1. Definition

Boolean algebra derives its name from the English mathematician George Boole 1815 - 1864, who published two major books, "*The Mathematical Analysis of Logic"* (1847) and "*The Laws of Thought* (1854)", where he founded the Mathematical theories of logic and probabilities.

Boolean algebra was first used by Claude E. Shannon (research assistant at the Massachusetts Institute of Technology) for the design of relay switching circuits in 1938. Instead of elementary algebra where the values of the variables are numbers, Boolean algebra deals only with binary number system 0 or 1 (false or true). Boolean algebra is very useful in designing logic circuits used in computers.

2. Fondamental concepts

2.1 Definition

Boolean algebra is defined with a set of elements (Boolean variables), a set of operators (+ or . and not $\overline{\ }$), and a number of postulates.

2.2 Boolean variable

A Boolean variable has two values, either 0 or 1.

2.3 Boolean function

A Boolean function is an expression formed with combination of Boolean variables Connected by Boolean Operators (+ or . and not $\overline{\ }$). The value of a function may be 0 or 1, depending on its variables' values.

***Example**:* let *A, B, C* three Boolean variables, the following expressions are Boolean functions.

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

2.4 Principle of duality

To form the dual of an Boolean expression we need to:

- Changing each OR (+) to an AND (.)
- Changing each AND (.) to an OR (+)
- Replacing each 0 by 1 and each 1 by 0

***Example**:* the dual of **1+1=1** is **0.0=0**

Note : Each postulate, each theorem and each expression of Boolean algebra has a dual equivalent, where the 0s are replaced by 1s, the 1s by 0s, the (.) by (+) and (+) by (.).

2.5 Postulates

Postulates are assumed to be true without any proof or demonstration.

	Postulate	**Dual Postulate**
P1	0+0=0	1.1=1

P2	1+1=1	0.0=0
P3	1+0=0+1=1	0.1=1.0=0
P4	$\overline{0}=1$	$\overline{1}=0$

2.6 Boolean Theorems

Assum A, B and C three Boolean variables:

Associative law	A+B)+C= A+(B+C)=A+B+C	(A.B).C=A.(B.C)=A.B.C
Commutative law	A+B=B+A	A.B=B.A
Distributive law	A.(B+C)= A.B + A.C	A+(B.C)=(A+B).(A+C)
Identity element	A+0=A	A.1=A
Complement Law	$A+\overline{A}=1$	$A.\overline{A}=0$
Involution	$\overline{\overline{A}}=A$	
Idempotence law	A+A+A+……+A = A	A.A.A……A=A
De Morgan's laws	$\overline{A+B}=\overline{A}.\overline{B}$	$\overline{A.B}=\overline{A}+\overline{B}$
Absorption law	A+A.B=A	A.(A+B)=A

Note :

De Morgan's laws can be extended for *n* variables as:

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

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

Note:

If a Boolean theorem/equality is proved, its dual automatically holds and need not to be proved separately.

2.7 Truth table

The truth table of a Boolean function is a table that gives the results (or outputs) of all possible input variables. For an **n** number of variables, 2^n combinations of inputs are arranged in columns on the left and the output result is listed in the rightmost column.

Example: construct the truth table of the Boolean function F

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

A	B	C	$\overline{B}$	$A.\overline{B}$	A.B.C	F(A,B,C)
0	0	0	1	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	1	1	0	1
1	0	1	1	1	0	1
1	1	0	0	0	0	0
1	1	1	0	0	1	1

Note:

A truth table can be used to prove Boolean algebra theorems and to determine if two Boolean functions are equivalent or not.

Example: verify the following equality using the truth table

A+A.B=A

A	B	A.B	A+A.B
0	0	0	**0**
0	1	0	**0**
1	0	0	**1**
1	1	1	**1**

2.8 Logic gates

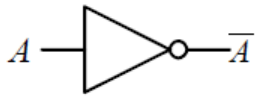
Logic gates are a basic building blocks of the electronic circuits, that have one (or more) input and only one output. The graphic symbol of each logic gate will be presented later in this chapter.

2.9 Boolean operators

There are three basic operators AND, Or, NOT and other derived operators (NAND, NOR, XOR…..) that are combinations of the basic operations. For each operator we will present the truth table and the corresponding logic gate.

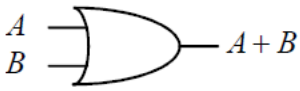
- **NOT (inverter)**

A	$\overline{A}$
0	1
1	0



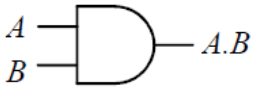
- **OR**

A	B	$A+B$
0	0	0
0	1	1
1	0	1
1	1	1



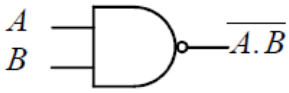
- **AND**

A	B	$A.B$
0	0	0
0	1	0
1	0	0
1	1	1



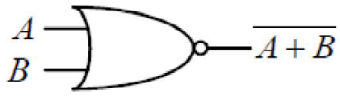
- **NAND**

A	B	$\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0



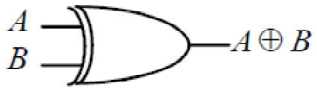
- **NOR**

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



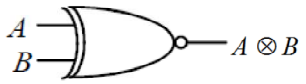
- **XOR** (Exclusive OR)

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0



- **XNOR**

A	B	$A \otimes B$
0	0	1
0	1	0
1	0	0
1	1	1



- *Properties of XOR*

$$A \oplus B = \overline{A}.B + A.\overline{B}$$
$$A \oplus B = \overline{A \otimes B}$$
$$A \oplus B = B \oplus A$$
$$A \oplus (B \oplus C) = (A \oplus B) \oplus C$$
$$A \oplus B = \overline{A} \oplus \overline{B}$$

- *Properties of XNOR*

$$A \otimes B = A.B + \overline{A}.\overline{B}$$
$$A \otimes B = \overline{A \oplus B}$$
$$A \otimes B = B \otimes A$$
$$A \otimes (B \otimes C) = (A \otimes B) \otimes C$$
$$A \otimes B = \overline{A} \otimes \overline{B}$$

2.10 Operators precedence

From highest to lowest precedence: NOT AND OR. If there are several logical operators of the same precedence, they will be examined left to right. Note that we must evaluate bracketed expressions first, if they exist.

3. Representation of Boolean Function

There are many equivalent representations of a Boolean function like: truth table, algebraic form (canonical form), numerical form, logic diagram , Karnaugh map, Venn diagram ….

3.1 Truth table

See the section 2.7

3.2 Algebraic form

The Boolean function is expressed in terms from complemented or uncomplemented variables connected with basic Boolean operators (+ or . and not $\overline{\ }$).

Example: $f(A,B) = A\overline{B} + AB + \overline{A}B$

- **Minterm :**

A minterm (called also standard product) is a product of all **n** variables of the function either complemented or uncomplemented.

Example: $\overline{A}.B.\overline{C}.D \quad A.B.C.D \quad \overline{A}.B.\overline{C}.\overline{D}$ 3 minterms for a function of 4 variables.

- **Maxterm :**

A maxterm (called also standard sum) is a sum of all **n** variables of the function either complemented or uncomplemented.

Example: $\overline{A}+B+\overline{C}+D \qquad A+B+C+D \qquad \overline{A}+\overline{B}+\overline{C}+\overline{D}$ 3 maxterms for a function of 4 variables.

3.2.1 Disjunctive canonical form (DCF)

Called also *sum of minterms*. The Boolean function is expressed as the logical sum of all the minterms for which the value of the function is 1 in the truth table.

3.2.2 Conjunctive canonical form (CCF)

Called also *product of maxterms*. The Boolean function is expressed as the logical product of all the maxterms for which the value of the function is 0 in the truth table.

Note: Disjunctive canonical form (DCF) and Conjunctive canonical form (CCF) are equivalent.

Example: Let F(A,B,C) a Boolean function represented by the following truth table.

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

1- Show all minterms and maxterms

2- Write the function F in DCF and CCF

A	B	C		F		
0	0	0		0	$A+B+C$	***Maxterm***
0	0	1		0	$A+B+\overline{C}$	***Maxterm***
0	1	0		0	$A+\overline{B}+C$	***Maxterm***
0	1	1		1	$\overline{A}.B.C$	***minterm***
1	0	0		0	$\overline{A}+B+C$	***Maxterm***
1	0	1		1	$A.\overline{B}.C$	***minterm***
1	1	0		1	$A.B.\overline{C}$	***minterm***
1	1	1		1	$A.B.C$	***minterm***

(DCF) ➔ $F(A,B,C)=\overline{A}.B.C + A.\overline{B}.C + A.B.\overline{C} + A.B.C$

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

Note:

Any Boolean function can be expressed as a DCF or CCF. The simplest way is to draw the truth table and write the two canonical forms.

Example: consider the following Boolean function $F(A,B)=\overline{(A+B).(\overline{A}+\overline{B})}$

Write the function F in DCF and CCF.

Answer (seen in the course)

(DCF) ➔ $F(A,B)=\overline{A}.\overline{B}+A.B$

(CCF) ➔ $F(A,B)=(A+\overline{B}).(\overline{A}+B)$

Note:

If there is a minterm/ maxterm that doesn't contain all **n** variables, the form is called SOP (sum of products)/ POS (product of sums).

To convert SOP/POS to DCF/CCF respectively we use the following rules:

1. Examine the missing variables in each product/ sum which is not a minterm/maxterm.
2. Multiply that product with 1 (**e.g** : $X+\overline{X}$)
3. Add 0 (**e.g** : $X.\overline{X}$) to that sum
4. Use the distributive laws
5. Remove the redundant terms if necessary

Example: find the DCF and CCF respectively for the following functions

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

Answer

$$F(A,B,C) = A.\overline{B} = A.\overline{B}.1 = A.\overline{B}.(\mathrm{C}+\overline{\mathrm{C}}) = \mathrm{A}.\overline{B}.\mathrm{C}+\mathrm{A}.\overline{B}.\overline{\mathrm{C}}$$

$$G(A,B,C) = (A+B) = (A+B+0) = (A+B+C.\overline{\mathrm{C}}) = (\mathrm{A}+\mathrm{B}+\mathrm{C}).(\mathrm{A}+\mathrm{B}+\overline{\mathrm{C}})$$

3.3 Numerical form

This form is used as a short notation, where each minterm/maxterm is replaced by the decimal equivalent.

Example: the numerical forms of the function F (section 5.2.2) are:

F(A,B,C,D)= $\sum$ (3,5,6,7)

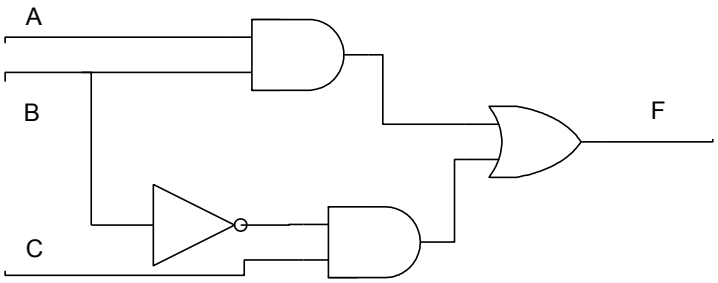
F(A,B,C,D)= $\prod$ (0,1,2,4)

3.4 Logic diagram

The logic diagram is a graphical representation of a Boolean function, it consists in replacing each logic operator by the corresponding logic gate.

Example: Draw the logic diagram of the following function.

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



4. Universal gates

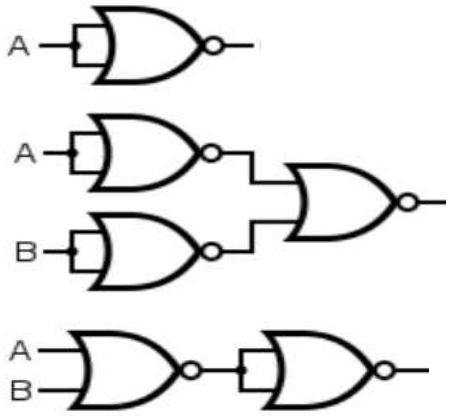
NAND gates and NOR gates are called universal gates, because any Boolean function can be implemented by using only one of these two. Universal gates allow reducing the circuit design complexity by reducing the number of different gate types required, and also reducing the number of transistors needed (minimize manufacturing costs).

- Basic logic operators (Not AND OR) are implemented using only **NOR** gates

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

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

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

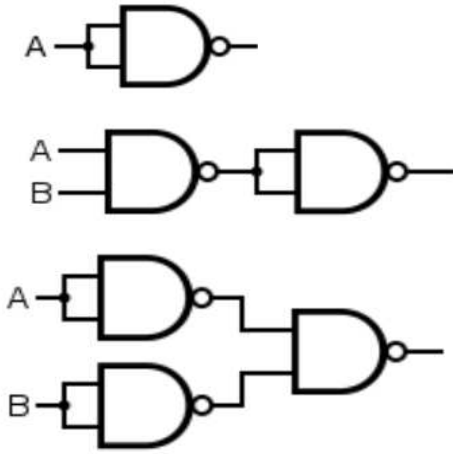


- Basic logic operators (Not AND OR) are implemented using only **NAND** gates

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

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

$$A + B = \overline{\overline{A + B}} = \overline{\overline{A}.\overline{B}} = \overline{\overline{AA}.\overline{BB}}$$



***Example**:*

1. Express the following expression with only NAND operator $A.B + C.\overline{D}$

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

2. Express the following expression with only NOR operator $(\mathrm{A} + B + C)(A + D)$

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

Note:

Generally, we prefer SOP (Sum Of Products) form to design the digital circuits using only NAND gates and POS (Product Of Sums) form to design the digital circuit using only NOR gates.

5. Minimization of Boolean functions

The objective of simplifying logic functions is to reduce the number of terms (reduce the number of gates) to obtain smaller, faster and cheaper circuit.

5.1 Algebraic minimization

It consists in applying the theorems/laws of Boolean algebra (see sections 2.5 and 2.6) in order to reduce the number of variables or terms.

Useful simplifications

$Rule1:$	$A + \overline{A} = 1$	$A.\overline{A} = 0$
$Rule2:$	$A.B + A.\overline{B} = A$	$(A + B)(A + \overline{B}) = A$
$Rule3:$	$A + AB = A$	$A(A + B) = A$
$Rule4:$	$A + \overline{A}B = A + B$	$A(\overline{A} + B) = AB$

Example: minimize the following Boolean function using algebraic manipulation

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

$= A.\overline{B} + B.(\overline{C} + C)$ using rule1

$= A.\overline{B} + B$ using rule4

$= A + B$

2. $F(A,B,C) = ABC + AB\overline{C} + A\overline{B}CD$

$= AB(C + \overline{C}) + A\overline{B}CD$ using rule1

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

$= A(B + \overline{B}(CD))$ using rule4

$= A(B + CD)$

3. Add an existing term

$$F(A,B,C) = A.B.C + \overline{A}.B.C + A.\overline{B}.C + A.B.\overline{C}$$
$$= A.B.C + \overline{A}.B.C + A.B.C + A.\overline{B}.C + A.B.C + A.B.\overline{C}$$
$$= B.C.(A+\overline{A}) + A.C..(B+\overline{B}) + A.B.(C+\overline{C}) \quad \text{using rule1}$$
$$= B.C + A.C + A.B$$

4. Simplify the complement of a function

$$F(A,B,C) = \sum (2,3,4,5,6,7)$$
$$\overline{F(A,B,C)} = \sum (0,1)$$
$$= \overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C$$
$$= \overline{A}.\overline{B}.(C+\overline{C}) \quad \text{using rule1}$$
$$= \overline{A}.\overline{B}$$
$$F(A,B,C) = \overline{\overline{A}.\overline{B}} = A + B \quad \text{using De Morgan's law}$$

5.2 Karnaugh Map

The algebraic simplification method becomes very difficult and cumbersome if the number of variables or terms increase. Karnaugh's method is a faster and can be used to solve Boolean functions of up to 6 variables.

- **Adjacency principle**

Two Boolean terms are adjacent when they contain the same variables and differ in the form of exactly one variable.

Example:

The following terms are adjacent

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

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

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

The following terms are not adjacent

A.B + $\overline{A}.\overline{B}$

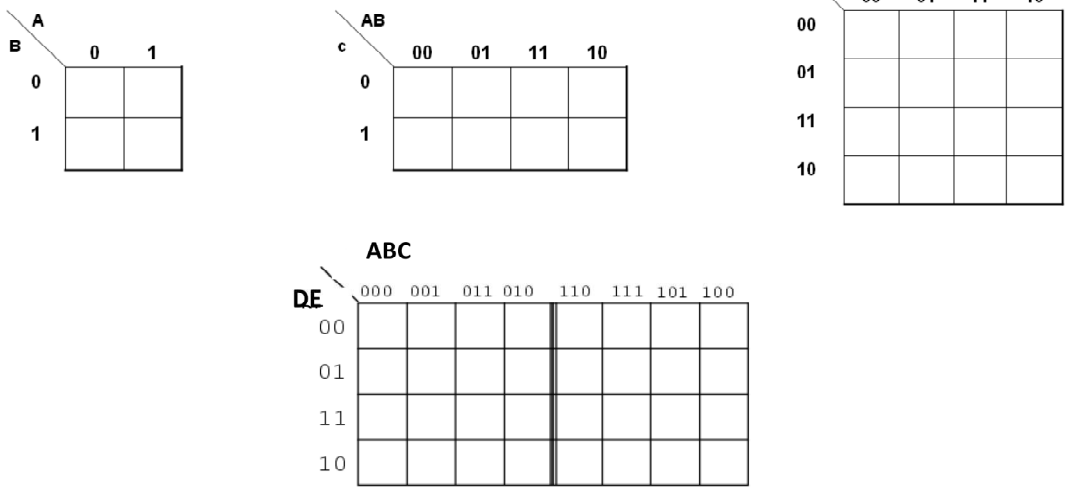
A.B.C + A.$\overline{B}.\overline{C}$

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

- **Karnaugh's principle**

- A Karnaugh map is a graphical form of a truth table consists of adjacent cells.
- The number of cells equal to $\mathbf{2^N}$, where **N** is the number of variables.
- Rows and columns are labeled using Gray code.
- Karnaugh maps for 2, 3, 4 and 5 variables are shown below.

-



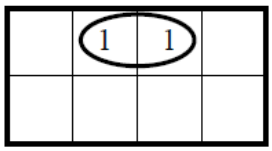
- Each minterm → cell =1 , Each maxterm → cell =0
- For simplicity, the maxterms (0's) are omitted
- To make implementation of Karnaugh map easier, the function must be represented in one of the two canonical forms (DCF or CCF).

- **Simplification rules**

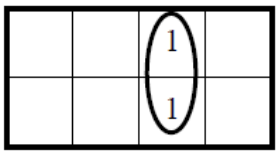
- Identify the adjacent cells containing **1** and make them in group of of 2^N (32, 16, 8, 4, 2, 1).
- Make the largest possible group of 1's. ***E.g.*** 1 group of 4 instead of 2 groups of 2.
- One or more cells can be common to several groupings.
- The objective is to cover all the 1's on the map with minimum number of groups (fewer terms) and to which contain the maximum cells of 1(fewer variables).
- Two cells are adjacent when they are located side by side horizontally or vertically
- Two cells located at the ends of the same row or of the same column are adjacent (cylindrical shape)
- The four corner cells are adjacent
- Diagonal cells are not adjacent

- **Writing simplified expression**

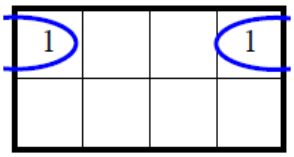
- Moving left to right or up to down in a group. The variables which change are ignored.
  - o A group of 2 1's eliminates one variable.
  - o A group of 4 1's eliminates two variables.
  - o A group of 8 1's eliminates three variables, and so on.
- Combine the unchanged variables by AND (.) operator
- The final expression is the sum (OR operator) of the previous terms.
- For the 1's which are not grouped, write the complet correcsponding term



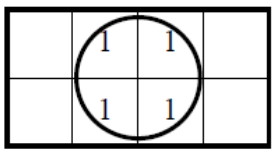
$F = B\overline{C}$



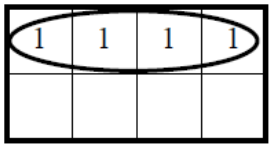
$F = AB$



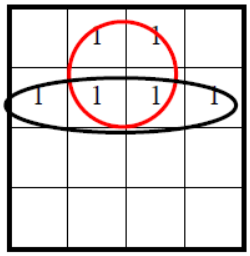
$F = \overline{B}\,\overline{C}$



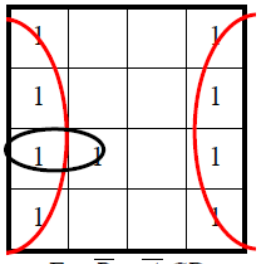
$F = B$



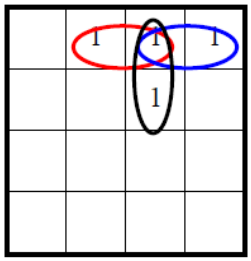
$F = \overline{C}$



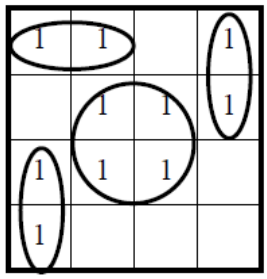
$F = B\overline{C} + \overline{C}D$



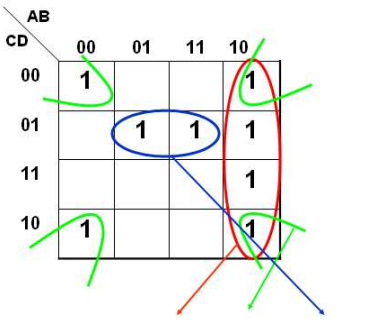
$F = \overline{B} + \overline{A}\,CD$



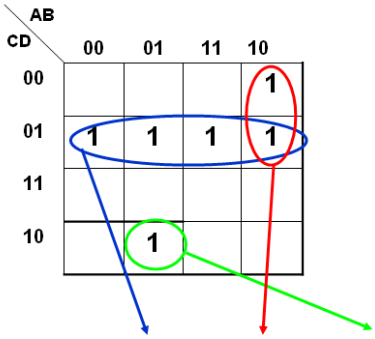
$= B\overline{C}\overline{D} + A\overline{C}\overline{D} + AB\overline{C}$



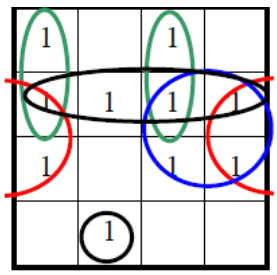
$= BD + \overline{A}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,C$



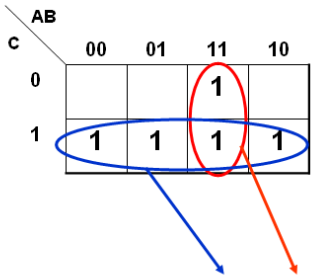
$F(A,B,C,D) = A\overline{B} + \overline{B}\overline{D} + B\overline{C}D$



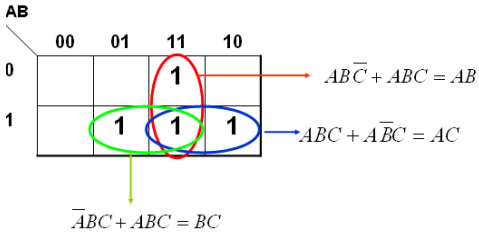
$F(A,B,C,D) = \overline{C}.D + A.\overline{B}.\overline{C} + \overline{A}.B.C.\overline{D}$



$= \overline{C}D + AD + \overline{B}D + \overline{A}\,\overline{B}\,\overline{C} + AB\overline{C} + \overline{A}\,BC\overline{D}$

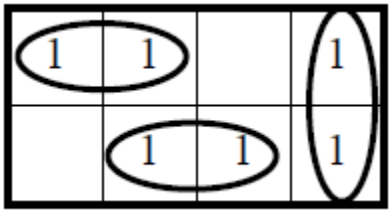


$F(A,B,C) = C + AB$

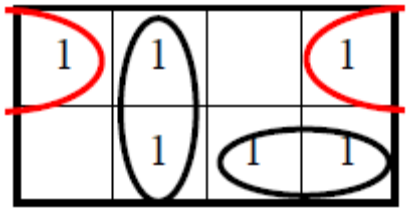


$F(A,B,C) = AB + AC + BC$

Note: grouping may not be unique, i.e. We can make grouping in more than one way.



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



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

Note: we can use maxterms instead of minterms (regrouping 0's)

1	1	1	1
1	1	1	1
1	0	0	1
1	1	1	1

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

- **Dont' Care states**

A function is said to be incompletely specified when its value is indifferent (we **don't care** what value may take on) or does not exist (never occurring) for certain combinations of input variables. We use the symbol **X** or **Ø** for **don't care** states.

In the Karnaugh table, the symbol X can take either a 1 or 0 indifferently, so we replace by 1 only those which helps in making a large group.

Example:

		X	
1	1		1
X	X	X	X
1	1	X	X

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

References

Library of faculty

1. Architecture des ordinateurs. Jean-Jacques Schwarz 2005
2. Architecture et technologie des ordinateurs. Zanella, Paolo 2005
3. Architecture des ordinateurs. Philippe Darche 2002
4. Architecture des machines et des systŁmes informatiques. Alain Cazes 2003
5. Architecture de l'ordinateur. Robert Strandh 2005
6. Architecture de l'ordinateur. Andrew Tanenbaum 2005
7. De l'AlgŁbre de Boole aux circuits numériques. Karima Khadidja Mokhtari 2014
8. Logique combinatoire et composants numériques. Mouloud Sbai 2013
9. Introduction aux circuits logiques. Letocha Jean 1985 (BibliothŁque centrale)

Downloadable books

1. D A Patterson & J L Hennessy, Computer Organization and Design : The hardware/software interface, Morgan-Kaufmann (Fifth edition) 2013
2. A S Tanenbaum and T Austin, Structured Computer Organization, Pearson (International edition), 2012
3. R E Bryant & D R O'Hallaron, Computer Systems : A Programmer's Perspective, Pearson (Global edition) 2015
4. Computer Organization and Architecture 8TH EDITION by William Stallings. Prentice Hall, Inc.,2010
5. Computer System Architecture (3rd Edition) M. Morris Mano 1992