# Intro to Digital Logic

## Boolean Algebra & Karnaugh Maps

By Apowware

Ref. Digital Design Principles & Practices (4th Ed.) John F. Wakerly

# Introduction to Boolean Algebra

## What is Boolean Algebra?

- Named after George Boole.
- Branch of algebra where variables are true (1) or false (0).
- Mathematical foundation for digital circuits and modern computing.

## Why is it Important?

- **Digital Circuit Design:** Essential for building and understanding logic gates.
- **Simplification:** Reduces complex expressions for efficient circuits.
- **Problem Solving:** Systematic approach to logical relationships.

# Basic Concepts - Variables & Literals

**Boolean Variables**

- Can only be one of two values:
  - **0 (False)**
  - **1 (True)**
- Also called **binary values** or **logic levels**.

**Literals**

- A variable or its complement.
- Examples: A, A' (or $\bar{A}$)

# Boolean Operators - AND

## AND Operator (Logical Conjunction)

- **Symbol:** . or implied (e.g., A.B or AB).
- **Rule:** Output is 1 (True) *only if all* inputs are 1 (True). Otherwise, 0 (False).
- **Example:** Y = A AND B or Y = A.B

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

# Boolean Operators - OR

## OR Operator (Logical Disjunction)

- **Symbol:** +
- **Rule:** Output is 1 (True) if *any* input is 1 (True). Output is 0 (False) *only if all* inputs are 0 (False).
- **Example:** Y = A OR B or Y = A + B

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

# Boolean Operations - NOT

**NOT Operator (Logical Negation/Inversion)**

- **Symbol:** ' or overbar (e.g., A' or $\bar{A}$).
- **Rule:** Inverts the input (1 becomes 0, 0 becomes 1).
- **Example:** Y = NOT A or Y = A' or Y=$\bar{A}$

| A | Y |
|---|---|
| 0 | 1 |
| 1 | 0 |

# Boolean Algebra - Commutative & Associative

**Boolean Algebra and Theorems**

- Fundamental rules for manipulating Boolean expressions.

**Commutative Laws**

- Order doesn't matter.
  - $A+B=B+A$
  - $A \cdot B=B \cdot A$

**Associative Laws**

- Grouping doesn't matter for the same operator.
  - $(A+B)+C=A+(B+C)$
  - $(A \cdot B) \cdot C=A \cdot (B \cdot C)$

# Boolean Algebra - Distributive, Identity, & Complement

**Distributive Laws**

- $A \cdot (B+C) = A \cdot B + A \cdot C$
- $A + (B \cdot C) = (A+B) \cdot (A+C)$ (Unique to Boolean Algebra!)

**Identity Laws**

- $A + 0 = A$ (Adding False doesn't change value)
- $A \cdot 1 = A$ (ANDing with True doesn't change value)

**Complement Laws**

- $A + A' = 1$ (A OR NOT A is always True)
- $A \cdot A' = 0$ (A AND NOT A is always False)

# Important Theorems - Idempotent, Null, & Absorption

**Idempotent Laws**

- $A+A=A$
- $A \cdot A=A$

**Null Laws (or Annulment Laws)**

- $A+1=1$ (ORing with True is always True)
- $A \cdot 0=0$ (ANDing with False is always False)

**Absorption Laws**

- $A+(A \cdot B)=A$
- $A \cdot (A+B)=A$

# De Morgan’s Theorems

**De Morgan's Theorems**

- Powerful for simplifying expressions with complements.
- Convert between sum-of-products and product-of-sums forms.

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

- **Explanation:** Complement of a sum (OR) is the product (AND) of individual complements.
- **Intuitive Example:** "NOT (Apple OR Banana)" is equivalent to "NOT Apple AND NOT Banana".

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

- **Explanation:** Complement of a product (AND) is the sum (OR) of individual complements.
- **Intuitive Example:** "NOT (Study AND Pass)" is equivalent to "NOT Study OR NOT Pass".

**Double Negation (Involution) Law**

- $(A')'=A$

# Summary of Boolean Algebra Theorems

| Law/Theorem | OR Form (Sum) | AND Form (Product) |
|---|---|---|
| Commutative | $A+B=B+A$ | $A \cdot B=B \cdot A$ |
| Associative | $(A+B)+C=A+(B+C)$ | $(A \cdot B) \cdot C=A \cdot (B \cdot C)$ |
| Distributive | $A+(B \cdot C)=(A+B) \cdot (A+C)$ | $A \cdot (B+C)=A \cdot B+A \cdot C$ |
| Identity | $A+0=A$ | $A \cdot 1=A$ |
| Complement | $A+A'=1$ | $A \cdot A'=0$ |
| Idempotent | $A+A=A$ | $A \cdot A=A$ |
| Null (Annulment) | $A+1=1$ | $A \cdot 0=0$ |
| Absorption | $A+(A \cdot B)=A$ | $A \cdot (A+B)=A$ |
| De Morgan's | $(A+B)'=A' \cdot B'$ | $(A \cdot B)'=A'+B'$ |
| Double Negation | $(A')'=A$ | - |

# Boolean Simplification Examples

**Example 1: Simplify $Y=A \cdot B+A \cdot B \cdot C+A \cdot B \cdot C'$**

1. **Factor out common terms:** $Y=A \cdot B \cdot (1+C+C')$
2. **Apply Identity Law ($1+X=1$):** $Y=A \cdot B \cdot (1)$
3. **Apply Identity Law ($X \cdot 1=X$):** $Y=A \cdot B$

**Example 2: Simplify $F=A+A' \cdot B$**

1. **Apply Distributive Law ($A+(B \cdot C)=(A+B) \cdot (A+C)$):** $F=(A+A') \cdot (A+B)$
2. **Apply Complement Law ($A+A'=1$):** $F=1 \cdot (A+B)$
3. **Apply Identity Law ($1 \cdot X=X$):** $F=A+B$

# More Examples

**Example 3: Simplify $G=(A+B)\cdot(A+B')$**

1. **Apply Distributive Law (**$A\cdot(B+C)=A\cdot B+A\cdot C$ **- in reverse):** $G=A+(B\cdot B')$
2. **Apply Complement Law (**$B\cdot B'=0$**):** $G=A+0$
3. **Apply Identity Law (**$A+0=A$**):** $G=A$

**Example 4: Simplify $H=A\cdot B+A\cdot B'+A'\cdot B$**

1. **Factor out A from the first two terms:** $H=A\cdot(B+B')+A'\cdot B$
2. **Apply Complement Law (**$B+B'=1$**):** $H=A\cdot 1+A'\cdot B$
3. **Apply Identity Law (**$A\cdot 1=A$**):** $H=A+A'\cdot B$
4. **Apply Distributive Law (as in Example 2):** $H=(A+A')\cdot(A+B)$
5. **Apply Complement Law (**$A+A'=1$**):** $H=1\cdot(A+B)$
6. **Apply Identity Law (**$1\cdot X=X$**):** $H=A+B$

# Karnaugh Maps

Karnaugh Maps, often abbreviated as K-Maps, are a graphical method used to simplify Boolean algebra expressions. Developed by Maurice Karnaugh in 1953, they provide a systematic and visual way to reduce complex logic circuits to their simplest form, making them easier and more cost-effective to implement.

While Boolean algebra laws and theorems can be used to simplify expressions, they can be cumbersome and prone to errors, especially for expressions with many variables. K-Maps offer several advantages:

- **Visual Aid:** They provide a clear visual representation of the Boolean function, making it easier to identify adjacent terms that can be combined.
- **Systematic Approach:** K-Maps offer a step-by-step method, reducing the chances of missing simplification opportunities.
- **Guaranteed Minimization (for small number of variables):** For up to 4 or 5 variables, K-Maps generally lead to the minimal sum-of-products (SOP) or product-of-sums (POS) expression.
- **Handling "Don't Care" Conditions:** K-Maps easily incorporate "don't care" conditions, which can further simplify expressions.

# Creating a K-Map

- Boolean function is plotted on a grid (the K-Map).
- Adjacent cells in the map differ by only one variable.
- Groups of '1's (minterms) are formed to identify common terms.
- The largest possible groups of '1's (in powers of 2: 1, 2, 4, 8, etc.) are circled.
- Each group represents a simplified product term.

# 3 Variable K-Map

F = (A, B, C) = Σ(0, 2, 4, 5, 6) (This means the function is 1 for minterms 0, 2, 4, 5, 6)

| A \ BC | 00 | 01 | 11 | 10 | |
|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | ← m0 & m2 |
| 1 | 1 | 1 | 0 | 1 | ← m4, m5, & m6 |

**Grouping:**

- Group 1: Cells m0, m2, m4, m6 (a group of 4 '1's). This group covers C′. (Notice B changes, A changes, but C is always 0).
- Group 2: Cells m4, m5 (a group of 2 '1's). This group covers AB′. (Notice C changes, but A=1 and B=0).

**Simplified Expression:** F=C′+AB′

| A | B | C | F |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

# Another K-Map Example

F = (A, B, C) = Σ(0, 2, 4, 6)

| BC<br>A \ | 00 | 01 | 11 | 10 | |
|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | ← m0 & m2 |
| 1 | 1 | 0 | 0 | 1 | ← m4 & m6 |

**Grouping:**

- Group 1: m0, m2, m4, m6 (a group of 4 '1's). This group covers C′.

**Simplified Expression:** F=C′

| A | B | C | F |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |