LOGIC GATES, BOOLEAN ALGEBRA & SIMPLIFICATION OF LOGIC

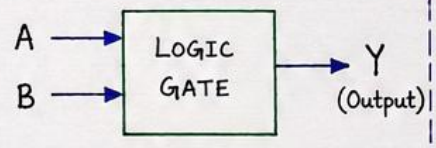
1. What is Logic Gate?

A **Logic Gate** is an electronic circuit that performs a logical operation on one or more **binary inputs** and produces a single **binary output**.

Key Points :

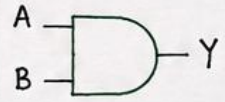
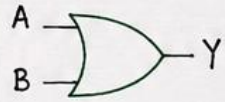
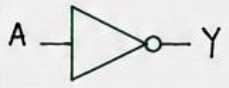
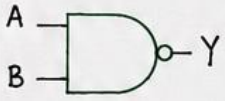
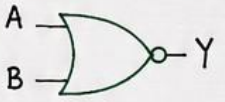
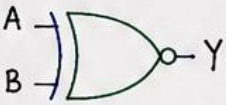
- Inputs and output are in binary form (0 or 1).
- Used in digital circuits to perform logical operations.
- Basic building blocks of digital systems.
- Implemented using diodes, transistors or ICs.

Example :



2. Types of Logic Gates

There are seven basic types of logic gates.

S. No.	Gate Name	Symbol	Truth Table	Boolean Expression	Description
1.	AND Gate ($Y = A \cdot B$)		A B | Y 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1	$Y = A \cdot B$	Output is 1 only when both inputs are 1.
2.	OR Gate ($Y = A + B$)		A B | Y 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1	$Y = A + B$	Output is 1 when atleast one input is 1.
3.	NOT Gate ($Y = \overline{A}$)		A | Y 0 | 1 1 | 0	$Y = \overline{A}$	Output is complement of the input.
4.	NAND Gate ($Y = \overline{A \cdot B}$)		A B | Y 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0	$Y = \overline{A \cdot B}$	Output is 0 only when both inputs are 1.
5.	NOR Gate ($Y = \overline{A + B}$)		A B | Y 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0	$Y = \overline{A + B}$	Output is 1 only when both inputs are 0.
6.	XOR Gate ($Y = \overline{A \oplus B}$)		A B | Y 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0	$Y = A \oplus B$	Output is 1 when inputs are different.
7.	XNOR Gate ($Y = \overline{A \oplus B}$)		A B | Y 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1	$Y = \overline{A \oplus B}$	Output is 1 when inputs are same.

★ **Note :** In Digital Electronics, all inputs and outputs are in terms of binary 0 (LOW) and 1 (HIGH).

Logic Gates, Boolean Algebra & Simplification of Logic

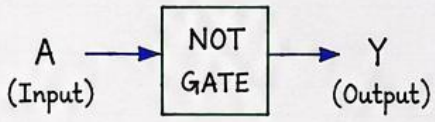
1. Not Gate – Definition

The NOT gate is a basic digital logic gate that inverts the input. It produces an output that is the complement of the input.

Key Points :

- It has only one input and one output.
- The output is always the opposite of the input.
- Also called Inverter or Complemeter.
- Represented by a small circle (bubble) at the output.

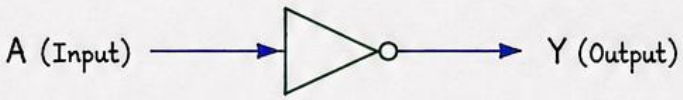
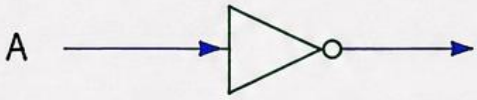
Example :



2. Not Gate – Working Principle

The NOT gate reverses the logic level of the input.

- If the input is HIGH (1), the output becomes LOW (0).
- If the input is LOW (0), the output becomes HIGH (1).

S. No.	Parameter	Details
1.	Diagram	
2.	Logic Symbol	
3.	Truth Table	A (Input) | Y (Output) 0 | 1 1 | 0
4.	Boolean Expression	$Y = \overline{A}$
5.	Description	Output is the complement of the input. When input is 0, output is 1. When input is 1, output is 0.

★ **Note :** The NOT gate is a fundamental gate and is used in many digital circuits and applications.

Logic Gates, Boolean Algebra & Simplification of Logic

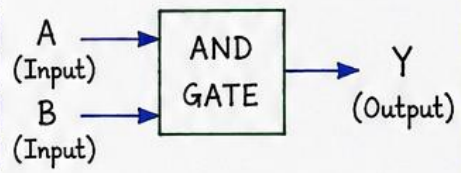
1. AND Gate – Definition

The AND gate is a fundamental digital logic gate that performs logical multiplication. It produces an output that is HIGH (1) only when all its inputs are HIGH (1).

Key Points :

- It has two or more inputs and one output.
- Output is HIGH (1) only when all inputs are HIGH (1).
- Also called Logical Multiplier.
- Symbol resembles the letter 'D' shape.
- Widely used in digital circuits and applications.

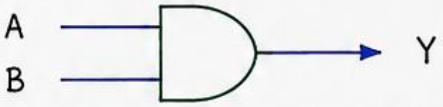
Example :



2. AND Gate – Working Principle

The AND gate follows the principle of logical multiplication.

- If all inputs are HIGH (1), the output is HIGH (1).
- If any one (or more) of the inputs is LOW (0), the output is LOW (0).

S. No.	Parameter	Details
1.	Diagram	
2.	Logic Symbol	
3.	Truth Table	A (Input) / B (Input) / Y (Output): 0 0 0; 0 1 0; 1 0 0; 1 1 1
4.	Boolean Expression	$Y = A \cdot B$
5.	Description	Output is HIGH (1) only when both inputs are HIGH (1). Otherwise, output is LOW (0).

★ **Note :** The AND gate is a basic building block in digital circuits and is used to perform logical multiplication.

Logic Gates, Boolean Algebra & Simplification of Logic

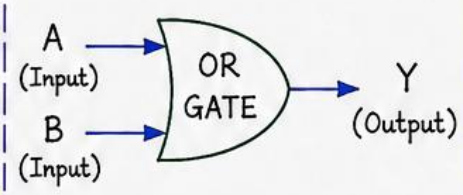
OR Gate – Definition

The OR gate is a fundamental digital logic gate that performs logical addition (OR) operation.
It produces an output that is HIGH (1) when any one or more of its inputs are HIGH (1).

Key Points :

- It has two or more inputs and one output.
- Output is HIGH (1) if any one (or more) input is HIGH (1).
- Output is LOW (0) only when all inputs are LOW (0).
- Also called Logical Adder.
- Symbol resembles the curved shape like the letter '≥' without the equal bar.
- Widely used in digital circuits and applications.

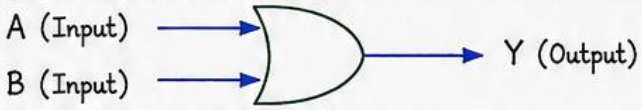
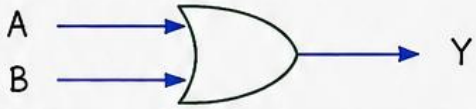
Example :



OR Gate – Working Principle

The OR gate follows the principle of logical addition.

- If any one (or more) of the inputs is HIGH (1), the output is HIGH (1).
- If all inputs are LOW (0), the output is LOW (0).

S. No.	Parameter	Details
1.	Diagram	
2.	Logic Symbol	
3.	Truth Table	A (Input) / B (Input) / Y (Output): 0 0 0; 0 1 1; 1 0 1; 1 1 1
4.	Boolean Expression	$Y = A + B$
5.	Description	Output is HIGH (1) when any one or more inputs are HIGH (1). Output is LOW (0) only when all inputs are LOW (0).

★ **Note :** The OR gate is a basic building block in digital circuits and is used to perform logical addition.

Logic Gates, Boolean Algebra & Simplification of Logic

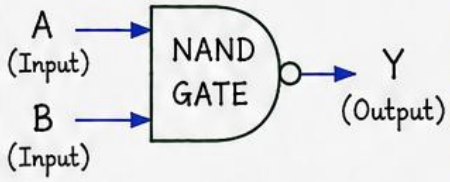
1. NAND Gate – Definition

The NAND gate is a fundamental digital logic gate that performs logical NAND operation. It produces an output that is LOW (0) only when all its inputs are HIGH (1).

Key Points :

- It has two or more inputs and one output.
- Output is LOW (0) only when all inputs are HIGH (1).
- Also called Logical NAND.
- Symbol resembles the AND gate with a small circle (bubble) at the output.
- Widely used in digital circuits and applications.

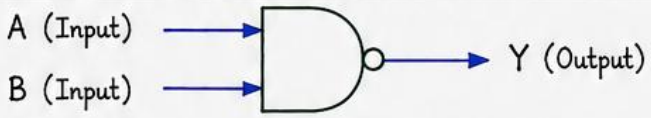
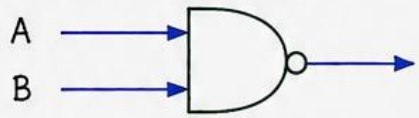
Example :



2. NAND Gate – Working Principle

The NAND gate follows the principle of logical NAND operation.

- If all inputs are HIGH (1), the output is LOW (0).
- If any one (or more) of the inputs is LOW (0), the output is HIGH (1).

S. No.	Parameter	Details
1.	Diagram	
2.	Logic Symbol	
3.	Truth Table	A (Input) / B (Input) / Y (Output): 0 0 1; 0 1 1; 1 0 1; 1 1 0
4.	Boolean Expression	$Y = (A \cdot B)'$ (' means NOT)
5.	Description	Output is LOW (0) only when both inputs are HIGH (1). Otherwise, output is HIGH (1).

★ **Note :** The NAND gate is a universal gate. Any digital circuit can be built using only NAND gates.

Logic Gates, Boolean Algebra & Simplification of Logic

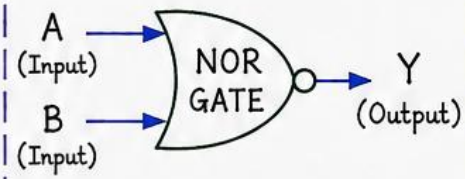
1. NOR Gate – Definition

The NOR gate is a fundamental digital logic gate that performs logical NOR operation. It produces an output that is HIGH (1) only when all its inputs are LOW (0).

Key Points :

- It has two or more inputs and one output.
- Output is HIGH (1) only when all inputs are LOW (0).
- Output is LOW (0) when any one (or more) input is HIGH (1).
- Also called Logical NOR.
- Symbol resembles the OR gate with a small circle (bubble) at the output.
- Widely used in digital circuits and applications.

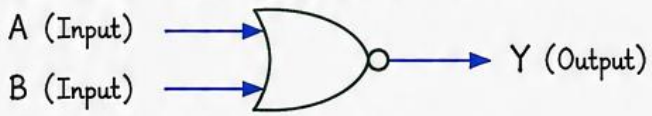
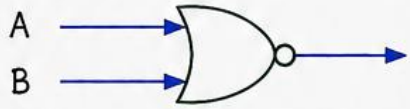
Example :



2. NOR Gate – Working Principle

The NOR gate follows the principle of logical NOR operation.

- If all inputs are LOW (0), the output is HIGH (1).
- If any one (or more) of the inputs is HIGH (1), the output is LOW (0).

S. No.	Parameter	Details
1.	Diagram	
2.	Logic Symbol	
3.	Truth Table	A (Input) / B (Input) / Y (Output): 0 0 1; 0 1 0; 1 0 0; 1 1 0
4.	Boolean Expression	$Y = (A + B)'$ (' means NOT)
5.	Description	Output is HIGH (1) only when both inputs are LOW (0). Otherwise, output is LOW (0).

★ Note : The NOR gate is a universal gate. Any digital circuit can be built using only NOR gates.

LOGIC GATES, BOOLEAN ALGEBRA & SIMPLIFICATION OF LOGIC

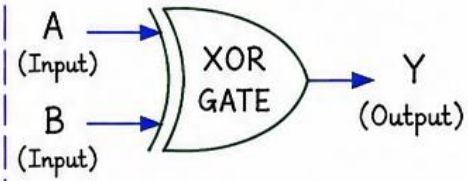
1. XOR Gate – Definition

The XOR gate is a fundamental digital logic gate that performs logical Exclusive-OR operation.
It produces an output that is HIGH (1) only when the inputs are different. If the inputs are same, output is LOW (0).

Key Points :

- It has two or more inputs and one output.
- Output is HIGH (1) only when inputs are different.
- Output is LOW (0) when inputs are same.
- Also called Exclusive-OR gate.
- Symbol has an extra curved line on the input side compared to OR gate.
- Widely used in digital circuits and applications.

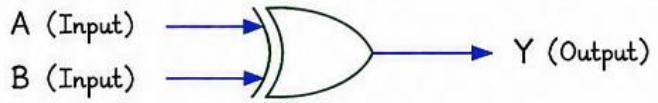
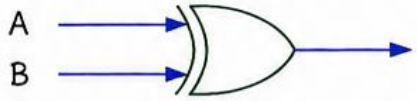
Example :



2. XOR Gate – Working Principle

The XOR gate follows the principle of logical Exclusive-OR operation.

- If the inputs are different (one is HIGH (1) and the other is LOW (0)), the output is HIGH (1).
- If the inputs are same (both HIGH (1) or both LOW (0)), the output is LOW (0).

S. No.	Parameter	Details
1.	Diagram	
2.	Logic Symbol	
3.	Truth Table	A (Input) / B (Input) / Y (Output): 0 0 0; 0 1 1; 1 0 1; 1 1 0
4.	Boolean Expression	$Y = A \oplus B$ (⊕ means XOR)
5.	Description	Output is HIGH (1) only when the inputs are different. Output is LOW (0) when the inputs are same.

★ **Note :** The XOR gate is a versatile gate used in adders, parity generators, code converters, and many digital circuits.

Logic Gates, Boolean Algebra & Simplification of Logic

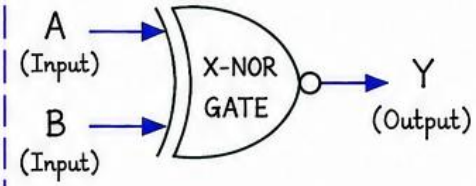
1. X-NOR Gate – Definition

The X-NOR gate is a fundamental digital logic gate that performs logical Exclusive-NOR operation.
It produces an output that is HIGH (1) only when the inputs are same. If the inputs are different, output is LOW (0).

Key Points :

- It has two or more inputs and one output.
- Output is HIGH (1) only when inputs are same.
- Output is LOW (0) when inputs are different.
- Also called Exclusive-NOR gate.
- Symbol has an extra curved line on the input side and a small circle (bubble) at the output.
- Widely used in digital circuits and applications.

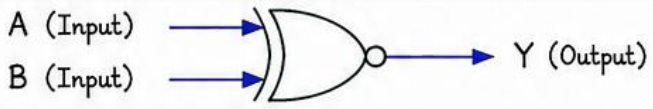
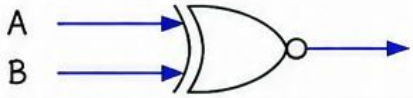
Example :



2. X-NOR Gate – Working Principle

The X-NOR gate follows the principle of logical Exclusive-NOR operation.

- If the inputs are same (both HIGH (1) or both LOW (0)), the output is HIGH (1).
- If the inputs are different (one is HIGH (1) and the other is LOW (0)), the output is LOW (0).

S. No.	Parameter	Details
1.	Diagram	
2.	Logic Symbol	
3.	Truth Table	A (Input) / B (Input) / Y (Output): 0 0 1; 0 1 0; 1 0 0; 1 1 1
4.	Boolean Expression	$Y = \overline{A \oplus B}$ (⊕ means XOR, overbar means NOT)
5.	Description	Output is HIGH (1) only when the inputs are same. Output is LOW (0) when the inputs are different.

★ **Note :** The X-NOR gate is a universal gate. Any digital circuit can be built using only X-NOR gates.

LOGIC GATES, BOOLEAN ALGEBRA & SIMPLIFICATION OF LOGIC

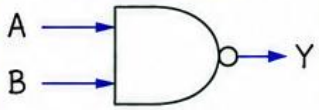
Universal Gates – NAND and NOR

NAND and NOR gates are called Universal Gates because any digital logic circuit can be built using only NAND gates or only NOR gates.

Key Points :

- Both NAND and NOR gates are universal, i.e., they can be used to implement NOT, AND, OR and any other logic function.
- This reduces the variety of ICs required in digital circuit design and manufacturing.
- NAND gate is more commonly used in digital ICs.

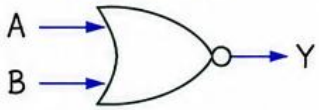
NAND Gate



$Y = (A \cdot B)'$

LOW (0) only when A = 1 and B = 1

NOR Gate

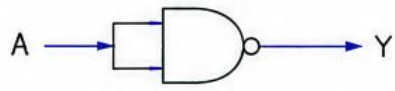
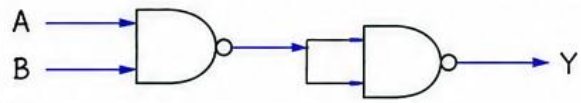
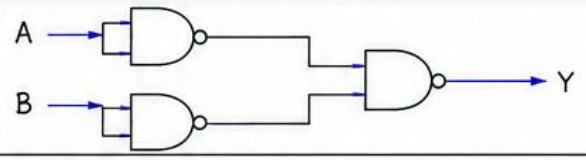


$Y = (A + B)'$

HIGH (1) only when A = 0 and B = 0

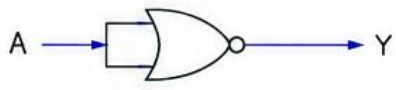
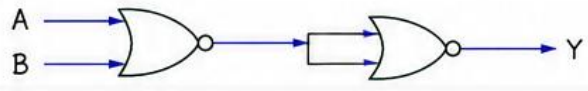
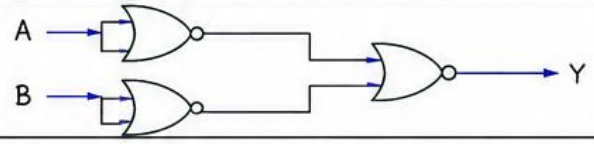
★ 1. Using NAND Gates Only

All basic gates can be implemented using NAND gates.

Logic Function	Implementation using NAND Gates	Boolean Expression
NOT Gate (Inverter)	 $Y = (A \cdot A)'$ $= A'$	$Y = A'$
AND Gate	 $Y = (A \cdot B)''$ $= A \cdot B$	$Y = A \cdot B$
OR Gate	 $Y = (A' \cdot B')'$ $= A + B$	$Y = A + B$

★ 2. Using NOR Gates Only

All basic gates can be implemented using NOR gates.

Logic Function	Implementation using NOR Gates	Boolean Expression
NOT Gate (Inverter)	 $Y = (A + A)'$ $= A'$	$Y = A'$
OR Gate	 $Y = (A + B)''$ $= A + B$	$Y = A + B$
AND Gate	 $Y = (A' + B')'$ $= A \cdot B$	$Y = A \cdot B$

★ **Note :** Since NAND and NOR gates are universal, designers can implement any logic circuit using only one type of gate.

Logic Gates, Boolean Algebra & Simplification of Logic

Rules and Laws of Boolean Algebra

Boolean algebra has a number of important rules and laws that are used to simplify logical expressions.

Key Points :

- These laws are the foundation of logic simplification.
- They help in reducing complex expressions to simple forms.
- They are used in digital circuit design, optimization and analysis.
- Here, A and B are variables which can have values 0 or 1.

Basic Notation :

A , B	→	Variables (0 or 1)
$\overline{A}$	→	Complement of A (NOT A)
A + B	→	OR operation
A . B	→	AND operation
1	→	Logic HIGH (True)
0	→	Logic LOW (False)

S. No.	Law / Rule	OR (+) Form	AND (.) Form	Explanation
1.	Identity Law	$A + 0 = A$	$A \cdot 1 = A$	Adding 0 or multiplying by 1 does not change the value.
2.	Null Law	$A + 1 = 1$	$A \cdot 0 = 0$	Adding 1 gives 1, multiplying by 0 gives 0.
3.	Idempotent Law	$A + A = A$	$A \cdot A = A$	Repeating the same variable does not change the result.
4.	Complement Law	$A + \overline{A} = 1$	$A \cdot \overline{A} = 0$	A variable OR its complement gives 1; AND gives 0.
5.	Involution Law	$\overline{\overline{A}} = A$	$\overline{\overline{A}} = A$	Complement of a complement is the original variable.
6.	Commutative Law	$A + B = B + A$	$A \cdot B = B \cdot A$	Changing the order of variables does not change the result.
7.	Associative Law	$(A + B) + C = A + (B + C)$	$(A \cdot B) \cdot C = A \cdot (B \cdot C)$	Grouping of variables does not change the result.
8.	Distributive Law	$A \cdot (B + C) = A \cdot B + A \cdot C$	$A + (B \cdot C) = (A + B) \cdot (A + C)$	AND over OR and OR over AND distribution.
9.	Absorption Law	$A + A \cdot B = A$	$A \cdot (A + B) = A$	Absorbs the extra term.
10.	De Morgan's Law	$\overline{A} + \overline{B} = \overline{(A \cdot B)}$	$\overline{A} \cdot \overline{B} = \overline{(A + B)}$	Relates AND and OR with complements.
11.	Consensus Theorem	$A \cdot B + \overline{A} \cdot C + B \cdot C = A \cdot B + \overline{A} \cdot C$	$(A+B) \cdot (\overline{A}+C) \cdot (B+C) = (A+B) \cdot (\overline{A}+C)$	Redundant term can be eliminated.

★ Note : These rules are very useful in simplifying Boolean expressions and in designing efficient digital circuits.

LOGIC GATES, BOOLEAN ALGEBRA & SIMPLIFICATION OF LOGIC

De-Morgan's Theorem

De-Morgan's Theorem is a very useful theorem in Boolean algebra which helps to simplify logical expressions and logic circuits.

Key Points :

- It provides a relationship between AND, OR and NOT operations.
- It is widely used to simplify Boolean expressions and to realize logic functions using NAND or NOR gates only.
- It helps in converting a complemented combination into a combination of complemented variables.

Basic De-Morgan's Theorems :

1. $(A \cdot B)' = A' + B'$

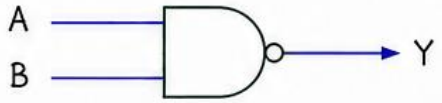
   The complement of an AND operation is equal to the OR of the complements.

2. $(A + B)' = A' \cdot B'$

   The complement of an OR operation is equal to the AND of the complements.

1. First Theorem : $(A \cdot B)' = A' + B'$

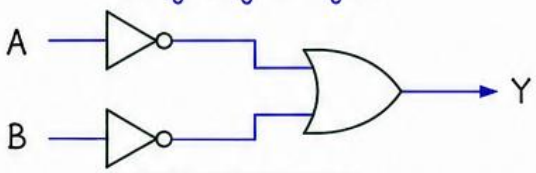
$(A \cdot B)'$ Using AND gate and NOT gate



$Y = (A \cdot B)'$

≡

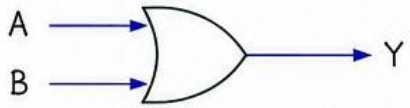
Using only OR gate



$Y = A' + B'$

2. Second Theorem : $(A + B)' = A' \cdot B'$

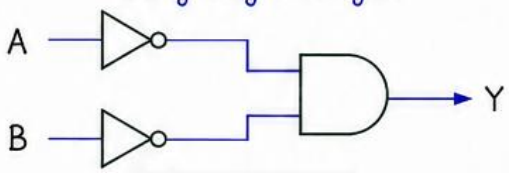
$(A + B)'$ Using OR gate and NOT gate



$Y = (A + B)'$

≡

Using only AND gate



$Y = A' \cdot B'$

Truth Table Verification

For $(A \cdot B)' = A' + B'$

A	B	A . B	(A . B)'	A'	B'	A' + 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

For $(A + B)' = A' \cdot B'$

A	B	A + B	(A + B)'	A'	B'	A' . 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

★ **Note :** De-Morgan's Theorems are very helpful in simplification of Boolean expressions and in implementing logic circuits using universal gates (NAND or NOR).

Logic Gates, Boolean Algebra & Simplification of Logic

Simplification of Boolean Expressions

Simplification of Boolean expressions means reducing a Boolean expression to its simplest form using Boolean algebra laws and theorems. This helps in reducing the number of gates in a circuit.

Key Points :

- Simplification reduces complexity.
- It reduces the cost of digital circuits.
- It makes implementation easier.
- It helps in faster calculation and analysis of logic circuits.

Basic Rules Used for Simplification :

1. $A + 0 = A$ (Identity Law)
2. $A \cdot 1 = A$ (Identity Law)
3. $A + 1 = 1$ (Null Law)
4. $A \cdot 0 = 0$ (Null Law)
5. $A + A = A$ (Idempotent Law)
6. $A \cdot A = A$ (Idempotent Law)
7. $A + \overline{A} = 1$ (Complement Law)
8. $A \cdot \overline{A} = 0$ (Complement Law)
9. $A + BC = (A + B)(A + C)$ (Distributive Law)
10. $A(B + C) = AB + AC$ (Distributive Law)

★ **Exercise :** Simplify the following Boolean expressions.

S. No.	Boolean Expression	Simplified Expression
1.	$Y = A + A \cdot B$	
2.	$Y = (A + \overline{A}) \cdot B$	
3.	$Y = (A + B)(A + \overline{B})$	
4.	$Y = A \cdot (\overline{A} + B)$	
5.	$Y = (A + B) + A \cdot \overline{B}$	

★ **Note :** Use the Boolean laws and theorems to simplify each expression and write the final simplest form.

LOGIC GATES, BOOLEAN ALGEBRA & SIMPLIFICATION OF LOGIC

Simplification of Boolean Expressions – Exercise (More Questions)

★ Simplify the following Boolean expressions using Boolean Algebra.

(1) $Y = A + A \cdot B$

Solution Steps :

$$
\begin{aligned}
Y &= A + A \cdot B \\
&= A\,(1 + B) && \text{(Factor A)} \\
&= A\,(1) && \text{(A + B = A (Distributive Law))} \\
&= A && \text{(1 + B = 1 (Null Law))} \\
& && \text{(A · 1 = A (Identity Law))}
\end{aligned}
$$

---

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

Solution Steps :

$$
\begin{aligned}
Y &= (A + \overline{A}) \cdot B \\
&= (1) \cdot B && (A + \overline{A} = 1 \text{ (Complement Law)}) \\
&= B && (1 \cdot B = B \text{ (Identity Law)})
\end{aligned}
$$

---

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

Solution Steps :

$$
\begin{aligned}
Y &= A \cdot \overline{A} + B \\
&= (0) + B && (A \cdot \overline{A} = 0 \text{ (Complement Law)}) \\
&= B && (0 + B = B \text{ (Identity Law)})
\end{aligned}
$$

---

(4) $Y = (A + B) \cdot (\overline{A} + C)$

Solution Steps :

$$
\begin{aligned}
Y &= (A + B) \cdot (\overline{A} + C) && \text{(Distributive Law)} \\
&= A\overline{A} + AC + B\overline{A} + BC && (A\overline{A} = 0) \\
&= 0 + AC + \overline{A}B + BC && \text{(Factor B from last two terms)} \\
&= AC + \overline{A}B + BC && \text{(Distributive Law)} \\
&= AC + B(\overline{A} + C) && \text{(No further simplification)} \\
&= AC + B(\overline{A} + C) \\
&= AC + B\overline{A} + BC
\end{aligned}
$$

---

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

Solution Steps :

$$
\begin{aligned}
Y &= (A + \overline{B})(\overline{A} + C) + A\overline{B} && \text{(Distributive Law)} \\
&= A\overline{A} + AC + B\overline{A} + \overline{B}C + A\overline{B} && (A\overline{A} = 0) \\
&= 0 + AC + A\overline{B} + \overline{B}C + A\overline{B} && \text{(Rearrange terms)} \\
&= AC + A\overline{B} + \overline{B}C + A\overline{B} && \text{(Factor } \overline{B} \text{ from terms with } \overline{B}) \\
&= AC + \overline{B}(\overline{A} + C + A) && (A + \overline{A} = 1) \\
&= AC + \overline{B}((\overline{A} + A) + C) && \text{(Associative Law)} \\
&= AC + \overline{B}(1 + C) && (1 + C = 1) \\
&= AC + \overline{B}(1) && (X \cdot 1 = X) \\
&= AC + \overline{B}
\end{aligned}
$$

★ Note : Practice these problems to improve your skills in simplifying Boolean expressions using Boolean Algebra.

LOGIC GATES, BOOLEAN ALGEBRA & SIMPLIFICATION OF LOGIC

Max. term and Min term

Max. terms and Min terms are standard product and standard sum terms used to represent Boolean functions.

Key Points :

- These terms are the basic building blocks for writing Boolean expressions.
- Minterms are used in Sum of Products (SOP) form.
- Maxterms are used in Product of Sums (POS) form.
- They help in converting truth tables into Boolean expressions.

Basic Notation :

★ Max. term → Represented by (M_i)
It is a sum (OR) term in which the output becomes 0 for only one combination of variables.

★ Min. term → Represented by (m_i)
It is a product (AND) term in which the output becomes 1 for only one combination of variables.

1. Min. term (m_i) :

- A minterm is an AND term that includes every variable in either true (uncomplemented) or complement form.
- For n variables, there are 2^n minterms.
- Each minterm is equal to 1 for only one row of the truth table and 0 for all others.

Example : For two variables A and B

A	B	Output (Decimal)	Min. term
0	0	0	$m_0 = \overline{A} \cdot \overline{B}$
0	1	1	$m_1 = \overline{A} \cdot B$
1	0	2	$m_2 = A \cdot \overline{B}$
1	1	3	$m_3 = A \cdot B$

- m_0 is 1 when A=0, B=0
- m_1 is 1 when A=0, B=1
- m_2 is 1 when A=1, B=0
- m_3 is 1 when A=1, B=1

2. Max. term (M_i) :

- A maxterm is an OR term that includes every variable in either true (uncomplemented) or complement form.
- For n variables, there are 2^n maxterms.
- Each maxterm is equal to 0 for only one row of the truth table and 1 for all others.

Example : For two variables A and B

A	B	Output (Decimal)	Max. term
0	0	0	$M_0 = A + B$
0	1	1	$M_1 = A + \overline{B}$
1	0	2	$M_2 = \overline{A} + B$
1	1	3	$M_3 = \overline{A} + \overline{B}$

- M_0 is 0 when A=0, B=0
- M_1 is 0 when A=0, B=1
- M_2 is 0 when A=1, B=0
- M_3 is 0 when A=1, B=1

★ Note : Min. terms are used to write functions in SOP form.
Max. terms are used to write functions in POS form.

LOGIC GATES, BOOLEAN ALGEBRA & SIMPLIFICATION OF LOGIC

Canonical form of equation

Canonical form of an equation is a standard form in which the expression is written as a sum of minterms (SOP form) or a product of maxterms (POS form).

Key Points :

- Canonical forms are unique for a given function.
- In canonical SOP form, all terms are minterms.
- In canonical POS form, all terms are maxterms.
- Every variable appears in each term in either true or complemented form.

Basic Notation :

m_i → Represents minterm number i

M_i → Represents maxterm number i

SOP (Sum of Products) → Sum of minterms

POS (Product of Sums) → Product of maxterms

1. Canonical SOP form (Sum of minterms) :

- A Boolean function can be expressed as a sum of minterms for which the function is 1.
- Each minterm contains all variables in either true or complemented form.
- It is also called Standard Sum of Products.

Example : For two variables A and B

A	B	Output (F)	Minterm
0	0	0	$m_0 = \bar{A}.\bar{B}$
0	1	1	$m_1 = \bar{A}.B$
1	0	1	$m_2 = A.\bar{B}$
1	1	0	$m_3 = A.B$

- If F = 1 for minterms m_1 and m_2 then canonical SOP form is $F(A,B) = \sum m(1,2) = \bar{A}.B + A.\bar{B}$
- If F = 1 for all minterms $F = \sum m(0,1,2,3) = 1$
- If F = 0 for all minterms $F = \sum m(\emptyset) = 0$

2. Canonical POS form (Product of maxterms) :

- A Boolean function can be expressed as a product of maxterms for which the function is 0.
- Each maxterm contains all variables in either true or complemented form.
- It is also called Standard Product of Sums.

Example : For two variables A and B

A	B	Output (F)	Maxterm
0	0	0	$M_0 = (A + B)$
0	1	1	$M_1 = (A + \bar{B})$
1	0	1	$M_2 = (\bar{A} + B)$
1	1	0	$M_3 = (\bar{A} + \bar{B})$

- If F = 0 for maxterms M_0 and M_3 then canonical POS form is $F(A,B) = \prod M(0,3) = (A + B).(\bar{A} + \bar{B})$
- If F = 0 for all maxterms $F = \prod M(0,1,2,3) = 0$
- If F = 1 for all maxterms $F = \prod M(\emptyset) = 1$

★ Note :
- Canonical forms are very helpful in using K-map for simplification.
- Any Boolean function can be written in both canonical SOP and POS forms.

LOGIC GATES, BOOLEAN ALGEBRA & SIMPLIFICATION OF LOGIC

Min Term Table (3 and 4 Variables)

A minterm is a product term that includes all variables in either true (uncomplemented) or complement form and evaluates to 1 for only one combination of variables.

Key Points :

- Minterms are used in SOP (Sum of Products) form.
- For n variables, there are 2^n minterms.
- Each minterm corresponds to one unique row in the truth table where the output is 1.
- Minterm m_i is written in the order of variables as A, B, C, D ...

Basic Notation :

m_i → Minterm number i

n → Number of variables

Minterm contains all variables in either true (uncomplemented) or complement form.

For n variables → Total minterms = 2^n

1. Minterm Table (3 Variables)

A	B	C	Minterm Number	Minterm Expression
0	0	0	m_0	$\bar{A}\ \bar{B}\ \bar{C}$
0	0	1	m_1	$\bar{A}\ \bar{B}\ C$
0	1	0	m_2	$\bar{A}\ B\ \bar{C}$
0	1	1	m_3	$\bar{A}\ B\ C$
1	0	0	m_4	$A\ \bar{B}\ \bar{C}$
1	0	1	m_5	$A\ \bar{B}\ C$
1	1	0	m_6	$A\ B\ \bar{C}$
1	1	1	m_7	$A\ B\ C$

For 3 variables (A, B, C):

- Total minterms = 2^3 = 8 (from m_0 to m_7)
- Each minterm is 1 for only one combination.

How to Write Minterm ?

1. Take the variables in a fixed order.
2. For each row of the truth table:
   - → Write the variable as it is if the value is 1
   - → Write it with bar (complement) if the value is 0.
3. Multiply (AND) all the variables.

2. Minterm Table (4 Variables)

A	B	C	D	Minterm Number	Minterm Expression
0	0	0	0	m_0	$\bar{A}\ \bar{B}\ \bar{C}\ \bar{D}$
0	0	0	1	m_1	$\bar{A}\ \bar{B}\ \bar{C}\ D$
0	0	1	0	m_2	$\bar{A}\ \bar{B}\ C\ \bar{D}$
0	0	1	1	m_3	$\bar{A}\ \bar{B}\ C\ D$
0	1	0	0	m_4	$\bar{A}\ B\ \bar{C}\ \bar{D}$
0	1	0	1	m_5	$\bar{A}\ B\ \bar{C}\ D$
0	1	1	0	m_6	$\bar{A}\ B\ C\ \bar{D}$
0	1	1	1	m_7	$\bar{A}\ B\ C\ D$
1	0	0	0	m_8	$A\ \bar{B}\ \bar{C}\ \bar{D}$
1	0	0	1	m_9	$A\ \bar{B}\ \bar{C}\ D$
1	0	1	0	m_{10}	$A\ \bar{B}\ C\ \bar{D}$
1	0	1	1	m_{11}	$A\ \bar{B}\ C\ D$
1	1	0	0	m_{12}	$A\ B\ \bar{C}\ \bar{D}$
1	1	0	1	m_{13}	$A\ B\ \bar{C}\ D$
1	1	1	0	m_{14}	$A\ B\ C\ \bar{D}$
1	1	1	1	m_{15}	$A\ B\ C\ D$

For 4 variables (A, B, C, D):

- Total minterms = 2^4 = 16 (from m_0 to m_{15})
- Each minterm is 1 for only one combination.

★ **Note :** Minterm tables are very useful in writing Boolean functions in canonical SOP form and in K-map simplification.

LOGIC GATES, BOOLEAN ALGEBRA & SIMPLIFICATION OF LOGIC

Max Term Table (3 and 4 Variables)

A maxterm is a sum term that includes all variables in either true (uncomplemented) or complement form and evaluates to 0 for only one combination of variables.

Key Points :

- Maxterms are used in POS (Product of Sums) form.
- For n variables, there are 2^n maxterms.
- Each maxterm corresponds to one unique row in the truth table where the output is 0.
- Maxterm M_i is written in the order of variables as A, B, C, D ...

Basic Notation :

M_i → Maxterm number i

n → Number of variables

Maxterm contains all variables in either true (uncomplemented) or complement form.

For n variables → Total maxterms = 2^n

1. Maxterm Table (3 Variables)

A	B	C	Maxterm Number	Maxterm Expression
0	0	0	M_0	$A + B + C$
0	0	1	M_1	$A + B + \overline{C}$
0	1	0	M_2	$A + \overline{B} + C$
0	1	1	M_3	$A + \overline{B} + \overline{C}$
1	0	0	M_4	$\overline{A} + B + C$
1	0	1	M_5	$\overline{A} + B + \overline{C}$
1	1	0	M_6	$\overline{A} + \overline{B} + C$
1	1	1	M_7	$\overline{A} + \overline{B} + \overline{C}$

For 3 variables (A, B, C):

- Total maxterms = $2^3 = 8$ (from M_0 to M_7)
- Each maxterm is 0 for only one combination.

How to Write Maxterm ?

1. Take the variables in a fixed order.
2. For each row of the truth table:
   - → Write the variable as it is if the value is 0
   - → Write it with bar (complement) if the value is 1.
3. Add all the variables with OR (+).

2. Maxterm Table (4 Variables)

A	B	C	D	Maxterm Number	Maxterm Expression
0	0	0	0	M_0	$A + B + C + D$
0	0	0	1	M_1	$A + B + C + \overline{D}$
0	0	1	0	M_2	$A + B + \overline{C} + D$
0	0	1	1	M_3	$A + B + \overline{C} + \overline{D}$
0	1	0	0	M_4	$A + \overline{B} + C + D$
0	1	0	1	M_5	$A + \overline{B} + C + \overline{D}$
0	1	1	0	M_6	$A + \overline{B} + \overline{C} + D$
0	1	1	1	M_7	$A + \overline{B} + \overline{C} + \overline{D}$
1	0	0	0	M_8	$\overline{A} + B + C + D$
1	0	0	1	M_9	$\overline{A} + B + C + \overline{D}$
1	0	1	0	M_{10}	$\overline{A} + B + \overline{C} + D$
1	0	1	1	M_{11}	$\overline{A} + B + \overline{C} + \overline{D}$
1	1	0	0	M_{12}	$\overline{A} + \overline{B} + C + D$
1	1	0	1	M_{13}	$\overline{A} + \overline{B} + C + \overline{D}$
1	1	1	0	M_{14}	$\overline{A} + \overline{B} + \overline{C} + D$
1	1	1	1	M_{15}	$\overline{A} + \overline{B} + \overline{C} + \overline{D}$

For 4 variables (A, B, C, D):

- Total maxterms = $2^4 = 16$ (from M_0 to M_{15})
- Each maxterm is 0 for only one combination.

★ **Note :** Maxterm tables are very useful in writing Boolean functions in canonical POS form and in K-map simplification.

Logic Gates, Boolean Algebra & Simplification of Logic

Realization of Boolean expression with different logic gates

A Boolean expression can be implemented using different combinations of logic gates. Above the expression may be realized in more than one way.

Key Points :

- Boolean expressions can be realized using different logic gates.
- Choose the gates that are available or most suitable.
- Complement can be realised using NOT gate.
- AND gate → Product term
  OR gate → Sum term

Basic Notation :

A + B	→	OR operation
A . B	→	AND operation
$\overline{A}$	→	NOT A (Complement of A)
AB + $\overline{C}$	→	A . B + NOT C

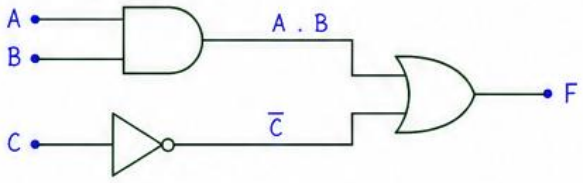
1. Realize the Boolean expression F = A . B + $\overline{C}$ using different logic gates.

Solution :

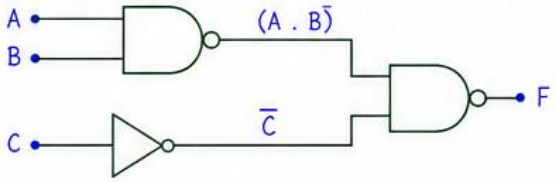
Steps :

1. Expression is F = A . B + $\overline{C}$
2. A . B is a product term → use AND gate.
3. $\overline{C}$ is complement of C → use NOT gate.
4. Now, OR the outputs of (A . B) and ($\overline{C}$) → use OR gate.
5. Output of OR gate is F.

(a) Using AND, OR, NOT Gates



(b) Using NAND and NOT Gates



Explanation :
NAND gate gives complement of AND. Hence, (A . B)‾ is obtained first. Then NANDing with $\overline{C}$ gives F.

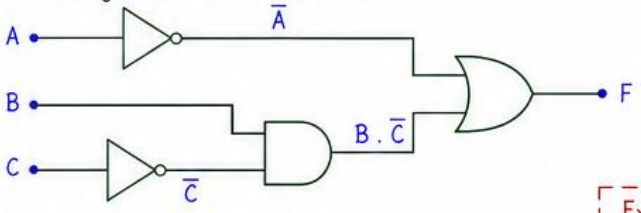
2. Realize the Boolean expression F = $\overline{A}$ + B . $\overline{C}$ using different logic gates.

Solution :

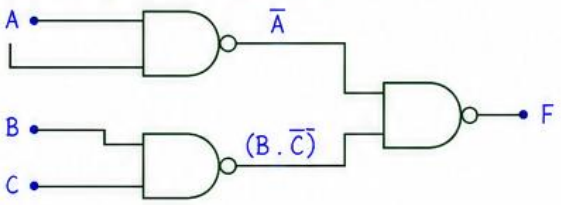
Steps :

1. Expression is F = $\overline{A}$ + B . $\overline{C}$
2. $\overline{A}$ is complement of A → use NOT gate.
3. B . $\overline{C}$ is product term → use AND gate.
4. OR the outputs of ($\overline{A}$) and (B . $\overline{C}$) → use OR gate.
5. Output of OR gate is F.

(a) Using AND, OR, NOT Gates



(b) Using NAND Gates Only



Explanation :
First NAND with same inputs gives NOT.
Second NAND gives complement of (B . $\overline{C}$).
Final NAND gives OR of the two terms.

★ **Note :** Any Boolean expression can be realized using basic gates in more than one way. Selection of gates depends on availability and convenience.

LOGIC GATES, BOOLEAN ALGEBRA & SIMPLIFICATION OF LOGIC

Realization of Boolean expression with different logic gates

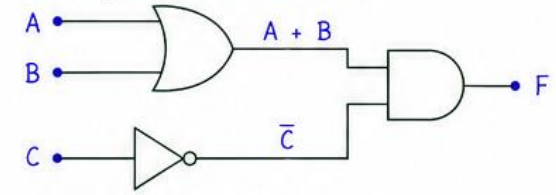
3. Realize the Boolean expression $F = (A + B) \cdot \overline{C}$ using different logic gates.

Solution :

Steps :

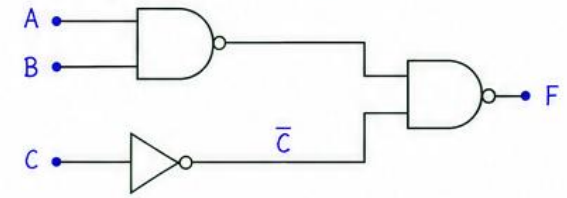
1. Expression is $F = (A + B) \cdot \overline{C}$
2. $(A + B)$ is sum term → use OR gate.
3. $\overline{C}$ is complement of C → use NOT gate.
4. AND the outputs of $(A + B)$ and $(\overline{C})$ → use AND gate.
5. Output of AND gate is F.

(a) Using AND, OR, NOT Gates



Explanation :
OR gate gives the sum of A and B.
NOT gate gives complement of C.
ANDing $(A + B)$ with $\overline{C}$ gives F.

(b) Using NAND Gate Only



Explanation :
First NAND gives $(\overline{A} + B)$.
Second NAND with $(\overline{A} + B)$ and $\overline{C}$ gives $(A + B) \cdot \overline{C}$.

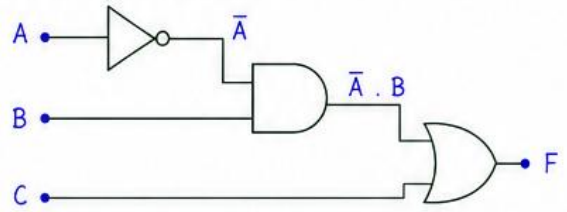
4. Realize the Boolean expression $F = \overline{A} \cdot B + C$ using different logic gates.

Solution :

Steps :

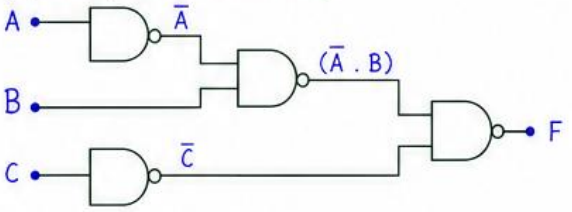
1. Expression is $F = \overline{A} \cdot B + C$
2. $\overline{A}$ is complement of A → use NOT gate.
3. $\overline{A} \cdot B$ is product term → use AND gate.
4. OR the output of $(\overline{A} \cdot B)$ with C → use OR gate.
5. Output of OR gate is F.

(a) Using AND, OR, NOT Gates



Explanation :
A is inverted to get $\overline{A}$.
AND with B gives $(\overline{A} \cdot B)$.
OR with C gives $F = \overline{A} \cdot B + C$.

(b) Using NAND Gates Only



Explanation :
First NAND gives $\overline{A}$.
Second NAND gives $(\overline{A} \cdot B)'$.
Third NAND with $(\overline{A} \cdot B)'$ and $\overline{C}$ gives $\overline{((\overline{A} \cdot B)' \cdot \overline{C})} = \overline{A} \cdot B + C$.

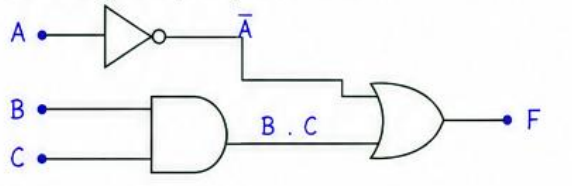
5. Realize the Boolean expression $F = \overline{A} + B \cdot C$ using different logic gates.

Solution :

Steps :

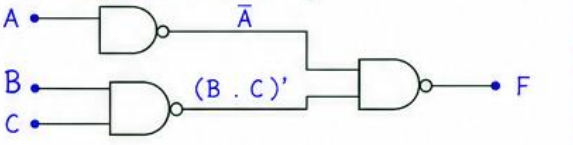
1. Expression is $F = \overline{A} + B \cdot C$
2. $\overline{A}$ is complement of A → use NOT gate.
3. $B \cdot C$ is product term → use AND gate.
4. OR the outputs of $\overline{A}$ and $(B \cdot C)$ → use OR gate.
5. Output of OR gate is F.

(a) Using AND, OR, NOT Gates



Explanation :
A is inverted to get $\overline{A}$.
AND of B and C gives $(B \cdot C)$.
ORing $\overline{A}$ with $(B \cdot C)$ gives $F = \overline{A} + B \cdot C$.

(b) Using NAND Gates Only



Explanation :
First NAND gives $\overline{A}$.
Second NAND gives $(B \cdot C)'$.
Third NAND with $\overline{A}$ and $(B \cdot C)'$ gives $\overline{(\overline{A} \cdot (B \cdot C)')} = \overline{A} + B \cdot C$.

★ Note : Any Boolean expression can be realized using basic gates in more than one way. Selection of gates depends on availability and convenience.

LOGIC GATES, BOOLEAN ALGEBRA & SIMPLIFICATION OF LOGIC

Realization of Boolean expression with different logic gates

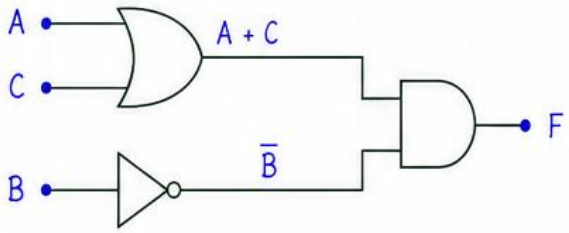
6. Realize the Boolean expression $F = (A + C) \cdot \overline{B}$ using different logic gates.

Solution :

Steps :

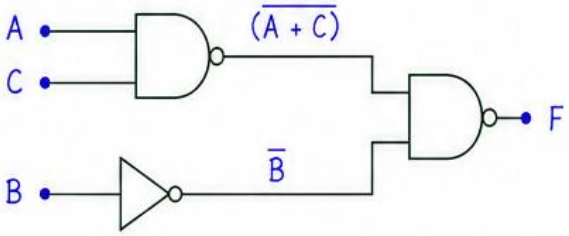
1. Expression is $F = (A + C) \cdot \overline{B}$
2. $(A + C)$ is sum term → use OR gate.
3. B is complemented → use NOT gate.
4. AND the outputs of $(A + C)$ and $(\overline{B})$ → use AND gate.
5. Output of AND gate is F.

(a) Using AND, OR, NOT Gates



Explanation :
OR gate gives the sum of A and C.
NOT gate complements B to get $\overline{B}$.
ANDing $(A + C)$ and $\overline{B}$ gives F.

(b) Using NAND Gates Only



Explanation :
First NAND gives $\overline{(A + C)}$.
Second NAND with $\overline{(A + C)}$ and $\overline{B}$ gives complement of $(\overline{A} + C) \cdot \overline{B})$.
Hence,
$F = (A + C) \cdot \overline{B}$

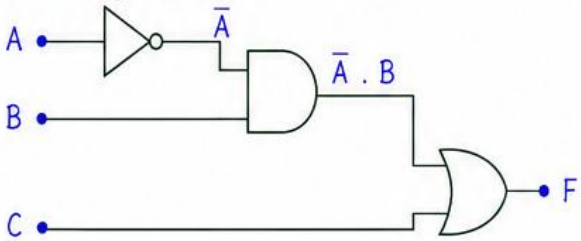
7. Realize the Boolean expression $F = \overline{A} \cdot B + C$ using different logic gates.

Solution :

Steps :

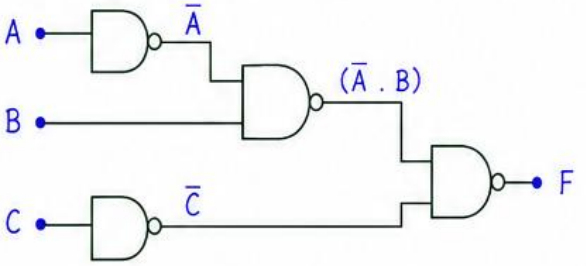
1. Expression is $F = \overline{A} \cdot B + C$
2. $\overline{A}$ is complemented → use NOT gate.
3. $\overline{A} \cdot B$ is product term → use AND gate.
4. OR the output of $(\overline{A} \cdot B)$ with C → use OR gate.
5. Output of OR gate is F.

(a) Using AND, OR, NOT Gates



Explanation :
NOT gate gives $\overline{A}$.
AND with B gives $(\overline{A} \cdot B)$.
OR with C gives $F = (\overline{A} \cdot B) + C$.

(b) Using NAND Gates Only



Explanation :
First NAND gives $\overline{A}$.
Second NAND gives $(\overline{A} \cdot B)$.
Third NAND with $(\overline{A} \cdot B)$ and $\overline{C}$ gives $F = (\overline{A} \cdot B) + C$.

★ Note : Boolean expressions can be realized using different combinations of gates. Choose the method that is simple and convenient.

LOGIC GATES, BOOLEAN ALGEBRA & SIMPLIFICATION OF LOGIC

From Logic Gate to Boolean Expression

A logic circuit can be converted into Boolean expression by writing the expression for each gate and then combining them step by step.

Key Points :

- Write the Boolean expression for each gate.
- Start from the inputs and move towards the output.
- Replace each gate by its corresponding Boolean operation.
- Combine the expressions to get the final output expression.

Basic Notation :

A + B	→	OR operation
A . B	→	AND operation
$\overline{A}$	→	NOT A (Complement of A)
AB + $\overline{C}$	→	A . B + NOT C

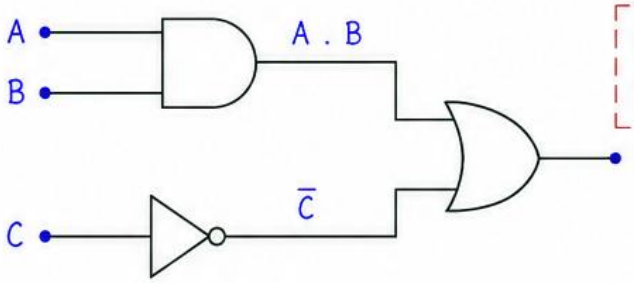
(Follow the order of gates to write expression.)

1. Obtain the Boolean expression for the output F of the given logic circuit.

Solution :

Steps :

1. A and B are connected to AND gate → Output = A . B
2. C is inverted by NOT gate → Output = $\overline{C}$
3. Outputs (A . B) and $\overline{C}$ are given to OR gate → Output F = (A . B) + $\overline{C}$



Final Expression :

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

Explanation :

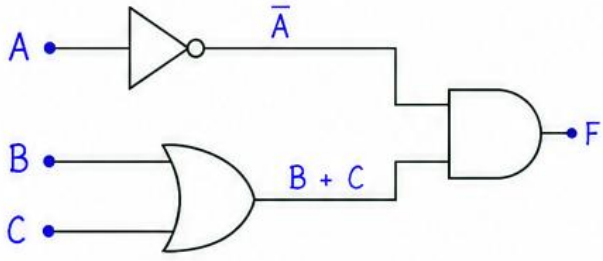
First AND the inputs A and B. Invert C. Finally OR the two results to get F.

2. Obtain the Boolean expression for the output F of the given logic circuit.

Solution :

Steps :

1. A is inverted → Output = $\overline{A}$
2. B and C are given to OR gate → Output = (B + C)
3. Outputs $\overline{A}$ and (B + C) are given to AND gate → Output F = $\overline{A}$. (B + C)



Final Expression :

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

Explanation :

Invert A to get $\overline{A}$. OR the inputs B and C. Finally AND the two results to get F.

★ **Note :** Always follow the flow of the circuit from inputs to output while writing the Boolean expression.

LOGIC GATES, BOOLEAN ALGEBRA & SIMPLIFICATION OF LOGIC

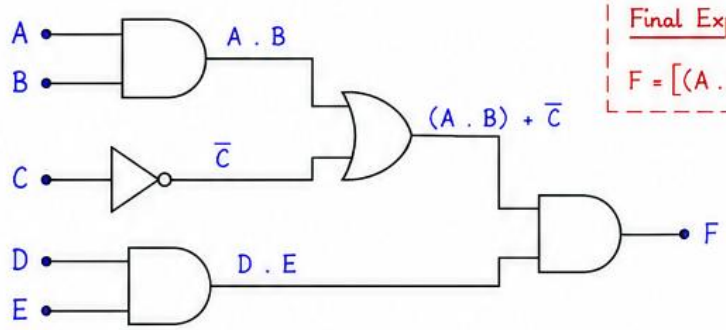
From Logic Gate to Boolean Expression

3. Obtain the Boolean expression for the output F of the given logic circuit.

Solution :

Steps :

1. A and B are given to AND gate → Output = (A . B)
2. C is inverted by NOT gate → Output = $\overline{C}$
3. (A . B) and $\overline{C}$ are given to OR gate → Output = (A . B) + $\overline{C}$
4. D and E are given to AND gate → Output = (D . E)
5. The outputs of OR gate and AND gate are given to AND gate → Output F = $[(A.B)+\overline{C}].(D.E)$



Final Expression :

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

Explanation :

AND A and B. Invert C. OR the results of (A . B) and $\overline{C}$. AND this result with (D . E) to get F.

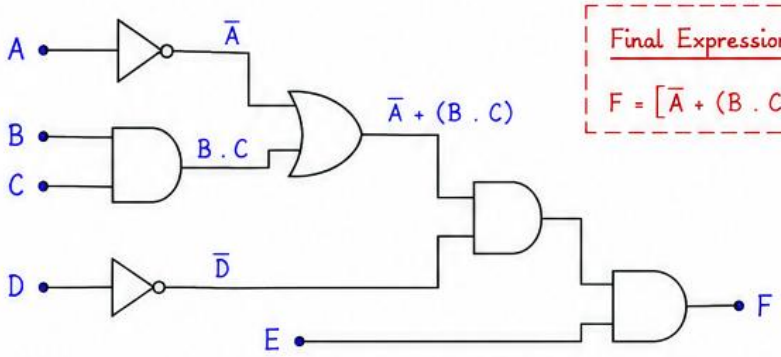
---

4. Obtain the Boolean expression for the output F of the given logic circuit.

Solution :

Steps :

1. A is inverted → Output = $\overline{A}$
2. B and C are given to AND gate → Output = (B . C)
3. $\overline{A}$ and (B . C) are given to OR gate → Output = $\overline{A}$ + (B . C)
4. D is inverted → Output = $\overline{D}$
5. ($\overline{A}$ + (B . C)) and $\overline{D}$ are given to AND gate → Output = $[\overline{A} + (B . C)] . \overline{D}$
6. E is given to AND gate together with above output → Output F = $[\overline{A} + (B . C)] . \overline{D} . E$



Final Expression :

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

Explanation :

Invert A and D. AND B and C. OR the results of $\overline{A}$ and (B . C). AND this with $\overline{D}$. Finally AND with E to get F.

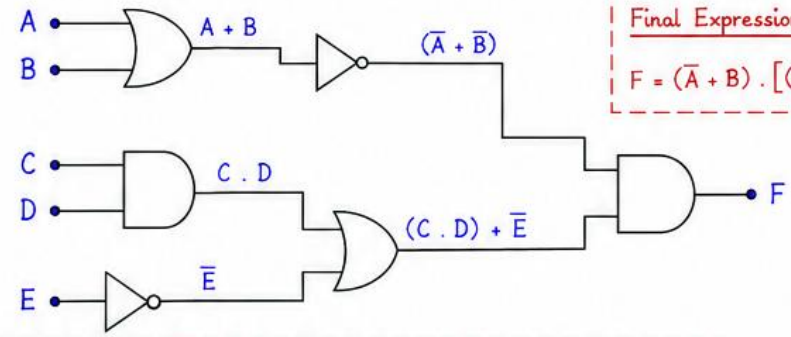
---

5. Obtain the Boolean expression for the output F of the given logic circuit.

Solution :

Steps :

1. A and B are given to OR gate → Output = (A + B)
2. Output is inverted by NOT gate → Output = $(\overline{A} + \overline{B})$
3. C and D are given to AND gate → Output = (C . D)
4. E is inverted by NOT gate → Output = $\overline{E}$
5. (C . D) and $\overline{E}$ are given to OR gate → Output = (C . D) + $\overline{E}$
6. The outputs of steps 2 and 5 are given to AND gate → Output F = $(\overline{A} + \overline{B}) . [(C . D) + \overline{E}]$



Final Expression :

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

Explanation :

OR A and B then invert. AND C and D. Invert E. OR the results of (C . D) and $\overline{E}$. Finally AND the two results to obtain F.

★ **Note :** Always write the expression in the order of gates from inputs to output.

LOGIC GATES, BOOLEAN ALGEBRA & SIMPLIFICATION OF LOGIC

KARNAUGH MAP TECHNIQUE

What is Karnaugh Map?

A Karnaugh Map (K-map) is a simple and effective graphical method used to minimize a Boolean function. It helps in getting the simplest form of a logic expression by grouping the adjacent 1's (or 0's) in a systematic way.

Key Points :

- K-map is used for minimization of Boolean functions.
- It is based on the concept of adjacency of minterms (or maxterms).
- Adjacent cells differ in only one variable.
- Used for 2, 3, 4 and more variables (we can extend the map for more variables).

Important Rules :

1. Group the 1's (for SOP) or 0's (for POS) in rectangular blocks of size 1, 2, 4, 8,...
2. The groups must be as large as possible.
3. Groups can overlap.
4. Each group should have a power of 2 number of cells.
5. The variables that do not change within a group are eliminated from the term.

Example : Minimize the Boolean expression for the given function using K-map.

Given Function : $F(A, B, C, D) = \Sigma\, m(0, 1, 2, 3, 8, 9, 10, 11)$

Step 1: Draw the K-map :

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

Step 2: Identify the groups :

- Group 1: 4 cells (m_0, m_1, m_8, m_9)
- Group 2: 4 cells (m_2, m_3, m_{10}, m_{11})
- Each group gives a single term.

Step 3: Write the simplified expression :

- Group 1 (m_0, m_1, m_8, m_9) → $\overline{B}\,\overline{D}$
- Group 2 (m_2, m_3, m_{10}, m_{11}) → $\overline{B}C$

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

Another Example (POS form) :

For the function : $F = \Pi M(0, 2, 5, 7)$, the simplified POS form is :

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

★ **Note :** K-map is very useful for small number of variables (up to 4). For more variables, we can use tabular or algorithmic method.

LOGIC GATES, BOOLEAN ALGEBRA & SIMPLIFICATION OF LOGIC

KARNAUGH MAP WITH MINTERM AND MAXTERM NOTATION

How to put the values in K-map using Minterm and Maxterm notation :

In K-map, minterm and maxterm notation gives the position of 1's and 0's respectively. The values are placed according to the binary form of the term.

Key Points :

- Minterm (m) : 1 is placed at the cell corresponding to the given minterm.
- Maxterm (M) : 0 is placed at the cell corresponding to the given maxterm.
- For n variables, the cell address is the binary value of the term.

K-map cell order (for 4 variables) :

AB \ CD	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

(Same order is used for maxterms also. Only the values will be 0 instead of 1.)

1. Example using Minterm Notation :

Given : $F(A, B, C, D) = \Sigma m(0, 2, 5, 7, 8, 10, 11, 14)$

Step 1 : Place 1's in the cells corresponding to the given minterms.

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

Explanation :

- $m_0 \rightarrow (0,0) \rightarrow 1$
- $m_2 \rightarrow (0,1) \rightarrow 1$
- $m_5 \rightarrow (1,1) \rightarrow 1$
- $m_7 \rightarrow (1,3) \rightarrow 1$
- $m_8 \rightarrow (2,0) \rightarrow 1$
- $m_{10} \rightarrow (2,3) \rightarrow 1$
- $m_{11} \rightarrow (2,2) \rightarrow 1$
- $m_{14} \rightarrow (3,2) \rightarrow 1$

Final Expression :

$F = \Sigma m(0, 2, 5, 7, 8, 10, 11, 14)$

(or)

$F = m_0 + m_2 + m_5 + m_7 + m_8 + m_{10} + m_{11} + m_{14}$

2. Example using Maxterm Notation :

Given : $F(A, B, C, D) = \Pi M(1, 3, 4, 6, 9, 12, 13, 15)$

Step 1 : Place 0's in the cells corresponding to the given maxterms.

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

Esplanation :

- $M_1 \rightarrow (0,1) \rightarrow 0$
- $M_3 \rightarrow (0,3) \rightarrow 0$
- $M_4 \rightarrow (1,0) \rightarrow 0$
- $M_6 \rightarrow (1,3) \rightarrow 0$
- $M_9 \rightarrow (2,1) \rightarrow 0$
- $M_{12} \rightarrow (3,0) \rightarrow 0$
- $M_{13} \rightarrow (3,1) \rightarrow 0$
- $M_{15} \rightarrow (3,3) \rightarrow 0$

Final Expression :

$F = \Pi M(1, 3, 4, 6, 9, 12, 13, 15)$

(or)

$F = (M_1 + M_3 + M_4 + M_6)(M_9 + M_{12} + M_{13} + M_{15})$

★ Note :

- In minterm notation, put 1's at the given minterms.
- In maxterm notation, put 0's at the given maxterms.
- Always use the binary form of the term to find the correct cell in K-map.

LOGIC GATES, BOOLEAN ALGEBRA & SIMPLIFICATION OF LOGIC

KARNAUGH MAP TECHNIQUE

Solved Questions (with Answer Steps)

1. Minimize the following function using Karnaugh Map technique :

$F(A, B, C, D) = \Sigma m(0, 2, 5, 7, 8, 10, 11, 14)$

Step 1 : Draw the K-map :

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

Step 2 : Identify the groups :

- Group 1 : m_0, m_2, m_8, m_{10} (4 cells) (using A' and C')
- Group 2 : m_5, m_7, m_{11}, m_{14} (4 cells) (using B and D)
- Group 3 : $m_{11}, m_{15}, m_{10}, m_{14}$ (using A and D')

Step 3 : Write the simplified expression :

Group 1 → A'C' Group 2 → BD

Group 3 → AD'

So, $F = A'C' + BD + AD'$

Final Answer :

$F = A'C' + BD + AD'$

2. Minimize the following function using Karnaugh Map technique :

$F(A, B, C, D) = \Sigma m(1, 3, 4, 7, 8, 9, 12, 13)$

Step 1 : Draw the K-map :

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

Step 2 : Identify the groups :

- Group 1 : m_1, m_3 (2 cells) → B'D
- Group 2 : m_4, m_5 (2 cells) → A'B
- Group 3 : m_8, m_9 (2 cells) → AB'
- Group 4 : m_{12}, m_{13} (2 cells) → AB

Step 3 : Write the simplified expression :

$F = B'D + A'B + AB' + AB$

(or $F = A \oplus B + C'D'$ (if further simplification needed)

Final Answer :

$F = B'D + A'B + AB' + AB$

★ **Note :** In K-map, don't care conditions (X) can also be used for grouping to get a simplified expression.

LOGIC GATES, BOOLEAN ALGEBRA & SIMPLIFICATION OF LOGIC

KARNAUGH MAP TECHNIQUE

3. Minimize the following function using Karnaugh Map technique :

$F(A, B, C, D) = \Sigma m(0, 2, 5, 7, 8, 10, 13, 15)$

Step 1 : Draw the K-map :

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

Step 2 : Identify the groups :

- Group 1 : 4 cells (m_0, m_2, m_8, m_{10}) (using A' and D')
- Group 2 : 2 cells (m_5, m_7) (using B and D)
- Group 3 : 2 cells (m_{13}, m_{15}) (using C and D)

Step 3 : Write the simplified expression :

$F = A'D' + B'D + CD'$

(order of terms can be changed)

Final Answer :

$F = B'D + A'D' + CD'$

4. Minimize the following function using Karnaugh Map technique :

$F(A, B, C, D) = \Sigma m(1, 3, 4, 6, 8, 9, 11, 14)$

Step 1 : Draw the K-map :

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

Step 2 : Identify the groups :

- Group 1 : 4 cells (m_1, m_3, m_{11}, m_{14}) (using B' and D)
- Group 2 : 4 cells (m_0, m_4, m_6, m_{10}) (using C and D')
- Group 3 : 2 cells (m_8, m_9) (using A' and B)

Step 3 : Write the simplified expression :

$F = B'D + CD' + A'B$

(or $F = B'D + A'B + CD'$)

Final Answer :

$F = B'D + CD' + A'B$

★ **Note :** In K-map, we can group the 1's in different ways. But we always choose the largest possible groups to get the simplest expression.

LOGIC GATES, BOOLEAN ALGEBRA & SIMPLIFICATION OF LOGIC

KARNAUGH MAP TECHNIQUE

5. Minimize the following function using Karnaugh Map technique :

$F(A, B, C, D) = \Sigma m(0, 2, 5, 7, 8, 10, 12, 14)$

Step 1 : Draw the K-map :

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

Step 2 : Identify the groups :

- Group 1 : 4 cells (m_0, m_2, m_8, m_{10}) (using A' and D')
- Group 2 : 4 cells (m_5, m_7, m_{12}, m_{14}) (using B and D')
- Group 3 : 2 cells (m_1, m_5) (using A' and B)

Step 3 : Write the simplified expression :

$F = A'D' + B'D' + AB'$

(order of terms can be changed)

Final Answer :

$F = A'D' + AB' + BD'$

6. Minimize the following function using Karnaugh Map technique :

$F(A, B, C, D) = \Sigma m(1, 3, 4, 7, 8, 9, 11, 13)$

Step 1 : Draw the K-map :

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

Step 2 : Identify the groups :

- Group 1 : 4 cells (m_1, m_4, m_7, m_9) (using B' and D)
- Group 2 : 4 cells (m_3, m_8, m_{11}, m_{13}) (using B and D')

Step 3 : Write the simplified expression :

$F = B'D + BD'$

(order of terms can be changed)

Final Answer :

$F = B'D' + BD'$

★ **Note :** In K-map, we can group the 1's in different ways. But we always choose the largest possible groups to get the simplest expression.

LOGIC GATES, BOOLEAN ALGEBRA & SIMPLIFICATION OF LOGIC

DON'T CARE CONDITION (KARNAUGH MAP)

What is Don't Care Condition?

A don't care condition is a condition for which the output is not specified in the given logic function. It means that the output can be either 0 or 1 according to our convenience. These conditions are represented by the symbol 'X' in the Karnaugh map.

Key Points :

- Don't care condition is used to further simplify the expression.
- It helps to get a more simplified result (minimum terms).
- In K-map, don't care conditions are written as 'X'.
- We can group the 'X' with 0 or 1 (whichever gives the larger group).

Important Rules :

1. 'X' can be grouped with 0 or 1 (whichever gives maximum simplification).
2. It can help to form larger groups than required.
3. Don't care conditions do not affect the final output, so they can be used for simplification.

Example : Simplify the following function using K-map with don't care condition.

Given Function : $F(A, B, C, D) = \Sigma m(0, 1, 2, 5, 8, 9) + d(3, 7)$

Step 1 : Draw the K-map :

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

Step 2 : Identify the groups :

- Group 1 : 4 cells (m_0, m_1, m_8, m_9) (using don't care X at 3).
- Group 2 : 4 cells (m_5, m_7, m_{12}, m_{13}) (using don't care X at 7).

Step 3 : Write the simplified expression :

- Group 1 gives $\overline{A}\,\overline{B}$ and Group 2 gives $B\overline{D}$.

So, $F = \overline{A}\,\overline{B} + B\overline{D}$.

Another Example (POS form) :

For the function : $F = \Sigma m(0, 2, 5, 7) + d(3, 15)$, the simplified POS form is :

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

★ **Note :** Don't care condition can be used only for simplification. It does not change the function, it just helps to get a simpler expression.

LOGIC GATES, BOOLEAN ALGEBRA & SIMPLIFICATION OF LOGIC

DON'T CARE CONDITION (KARNAUGH MAP)

Question 1 : Simplify the following function using K-map with don't care condition.

$F(A, B, C, D) = \Sigma m(0, 1, 2, 5, 6, 9) + d(3, 7)$

Step 1 : Draw the K-map :

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

Step 2 : Identify the groups :

- Group 1 : (0,0), (0,1), (1,0), (1,1) → $\bar{A}\bar{C}$
  (using minterms 0, 1, 2 and don't cares 3, 7)
- Group 2 : (1,0), (1,1) → $\bar{B}\bar{D}$
  (using minterms 8, 9)
- Group 3 : (1,0) → $\bar{B}\bar{D}$
  (using minterm 9)

Step 3 : Write the simplified expression :

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

Final Answer :

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

Question 2 : Simplify the following function using K-map with don't care condition.

$F(A, B, C, D) = \Sigma m(0, 2, 3, 5, 8, 10) + d(1, 7, 15)$

Step 1 : Draw the K-map :

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

Step 2 : Identify the groups :

- Group 1 : (0,0), (0,10), (3,0), (3,10)
  (using minterms 0, 2, 8, 10) → $\bar{B}\bar{C}$
- Group 2 : (2,1), (2,11) → $\bar{C}\bar{D}$
  (using minterms 5, 3)
- Group 3 : (0,0), (0,10) → $\bar{B}\bar{D}$
  (using minterms 0, 8)

Step 3 : Write the simplified expression :

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

Final Answer :

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

★ **Note :** Don't care conditions (X) can be used to form larger groups in K-map which helps to get a simpler expression.

LOGIC GATES, BOOLEAN ALGEBRA & SIMPLIFICATION OF LOGIC

DON'T CARE CONDITION (KARNAUGH MAP)

3. Simplify the following function using K-map with don't care condition:

$F(A, B, C, D) = \Sigma m(0, 1, 2, 5, 6, 7, 8, 9, 10, 13) + d(3, 11, 15)$

Step 1 : Draw the K-map :

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

Step 2 : Identify the groups :

- Group 1 : 4 cells (m_0, m_1, m_2, m_3) (using A' and C') → A'C'
- Group 2 : 4 cells (m_5, m_6, m_7, m_{13}) (using B and D) → $B\overline{D}$
- Group 3 : 4 cells $(m_8, m_9, m_{10}, m_{11})$ (using B' and C) → $B'\overline{C}$

Step 3 : Write the simplified expression :

$F = A'C' + BD + B'C$

Final Answer :

$F = B + \overline{D}$

4. Simplify the following function using K-map with don't care condition :

$F(A, B, C, D) = \Sigma m(1, 2, 3, 5, 7, 8, 9, 11, 13, 14) + d(0, 4, 10, 12, 15)$

Step 1 : Draw the K-map :

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

Step 2 : Identify the groups :

- Group 1 : 4 cells (m_0, m_1, m_8, m_9) (using B' and D) → B'D
- Group 2 : 4 cells $(m_3, m_7, m_{11}, m_{15})$ (using C and D') → CD'
- Group 3 : 8 cells $(m_5, m_6, m_7, m_{13}, m_{14}, m_{15})$ (using C) → C

Step 3 : Write the simplified expression :

$F = C + B'D + CD'$

Final Answer :

$F = C + B'D + CD'$

★ **Note :** Don't care conditions (d) can be used to form larger groups in K-map which helps to get a simpler expression.

LOGIC GATES, BOOLEAN ALGEBRA & SIMPLIFICATION OF LOGIC

QUINE-McCLUSKY METHOD

A systematic method to minimize the Boolean function :

The Quine-McClusky method is a tabular technique used to obtain the minimal sum of products (SOP) or product of sums (POS) expression from a given set of minterms and/or maxterms.

Key Points :

- Group the minterms (or maxterms) having the same number of 1's (or 0's).
- Combine terms that differ in only one bit (→ one variable).
- Repeat the process until no further combination is possible.
- The remaining implicants are the prime implicants.
- Use a prime implicant chart to select essential implicants and obtain the minimal expression.

Steps of Quine-McClusky Method :

1. Write the given minterms (or maxterms).
2. Group the terms having the same number of 1's (or 0's).
3. Combine terms that differ in only one bit and put the result in the next group.
4. Repeat step 3 until no more combination is possible.
5. Mark the prime implicants and select the essential prime implicants using a prime implicant chart.
6. Write the simplified expression.

Example : Minimize $F(A, B, C, D) = \Sigma m(0, 1, 2, 5, 6, 8, 9, 10, 14)$ using Quine-McClusky method.

Step 1 : Group the minterms according to the number of 1's.

No. of 1's	Minterms
0	0, 1
1	2, 5
2	6, 8, 9, 10
3	14

Step 2 : Combine the terms (differ in only one bit).

Group	Minterms	Combinations
0	0, 1	0- , -1
1	2, 5	-2 , 5-
2	6, 8, 9, 10	6- , 8- , -9 , -10
3	14	-14

Step 3 : Repeat combination (if possible).

Group	Minterms	New Combinations
0	0- , -1	- -
1	-2 , 5-	- -
2	6- , 8- , -9 , -10	- -

Now - - is the final implicant (covers 0, 1, 2, 5, 6, 8, 9, 10, 14).

Prime Implicant Chart :

PI	Minterms Covered
- -	0, 1, 2, 5, 6, 8, 9, 10, 14

Since this single implicant covers all the minterms, it is the essential prime implicant.

Final Simplified Expression :

$$F = \overline{A}\,\overline{D}$$

★ **Note :**

- The Quine-McClusky method is suitable for a large number of variables.
- It is more systematic and efficient than the Karnaugh map for large functions.