- Karnaugh Map Simplification of SOP expressions

The process that results in an expression containing the fewest possible terms with the fewest possible variables is called **minimization**. After an SOP expression has been mapped , a minimum SOP expression is obtained by ***grouping the 1's*** and ***determining the minimum SOP expression from the map***.

- ***grouping the 1's*** : you can group 1s on the Karnaugh map according to the following rules by enclosing those adjacent cells containing 1s.the goal is to maximize the size of the groups and to minimize the number of groups.

  1. A group must contain either 1,2,4,8 , or 16 cells,which are all powers of two. In the case of a 3-variables map,2^3=8 cells is the maximum group.

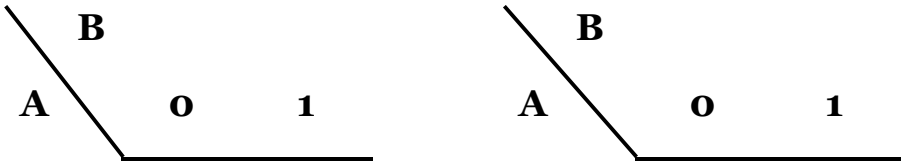
  2. Each cell in a group must be adjacent to one or more cells in that some group, but all cells in the group do not have to be adjacent to each other.

  3. always include the largest possible number of 1s in a group in accordance with rule 1.

  4. Each1 on the map must be included in at least one group. The 1s already in a group can be included in another group as long as the overlapping groups include non common 1s.

Example :group the 1s in each of the Karnauph maps in the following 2-varable map.

2-variable map

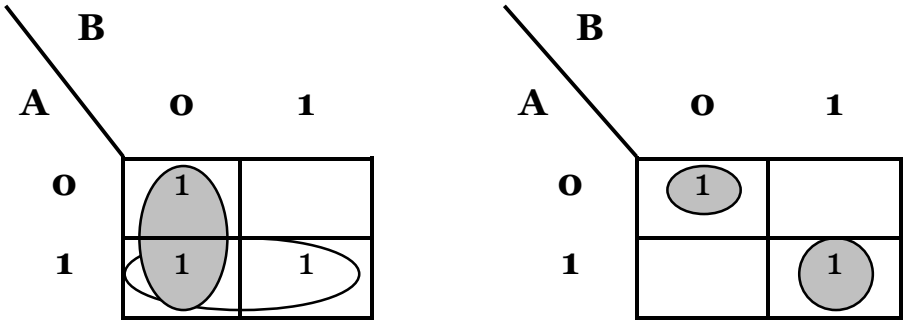


	0	1
0	1	
1	1	1

	0	1
0	1	
1		1

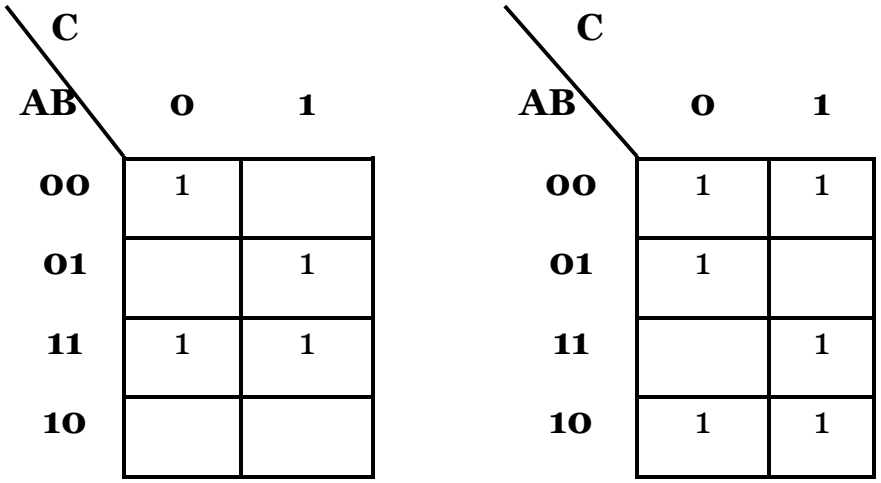
Solution: the grouping are shown the next figure. In same cases, there may be more than one way to group the1s to form maximum grouping .

2-variable map



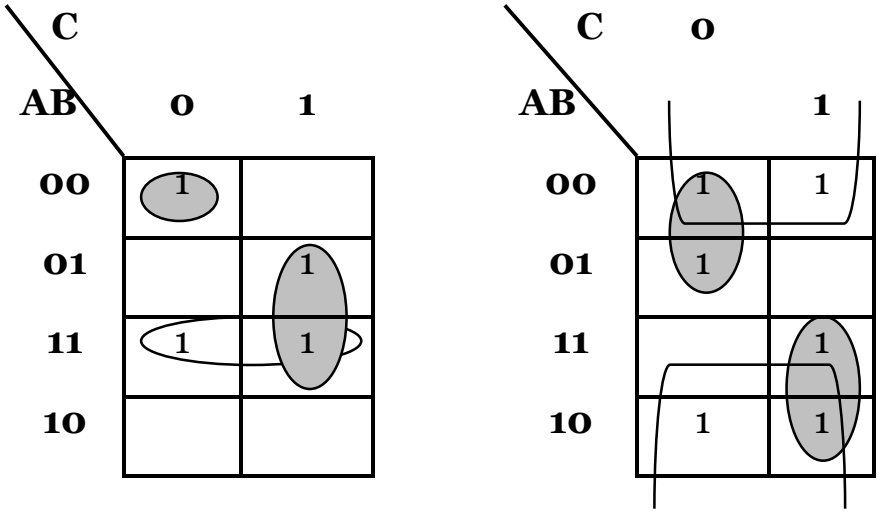
Example :group the 1s in each of the Karnauph maps in the following 3-varable map.

3- variable map



Solution: the grouping are shown the next figure. In same cases, there may be more than one way to group the1s to form maximum grouping .

3- variable map



Example :group the 1s in each of the Karnauph maps in the following 4-varable map

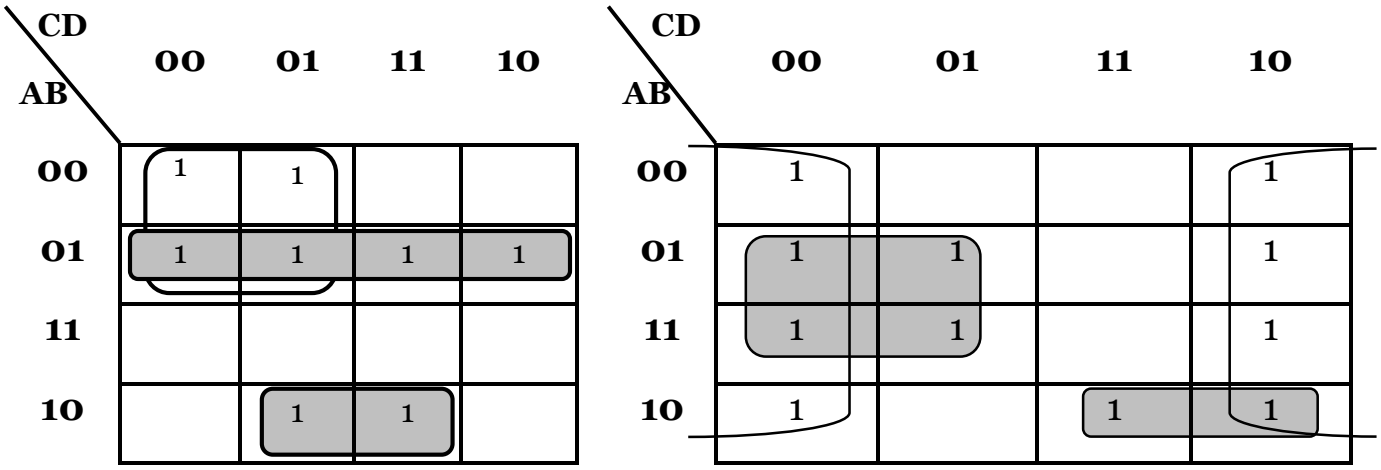
4-variable map

CD / AB	00	01	11	10
00	1	1		
01	1	1	1	1
11				
10		1	1	

CD / AB	00	01	11	10
00	1			1
01	1	1		1
11	1	1		1
10	1		1	1

Solution: the grouping are shown the next figure. In same cases, there may be more than one way to group the1s to form maximum grouping .

4-variable map



- ***Determining the minimum SOP Expression From the map***

When all the 1s representing the standard product terms in an expression are properly mapped and grouped, the process of determining thee resulting minimum SOP expression begins.

The following rules ate applied to find the minimum product terms and the minimum SOP expression:

1. Group the cell that have 1's. Each group of cells containing 1's creates one product item composed of all variables that occur in only one form (either uncomplemented or complemented) within the group. ***Variable that occur both uncomplemented and complemented within the group are eliminated . these are called contradictory variables***

2. Determine the minimum product term for each group.

   a. **For a 2-Variable map**

      (1) A 1-cell group yields a 2-variable product term .

(2) A 2-cell group yields a 1-variable product term.

(3) An 4-cell group yields a value of 1 for the expression.

b. For a 3-Variable map

(1) A 1-cell group yields a 3-variable product term .

(2) A 2-cell group yields a 2-variable product term.

(3) A 4-cell group yields a 1-variable product term.

(4) An 8-cell group yields a value of **1** for the expression

c. For a 4-Variable map

(1) A 1-cell group yields a 4-variable product term**.**

(2) A 2-cell group yields a 3-variable product term.

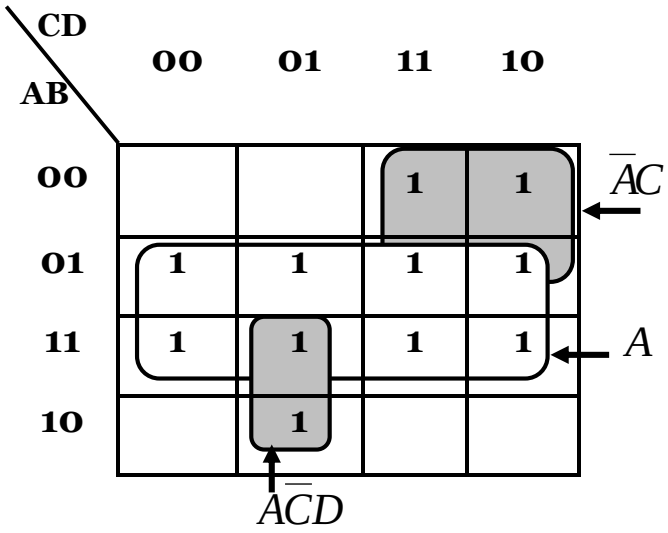
(3) A 4-cell group yields a 2-variable product term.

(4) An 8-cell group yields a 1-variable term.

(5) A 16-cell group yields a value of 1 for the expression .

3. When all the minimum product terms are derived from the Karnaugh map, they are summed to form the minimum SOP expression,

Example : Determine the product terms for the Karnaugh map in figure bellow , and write the resulting minimum SOP expression.



Solution: Eliminate variables that are in a grouping in both complemented and uncomplemented forms. In the above figure ,

- the product term for the 8-cell group is B because the cells withen that group contain both A and $\overline{A}$, C and $\overline{C}$, and D and $\overline{D}$, which are eliminated.
- The 4-cell group contains $B, \overline{B}, D, and\ \overline{D}$, leaving the variables $\overline{A}$ and C. which form the product term $\overline{A}C$.
- The 2-cell group contains B and $\overline{B}$, leaving variables A , $\overline{C}$, and D which form the product term $A\overline{C}D$.

Notes: how overlapping is used to maximize the size of the groups. The resulting minimum SOP expression is the sum of these product terms:

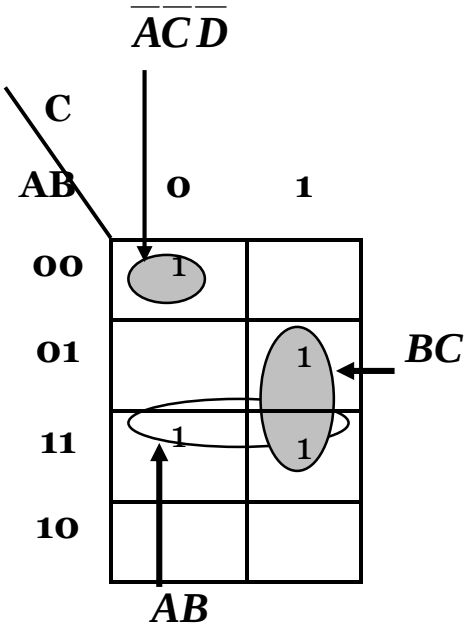
$$B + \overline{A}C + A\overline{C}D$$

✍ Home work

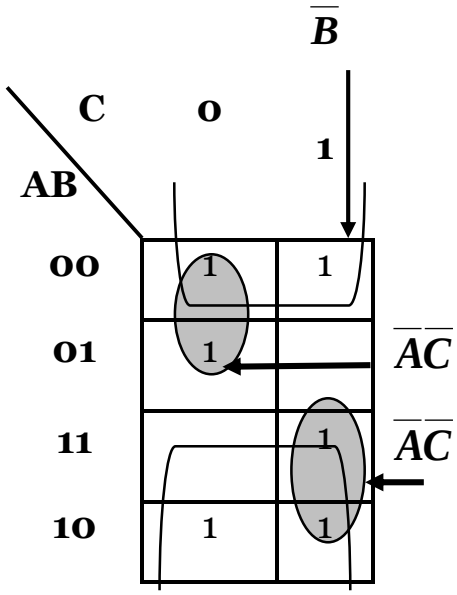
for the Karnagh map in the above example , add a 1 in the lower right cell (1010) and determine the resulting SOP expression.

Example **:** Determine the product terms for the Karnaugh map in two figures bellow , and write the resulting minimum SOP expression.

Figure(a)



Figure(b)



Solution: *the resulting minimum product term for each group shown in figure(a) and (b) are:*

(a) $AB + BC + \overline{A}\overline{B}\overline{C}$

(b) $\overline{B} + \overline{A}\overline{C} + AC$

Example **:** Determine the product terms for the Karnaugh map in two figures bellow , and write the resulting minimum SOP expression.

Solution: *the resulting minimum product term for each group shown in figure(c) and (d) are:*

(c) $\overline{A}B + \overline{A}\overline{C} + A\overline{B}D$

(d) $\overline{D} + A\overline{B}C + B\overline{C}$

Figure (c)

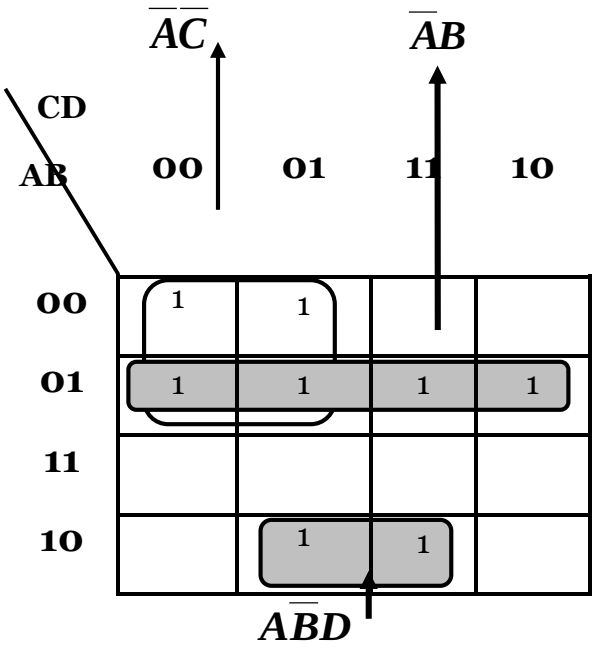
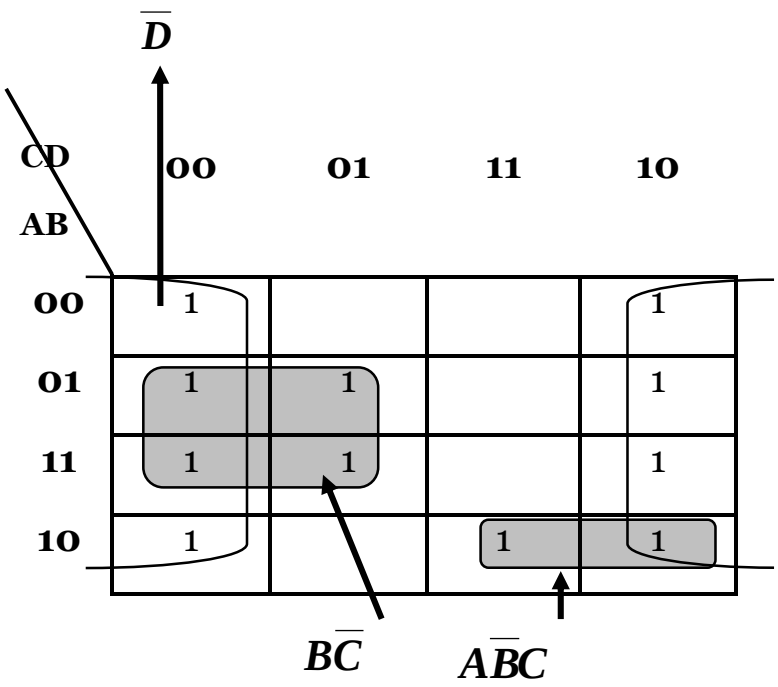


Figure (d)



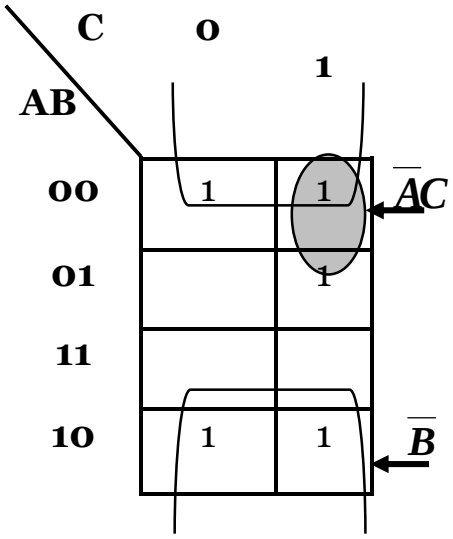
Example: *Use a Karnaugh map to minimize the following standard SOP expression:*

$$A\overline{B}C + \overline{A}BC + \overline{A}\overline{B}C + \overline{A}\overline{B}\overline{C} + A\overline{B}\overline{C}$$

Solution: the binary values of the expression are

101 + 011 + 001 + 000 + 100

Map the standard SOP expression and group the cells are shown in the following figure.



The resulting minimum SOP expression is : $\overline{B} + \overline{A}C$

Home work

Use the Karnaugh map to minimize the following SOP expression:

1 – $\overline{B}\overline{C}\overline{D} + \overline{A}B\overline{C}\overline{D} + AB\overline{C}\overline{D} + \overline{A}\overline{B}CD + A\overline{B}CD + \overline{A}\overline{B}C\overline{D} + \overline{A}BC\overline{D} + ABC\overline{D} + A\overline{B}C\overline{D}$

2 – $\overline{W}\overline{X}\overline{Y}\overline{Z} + W\overline{X}YZ + W\overline{X}\overline{Y}Z + \overline{W}YZ + W\overline{X}\overline{Y}\overline{Z}$

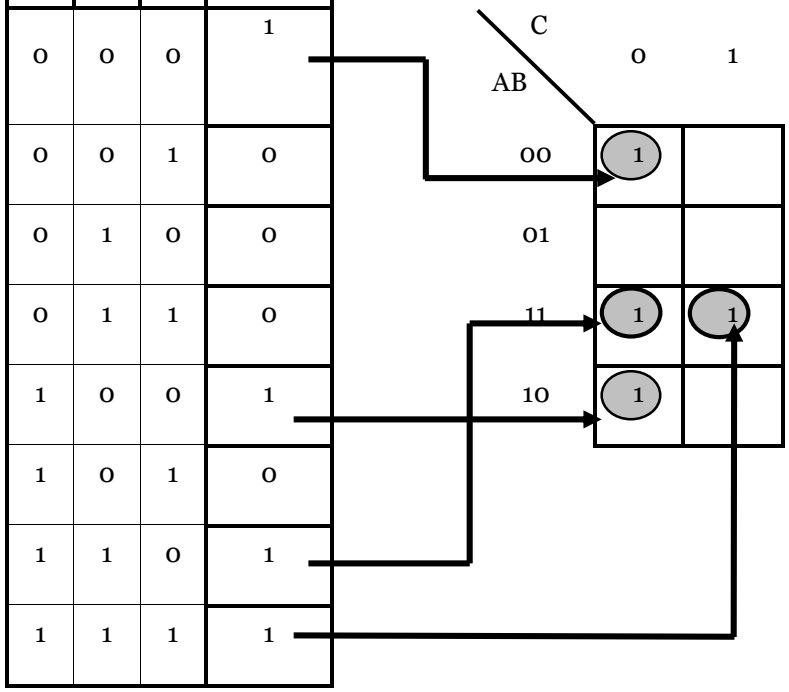
Mapping Directly from the Truth Table

You have seen how to map a Boolean expression; now you will learn how to go directly from a truth table to a Karnauph map. Recall that a truth table gives the output of a Boolean expression for all possible input variable combination.

An example of a Boolean expression and its troth table representation is shown in the next figure. Notice in the truth table that the output X is 1 for four different input variable combinations.

$$X = \overline{A}B\overline{C} + A\overline{B}\overline{C} + AB\overline{C} + ABC$$

Input			Output
A	B	C	X
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



The 1s in the output column of the truth table are mapped directly onto a Karnaugh map into the cells corresponding to the values of the associated input variable combinations.

Notes :*In the above figure you can see that the Boolean expression , the truth table , and the Karnaugh map are simply different ways to represent a logic function.*

♦ "Don't care " Condition

Sometime a situation arises in which some input variables combinations are not allowed. For example , recall tat in BCD code covered in previous sections , there are six invalid combinations: 1010 ,1011 ,1100,1101 , and 1111.

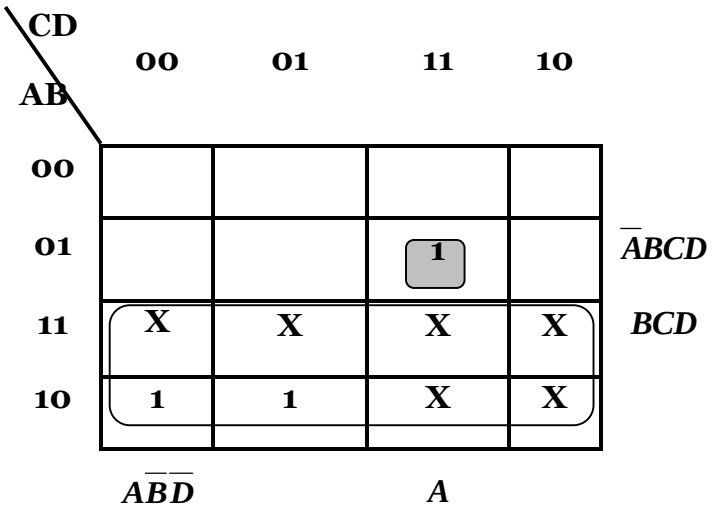
Since these un allowed states will never occur in an application involving the BCD code, they can be treated as **"Don't care"** terms with respect to their effect on the output. That is , for these **"don't care "** terms either a **1** or a **0** may be assigned to the output; it really does not matter since they will never occur.

The **"don't care"** terms can be used to advantage on the Karnaugh map. The figure bellow shows that for each "don't care" term, an X is placed in the cell. When grouping the 1s , 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.***

Input				Output
A	**B**	**C**	**D**	**X**
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

Don't cares

(a) Truth Table



(b)without "don't care" Y= $A\bar{B}\bar{C} + \bar{A}BCD$ **,with "don't care"** Y= $A + BCD$

Notes: the truth table (a) describe 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 care"** are used as 1s the resulting expression for the function is $A + BCD$, a indicated in part(b). if the "don't cares" are not used as 1s ,the resulting expression is $A\bar{B}\bar{C} + \bar{A}BCD$; so you can see the advantage of using **"don't care"** terms to get the simplest expression.

6- Combinational Logic Analysis

Combinational circuit is circuit in which we combine the different gates(AND ,OR ,NAND ,NOR ,XOR ,XNOR ,NOT ,..) in the circuit for example encoder, decoder, multiplexer , demultiplexer and any circuit build bay any combinations of theses gates . Some of the characteristics of combinational circuits are following.

- The output of combinational circuit at any instant of time, depends only on the levels present at input terminals.
- The combinational circuit do not use any memory. The previous state of input does not have any effect on the present state of the circuit.
- A combinational circuit can have n number of inputs and m number of outputs.

- Combinational circuit black Diagram

- Implementing Combinational Logic

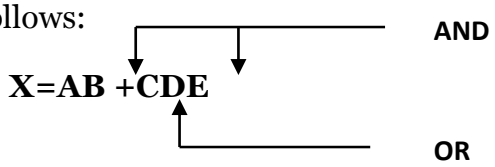
In this section , examples are used to illustrate how to implement a logic circuit from a Boolean expression or a truth table. Minimization of a logic circuit using the methods covered in previous section.

- **From a Boolean Expression to a logic circuit**

Let's examine the following Boolean expression:

$$X=AB+CDE$$

A brief inspection shows that this expression is composed of two terms , AB and CDE , with a domain of five variables. The first term is formed by ANDing A with B, and the second term is formed by ANDing C,D, and D. the two terms are then ORed to form the output X . These operations are indicated in the structure of the expression as follows:

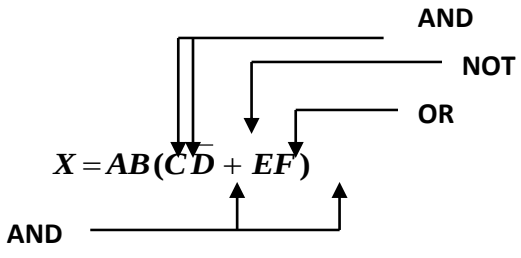


Homework: ***Draw the logic circuit for the above Boolean expression***

Example: let's implement the following expression:

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

Solution:



Notes: Unless an intermediate term, such as $C\overline{D}+EF$, is required as an output for some other purpose, it usually best to reduce a circuit to its SOP form in order to reduce the overall propagation delay time .the expression is converted to SOP as follows, $AB(C\overline{D}+EF)=ABC\overline{D}+ABEF$.

Homework: ***Draw the logic circuit for the above SOP Boolean expression***

- ## From a Truth Table to a Logic Circuit

If you begin with a truth table instead of an expression, you can write the SOP expression from the truth table and then implement the logic circuit. The table bellow specifies a logic function.

Input			Output	Product term
A	B	C	X	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\overline{A}BC$
1	0	0	1	$A\overline{B}\overline{C}$
1	0	1	0	
1	1	0	0	
1	1	1	0	

The Boolean SOP expression obtained from the truth table by ORing the product terms for which **X=1** is $X=\overline{A}BC+A\overline{B}\overline{C}$

Homework:

1- ***Draw the logic circuit for the above SOP Boolean expression .***

2- ***Develop a logic circuit with four input variable that will only produce a 1 output when exactly three input variables are 1s.***

Function Of Combinational Logic

As we mention above , Combinational circuit is circuit in which we combine the different gates, We're going to elaborate few important combinational circuits as follows.

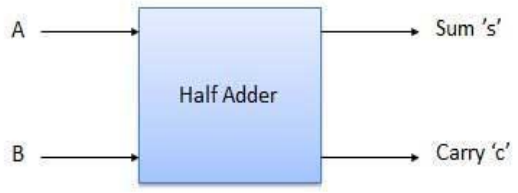
1- Basic Adders

Adders are important in computers and also in other types of digital systems in which numerical data are processed. An understanding of basic adder operation is fundamental to the study of digital systems. In this section, the half-adder and the full-adder are introduced.

- **Half - Adder (HA)**

Half adder is a combinational logic circuit with two input and two output. The half adder circuit is designed to add two single bit binary number A and B. It is the basic building block for addition of two **single** bit numbers. This circuit has two outputs **carry** and **sum**.

Block Diagram

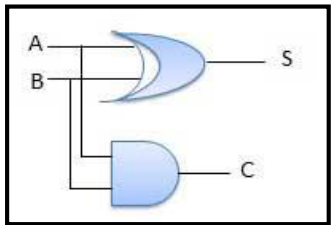


Truth Table

Input		Outputs	
A	B	$\Sigma(s)$	C_o
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

Binary digits to be added	Sum	Carry out
	XOR	AND

Circuit Diagram



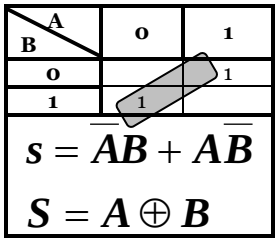
Now, what is the Boolean expression needed for the (Co) output ?

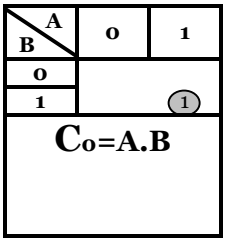
the Boolean expression is $\mathbf{C_o=A.B}$ which represent AND gate

Now, what is the Boolean expression needed for the ($\sum(s)$) output ?

The Boolean expression I s $\sum(s)= \overline{A}.B + A.\overline{B}$ which represent XOR gate.

We Can also simplify HA function using Karnough Map.



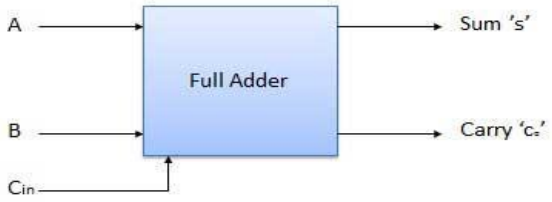


Note: the half-adder circuit adds only the LSB column (1s column) in a binary addition problem.

- **Full – Adder (FA)**

Full adder is developed to overcome the drawback of Half Adder circuit. It can add two one-bit numbers A and B, and carry c. The full adder is a three input and two output combinational circuit.

Block Diagram



Truth Table

The full adder must be used when it possible to have an extra Carry input ,then the full adder has three inputs: **A , B , and** $\mathbf{C_{in}}$. These three inputs must be added to get the $\sum(s)$ and $\mathbf{C_o}$ output .

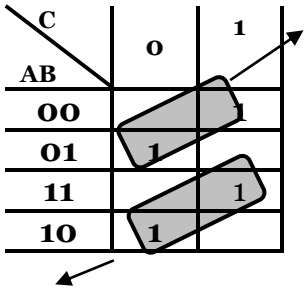
Input			Outputs	
A	**B**	**C_{in}**	**$\sum(s)$**	**C_o**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1
A + B + C_{in}			**Sum**	**Carry out**

$\sum(s) = \sum(1,2,4,7)$

$C_o = \sum(3,5,6,7)$

We can simplify **FA** Function using K-Map

$\sum(s) = \sum(1,2,4,7)$



$\bar{A}B\overline{C_{in}} + \bar{A}\bar{B}C_{in}$

$= \bar{A}(B\overline{C_{in}} + \bar{B}C_{in})$

$= \bar{A}.(B \oplus C_{in})$

$ABC_{in} + A\overline{B}\overline{C_{in}}$

$= A.(BC_{in} + \overline{B}\overline{C_{in}})$

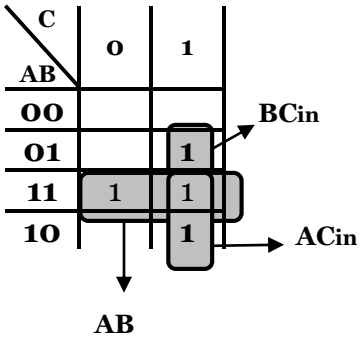
$= A.\overline{(B \oplus C_{in})}$

Note : Let X=(B $\oplus$ C_{in}) then

$\therefore \sum = \bar{A}X + A\bar{X} \longrightarrow \sum = A \oplus X \quad \therefore \sum = A \oplus B \oplus C_{in}$

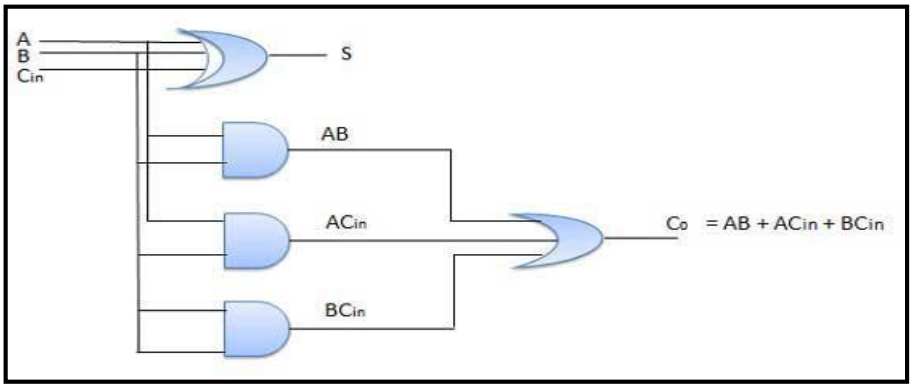
By the same way

$C_o = \sum(3,5,6,7)$



$$\therefore C_o = \sum BC_{in} + AB + AC_{in}$$

Circuit Diagram



Homework:

How to construct Full -Adder from two Half –Adders ?

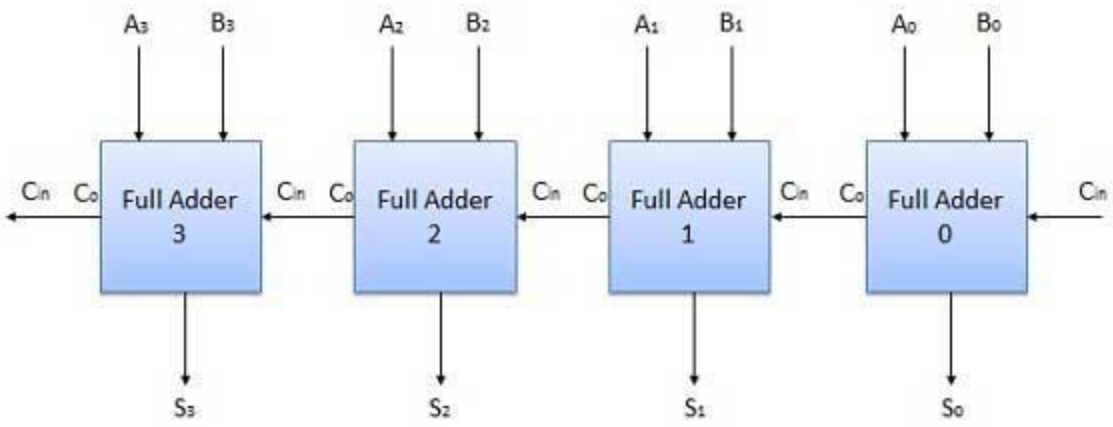
- **N- bit Adder**

The Full Adder is capable of adding only two single digit binary number along with a carry input. But in practical we need to add binary numbers which are much longer than just one bit. To add two n-bit binary numbers we need to use the n-bit parallel adder. It uses a number of full adders in cascade. The carry output of the previous full adder is connected to carry input of the next full adder.

- **4 Bit Parallel Adder**

In the block diagram, A0 and B0 represent the LSB of the four bit words A and B. Hence Full Adder-0 is the lowest stage. Hence its Cin has been permanently made 0. The rest of the connections are exactly same as those of n-bit parallel adder are shown in fig. The four bit parallel adder is a very common logic circuit.

Block Diagram



Note : we can replace the first full adder with Half adder

2- Subtractors

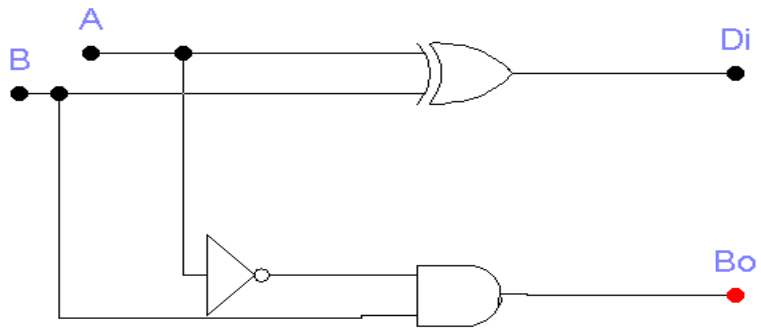
- **half subtractors**

You will find that adders and suntractors are very similar. You use half subtractors and full subtractors just as you use half and full adders. Converting the rules to truth table from as in bellow .

Inputs		Outputs	
A	B	D_i	B_o
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0
A - B		Difference	Borrow out

On the input side,(B) is subtracted from(A) to give output D_i(Diffrence). If B is larger then A , we need a borrow , which is shown in the column labeled B_o (borrow out).

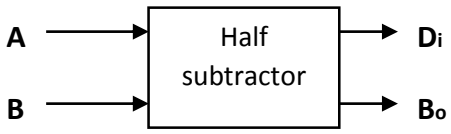
* form the truth table, we can determine the Boolean expression for the half-subtractor.



The expression for the Di column is : **Di= A ⊕ B** , this is the same as for the half adder .

The Boolean expression for the B_o column is $B_o = \overline{A}, B$, combining these two expression in a logic diagram gives the logic circuit for a half subtractor.

Block Diagram



- **Full subtractors**

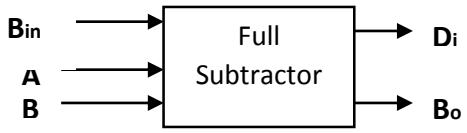
When you subtract several columns of binary digits, you must take into account the borrowing . the truth table that describe the full subtractors as fellow:

Input			Outputs	
A	**B**	**B_{in}**	**D_i**	**B_o**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0

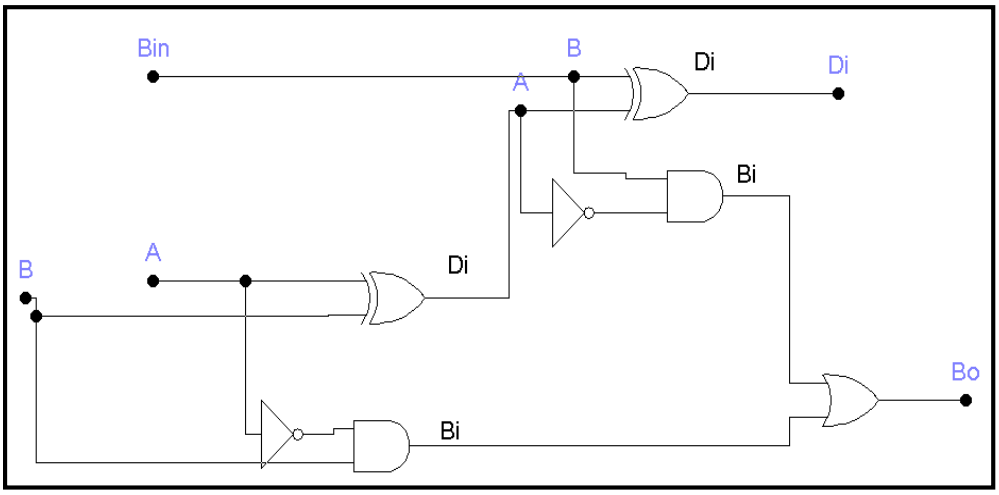
1	1	0	0	0
1	1	1	1	1
A - B - Bin			D_i	B_o

You might keep track of the differences and borrow . to solve such problem we must use full subtractor which has pervious truth table that considers all the possible combinations in Binary subtraction .

Block Diagram



Logic circuit



- **Parallel Subtractors**

Half and full subtractors are wired together to perform as a parallel subtractor , foe example , to subtract binary number B3 B2 B1 B0 from binary number A3 A2 A1 A0 we make a 4-bit parallel subtractors.

A3 A2 A1 A0

- B3 B2 B1 B0