## 2 Boolean Algebra and Logic Simplification

## Introduction to Boolean Algebra

Boolean algebra is a mathematical framework used for analyzing and simplifying digital logic circuits. It consists of a set of rules and operations, such as AND, OR, and NOT, to represent logic gates and their interactions. In this section, we will cover the foundational rules and properties of Boolean algebra.

### Basic Boolean Operations and Notation

In Boolean algebra, we commonly use the following operations:

- AND operation $(\cdot)$: Often written as $A \cdot B$ or simply $AB$.
- OR operation $(+)$: Written as $A + B$.
- NOT operation $(\overline{A})$: Denotes the complement or negation of $A$.

Alternative notations include:

$$\begin{aligned} A \cdot B &\equiv A \wedge B \quad \text{(AND)} \\ A + B &\equiv A \vee B \quad \text{(OR)} \\ \overline{A} &\equiv \neg A \quad \text{(NOT)} \end{aligned}$$

### Boolean Algebra Rules

Boolean algebra includes fundamental rules for simplifying logic expressions. Key rules include:

- Identity: $A + 0 = A$, $A \cdot 1 = A$
- Null: $A + 1 = 1$, $A \cdot 0 = 0$
- Idempotent: $A + A = A$, $A \cdot A = A$
- Complement: $A + \overline{A} = 1$, $A \cdot \overline{A} = 0$
- Double Negation: $\overline{\overline{A}} = A$
- Absorption: $A + (A \cdot B) = A$, $A \cdot (A + B) = A$
- Distributive: $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$
- De Morgan's Theorems:

$$\overline{A + B} = \overline{A} \cdot \overline{B}, \quad \overline{A \cdot B} = \overline{A} + \overline{B}$$

## Boolean Simplification Techniques

The goal of Boolean simplification is to reduce the complexity of logic circuits, which in turn minimizes cost, power consumption, and space. There are two primary techniques: Boolean algebra simplification and Karnaugh maps.

### Example: Simplifying a Fault-Tolerant Voting System

Consider a system that outputs a signal $Y$ if at least two out of three sensors agree (e.g., $A$, $B$, and $C$). The initial expression for such a system can be written as:

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

Using Boolean simplification, we find:

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

## 3 Karnaugh Maps (K-Maps)

Karnaugh maps (K-maps) are a visual method for simplifying Boolean expressions by grouping terms to minimize the expression. K-maps are particularly useful for expressions with 2-4 variables.

### Constructing a Karnaugh Map

1. Draw a grid where each cell represents a possible variable combination. 2. Fill in the cells based on the output values from the truth table. 3. Group 1s in powers of 2 (e.g., 1, 2, 4). 4. Each group corresponds to a simplified term in the Boolean expression.

### K-Map Simplification Example 1: Two-Variable Expression

Suppose we have the Boolean function:

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

Using a K-map:

| A | B | F(A, B) |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

The K-map layout:

| | |
|---|---|
| $F(0,0)$ | $F(0,1)$ |
| $F(1,0)$ | $F(1,1)$ |

After grouping, we find that $F = A + B$.

### K-Map Simplification Example 2: Three-Variable Expression

Given:

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

Construct a K-map:

| $AB\backslash C$ | 0 | 1 |
|---|---|---|
| 00 | 0 | 0 |
| 01 | 1 | 1 |
| 10 | 0 | 1 |
| 11 | 1 | 1 |

Grouping and simplifying yields the final expression $F = B + A \cdot C$.

## Example Problems and Practice

### Practice Problem 1: Simplify Using K-Map

Given the Boolean expression $F(A,B,C,D) = (A \cdot B \cdot C) + (A \cdot B \cdot \overline{C}) + (A \cdot \overline{B} \cdot D)$, simplify the function using a K-map.

### Practice Problem 2: Design a Simple Circuit

Design a logic circuit that outputs 1 only when exactly two out of three inputs are 1. Use the K-map method to find the minimal expression.

## Implementing Simplified Boolean Expressions

Simplified Boolean expressions can be implemented directly using logic gates. For example, the expression $F = A + B \cdot C$ would translate into an OR gate with one input as $A$ and the other input as the output of an AND gate with inputs $B$ and $C$.

## Conclusion

In this lecture, we explored Boolean algebra, simplified expressions, and applied Karnaugh maps.

# Appendix: Detailed Solution for the 2-bit Multiplier Circuit

## Introduction to the 2-bit Multiplier Problem

A 2-bit multiplier circuit takes two 2-bit binary inputs, $A = (A_1, A_0)$ and $B = (B_1, B_0)$, and produces a 4-bit output, $P = (P_3, P_2, P_1, P_0)$, where each bit of $P$ represents a Boolean function of the inputs.

This appendix details the process of generating and simplifying the Boolean expressions for each output bit $P_0$, $P_1$, $P_2$, and $P_3$ using truth tables and Karnaugh maps.

## Step 1: Truth Table for the Multiplier

The truth table below shows all possible combinations of the inputs $A_1$, $A_0$, $B_1$, and $B_0$, along with their respective output bits $P_3$, $P_2$, $P_1$, and $P_0$.

| $A_1$ | $A_0$ | $B_1$ | $B_0$ | $P_3$ | $P_2$ | $P_1$ | $P_0$ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Each column in the output corresponds to a bit in the product $P$, which we will simplify using Karnaugh maps.

## Step 2: Simplification of Output Bits Using Karnaugh Maps

Output Bit $P_0$

The least significant bit $P_0$ is straightforward since it is simply the result of the AND operation between $A_0$ and $B_0$:

$$P_0 = A_0 \cdot B_0$$

No further simplification is required for $P_0$.

Output Bit $P_1$

The second output bit, $P_1$, involves a combination of terms:

$$P_1 = (A_0 \cdot B_1) \oplus (A_1 \cdot B_0)$$

Creating the Karnaugh map for $P_1$:

| $A_1A_0 \backslash B_1B_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | 0 | 0 |
| 01 | 0 | 0 | 1 | 0 |
| 11 | 0 | 1 | 0 | 1 |
| 10 | 0 | 1 | 0 | 0 |

From the map, we identify and group terms to simplify:

$$P_1 = A_0 \cdot B_1 + A_1 \cdot B_0$$

Output Bit $P_2$

The third bit, $P_2$, requires analyzing additional combinations:

$$P_2 = (A_1 \cdot B_1) \vee (A_0 \cdot B_1 \cdot B_0) \vee (A_1 \cdot A_0 \cdot B_0)$$

Karnaugh map for $P_2$:

| $A_1A_0 \backslash B_1B_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | 0 | 0 |
| 01 | 0 | 0 | 0 | 1 |
| 11 | 1 | 0 | 1 | 0 |
| 10 | 0 | 1 | 0 | 1 |

After grouping and simplifying:

$$P_2 = A_1 \cdot B_1 + A_0 \cdot B_1$$

Output Bit $P_3$

The most significant bit $P_3$ only depends on the highest order inputs:

$$P_3 = A_1 \cdot B_1$$

## Summary of Simplified Output Expressions

After constructing and simplifying the K-maps for each bit, we have the final Boolean expressions:

$$P_0 = A_0 \cdot B_0$$
$$P_1 = A_0 \cdot B_1 + A_1 \cdot B_0$$
$$P_2 = A_1 \cdot B_1 + A_0 \cdot B_1$$
$$P_3 = A_1 \cdot B_1$$

These expressions represent the minimal logic needed to implement a 2-bit multiplier circuit.

## Further Steps and Implementation

These simplified expressions can be implemented with basic logic gates, enabling efficient hardware realization of the 2-bit multiplier circuit. In subsequent lectures, students will learn how to convert these expressions into practical digital circuits and simulate their behavior.