UNIT 2

BOOLEAN ALGEBRA AND LOGIC GATES

Logic gates and logic operators

- **Logic**- certain declarative statement to state that the condition is whether ***true*** **or** ***false.***
- ***True-*** **condition fulfilled**
- ***False-*** **condition not met**
- ***The manner in which the digital circuits responds to an input is referred to as circuit's logic.***
- ***Logic gates*****- building blocks of digital system, manipulates binary information.**
- ***Gates*****- the logic circuits having more than one input and only one output .**

- **To simplify the logical expression, used in digital circuits, we need to use logical operators.**
- **Three types:**
    1. ***AND operator (A.B → logical multiplication)***
    2. ***OR operator (A+B → logical addition)***
    3. ***NOT operator (A→ A' logical inversion)***
- **The operation of logic gates are best understood by the help of *Truth table*.**
- **Logic gates can be categorized as:**
    1. **Basic gates: AND, OR, NOT**
    2. **Universal gates: NAND, NOR**
    3. **Special purpose gates: EX-OR, EX-NOR**

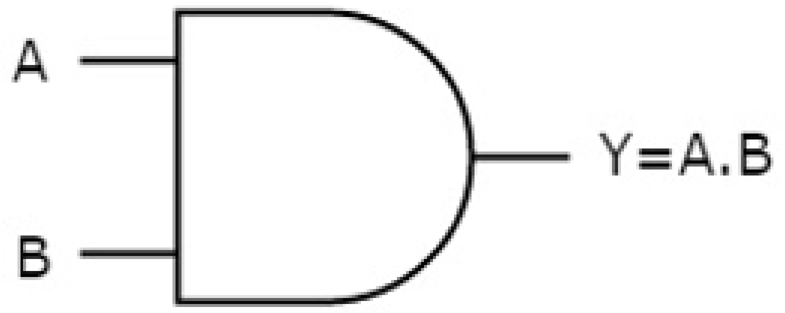
Basic Gates

- The basic gates are AND, OR & NOT gates.

AND gate:

- A digital circuit that has two or more inputs and produces an output.
- It is optional to represent the **Logical AND** with the symbol '.'.
- ***Truth table*** and ***Logic diagram*** of 2-input AND gate:

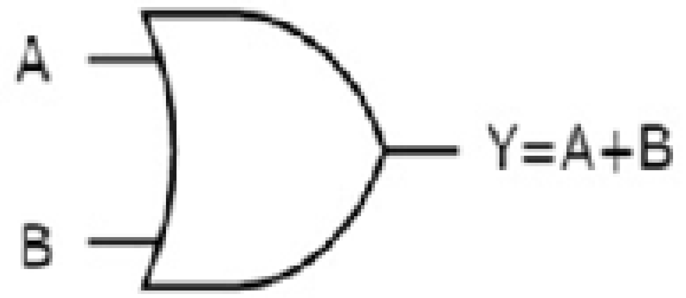
A	B	Y = A.B
0	0	0
0	1	0
1	0	0
1	1	1



OR gate

- An OR gate is a digital circuit that has two or more inputs and produces an output.
- This **logical or** is represented with the symbol ‘+’.
- **Truth table** of 2-input or gate.

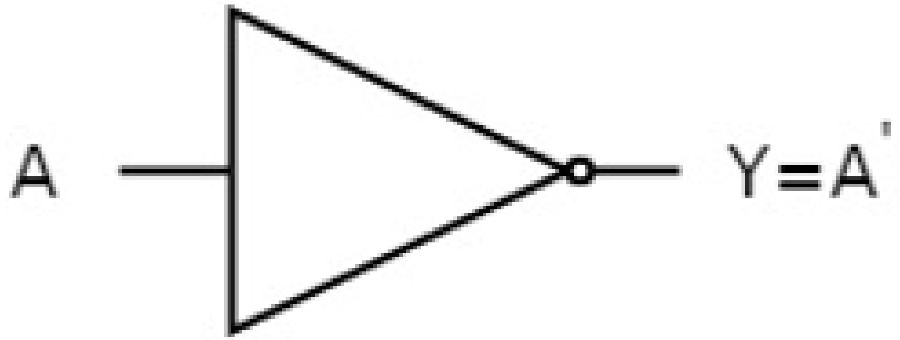
A	B	Y = A + B
0	0	0
0	1	1
1	0	1
1	1	1



NOT gate

- A NOT gate is a digital circuit that has single input and single output.
- The output of NOT gate is the **logical inversion** of input.
- Hence, the NOT gate is also called as inverter.
- **Truth table** of NOT gate.

A	Y = A’
0	1
1	0



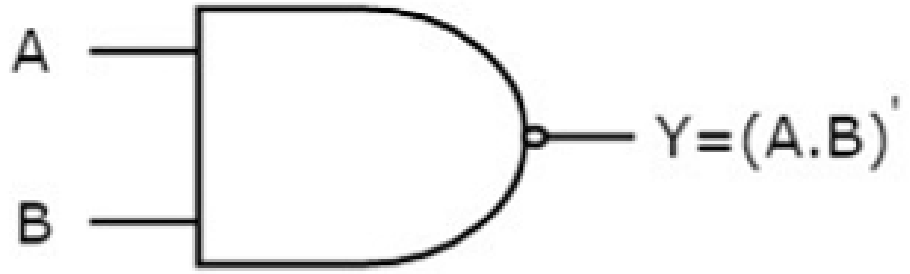
UNIVERSAL GATES

- **NAND** & **NOR** gates are called as **universal gates**.
- WE can implement any Boolean function, which is in ***sum of products (SOP)*** form by using ***NAND gates alone***.
- we can implement any Boolean function, which is in ***product of sums (POS)*** form by using ***NOR gates alone***.

NAND gate

- NAND gate is a digital circuit that has two or more inputs and produces an output.
- Combination of ***AND gate*** and ***NOT gate***.
- **Inversion of logical AND gate.**
- Equivalent to ***AND gate*** followed by ***NOT gate***.

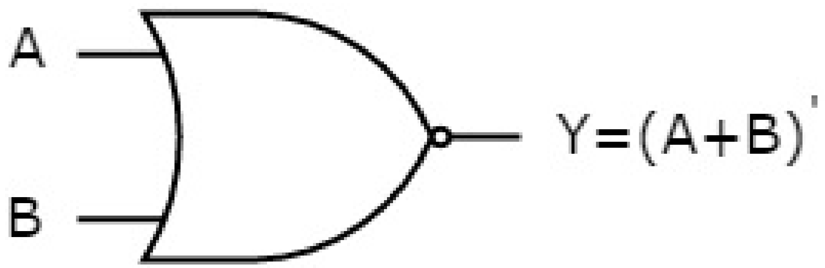
A	B	Y = (A.B)’
0	0	1
0	1	1
1	0	1
1	1	0



NOR gate

- NOR gate is a digital circuit that has two or more inputs and produces an output
- **Inversion of logical OR**.
- ***OR gate followed by NOT gate.***

A	B	Y = (A+B)’
0	0	1
0	1	0
1	0	0
1	1	0



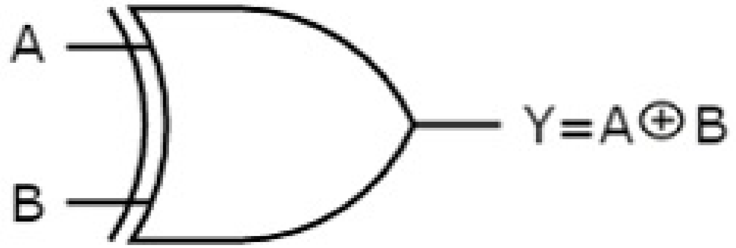
Special Purpose Gates

- ***X-OR*** & ***X-NOR*** gates are called as special purpose gates.
- Special cases of ***OR*** & ***NOR*** gates.

X-OR gate:

- The full form of X-OR gate is **Exclusive-OR** gate.
- Its function is same as that of OR gate except for some cases, when the inputs having even number of ones.
- The output of X-OR gate is also called as an **odd function**.

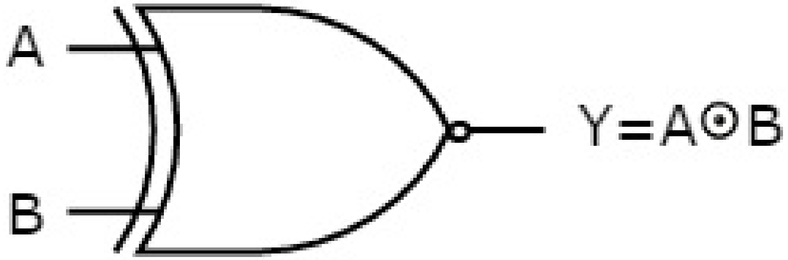
A	B	$Y = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

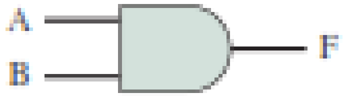
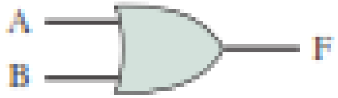
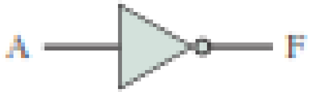
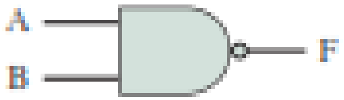
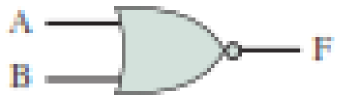
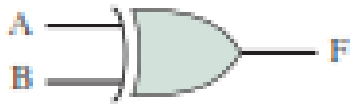


Exclusive-NOR gate:

- The full form of ***X-NOR*** gate is **exclusive-nor** gate.
- Its function is same as that of NOR gate except for some cases, when the inputs having even number of ones.
- The output of ***X-NOR*** gate is also called as an **even function**.

A	B	Y = A⊙B
0	0	1
0	1	0
1	0	0
1	1	1



<table>
<tr><th>Name</th><th>Graphical Symbol</th><th>Algebraic Function</th><th>Truth Table</th></tr>
<tr><td>AND</td><td></td><td>$F = A \cdot B$
or
$F = AB$</td><td>A B | F
0 0 | 0
0 1 | 0
1 0 | 0
1 1 | 1</td></tr>
<tr><td>OR</td><td></td><td>$F = A + B$</td><td>A B | F
0 0 | 0
0 1 | 1
1 0 | 1
1 1 | 1</td></tr>
<tr><td>NOT</td><td></td><td>$F = \overline{A}$
or
$F = A'$</td><td>A | F
0 | 1
1 | 0</td></tr>
<tr><td>NAND</td><td></td><td>$F = \overline{AB}$</td><td>A B | F
0 0 | 1
0 1 | 1
1 0 | 1
1 1 | 0</td></tr>
<tr><td>NOR</td><td></td><td>$F = \overline{A + B}$</td><td>A B | F
0 0 | 1
0 1 | 0
1 0 | 0
1 1 | 0</td></tr>
<tr><td>XOR</td><td></td><td>$F = A \oplus B$</td><td>A B | F
0 0 | 0
0 1 | 1
1 0 | 1
1 1 | 0</td></tr>
</table>

Boolean Algebra

- Introduced by Gorge Boole in 1854.
- Used for systematic treatment of logic.
- Two valued Boolean algebra called switching algebra, demonstrates the bi-stable electrical system.
- Deals with binary system.
- Useful for designing logical circuits, used in processor of computer.
- Logic gates are the building block of modern computer.

Rules of Boolean Algebra

- Variables can have only two values, ***'0' for low*** and ***'1' for high***.
- ***Complement*** of the variable is represented by " ' " or " ‾ ".

  e.g., complement of B is B' or $\overline{B}$. If B=0 then $\overline{B}$=1 and vice versa.
- ORing of variables is represented by (+) sign between them.

  e.g, ORing of A,B is A+B and so on.
- ANDing of variables is represented by (.) sign between them.

  e.g, ANDing of A, b=B is A.B and so on.

Basic Boolean Laws are:

1. Commutative laws
2. Associative laws
3. Distributive laws

1. Commutative laws

- Law of addition of two variables is written as,

  $A+B=B+A$

- Law of multiplication of two variables is written as,

  $A.B=B.A$

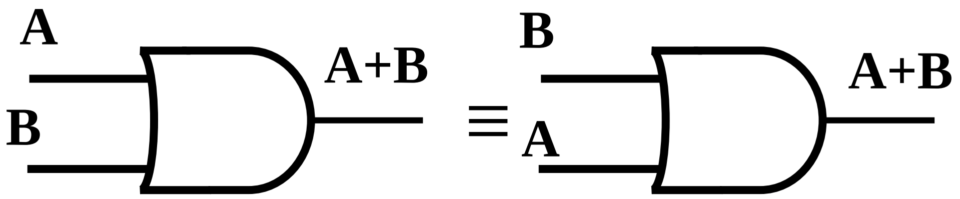


Fig: application of commutative law of addition

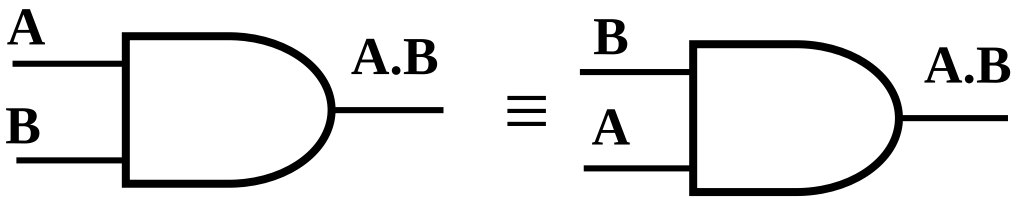


Fig: application of commutative law of multiplication

2. Associative laws

- Associative law of addition

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

- Associative law of multiplication

  $A.(B.C)=(A.B).C$

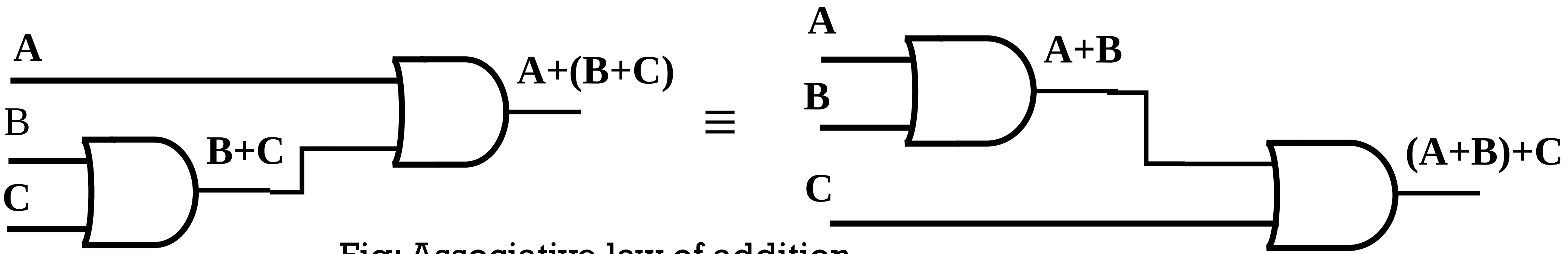


Fig: Associative law of addition

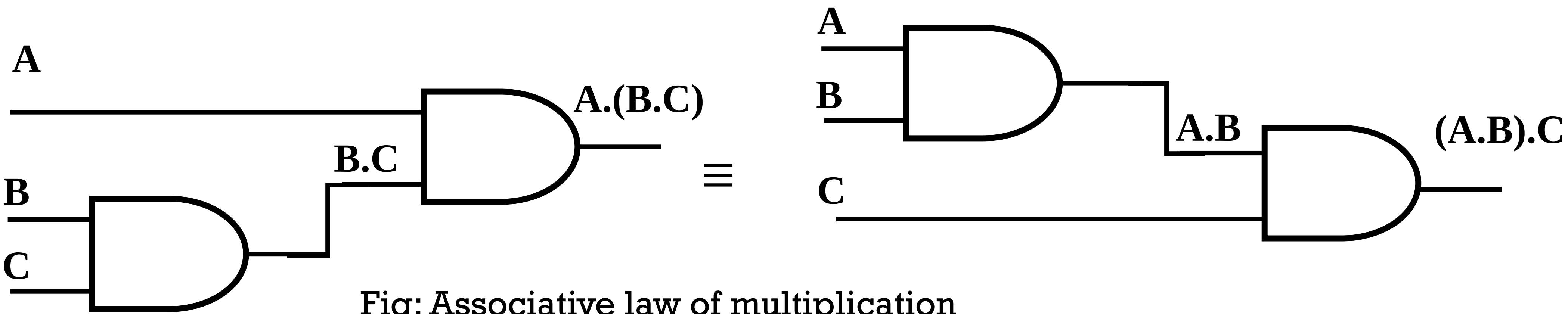


Fig: Associative law of multiplication

3. Distributive laws

- The distributive law of three variables can be defined as,

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

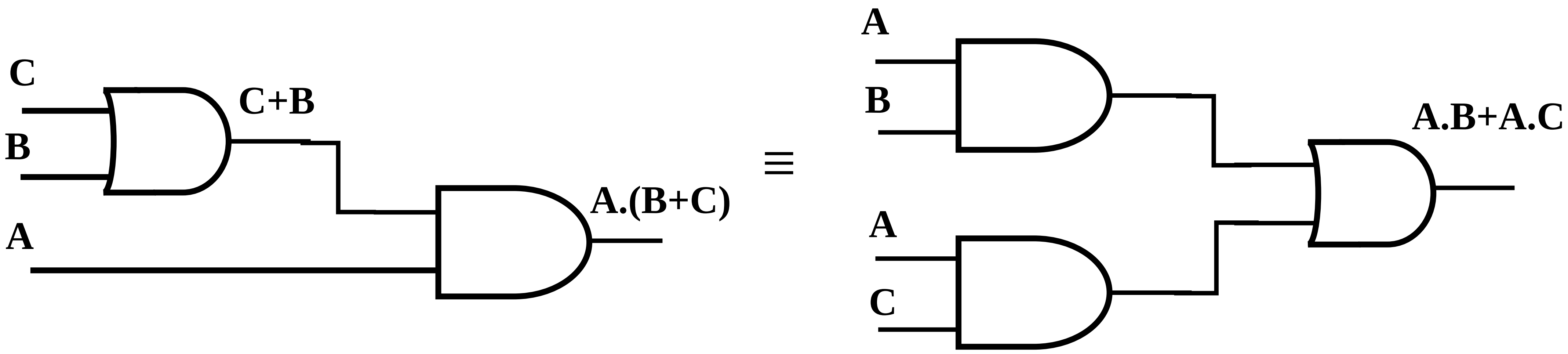


Fig: Distributive law of three variables

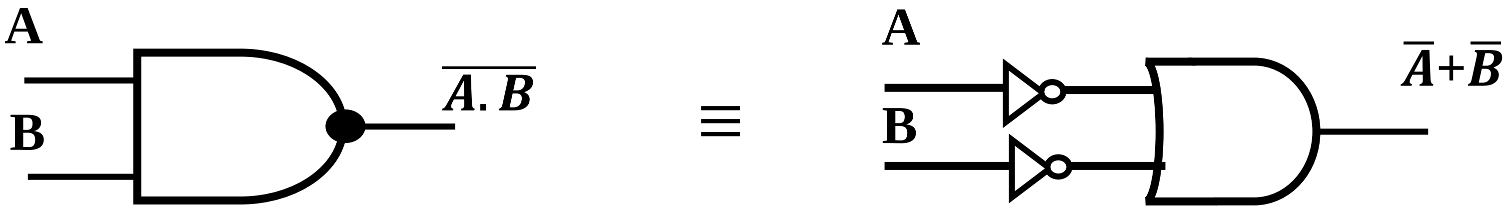
Boolean expression rules

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

De-Morgan's Theorem

- **Theorem 1:** *"The complement of the product is equal to addition of the complements".*
  - It shows that **NAND** gate is equivalent to bubbled **OR** gate.

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

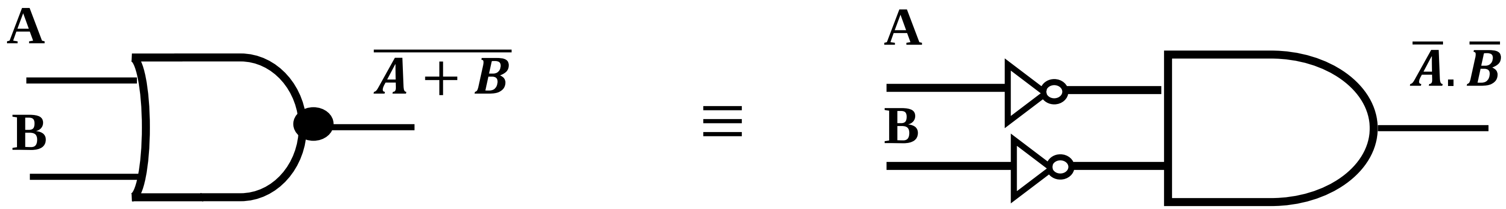


A	B	A.B	$\overline{A.B}$	$\overline{A}$	$\overline{B}$	$\overline{A}+\overline{B}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

Fig: Verification table

- **Theorem 2:** *"The complement of the* sum *is equal to* product *of the complements".*
  - It shows that **NOR** gate is equivalent to bubbled **AND** gate.

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



A	B	A+B	$\overline{A+B}$	$\bar{A}$	$\bar{B}$	$\bar{A}.\bar{B}$
0	0	0	**1**	1	1	**1**
0	1	1	**0**	1	0	**0**
1	0	1	**0**	0	1	**0**
1	1	1	**0**	0	0	**0**

Fig: Verification table

EXERCISE:

Apply De-Morgan's theorems to each of the following expressions:

Example1: ((A+B+C). D)'

Example2: ((ABC).D)'

1. **OPERATOR PRECEDENCE:**
   i. **Parentheses**
   ii. **NOT**
   iii. **AND**
   iv. **OR**

DUALITY THEOREM

- **Statement:** ***"Every algebraic expression deducible from the postulates of Boolean algebra remains valid if the operators and identity elements are interchanged".***
- There is precise duality between the operators AND and OR and the digits '0' and '1'.
- ***Step 1***:change each OR sign to an AND sign and vice-versa.
- ***Step 2:*** complement any '0' or '1' appearing in the statement.
- **Example:** For expression, A+0=A, the **dual** is A.1=A

  A(B+C)=A.B+A.C, its **dual** is (A+B)(A+C)

Realization of various logic gates using universal gates

1. Realization of various logic gates using NAND:

i. NOT gate using NAND

$$Y = (A.B)'$$
$$= (A.A)' \qquad (\because A=B)$$
$$= A'$$

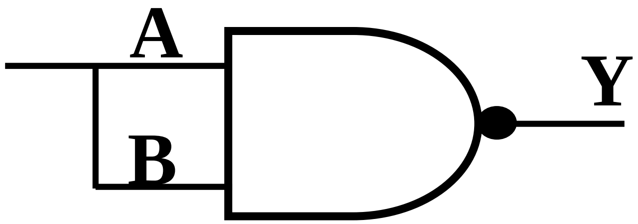


Fig: NOT gate Using NAND

ii. AND gate using NAND

$$Y1 = (A.B)'$$
$$Y=Y1' = ((A.B)')'$$
$$= A.B$$

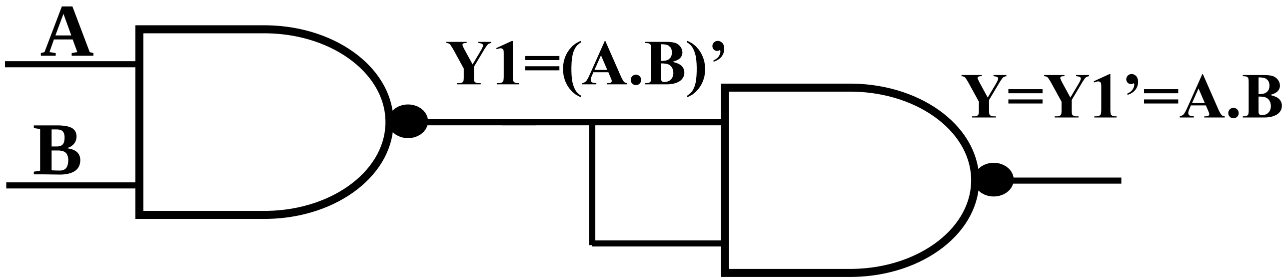


Fig: AND gate Using NAND

iii. OR gate using NAND

$Y = A + B$

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

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

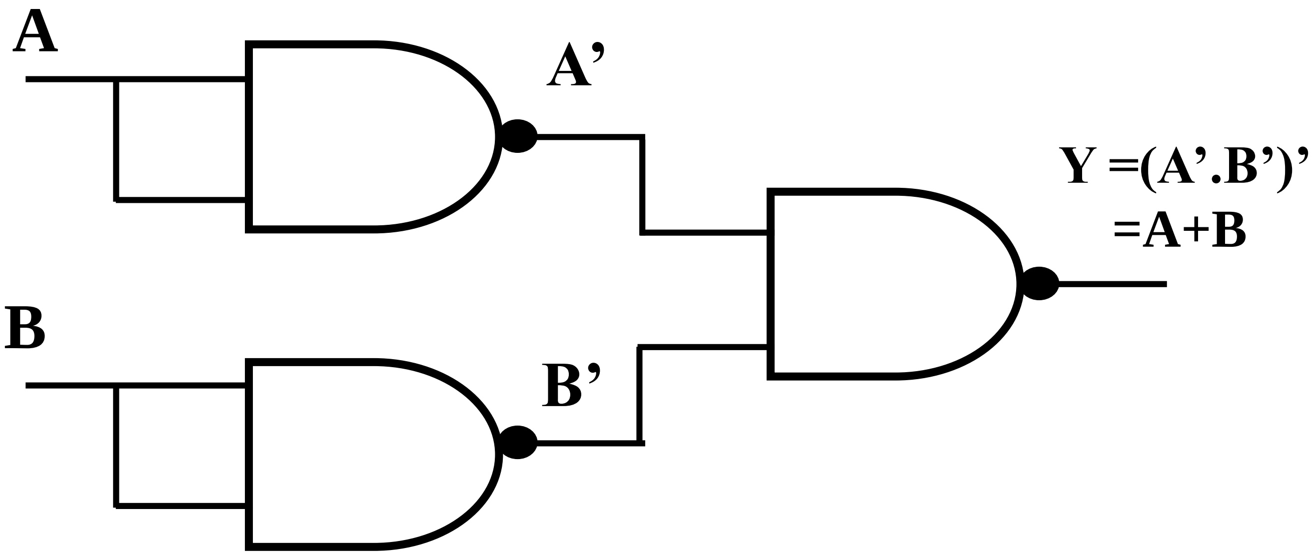


Fig: OR gate Using NAND

iv. NOR gate using NAND

Y= (A + B)’

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

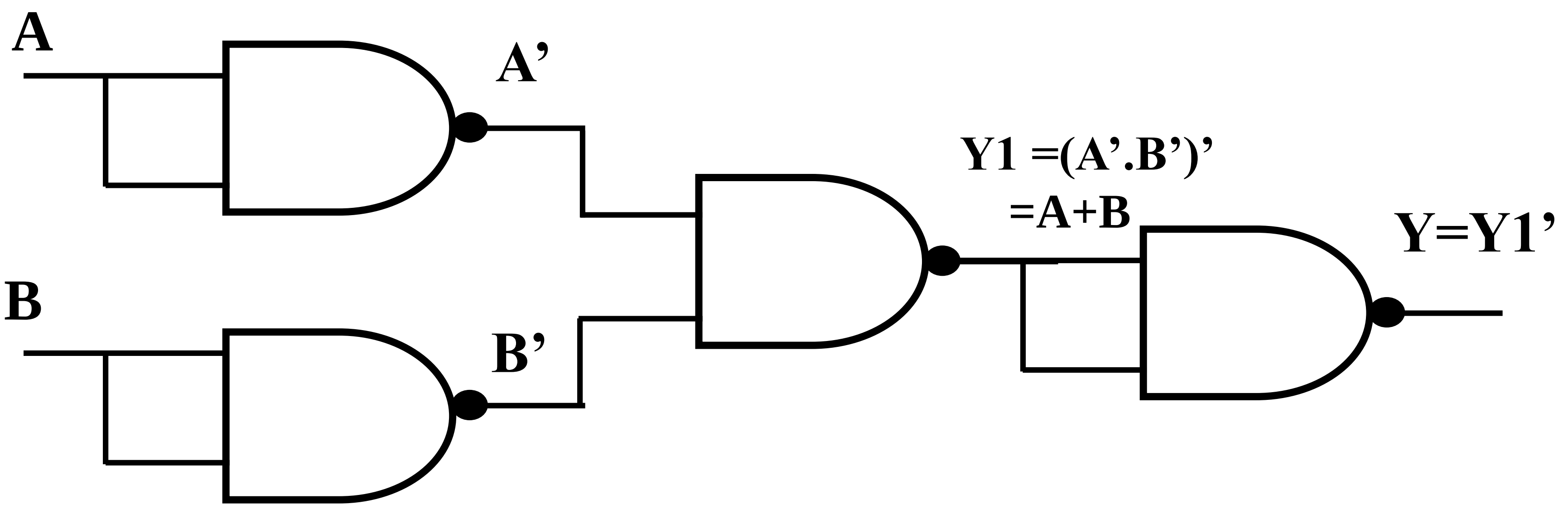


Fig: NOR gate Using NAND

v. X-OR gate using NAND

$$Y = (A \oplus B) = (A\overline{B} + \overline{A}B)$$

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

$$= \overline{(A'B)'.(AB')'}$$

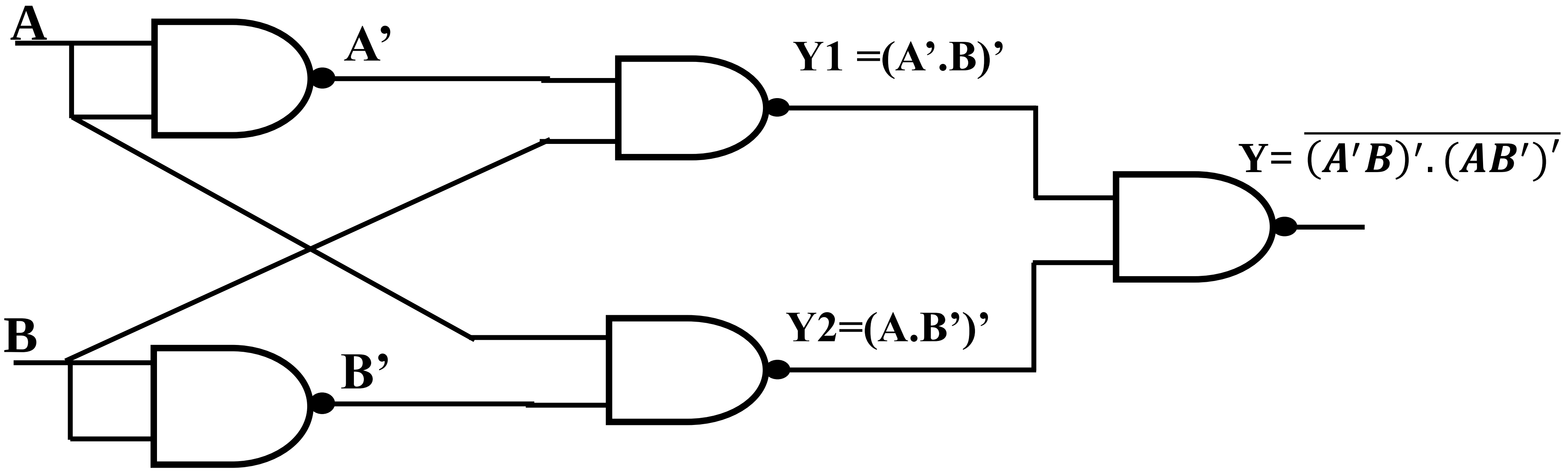


Fig: XOR gate Using NAND

vi. X-NOR gate using NAND

$$Y = (A \odot B) = (\bar{A}\bar{B} + AB)$$

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

$$= \overline{(A'B')'.(AB)'}$$

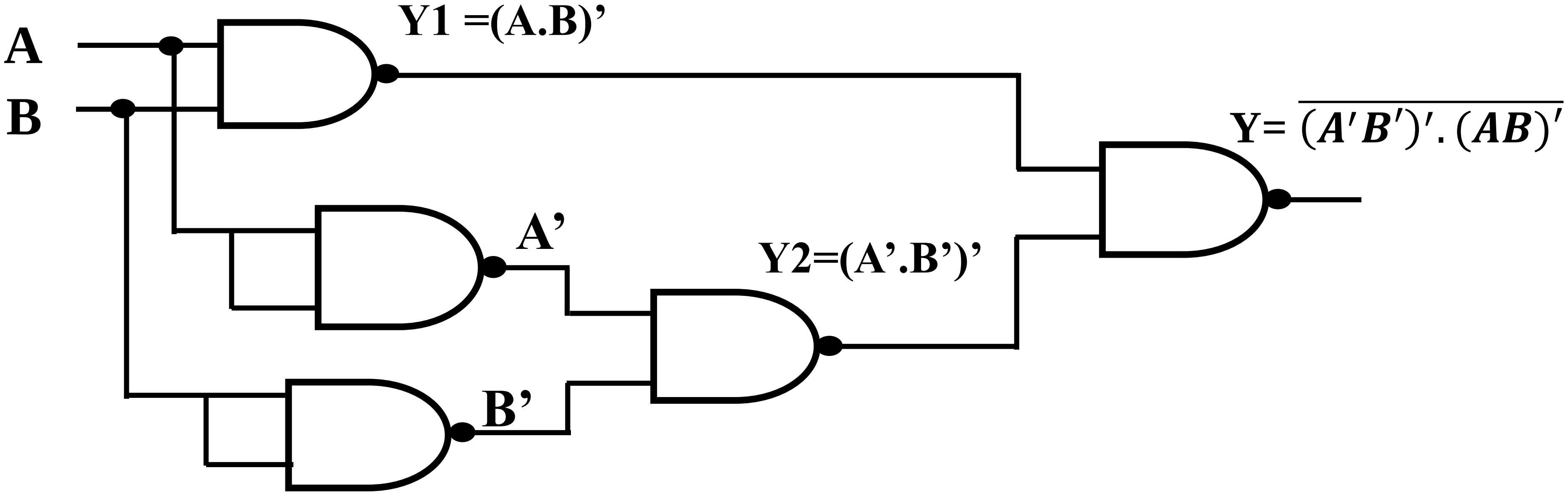
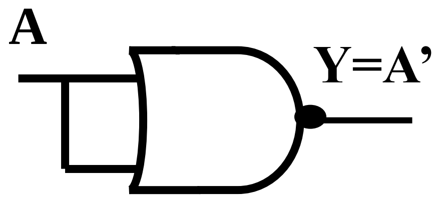


Fig: X-NOR gate Using NAND

2. Realization of various logic gates using NOR:

1. NOT gate using NOR

Y=A’



2. OR gate using NOR
3. AND gate using NOR
4. NAND gate using NOR
5. X-OR gate using NOR
6. X-NOR gate using NOR

Boolean Functions

- **Terminology:** *F(a,b,c)=a'bc+abc'+ab+c+....*
  - **Variables:** a,b,c and can have values '0' or '1'
  - **Prime Implicant:** Product or sum term obtained by combining the maximum possible number of adjacent squares (min/max-terms) in map.
  - **Essential Prime Implicant:** Sequence covered by only 1-prime implicant.
  - **Literal:** Appearance of variables, complemented or non-complemented.
  - **Product terms:** Logical anded terms
  - **Minterms:** *ANDing terms(Product) of literals, input appears exactly once.*
  - **Maxterms:** *ORing terms(Sum) of literals, input appears exactly once*

- **Minterms/Maxterms for 3-binary variables**

Inputs			Minterms		Maxterms	
A	B	C	TERMS	NOTATION	TERMS	NOTATION
0	0	0	A’B’C’	m0	A+B+C	M0
0	0	1	A’B’C	m1	A+B+C’	M1
0	1	0	A’BC’	m2	A+B’+C	M2
0	1	1	A’BC	m3	A+B’+C’	M3
1	0	0	AB’C’	m4	A’+B+C	M4
1	0	1	AB’C	m5	A’+B+C’	M5
1	1	0	ABC’	m6	A’+B’+C	M6
1	1	1	ABC	m7	A’+B’+C’	M7

Canonical and standard forms

- Boolean functions can be expressed as:
  - ➢ ***Sum of Minterms*** or ***Product of Maxterms*** and are said to be in ***canonical form***.
  - ➢ An arbitrary logic function can be expressed in:
    1. Sum of product (SOP)
    2. Product of Sum (POS)

1. Sum of product (SOP)/sum of Minterms

- For n-bit binary input there will be 2^n distinct Minterms.
- Minterms= 1's function in the truth table.
- If the terms are not in SOP form, then can be expressed into sum of AND terms.
- Each term is inspected to see if it contains all the variables.
- If any variable is missing then it is ANDed with an expression such as A+A', where A is a missing variable.

Ex: Express the Boolean function in a sum of Minterms F(A,B,C)=A+B'C

Solution: *The given function has three variables A,B and C.*

The first variable contains only one variable A so, variable B and C are missing.

A=A(B+B')=AB+AB' **(∵ B+B'=1)**

Again the variable c is missing,

AB(C+C')+AB'(C+C')=ABC+ABC'+AB'C+AB'C'

In the second term the variable A is missing,

B'C(A+A')=AB'C+A'B'C

Now, combining the first and second term,

F(A,B,C)=A+B'C) =ABC+ABC'+AB'C+AB'C'+AB'C+A'B'C

=ABC+ABC'+AB'C+AB'C'+A'B'C

=m7+m6+m5+m4+m1

∴F(A,B,C)=∑m(1,4,5,6,7)

- ***The ∑ symbol represents the ORing of the terms and the number following are Minterms of the function.***

2. Product of sum(POS)/Product of Maxterms

- n-bit binary input=2^n Maxterms
- Must be in ***OR*** terms.
- Use of distributive law, ***A+BC=(A+B)(A+C)***
- Any missing variable ***X*** is ***ORed*** with base expression using ***XX'***.

Ex1: Express the Boolean function in a sum of Minterms F(A,B,C)=A+B'C

Solution: *The given function has three variables A,B and C.*

using distributive law,

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

$A+B' = A+B'+CC'= (A+B'+C)(A+B'+C')$ $(\because CC'=0)$

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

Combining all,

$F(A,B,C)=A+B'C=(A+B+C)(A+B'+C)(A+B'+C')=M_0M_1M2=\Pi(0,2,3)$

Ex2: Express the Boolean function in a sum of Minterms
$F(A,B,C)=AB+A'C$

Solution: *The given function has three variables A,B and C.*

Using the distributive law,

$A=A+(BB')=(A+B)(A+B')$ $(\because BB'=0)$

Again the variable c is missing,

$AB(C+C')+AB'(C+C')=ABC+ABC'+AB'C+AB'C'$

In the second term the variable A is missing,

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

Now, combining the first and second term,

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

$=ABC+ABC'+AB'C+AB'C'+A'B'C$

$=m7+m6+m5+m4+m1$

$\therefore F(A,B,C)=\sum m(1,4,5,6,7)$

The $\sum$ symbol represents the ORing of the terms and the number following are Minterms of the function.

Conversion between canonical forms

- *Let the expression be* $F(A,B,C) = \sum m(1,4,5,6,7)$

$$= m1+m4+m5+m6+m7$$

It's complement can be expressed as,

$$\overline{F}(A,B,C)=\sum m(0,2,3)=m0+m2+m3$$

- *Now, if we use De-Morgan's theorem F' is obtained as,*

$$F(A,B,C) = \overline{m0+m2+m3}$$

$$= \overline{mo}.\overline{m2}.\ \overline{m3}$$

$$=M0.M2.M3$$

$$=\Pi(0,2,3)$$

- $\overline{m}_i = M_i$

1. Find the Minterms and Maxterms of a Boolean function ***F(A,B,C)=A'B'+C.***

INPUTS			**OUTPUTS**
A	**B**	**C**	**F(A,B,C)=A'B'+C**
0	**0**	**0**	**1**
0	**0**	**1**	**1**
0	**1**	**0**	**0**
0	**1**	**1**	**1**
1	**0**	**0**	**0**
1	**0**	**1**	**1**
1	**1**	**0**	**0**
1	**1**	**1**	**1**

Fig: Truth table for the function F(A,B,C)=A'B'+C

- The ***Minterms*** of the Boolean function from the truth table are:

  m0,m1,m3,m5,and m7.
- **SOP: F(A,B,C) =A'B'C'+A'B'C+A'BC+AB'C+ABC**

  =m0+m1+m3+m5+m7

  =∑m(0,1,3,5,7)
- Since, there are 3-input variables, there are 8-Min/Maxterms.
- The non-Minterms are the ***Maxterms (POS)***.

  ∴**F(A,B,C)=Π(2,4,6)=(A+B'+C)(A'+B+C)(A'+B'+C)**

Karnaugh-MAP

- The digital circuit are mostly constructed using digital gates.
- Minimization of digital gate is major concern in digital circuits.
- Minimization reduces the size and cost with improved performance.
- Karnaugh-MAP is one of the popular method of minimization.
- Boolean expression can be minimized in two different ways:
    1. ***Boolean Algebra method***
    2. ***MAP method***

- Karnaugh-MAP is a graphical representation of truth table of given expression.
- Can be used for:
    1. ***One and two variables***
    2. ***Three variables***
    3. ***Four variables***

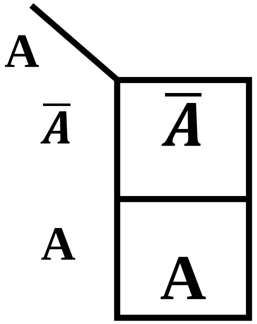


Fig1: K-map For One variable

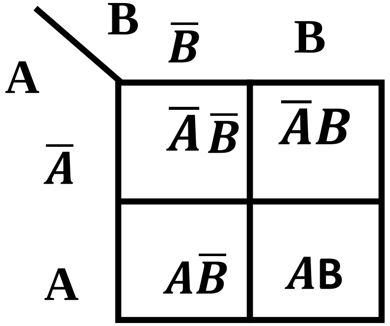


Fig2: K-map For Two variable

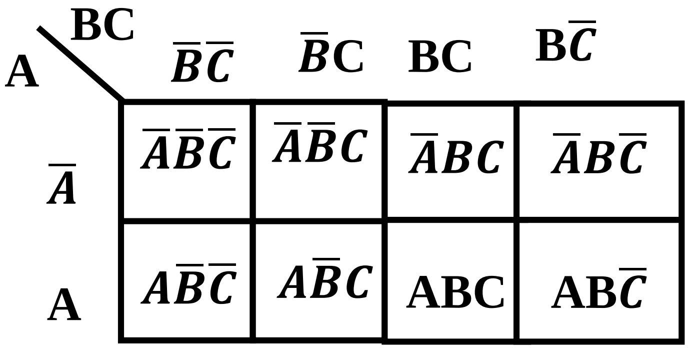


Fig3: K-map For Three variable

A B C D	Min-term,m
0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7
1 0 0 0	8
1 0 0 1	9
1 0 1 0	10
1 0 1 1	11
1 1 0 0	12
1 1 0 1	13
1 1 1 0	14
1 1 1 1	15

AB \ CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}\bar{B}\bar{C}D$	$\bar{A}\bar{B}CD$	$\bar{A}\bar{B}C\bar{D}$
$\bar{A}B$	$\bar{A}B\bar{C}\bar{D}$	$\bar{A}B\bar{C}D$	$\bar{A}BCD$	$\bar{A}BC\bar{D}$
AB	$AB\bar{C}\bar{D}$	$AB\bar{C}D$	ABCD	$ABC\bar{D}$
$A\bar{B}$	$A\bar{B}\bar{C}\bar{D}$	$A\bar{B}\bar{C}D$	$A\bar{B}CD$	$A\bar{B}C\bar{D}$

Fig4: K-map For Four variable

Representing Minterms (SOP) in K-map

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

A \ B	$\bar{B}$	B
$\bar{A}$	$\bar{A}\bar{B}$ 0	$\bar{A}B$ 1
A	$A\bar{B}$ 2	AB 3

Fig5: K-map For One and Two variable

A \ BC	$\bar{B}\bar{C}$	$\bar{B}C$	BC	$B\bar{C}$
$\bar{A}$	$\bar{A}\bar{B}\bar{C}$ 0	$\bar{A}\bar{B}C$ 1	$\bar{A}BC$ 3	$\bar{A}B\bar{C}$ 2
A	$A\bar{B}\bar{C}$ 4	$A\bar{B}C$ 5	ABC 7	$AB\bar{C}$ 6

Fig6: K-map For Three variable

AB \ CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	$\bar{A}\bar{B}\bar{C}\bar{D}$ 0	$\bar{A}\bar{B}\bar{C}D$ 1	$\bar{A}\bar{B}CD$ 3	$\bar{A}\bar{B}C\bar{D}$ 2
$\bar{A}B$	$\bar{A}B\bar{C}\bar{D}$ 4	$\bar{A}B\bar{C}D$ 5	$\bar{A}BCD$ 7	$\bar{A}BC\bar{D}$ 6
AB	$AB\bar{C}\bar{D}$ 12	$AB\bar{C}D$ 13	$ABCD$ 15	$ABC\bar{D}$ 14
$A\bar{B}$	$A\bar{B}\bar{C}\bar{D}$ 8	$A\bar{B}\bar{C}D$ 9	$A\bar{B}CD$ 11	$A\bar{B}C\bar{D}$ 10

Fig7: K-map For Four variable

A	
0	$m0$
1	$m1$

A \ B	0	1
0	$m0$	$m1$
1	$m2$	$m3$

Fig8: K-map For One and Two variable

A \ BC	00	01	11	10
0	$m0$	$m1$	$m3$	$m2$
1	$m4$	$m5$	$m7$	$m6$

Fig9: K-map For Three variable

AB \ CD	00	01	11	10
00	$m0$	$m1$	$m3$	$m2$
01	$m4$	$m5$	$m7$	$m6$
11	$m12$	$m13$	$m15$	$m14$
10	$m8$	$m9$	$m11$	$m10$

Fig10: K-map For Four variable

Representing Maxterms (POS) in K-map

A	
0	$M0$
1	$M1$

A \ B	0	1
0	$M0$	$M1$
1	$M2$	$M3$

Fig8: K-map For One and Two variable

A \ BC	00	01	11	10
0	$M0$	$M1$	$M3$	$M2$
1	$M4$	$M5$	$M7$	$M6$

Fig9: K-map For Three variable

AB \ CD	00	01	11	10
00	$M0$	$M1$	$M3$	$M2$
01	$M4$	$M5$	$M7$	$M6$
11	$M12$	$M13$	$M15$	$M14$
10	$M8$	$M9$	$M11$	$M10$

Fig10: K-map For Four variable

Plotting a K-map

- *K-map provides a systematic method for simplifying Boolean expression.*
- *Provides simplest SOP or POS, minimized expression.*

Representing Truth table in K-Map

1. **Cell:** *smallest unit of K-map representing to one line of Truth table.*

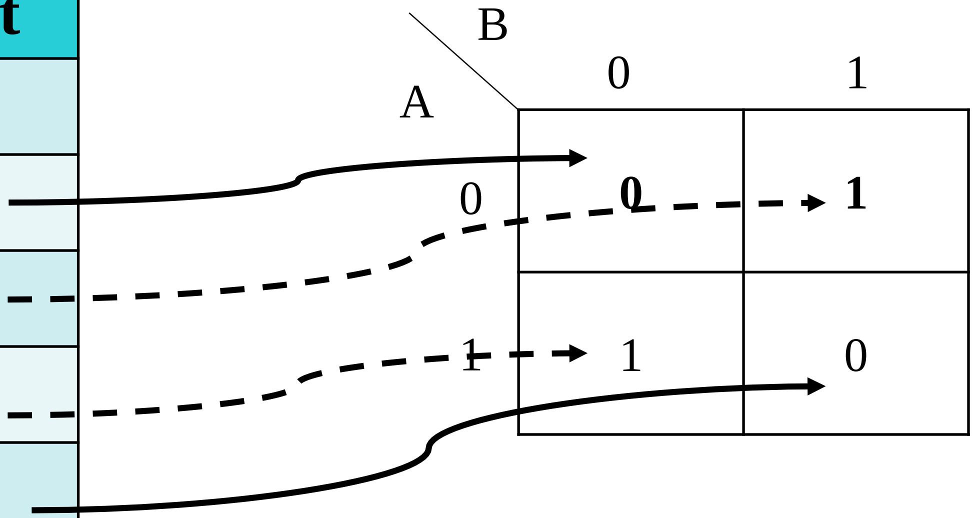
   Input variables → cell’s coordinate

   Output variables → cell’s content

- ***Ex: let us take an X-OR gate***

  Y=A’B+AB’

Input		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0



Representing standard SOP on K-map

- Place ‘1’ in each cell corresponding to a term (Minterm) in SOP expression.
- Remaining cells are filled with ‘0’.
- ***EX:*** **Plot Boolean expression Y=ABC’+ABC+A’B’C ON THE K-map.**

Solution: The Boolean expression has 3-variables and hence it can be plotted using 3-variables.

A \ BC	00	01	11	10
0	0	1	0	0
1	0	0	1	1

Representing standard POS on K-map

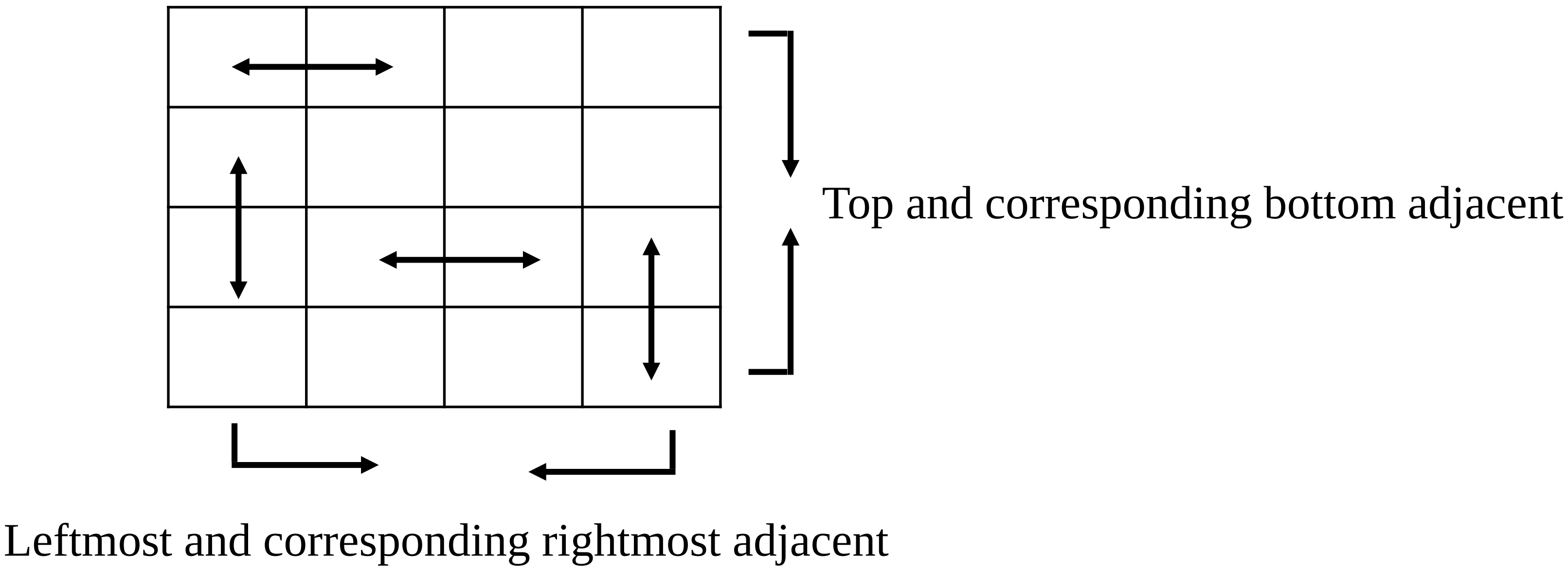
- Place ‘0’ in each cell corresponding to a term (Maxterm) in SOP expression.
- Remaining cells are filled with ‘1’.
- ***EX:*** **Plot Boolean expression Y=ABC’+ABC+A’B’C ON THE K-map.**

Solution: The Boolean expression has 3-variables and hence it can be plotted using 3-variables.

A \ BC	00	01	11	10
0	1	0	1	1
1	1	1	0	0

Grouping cells for simplification

- Cells are said to be adjacent if they form the single change rule.



1. Grouping two adjacent ones (pair)

- **Two adjacent 1's** present may be horizontally or vertically.
- **EX: Let the expression Y=$\bar{A}\bar{B}C + \bar{A}$BC**

A \ BC	$\bar{B}\bar{C}$ 00	$\bar{B}C$ 01	BC 11	$B\bar{C}$ 10
$\bar{A}$ 0	0	1	1	0
A 1	0	0	0	0

$\bar{A}C$

$Y = \bar{A}C$

A \ BC	$\bar{B}\bar{C}$ 00	$\bar{B}C$ 01	BC 11	$B\bar{C}$ 10
$\bar{A}$ 0	0	1	0	0
A 1	0	1	0	0

Y=$\bar{B}$C

A \ BC	$\bar{B}\bar{C}$ 00	$\bar{B}C$ 01	BC 11	$B\bar{C}$ 10
$\bar{A}$ 0	0	0	1	0
A 1	1	0	0	1

$Y = A\bar{C}$

A \ BC	$\bar{B}\bar{C}$ 00	$\bar{B}C$ 01	BC 11	$B\bar{C}$ 10
$\bar{A}$ 0	0	0	1	1
A 1	0	0	0	1

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

2. Grouping four adjacent ones (Quad)

- **Four adjacent 1's** present may be horizontally or vertically.
- **EX: Let the expression Y=$\bar{A}\bar{B}C + \bar{A}\mathbf{BC} + \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C}$**

A \ BC	$\bar{B}\bar{C}$ 00	$\bar{B}$C 01	**BC** 11	B$\bar{C}$ 10	
$\bar{A}$ 0	**1**	**1**	**1**	**1**	$\bar{A}$
A 1	**0**	**0**	**0**	**0**	

Fig: Horizontal grouping of four adjacent cells

A \ BC	$\bar{B}\bar{C}$ 00	$\bar{B}$C 01	**BC** 11	B$\bar{C}$ 10
$\bar{A}$ 0	**0**	**0**	**1**	**1**
A 1	**0**	**0**	**1**	**1**

B

Y=B

- This gives **Y=** $\bar{A}$ which is the solution of the given expression.

3. Grouping eight adjacent ones (Octet)

- **Eight adjacent 1's** present may be horizontally or vertically.
- **EX: Let the expression Y=$\boldsymbol{\bar{A}\bar{B}C + \bar{A}\mathrm{BC} + \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C}}$**

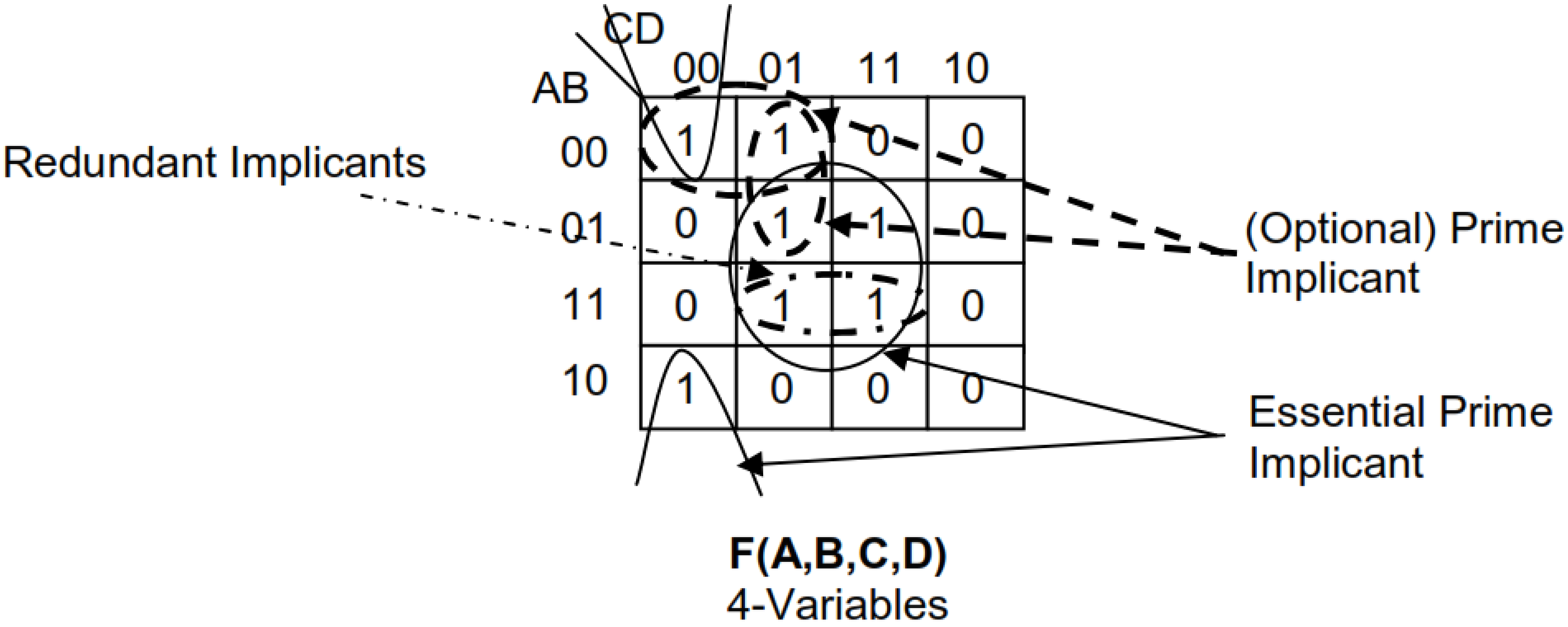
AB \ CD	$\bar{C}\bar{D}$ 00	$\bar{C}D$ 01	CD 11	$C\bar{D}$ 10
$\bar{A}\bar{B}$ 00	**0**	**0**	**0**	**0**
$\bar{A}B$ 01	**0**	**0**	**0**	**0**
AB 11	**1**	**1**	**1**	**1**
$A\bar{B}$ 10	**1**	**1**	**1**	**1**

A

Fig: Horizontal grouping of four adjacent cells

- This gives **Y= A** which is the solution of the given expression.

- **More K-map related definitions:**
  - ➢ Example: A function with the following K-Map



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

Example:

Use K-map to minimize F(A,B,C) =$\bar{A}\bar{B}C + \bar{A}BC$+A$\bar{B}C + ABC$

Solution:

1. Use a truth table to identify all the Min-terms (Over time you can do this mentally, so it would not be necessary to draw it).

A B C	F	Min-term, m_i
0 0 0	0	0
0 0 1	1	1
0 1 0	0	2
0 1 1	1	3
1 0 0	0	4
1 0 1	1	5
1 1 0	0	6
1 1 1	1	7

2. *Fill in the K-map:*

a) Select the K-Map that matches the number of variables in your function, (3 for the Example)

b) Draw the K-map (remember the labels are reflective Gray Code)

c) Enter the value '1' of the function for the corresponding min-term.

AB \ C	0	1
00	0	1
01	0	1
11	0	1
10	0	1

F(A,B,C)
3-Variables

≡

A \ BC	00	01	11	10
0	**0**	**1**	**1**	**0**
1	**0**	**1**	**1**	**0**

3. The next step is to group as many neighbouring ones as possible.
   a. Grouping adjacent min-terms (boxes) is applying the Adjacency theorem graphically.
   a. The goal is to get as large a grouping of 1s as possible (Must form a full rectangle – cannot group diagonally).
4. For each identified group, look to see which variable has a unique value. In this case, F(A,B,C) = C since F's value is not dependent on the value of A and B.

Example: Write the minimized SOP function represented by the following K-Map

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

Example: Use K-map to write the minimized SOP and POS forms of the following function:

a. $F(A,B,C,D)=\sum(0,1,4,5,8,9,12,15)$
b. $F(A,B,C,D)=\prod(2,3,5,6,7,8,12,13,14)$

Special Case: “Don’t Care” Terms

- In K-map, we can use the unspecified values of a function ***“don’t care”*** as 1 or 0, allowing us to create larger cubes to write products with smaller ***Literal Count (LCs).***
- ***Example:*** F(W,X,Y,Z) with unspecified values (don’t cares, “-“)

WX \ YZ	00	01	11	10
00	0	1	-	-
01	1	1	-	0
11	1	0	1	1
10	0	-	1	0

- We have an option of assuming “-“ as 0 or 1 whichever ends up with a lower Literal Count (LC) and therefore lower hardware (gates) cost during the implementation phase.

WX \ YZ	00	01	11	10
00	0	1	-	-
01	1	1	-	0
11	1	0	1	1
10	0	-	1	0

- $F(W,X,Y,Z)=X\overline{Y}\overline{Z} + \overline{W}Z + YZ + WXY$
- For this function the Literal Count (LC) is 10.

- Sometimes it makes sense to use the 0s and write the complement to get a lower LC.

WX \ YZ	00	01	11	10
00	0	1	-	-
01	1	1	-	0
11	1	0	1	1
10	0	-	1	0

- $\overline{F(W,X,Y,Z)} = W\bar{Y}Z + \bar{X}\bar{Z} + \bar{W}Y$
- For this function, the Literal Count (LC) is 7.
- Function in POS form, ***F(W,X,Y,Z)= (W'+Y+Z').(X+Y).(W+Y')***

Code Conversion

1. Binary to BCD code conversion

i. First, we will convert the binary number into decimal.

ii. We will convert the decimal number into BCD.

- **Example 1: $(11110)_2$**

Step 1: First, convert the given binary number into a decimal number.

Finding Decimal Equivalent of the number:

Steps	Binary Number	Decimal Number
1)	(11110)	$((1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0))$
2)	(11110)	(16 + 8 + 4 + 2 + 0)
3)	(11110)	(30)

Decimal number of the Binary number $(11110)_2$ is $(30)_{10}$

2. Convert the decimal to the BCD

Steps	Decimal Number	Conversion
Step 1	**30**	**(0011) (0000)**
Step 2	**30**	**$(00110000)_{BCD}$**

- The most significant bit of the decimal number is represented by the bit B4, and the least significant bits are represented by B3, B2, B1, and B0.

SOP function for different bits of BCD code are as follows:

$$B4 = \sum m(10,11,12,13,14,15)$$

$$B3 = \sum m(8,9)$$

$$B0 = \sum m(1,3,5,7,9,11,13,15)$$

$$B3 = \sum m(4,5,6,7,14,15)$$

$$B2 = \sum m(2,3,6,7,12,13)$$

Binary Code	Decimal Number	BCD Code
A B C D		**B4 :B3B2B1B0**
0 0 0 0	0	**0 : 0 0 0 0**
0 0 0 1	1	**0 : 0 0 0 1**
0 0 1 0	2	**0 : 0 0 1 0**
0 0 1 1	3	**0 : 0 0 1 1**
0 1 0 0	4	**0 : 0 1 0 0**
0 1 0 1	5	**0 : 0 1 0 1**
0 1 1 0	6	**0 : 0 1 1 0**
0 1 1 1	7	**0 : 0 1 1 1**
1 0 0 0	8	**0 : 1 0 0 0**
1 0 0 1	9	**0 : 1 0 0 1**
1 0 1 0	10	**1 : 0 0 0 0**
1 0 1 1	11	**1 : 0 0 0 1**
1 1 0 0	12	**1 : 0 0 1 0**
1 1 0 1	13	**1 : 0 0 1 1**
1 1 1 0	14	**1 : 0 1 0 0**
1 1 1 1	15	**1 : 0 1 0 1**

The K-maps of the above SOP functions are as follows:

AB \ CD	$\bar{C}\bar{D}$ 00	$\bar{C}D$ 01	CD 11	$C\bar{D}$ 10
$\bar{A}\bar{B}$ 00	**0**	**1**	**1**	**0**
$\bar{A}B$ 01	**0**	**1**	**1**	**0**
AB 11	**0**	**1**	**1**	**0**
$A\bar{B}$ 10	**0**	**1**	**1**	**0**

$B_0=D$

AB \ CD	$\bar{C}\bar{D}$ 00	$\bar{C}D$ 01	CD 11	$C\bar{D}$ 10
$\bar{A}\bar{B}$ 00	**0**	**0**	**1**	**1**
$\bar{A}B$ 01	**0**	**0**	**1**	**1**
AB 11	**1**	**1**	**0**	**0**
$A\bar{B}$ 10	**0**	**0**	**0**	**0**

$B_1= \bar{A}C+AB\bar{C}$

AB \ CD	$\bar{C}\bar{D}$ 00	$\bar{C}D$ 01	CD 11	$C\bar{D}$ 10
$\bar{A}\bar{B}$ 00	**0**	**0**	**0**	**0**
$\bar{A}B$ 01	**1**	**1**	**1**	**1**
AB 11	**0**	**0**	**1**	**1**
$A\bar{B}$ 10	**0**	**0**	**0**	**0**

$B_2= \bar{A}B+BC$

AB \ CD	$\bar{C}\bar{D}$ 00	$\bar{C}D$ 01	CD 11	$C\bar{D}$ 10
$\bar{A}\bar{B}$ 00	**0**	**0**	**0**	**0**
$\bar{A}B$ 01	**0**	**0**	**0**	**0**
AB 11	**0**	**0**	**0**	**0**
$A\bar{B}$ 10	**1**	**1**	**0**	**0**

$B_3= A\bar{B}\bar{C}$

AB \ CD	$\bar{C}\bar{D}$ 00	$\bar{C}D$ 01	CD 11	$C\bar{D}$ 10
$\bar{A}\bar{B}$ 00	0	0	0	0
$\bar{A}B$ 01	0	0	0	0
AB 11	1	1	1	1
$A\bar{B}$ 10	0	0	1	1

$B_4 = AB + AC$

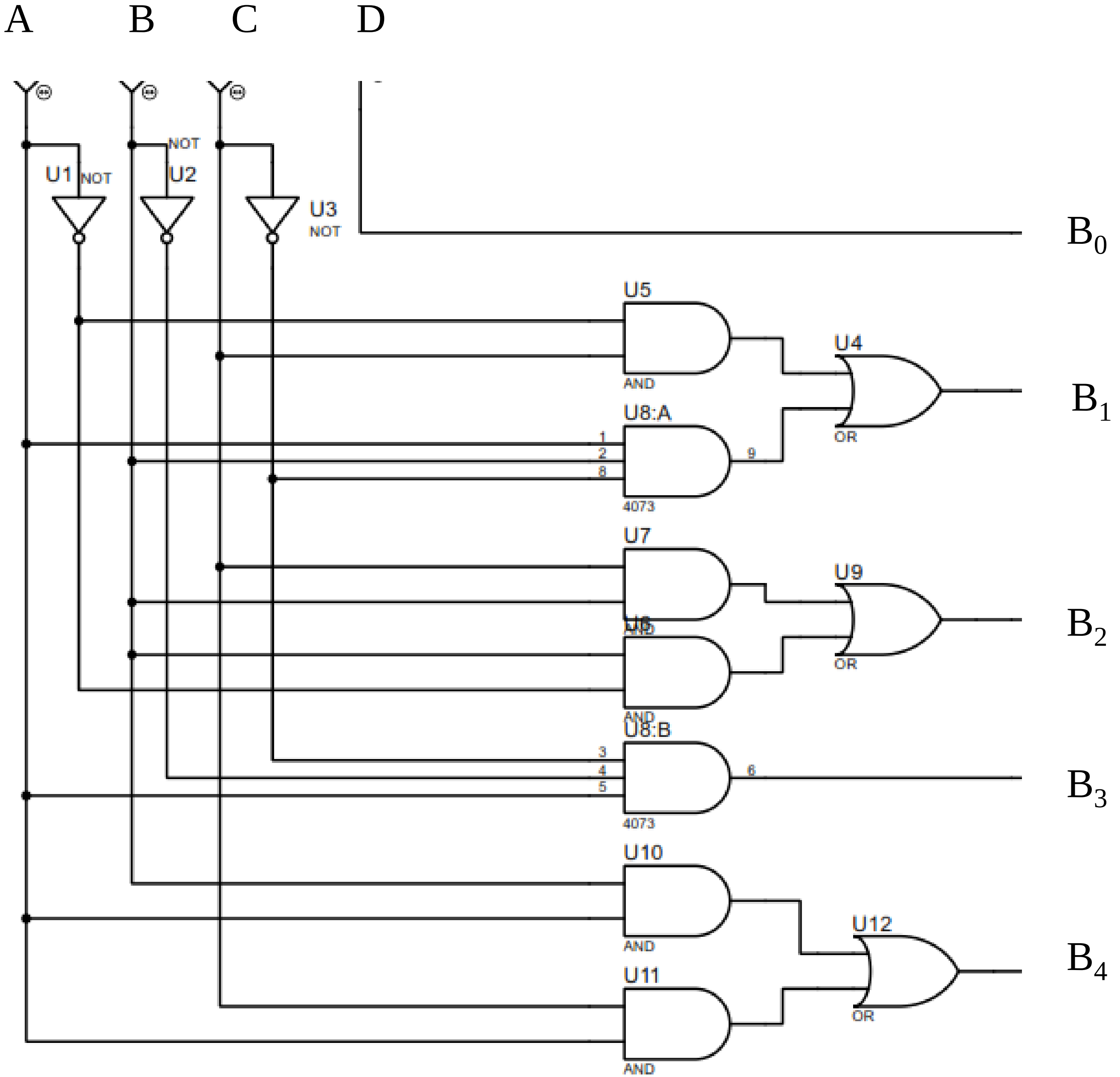


Fig: Circuit diagram for binary to BCD converter

Binary to Gray code conversion

- The 4-bit binary to Gray code conversion table is as follows:

Decimal Number	4-bit Binary Code	4-bit Gray Code
	ABCD	**G3G2G1G0**
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

$$G0 = \sum m(1,2,5,6,9,10,13,14)$$

$$G1 = \sum m(2,3,4,5,10,11,12,13)$$

$$G2 = \sum m(4,5,6,7,8,9,10,11)$$

$$G3 = \sum m(8,9,10,11,12,13,14,15)$$

AB \ CD	$\bar{C}\bar{D}$ 00	$\bar{C}D$ 01	CD 11	$C\bar{D}$ 10
$\bar{A}\bar{B}$ 00	0	1	0	1
$\bar{A}B$ 01	0	1	0	1
AB 11	0	1	0	1
$A\bar{B}$ 10	0	1	0	1

$G_0=\bar{C}D+C\bar{D} = C\oplus D$

AB \ CD	$\bar{C}\bar{D}$ 00	$\bar{C}D$ 01	CD 11	$C\bar{D}$ 10
$\bar{A}\bar{B}$ 00	0	0	1	1
$\bar{A}B$ 01	1	1	0	0
AB 11	1	1	0	0
$A\bar{B}$ 10	0	0	1	1

$G_1=B\bar{C} + \bar{B}C = B\oplus C$

AB \ CD	$\bar{C}\bar{D}$ 00	$\bar{C}D$ 01	CD 11	$C\bar{D}$ 10
$\bar{A}\bar{B}$ 00	0	0	0	0
$\bar{A}B$ 01	1	1	1	1
AB 11	0	0	0	0
$A\bar{B}$ 10	1	1	1	1

$G_2=A\bar{B} + \bar{A}B = A\oplus B$

AB \ CD	$\bar{C}\bar{D}$ 00	$\bar{C}D$ 01	CD 11	$C\bar{D}$ 10
$\bar{A}\bar{B}$ 00	0	0	0	0
$\bar{A}B$ 01	0	0	0	0
AB 11	1	1	1	1
$A\bar{B}$ 10	1	1	1	1

$G_3=A$

Note: Binary to Excess-3 Do yourself

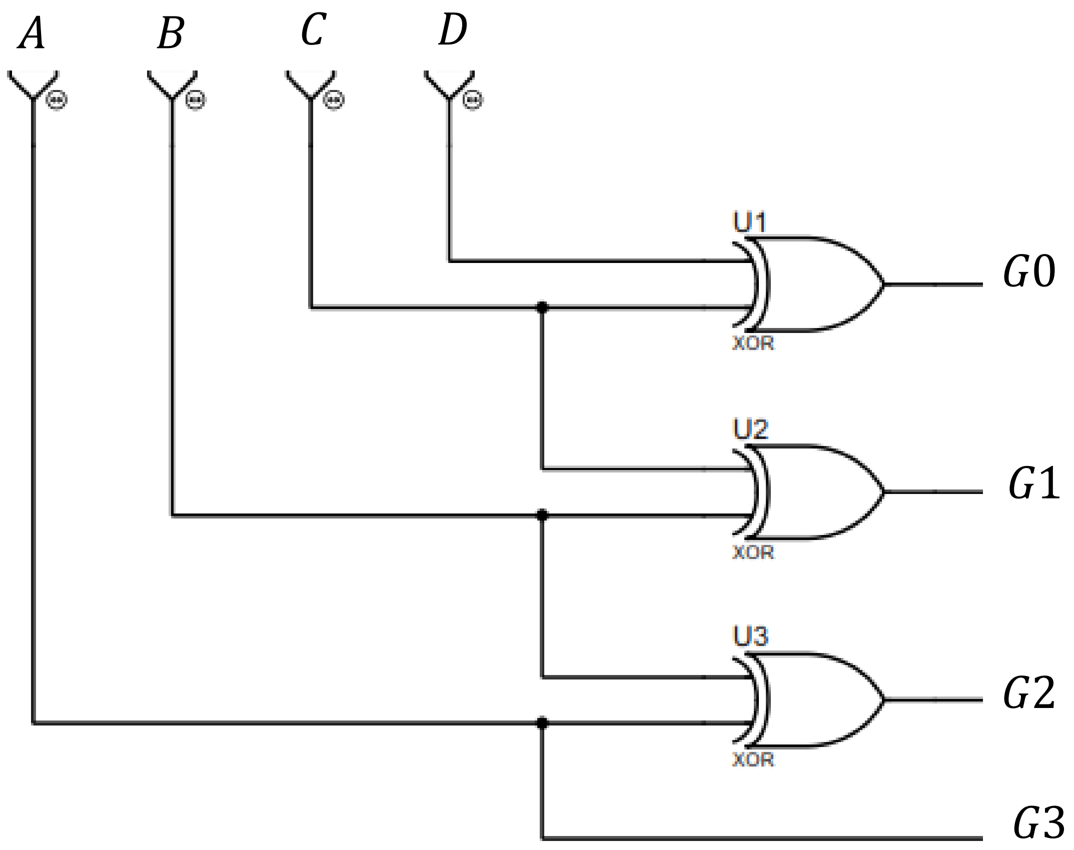


Fig: Circuit diagram for binary to Gray code converter