Chapter 06

Boolean Algebra and Logic Circuits

Computer Fundamentals - Pradeep K. Sinha & Priti Sinha
In this chapter you will learn about:

- Boolean algebra
- Fundamental concepts and basic laws of Boolean algebra
- Boolean function and minimization
- Logic gates
- Logic circuits and Boolean expressions
- Combinational circuits and design
Boolean Algebra

- An algebra that deals with binary number system
- George Boole (1815-1864), an English mathematician, developed it for:
  - Simplifying representation
  - Manipulation of propositional logic
- In 1938, Claude E. Shannon proposed using Boolean algebra in design of relay switching circuits
- Provides economical and straightforward approach
- Used extensively in designing electronic circuits used in computers
Fundamental Concepts of Boolean Algebra

- Use of Binary Digit
  - Boolean equations can have either of two possible values, 0 and 1

- Logical Addition
  - Symbol ‘+’, also known as ‘OR’ operator, used for logical addition. Follows law of binary addition

- Logical Multiplication
  - Symbol ‘.’, also known as ‘AND’ operator, used for logical multiplication. Follows law of binary multiplication

- Complementation
  - Symbol ‘∼’, also known as ‘NOT’ operator, used for complementation. Follows law of binary compliment
Each operator has a precedence level
Higher the operator’s precedence level, earlier it is evaluated
Expression is scanned from left to right
First, expressions enclosed within parentheses are evaluated
Then, all complement (NOT) operations are performed
Then, all ‘⋅’ (AND) operations are performed
Finally, all ‘+’ (OR) operations are performed

(Continued on next slide)
Operator Precedence

(Continued from previous slide..)

\[ X + \overline{Y} \cdot Z \]

1\textsuperscript{st}  2\textsuperscript{nd}  3\textsuperscript{rd}
Postulate 1:
(a) \( A = 0 \), if and only if, \( A \) is not equal to 1
(b) \( A = 1 \), if and only if, \( A \) is not equal to 0

Postulate 2:
(a) \( x + 0 = x \)
(b) \( x \cdot 1 = x \)

Postulate 3: Commutative Law
(a) \( x + y = y + x \)
(b) \( x \cdot y = y \cdot x \)
Postulate 4: Associative Law

(a) \( x + (y + z) = (x + y) + z \)
(b) \( x \cdot (y \cdot z) = (x \cdot y) \cdot z \)

Postulate 5: Distributive Law

(a) \( x \cdot (y + z) = (x \cdot y) + (x \cdot z) \)
(b) \( x + (y \cdot z) = (x + y) \cdot (x + z) \)

Postulate 6:

(a) \( x + \bar{x} = 1 \)
(b) \( x \cdot \bar{x} = 0 \)
The Principle of Duality

There is a precise duality between the operators . (AND) and + (OR), and the digits 0 and 1.

For example, in the table below, the second row is obtained from the first row and vice versa simply by interchanging ‘+’ with ‘.’ and ‘0’ with ‘1’

<table>
<thead>
<tr>
<th></th>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Row 1</strong></td>
<td>1 + 1 = 1</td>
<td>1 + 0 = 0 + 1 = 1</td>
<td>0 + 0 = 0</td>
</tr>
<tr>
<td><strong>Row 2</strong></td>
<td>0 · 0 = 0</td>
<td>0 · 1 = 1 · 0 = 0</td>
<td>1 · 1 = 1</td>
</tr>
</tbody>
</table>

Therefore, if a particular theorem is proved, its dual theorem automatically holds and need not be proved separately.
## Some Important Theorems of Boolean Algebra

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Theorems/Identities</th>
<th>Dual Theorems/Identities</th>
<th>Name (if any)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$x + x = x$</td>
<td>$x \cdot x = x$</td>
<td>Idempotent Law</td>
</tr>
<tr>
<td>2</td>
<td>$x + 1 = 1$</td>
<td>$x \cdot 0 = 0$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$x + x \cdot y = x$</td>
<td>$x \cdot x + y = x$</td>
<td>Absorption Law</td>
</tr>
<tr>
<td>4</td>
<td>$\overline{x} = x$</td>
<td></td>
<td>Involution Law</td>
</tr>
<tr>
<td>5</td>
<td>$x \cdot \overline{x} + y = x \cdot y$</td>
<td>$x + \overline{x} \cdot y = x + y$</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>$\overline{x+y} = \overline{x} \overline{y}$</td>
<td>$x \cdot y = \overline{x} \overline{y} + \overline{y}$</td>
<td>De Morgan’s Law</td>
</tr>
</tbody>
</table>
The theorems of Boolean algebra may be proved by using one of the following methods:

1. By using postulates to show that L.H.S. = R.H.S
2. By *Perfect Induction* or *Exhaustive Enumeration* method where all possible combinations of variables involved in L.H.S. and R.H.S. are checked to yield identical results
3. By the *Principle of Duality* where the dual of an already proved theorem is derived from the proof of its corresponding pair
**Theorem:**

\[ x + x \cdot y = x \]

**Proof:**

L.H.S.

\[
= x + x \cdot y \\
= x \cdot 1 + x \cdot y \quad \text{by postulate 2(b)} \\
= x \cdot (1 + y) \quad \text{by postulate 5(a)} \\
= x \cdot (y + 1) \quad \text{by postulate 3(a)} \\
= x \cdot 1 \quad \text{by theorem 2(a)} \\
= x \quad \text{by postulate 2(b)} \\
= R.H.S.
\]
**Theorem:**

\[ x + x \cdot y = x \]

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x \cdot y</th>
<th>x + x \cdot y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
**Theorem:**

\[ x + x = x \]

**Proof:**

L.H.S.

= \( x + x \)

= \((x + x) \cdot 1 \)  

= \((x + x) \cdot (x + \overline{x}) \)  

= \( x + x \cdot \overline{x} \)  

= \( x + 0 \)  

= \( x \)  

= R.H.S.

(Continued on next slide)
Dual Theorem:

\[ x \cdot x = x \]

Proof:

L.H.S.

\[
\begin{align*}
= x \cdot x \\
= x \cdot x + 0 & \quad \text{by postulate 2(a)} \\
= x \cdot x + x \cdot \bar{x} & \quad \text{by postulate 6(b)} \\
= x \cdot (x + \bar{x}) & \quad \text{by postulate 5(a)} \\
= x \cdot 1 & \quad \text{by postulate 6(a)} \\
= x & \quad \text{by postulate 2(b)} \\
= R.H.S.
\end{align*}
\]

Notice that each step of the proof of the dual theorem is derived from the proof of its corresponding pair in the original theorem.
A Boolean function is an expression formed with:

- Binary variables
- Operators (OR, AND, and NOT)
- Parentheses, and equal sign

The value of a Boolean function can be either 0 or 1

A Boolean function may be represented as:

- An algebraic expression, or
- A truth table
Representation as an Algebraic Expression

\[ W = X + \overline{Y} \cdot Z \]

- Variable \( W \) is a function of \( X, Y, \) and \( Z \), can also be written as \( W = f(X, Y, Z) \).
- The RHS of the equation is called an **expression**.
- The symbols \( X, Y, Z \) are the **literals** of the function.
- For a given Boolean function, there may be more than one algebraic expressions.
## Representation as a Truth Table

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
The number of rows in the table is equal to $2^n$, where $n$ is the number of literals in the function.

The combinations of 0s and 1s for rows of this table are obtained from the binary numbers by counting from 0 to $2^n - 1$. 
Minimization of Boolean Functions

Minimization of Boolean functions deals with
- Reduction in number of literals
- Reduction in number of terms

Minimization is achieved through manipulating expression to obtain equal and simpler expression(s) (having fewer literals and/or terms)

(Continued on next slide)
Minimization of Boolean Functions

(Continued from previous slide..)

\[ F_1 = \overline{x} \cdot \overline{y} \cdot z + \overline{x} \cdot y \cdot z + x \cdot \overline{y} \]

\( F_1 \) has 3 literals (x, y, z) and 3 terms

\[ F_2 = x \cdot \overline{y} + \overline{x} \cdot z \]

\( F_2 \) has 3 literals (x, y, z) and 2 terms

\( F_2 \) can be realized with fewer electronic components, resulting in a cheaper circuit

(Continued on next slide)
Both $F_1$ and $F_2$ produce the same result

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$z$</th>
<th>$F_1$</th>
<th>$F_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Try out some Boolean Function Minimization

(a) \( x + \overline{x} \cdot y \)

(b) \( x \cdot (\overline{x} + y) \)

(c) \( \overline{x} \cdot \overline{y} \cdot z + x \cdot \overline{y} \cdot z + x \cdot \overline{y} \)

(d) \( x \cdot y + \overline{x} \cdot z + y \cdot z \)

(e) \( (x + y) \cdot (\overline{x} + z) \cdot (y + z) \)
The complement of a Boolean function is obtained by interchanging:

- Operators OR and AND
- Complementing each literal

This is based on De Morgan’s theorems, whose general form is:

\[
A_1 + A_2 + A_3 + \ldots + A_n = \overline{A_1 \cdot \overline{A}_2 \cdot \overline{A}_3 \cdot \ldots \cdot \overline{A}_n}
\]

\[
A_1 \cdot A_2 \cdot A_3 \cdot \ldots \cdot A_n = \overline{A}_1 + \overline{A}_2 + \overline{A}_3 + \ldots + \overline{A}_n
\]
Complementing a Boolean Function (Example)

\[ F_1 = \overline{x} \cdot y \cdot \overline{z} + \overline{x} \cdot \overline{y} \cdot z \]

To obtain \( \overline{F}_1 \), we first interchange the OR and the AND operators giving

\[ (\overline{x} + y + \overline{z}) \cdot (\overline{x} + \overline{y} + z) \]

Now we complement each literal giving

\[ \overline{F}_1 = (x + \overline{y} + z) \cdot (x + y + \overline{z}) \]
**Canonical Forms of Boolean Functions**

**Minterms**: \( n \) variables forming an AND term, with each variable being primed or unprimed, provide \( 2^n \) possible combinations called *minterms* or *standard products*.

**Maxterms**: \( n \) variables forming an OR term, with each variable being primed or unprimed, provide \( 2^n \) possible combinations called *maxterms* or *standard sums*.
### Minterms and Maxterms for three Variables

<table>
<thead>
<tr>
<th>Variables</th>
<th>Minterms</th>
<th>Maxterms</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Term</td>
<td>Designation</td>
</tr>
<tr>
<td>x y z</td>
<td>m₀</td>
<td>x + y + z</td>
</tr>
<tr>
<td>0 0 0</td>
<td>x · y · z</td>
<td>m₁</td>
</tr>
<tr>
<td>0 0 1</td>
<td>x · y · z</td>
<td>m₂</td>
</tr>
<tr>
<td>0 1 0</td>
<td>x · y · z</td>
<td>m₃</td>
</tr>
<tr>
<td>0 1 1</td>
<td>x · y · z</td>
<td>m₄</td>
</tr>
<tr>
<td>1 0 0</td>
<td>x · y · z</td>
<td>m₅</td>
</tr>
<tr>
<td>1 0 1</td>
<td>x · y · z</td>
<td>m₆</td>
</tr>
<tr>
<td>1 1 0</td>
<td>x · y · z</td>
<td>m₇</td>
</tr>
</tbody>
</table>

Note that each minterm is the complement of its corresponding maxterm and vice-versa.
A sum-of-products (SOP) expression is a product term (minterm) or several product terms (minterms) logically added (ORed) together. Examples are:

\[
\begin{align*}
  x & \quad x + y \\
  x + y \cdot z & \quad x \cdot y + z \\
  x \cdot \overline{y} + \overline{x} \cdot y & \quad \overline{x} \cdot \overline{y} + x \cdot \overline{y} \cdot z
\end{align*}
\]
Steps to Express a Boolean Function in its Sum-of-Products Form

1. Construct a truth table for the given Boolean function

2. Form a minterm for each combination of the variables, which produces a 1 in the function

3. The desired expression is the sum (OR) of all the minterms obtained in Step 2
Expressing a Function in its Sum-of-Products Form (Example)

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>F₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The following 3 combinations of the variables produce a 1: 001, 100, and 111

(Continued on next slide)
Expressing a Function in its Sum-of-Products Form (Example)

(Continued from previous slide...)

Their corresponding minterms are:

\[ \overline{x} \cdot \overline{y} \cdot z, \quad x \cdot \overline{y} \cdot \overline{z}, \quad \text{and} \quad x \cdot y \cdot z \]

Taking the OR of these minterms, we get

\[ F_1 = \overline{x} \cdot \overline{y} \cdot z + x \cdot \overline{y} \cdot \overline{z} + x \cdot y \cdot z = m_1 + m_4 + m_7 \]

\[ F_1(x \cdot y \cdot z) = \sum(1, 4, 7) \]
A product-of-sums (POS) expression is a sum term (maxterm) or several sum terms (maxterms) logically multiplied (ANDed) together. Examples are:

\[ x \quad (x+y) \cdot (\overline{x}+y) \cdot (\overline{x}+\overline{y}) \]

\[ \overline{x}+y \quad (x+y) \cdot (\overline{x}+y+z) \]

\[ (\overline{x}+\overline{y}) \cdot z \quad (\overline{x}+y) \cdot (x+\overline{y}) \]
Steps to Express a Boolean Function in its Product-of-Sums Form

1. Construct a truth table for the given Boolean function
2. Form a maxterm for each combination of the variables, which produces a 0 in the function
3. The desired expression is the product (AND) of all the maxterms obtained in Step 2
### Expressing a Function in its Product-of-Sums Form

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>$F_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The following 5 combinations of variables produce a 0: 000, 010, 011, 101, and 110.

(Continued on next slide)
Their corresponding maxterms are:

\((x + y + z), (x + \overline{y} + z), (x + \overline{y} + \overline{z}),\)

\((\overline{x} + y + \overline{z})\) and \((\overline{x} + \overline{y} + z)\)

Taking the AND of these maxterms, we get:

\[ F_1 = (x + y + z) \cdot (x + \overline{y} + z) \cdot (x + \overline{y} + \overline{z}) \cdot (\overline{x} + y + \overline{z}) \cdot (\overline{x} + \overline{y} + z) \cdot M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6 \]

\[ F_1(x,y,z) = \prod(0, 2, 3, 5, 6) \]
Conversion Between Canonical Forms (Sum-of-Products and Product-of-Sums)

To convert from one canonical form to another, interchange the symbol and list those numbers missing from the original form.

Example:

\[ F(x,y,z) = \Pi(0,2,4,5) = \Sigma(1,3,6,7) \]
\[ F(x,y,z) = \Pi(1,4,7) = \Sigma(0,2,3,5,6) \]
Logic Gates

- Logic gates are electronic circuits that operate on one or more input signals to produce standard output signal.
- Are the building blocks of all the circuits in a computer.
- Some of the most basic and useful logic gates are AND, OR, NOT, NAND and NOR gates.
AND Gate

- Physical realization of logical multiplication (AND) operation
- Generates an output signal of 1 only if all input signals are also 1
### AND Gate (Block Diagram Symbol and Truth Table)

A diagram of an AND gate is shown with inputs A and B, and output C. The equation for the output is $C = A \cdot B$.

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
OR Gate

- Physical realization of logical addition (OR) operation
- Generates an output signal of 1 if at least one of the input signals is also 1
**OR Gate (Block Diagram Symbol and Truth Table)**

\[ C = A + B \]

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

C = A + B
NOT Gate

- Physical realization of complementation operation
- Generates an output signal, which is the reverse of the input signal
NOT Gate (Block Diagram Symbol and Truth Table)

\[
\begin{array}{c}
A \\
\hline
0 & 1 \\
1 & 0 \\
\end{array}
\]

**Input** | **Output**
--- | ---
A | \(\bar{A}\)
0 | 1
1 | 0
NAND Gate

- Complemented AND gate
- Generates an output signal of:
  - 1 if any one of the inputs is a 0
  - 0 when all the inputs are 1
NAND Gate (Block Diagram Symbol and Truth Table)

\[ C = A \uparrow B = \overline{A} \cdot \overline{B} = A + B \]

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
NOR Gate

- Complemented OR gate
- Generates an output signal of:
  - 1 only when all inputs are 0
  - 0 if any one of inputs is a 1
NOR Gate (Block Diagram Symbol and Truth Table)

\[
C = A \downarrow B = \overline{A + B} = \overline{A} \cdot \overline{B}
\]

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
When logic gates are interconnected to form a gating / logic network, it is known as a *combinational logic circuit*.

The Boolean algebra expression for a given logic circuit can be derived by systematically progressing from input to output on the gates.

The three logic gates (AND, OR, and NOT) are logically complete because any Boolean expression can be realized as a logic circuit using only these three gates.
Finding Boolean Expression of a Logic Circuit (Example 1)

\[ D = \overline{A} \cdot (B + C) \]
Finding Boolean Expression of a Logic Circuit (Example 2)

C = (A + B) \cdot (\overline{A} \cdot \overline{B})
Constructing a Logic Circuit from a Boolean Expression (Example 1)

Boolean Expression = $A \cdot B + C$

Diagram showing the logic circuit with inputs A, B, and C, and outputs $A \cdot B + C$.
Constructing a Logic Circuit from a Boolean Expression (Example 2)

Boolean Expression = \(\overline{A \cdot B} + C \cdot D + \overline{E \cdot F}\)

![Logic Circuit Diagram]

Ref. Page 83
NAND gate is an universal gate, it is alone sufficient to implement any Boolean expression.

To understand this, consider:

- Basic logic gates (AND, OR, and NOT) are logically complete.
- Sufficient to show that AND, OR, and NOT gates can be implemented with NAND gates.
Implementation of NOT, AND and OR Gates by NAND Gates

(a) NOT gate implementation.

\[
\overline{A \cdot A} = \overline{A} + \overline{A} = \overline{A}
\]

(b) AND gate implementation.

\[
\overline{A \cdot B} = A \cdot B
\]
Implementation of NOT, AND and OR Gates by NAND Gates

(c) OR gate implementation.
Method of Implementing a Boolean Expression with Only NAND Gates

Step 1: From the given algebraic expression, draw the logic diagram with AND, OR, and NOT gates. Assume that both the normal \((A)\) and complement \((\overline{A})\) inputs are available.

Step 2: Draw a second logic diagram with the equivalent NAND logic substituted for each AND, OR, and NOT gate.

Step 3: Remove all pairs of cascaded inverters from the diagram as double inversion does not perform any logical function. Also remove inverters connected to single external inputs and complement the corresponding input variable.
Boolean Expression = \( A \cdot \overline{B} + C \cdot (A + B \cdot D) \)

(a) **Step 1:** AND/OR implementation
Implementing a Boolean Expression with Only NAND Gates (Example)

(Continued from previous slide..)

(b) **Step 2:** Substituting equivalent NAND functions

(Continued on next slide)
(c) **Step 3:** NAND implementation.

\[ A \cdot \overline{B} + C \cdot (A + B \cdot D) \]
Universal NOR Gate

- NOR gate is an universal gate, it is alone sufficient to implement any Boolean expression

- To understand this, consider:
  - Basic logic gates (AND, OR, and NOT) are logically complete
  - Sufficient to show that AND, OR, and NOT gates can be implemented with NOR gates
Implementation of NOT, OR, and AND Gates by NOR Gates

(a) NOT gate implementation.

\[ \overline{A + A} = \overline{A} \cdot \overline{A} = \overline{A} \]

(b) OR gate implementation.

\[ \overline{A + B} = A + B \]
Implementation of NOT, OR and AND Gates by NOR Gates

(Continued from previous slide..)

(c) AND gate implementation.
Method of Implementing a Boolean Expression with Only NOR Gates

Step 1: For the given algebraic expression, draw the logic diagram with AND, OR, and NOT gates. Assume that both the normal \((A)\) and complement \((\overline{A})\) inputs are available.

Step 2: Draw a second logic diagram with equivalent NOR logic substituted for each AND, OR, and NOT gate.

Step 3: Remove all parts of cascaded inverters from the diagram as double inversion does not perform any logical function. Also remove inverters connected to single external inputs and complement the corresponding input variable.
Boolean Expression \( A \cdot \overline{B} + C \cdot (A + B \cdot D) \)

(a) **Step 1:** AND/OR implementation.
(b) **Step 2:** Substituting equivalent NOR functions.

(Continued on next slide)
Implementing a Boolean Expression with Only NOR Gates (Examples)

(Continued from previous slide...)

(c) **Step 3:** NOR implementation.

\[
\overline{A} \cdot \overline{B} + C \cdot (A + B \cdot D)
\]
The Exclusive-OR Function is defined as:

\[ A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B \]

And can be illustrated as follows:

\[ C = A \oplus B = \overline{A} \cdot B + A \cdot \overline{B} \]

Also,

\[ (A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C \]

(Continued on next slide)
### Exclusive-OR Function (Truth Table)

(Continued from previous slide..)

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
\[ A \bar{A} \bar{B} = A \cdot B + \bar{A} \cdot \bar{B} \]

Also, 

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

(Continued on next slide)
## Equivalence Function (Truth Table)

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Steps in Designing Combinational Circuits

1. State the given problem completely and exactly
2. Interpret the problem and determine the available input variables and required output variables
3. Assign a letter symbol to each input and output variables
4. Design the truth table that defines the required relations between inputs and outputs
5. Obtain the simplified Boolean function for each output
6. Draw the logic circuit diagram to implement the Boolean function
Designing a Combinational Circuit
Example 1 – Half-Adder Design

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ S = \overline{A} \cdot B + A \cdot \overline{B} \]
\[ C = A \cdot B \]

Boolean functions for the two outputs.
Designing a Combinational Circuit

Example 1 – Half-Adder Design

(Continued from previous slide..)

Logic circuit diagram to implement the Boolean functions

\[ S = \overline{A} \cdot B + A \cdot \overline{B} \]

\[ C = A \cdot B \]
## Designing a Combinational Circuit

### Example 2 – Full-Adder Design

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Truth table for a full adder

*(Continued on next slide)*
Boolean functions for the two outputs:

\[ S = \overline{A} \cdot \overline{B} \cdot D + \overline{A} \cdot B \cdot \overline{D} + A \cdot \overline{B} \cdot \overline{D} + A \cdot B \cdot D \]

\[ C = \overline{A} \cdot B \cdot D + A \cdot \overline{B} \cdot D + A \cdot B \cdot \overline{D} + A \cdot B \cdot D \]

\[ = A \cdot B + A \cdot D + B \cdot D \quad \text{(when simplified)} \]
Designing a Combinational Circuit
Example 2 – Full-Adder Design

(Continued from previous slide..)

(a) Logic circuit diagram for sums

(Continued on next slide)
Designing a Combinational Circuit

Example 2 – Full-Adder Design

(Continued from previous slide..)

(b) Logic circuit diagram for carry
### Key Words/Phrases

- Absorption law
- AND gate
- Associative law
- Boolean algebra
- Boolean expression
- Boolean functions
- Boolean identities
- Canonical forms for Boolean functions
- Combination logic circuits
- Cumulative law
- Complement of a function
- Complementation
- De Morgan’s law
- Distributive law
- Dual identities
- Equivalence function
- Exclusive-OR function
- Exhaustive enumeration method
- Half-adder
- Idempotent law
- Involution law
- Literal
- Logic circuits
- Logic gates
- Logical addition
- Logical multiplication
- Maxterms
- Minimization of Boolean functions
- Minterms
- NAND gate
- NOT gate
- Operator precedence
- OR gate
- Parallel Binary Adder
- Perfect induction method
- Postulates of Boolean algebra
- Principle of duality
- Product-of-Sums expression
- Standard forms
- Sum-of Products expression
- Truth table
- Universal NAND gate
- Universal NOR gate