in Fig: (5-9) (a) describes a logic function that has a 1 output only when the BCD code for 7, 8, or 9 is present on the inputs. If the "don't cares" are used as 1s, the resulting expression for the function is $A + BCD$, as indicated in part (b). If the "don't cares" are not used as 1s, the resulting expression is $A\overline{B}\overline{C} + \overline{A}BCD$: so you can see the advantage of using "don't care" terms to get the simplest expression.

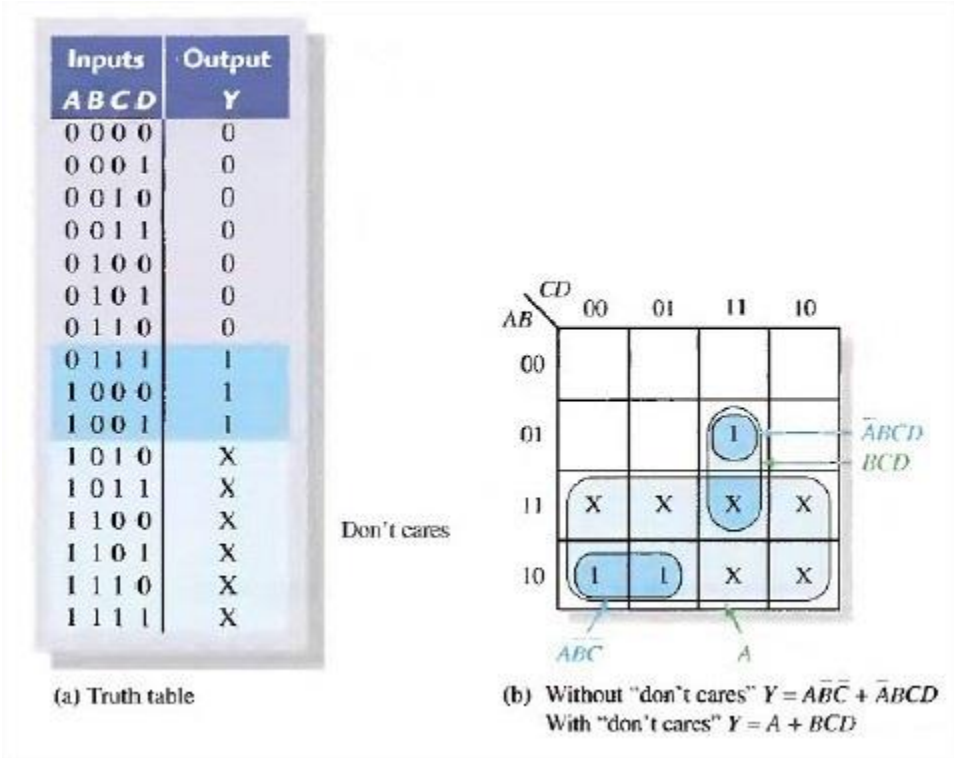


Fig: (5.9).

Example: - Simplify the following Boolean function using K-map: F= ∑1, 3, 7, 11, 15

Which has the don't car conditions d = ∑ 0, 2, 5

Solution:

a)

AB \ CD	00	01	11	10
00	X	1	1	X
01	0	X	1	0
11	0	0	1	0
10	0	0	1	0

$$F = \bar{A}D + CD$$

b)

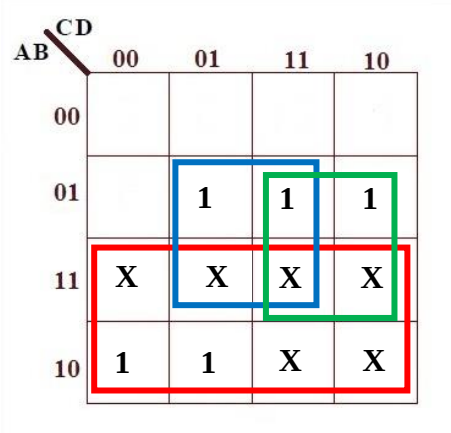
AB \ CD	00	01	11	10
00	X	1	1	X
01	0	X	1	0
11	0	0	1	0
10	0	0	1	0

$$F = \bar{A}\bar{B} + CD$$

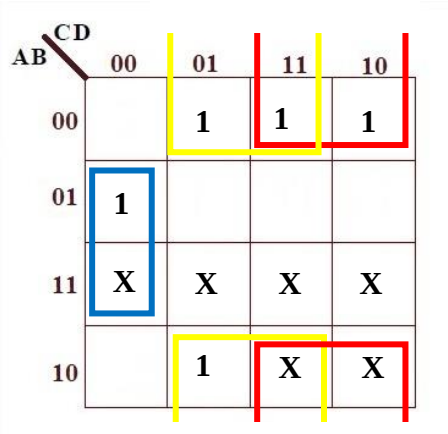
Example: Design a BCD code to excess-3 code converter and implement the function.

Solution:

BCD I/P				Excess-3 O/P			
A	B	C	D	X	Y	W	Z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	0	X	X	X	X



$X = A + BC + BD$



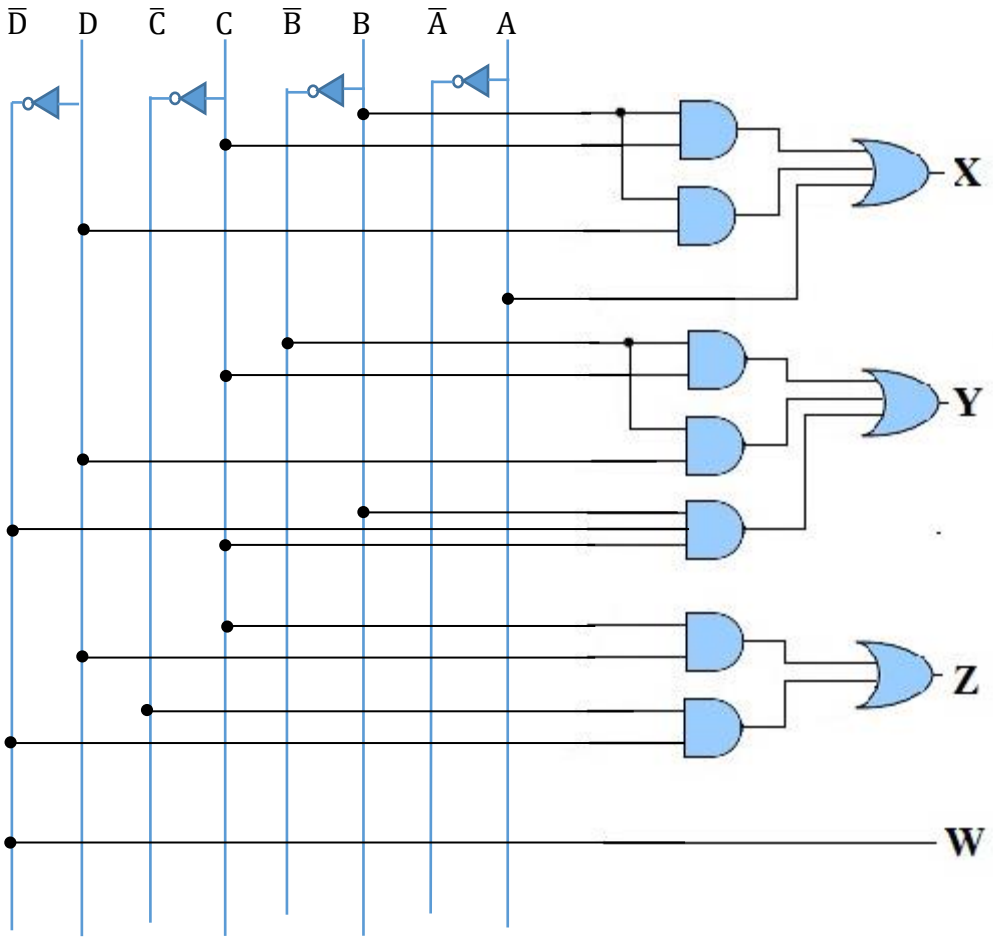
$Y = \bar{B}C + \bar{B}D + B\bar{C}\bar{D}$

AB \ CD	00	01	11	10
00	1		1	
01	1		1	
11	X	X	X	X
10	1		X	X

$W = CD + \bar{C}\bar{D}$

AB \ CD	00	01	11	10
00	1			1
01	1			1
11	X	X	X	X
10	1		X	X

$Z = \bar{D}$



Homework: Design a BCD code to gray code converter and implement the function.