UNIT 3

Digital Systems and Logic Design

Student Learning Outcomes

By the end of this chapter, you will be able to:

- Understand Boolean functions and operations it, such as Boolean AND, and OR.
- Construct Boolean expressions using variables and Boolean operators.
- Relate common Boolean identities and Boolean simplification procedures.
- Understand the concept of duality in Boolean algebra.
- Subtopics such analog and digital signals
- Introduce several types of gates and their functions.
- Build truth tables for the operations of logical expressions.
- Employ the K-Maps in minimizing Boolean expressions.
- Introduce logic diagrams of digital system.
- Analyze and design half-adder and full-adder circuits.

Introduction

In this chapter, we will discuss the Boolean functions, logic, digital logic, and difference between analog and digital signals. We will also discuss several types of gates, their truth tables, and digital devices including half and full adders. At the completion of this chapter, you should be construct Boolean expressions, simplify them, create truth table, and understand the basics of digital logic.

3.1 Basics of Digital Systems

Digital systems are the backbone of today's electronics and computing. They manipulate digital information in the form of binary digits, which are either 0 or 1 and are used in calculation devices such as calculators and computers, among others.

3.1.1 What is an Analog Signal

Analog signals are signals that changes with time smoothly and continuously over time. They can have any value within given range. Examples include voice signal (speaking), body's temperature and radio-wave signals. Digital signals are the signals which have only two values that are in the form of '0' and '1'. These are utilized in digital electronics and computing systems. Analog to digital converter (ADC) and digital to analog converters (DAC) are important operations in today's technological developments, enabling the transmission and control of signals.

Analog Signal	Digital Signal
Continuous Infinite possible values Example: Sound waves	Discrete Finite (0 or 1) Example: Binary data in computers

Analog to Digital Conversion (ADC): ADC is the conversion of analog signals into digital signals, which are discrete and can be easily processed by computerized devices like computers and smart phones.

Digital to Analog Conversion (DAC): DAC is the conversion whereby analog signals are converted to digital signals, making it possible for human to perceive the information, for instance through speakers, as depicted in figure 3. 1.

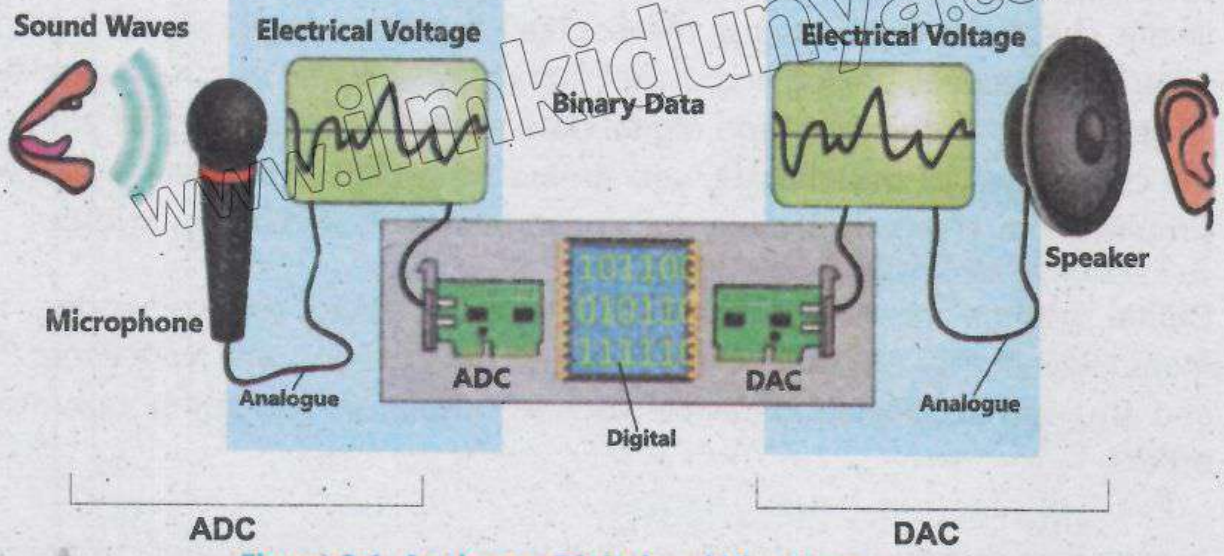


Figure 3.1: Analog to Digital and Vice Versa

ADC and DAC Conversion: Why is it needed?

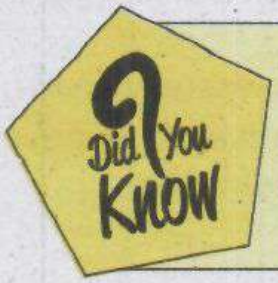
Digital to analog conversion, and vice versa, is critical since it enables data processing, storage, and transmission. Digital signals are much less affected by noise and signal degradation and are therefore better suited for transmitting and storing information over long distances.

Example: Sound Waves

Let us consider a situation where one person is speaking into a microphone while the other person is receiving sound through speakers as illustrated in the figure 3.1.

1. Microphone (ADC): When you speak into the microphone, your voice produces sound waves (analog signals) that are captured by the system. This is done by converting the sound waves into digital form using an ADC with the microphone. Finally, this digital data can be transmitted over long distances with little or no degradation in quality.

2. Speakers (DAC): At the receiver end, the digital signals are then converted back into analog signals with the help of DAC. The speakers then translate these analog signals back into sound waves to enable you hear to the other person's voice as if they were speaking directly to you.



Analog signals are sometimes changed to digital signals in an action known as Analog to Digital Conversion or ADC. This enables analog information such as music, to be recorded and manipulated by digital gadgets.

3.1.2 Fundamentals of Digital Logic

Digital logic is the basis of digital systems. It involves the use of binary numbers that is 0 and 1, to represent and manipulate information. Digital logic circuits use of these binary values to perform various operations, and they are essential to the functioning in operation of computers and many other electronic devices.

In digital circuits, the two states, 0 and 1, are represented by different voltage levels. Conventionally, a higher voltage, such as, 5 volts refer to a binary '1', while a low voltage, for instance, 0 volts refer to a binary '0'. These voltage levels are termed as the logic levels. Logic levels are needed to switch on and switch off the devices and to define ways through which digital circuits execute operations and process information.

3.2 Boolean Algebra and Logic Gates

Boolean algebra is a branch of mathematics relate to logic and symbolic computation, using two values namely True and False. It is an essential branch of digital circuits since it is the basis for the analysis and design of circuits. Here in this section we will cover of Boolean functions and expressions, the working, and functions of logic gates, Building and evaluating Truth Tables and Logic Diagrams.

3.2.1 Boolean Functions and Expressions

Binary values are used to describe the relationship between variables in the Boolean function and Boolean expressions. The expressions are built using AND, OR, and other logic operations and can in several ways be reduced to optimize digital circuits.

3.2.1.1 Binary Variables and Logic Operations

Binary variables that can have only have two values, 0 and 1. Logic operations are basic operations implemented in Boolean algebra for processing of these binary variables. The primary logic operations are AND, OR and NOT.

AND Operation:

AND is the basic logical operator which is used in Boolean algebra. It requires two binary inputs which will give a single binary output. The symbol '.' is used for the AND operation. The output of the AND operation is "1" only when both inputs are "1". Otherwise, the result is "0".

Example:

Consider two binary variables:

$$A = 1 (True)$$
$$B = 0 (False)$$

The AND operation for these variables can be written mathematically as:

$$P = A \cdot B$$

In this example:

$$A=1 \quad B=0$$

Therefore, then, the result P of the AND operation is 0 (false).

Truth Table:

A truth table is useful in demonstrating the functionality of the AND operation with all possibilities of the input variables. Below is the truth table for the AND operation.

A	B	A AND B (P)
0	0	0
0	1	0
1	0	0
1	1	1

Table 3.1: Truth Table for AND Operation

Explanation:

If both A and B are off, that is equal to zero then the desired output P is off (0).

if A is 0 and B is 1 the output P is 0.
When A is 1 and B is 0 P is resulting 0.
When A is 1 and B is 1, the output P also becomes 1.

OR Operation:

The OR operation is an other basic logical operator in Boolean algebra. To be specific this is also a function tables two binary variables as input produces a single binary output. According to Table 3.2, the OR operation yields true (1) output when at least of '1' of the inputs is true (1). The output is 0 only when both inputs are '0'.

Example:

Consider two binary variables:

A = 1 (true)
B = 0 (false)

The OR operation for these variables can be written mathematically as:

P = A+B

In this example:

A = 1 B= 0

Therefore, result P of the OR gate will be 1.

Truth Table:

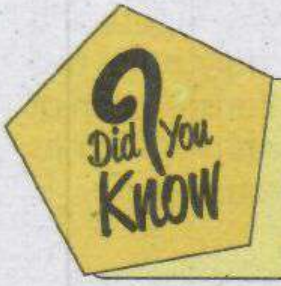
A truth table is useful for better understanding of how the OR operation is organized and what the result of the OR's application is for all variants of the input variables. Below is the truth table for the OR operation.

Explanation:

If A is equal to 0 and B is equal to 0 the output P is equal to 0. When A is zero and B is one, the output P is also one. When A is equal to 1 and B is equals to 0 the values of P equal to 1. When both A and B are 1 then the output P equal to one.

A	B	A OR B (P)
0	0	0
0	1	1
1	0	1
1	1	1

Table 3.2: Truth Table for OR Operation



In binary logic, 1 + 1 does not equal 2 but equals 1 in logical operation. This is because the OR operation returns a value of 1 if any or both of the inputs to this operator are 1.

NOT Operation:

The NOT operation is one of the basic Boolean algebra operations which takes a single binary variable and simply negates its value. If the input is one, the output is zero and if the input is zero, the output is one.

Example:

Consider a binary variable:

$$A = 1 \text{ (true)}$$

The NOT operation for this variable can be written mathematically as:

$$P = \overline{A} \quad \text{or} \quad P = \neg A$$

In this example:

$$P = 0$$

This signifies that if you have A = 1 (true), the result of NOT operation is going to be 0 (false).

Truth Table:

The following table will illustrate the working of NOT operation for all possible inputs of the variable. Below is the truth table for the NOT operation.

A	NOT A (P)
0	1
1	0

Table 3.3: Truth Table for NOT Operation

Explanation:

When the input A is 0, the output P is 1. When A is 1 the output value P is 0. A NOT operation performs the negative of the input variable i. e., it gives the opposite value. This operation is important in digital logic design to generate more complex logic functions and verify the functionality of digital circuits.

3.2.1.2 Construction of Boolean Functions

Boolean functions are algebraic statements that describe the relationship between binary variables and logical operations. These functions are particularly important for digital logic design and are employed in formation of various digital circuits, which are the basis of current computers, mobile phones and even simple calculator.

Understanding Boolean Functions:

A Boolean function is a function which has a one or more binary inputs and produces a single binary output. The inputs and outputs can only have two values: False (represented by 0) and True (represented by 1). The construction of Boolean functions is done by employing the basic logical operations such as AND, OR and NOT, which connect the inputs to generate the correct output.

Example 1: Simple Boolean Function

Consider a Boolean function with two inputs, A and B. We can construct a function F that represents the AND operation:

$$F(A, B) = A \cdot B$$

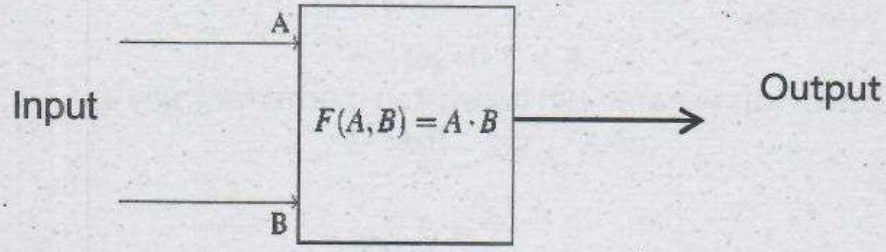


Figure 3.2: Simple Boolean Function

The diagram shown above demonstrates a basic digital circuit, which is an AND gate. The box symbolizes the AND function F (A, B) = A . B. This box has two inputs A and B. If both A and B are 1, the output will be 1. In any other case, the output will be 0. The input are shown at the entrance to the box, while the output is depicted at the exit of the block. The truth table for this function is as follows:

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

Table 3.4: Truth Table for $F(A,B) = A.B$

Example 2: Now, let us construct a more complex Boolean function with three inputs, A, B, and C:

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

This function uses AND, OR and NOT at the same time. The truth table for this function is as follows:

Explanation:

- The parameters A, B, and C are included in the following example as the input columns.
- The results of AND operation between two variable A and B are presented in the column $A \cdot B$.
- The column $\overline{A}$ standing for the NOT operation of A.
- Every value in the column $\overline{A} \cdot C$ displays the result of AND operation between the values in the Fifth column and the third column.
- The final column $F(A, B, C)$ shows the output of the Boolean function $(A \cdot B) + (\overline{A} \cdot C)$.

A	B	C	A·B	$\overline{A}$	$\overline{A}$· C	F(A, B, C)
0	0	0	0	1	0	0
0	0	1	0	1	1	1
0	1	0	0	1	0	0
0	1	1	0	1	1	1
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	1	0	0	1
1	1	1	1	0	0	1

Table 3.5: Detailed Truth Table for $F(A, B, C) = (A \cdot B) + (\overline{A} \cdot C)$

Usage in Computers:

There are many uses of Boolean functions in the computers for various operations. Here are some examples of their usage:

- **Arithmetic Operations:** Boolean functions are used in Arithmetical Logic Units (ALUs) of CPUs to perform operations like addition, subtraction, multiplication, and even division.

- **Data Processing:** Boolean functions are used to process binary data in memory and storage devices, ensuring efficient data manipulation and retrieval.
- **Control Logic:** Boolean functions are applied in computers to control various parts of the system's operation to function in co-ordinated manner.

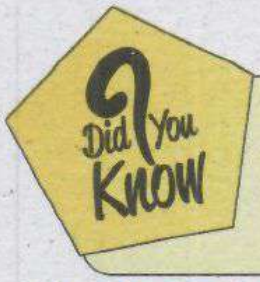
Boolean functions are also present in our everyday devices like cell phones and calculators:

Cell Phones: In cell phone processing, when you dail a number, or press a button on a phone, a Boolean function evaluates these inputs as true or false and makes the necessary output.

Calculators: Basic calculators use Boolean functions. When you feed it with numbers and the operations to be performed, Boolean logic is used to arrive at the right result.

Class activity

Consider what do you do with your cell phone or calculator on daily basis. Can you distinguish activities that require logical choices, like entering a password to unlock your smart phone or solving a math problem? Ask your group members how Boolean functions may be utilized in the background.



George Boole, a mathematician who invented Boolean algebra was born in Lincoln, England in the year 1815. His work laid the debate and the basis for future digital revolution and computer science as well as subsequent technologies of the future.

3.2.2 Logic Gates and their Functions

Logic gates are physical devices in electronic circuits that perform Boolean operations. Each type of logic gate corresponds to a basic Boolean operation. Examples of the logic gates are:

AND Gate: Implements the AND function. It outputs true only when both inputs are True (1).

Figure 3.3: AND Gate

Imagine a simple electronic circuit with an AND gate. If you press two switches (both must be ON), a light bulb will turn on.

- Switch 1: ON (True)
- Switch 2: ON (True)
- Light bulb: ON (True) because both switches are ON.

If either switch is OFF, the light bulb will be OFF.

OR Gate: Implements the OR function. It outputs true when at least one input is true.

Figure 3.4: OR Gate

NOT Gate: Implements the NOT function. It outputs the opposite of the input. See Figure 3.4

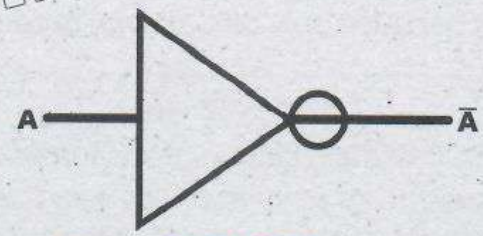


Figure 3.5: NOT Gate

NAND Gate: This gate is achieved when an AND gate is combined with a NOT gate. It generates true when at least one of the inputs is false. In other words, it is the inverse of the AND gate, as presented in Figure 3.6.

Figure 3.6: NAND Gate

Example:

Imagine a safety system where an alarm should go on if either one of two sensors detects an issue.

- Sensor 1: No issue (False)
- Sensor 2: Issue detected (True)
- Alarm: ON (True) because one sensor detects an issue.

XOR Gate:

The XOR (Exclusive OR) gate outputs true only when exactly one of the inputs is true. It differs from the OR gate in that it does not output true when both inputs are true. It is shown in Figure 3.7.

Figure 3.7: XOR Gate

Example:

Imagine a scenario where you can either play video games or do homework, but not both at the same time.

- Play video games: Yes (True)
- Do homework: No (False)
- Allowed? Yes (True) because only one activity is being done.

Class activities

Let's make learning these logical functions fun with an activity!

1. **AND Adventure**: Form pairs and give each pair two conditions they need to meet to win a prize (like both wearing a specific color shirt).
2. **OR Options**: Make a list of fun activities. If at least one activity is possible, the class gets extra playtime.
3. **NOT Negatives:** Ask true/false questions and have students shout the opposite answer. For example, "Is the sky green?" Students should shout "No!" (NOT True).
4. Construct a basic circuit using a breadboard, a battery, and LED lights to represent an AND gate. Connect two switches which will serve as, inputs A and B. In this experiment the LED will light up only when both switches are pressed.

3.3 Simplification of Boolean Functions

Simplification of Boolean functions is a particularly important process in designing an efficient digital circuit. Such simplified functions require fewer gates making them compact in size, energy efficient and faster than the complicated ones. Simplification means applying of some Boolean algebra rules to make the functions less complicated.

Basic Boolean Algebra Rules:

Here are some fundamental Boolean algebra rules used for simplification:

1. **Identity Laws**

$$A + 0 = A$$
$$A \cdot 1 = A$$

2. **Null Laws**

$$A + 1 = 1$$
$$A \cdot 0 = 0$$

3. **Idempotent Laws**

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

4. **Complement Laws**

$$A + \overline{A} = 1$$
$$A \cdot \overline{A} = 0$$

5. **Commutative Laws**

$$A + B = B + A$$
$$A \cdot B = B \cdot A$$

6. **Associative Laws**

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

7. **Distributive Laws**

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

8. **Absorption Laws**

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

9. **De Morgan's Theorems**

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

10. **Double Negation Law**

$$\overline{\overline{A}} = A$$

Simplification Examples

Example 1

Simplify the expression $A + \overline{A} \cdot B$.

Solution:

$A + \overline{A} \cdot B = (A + \overline{A}) \cdot (A + B)$ (Distributive Law)

$= 1 \cdot (A + B)$ (Complement Law)

$= A + B$ (Identity Law)

Example 2

Simplify the expression $\overline{A \cdot B} + \overline{A} \cdot B$.

Solution:

$\overline{A \cdot B} + \overline{A} \cdot B = \overline{A} + \overline{B} + \overline{A} \cdot B$ (De Morgan's Theorem)

$= (\overline{A} + \overline{B})$ Since $\overline{A}$ is already present in $(\overline{A} \cdot \overline{B})$, we can use absorption law i.e $\overline{A} + (\overline{A} \cdot B) = \overline{A}$

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

Example 3

Simplify the expression $(A + B) \cdot (A + \overline{B})$

Solution:

$(A + B) \cdot (A + \overline{B}) = A \cdot (A + \overline{B}) + B \cdot (A + \overline{B})$ (Distributive Law)

$= A + A \cdot \overline{B} + B \cdot \overline{B}$ (Absorption Law)

$= A + A \cdot B$ (Identity Law)

$= A \cdot (1 + B)$ (Distributive Law)

$= A \cdot 1$ (Null Law)

$= A$ (Identity Law)

Example 4

Simplify the expression $\overline{\overline{A} + B \cdot (A + \overline{B})}$

Solution:

$\overline{\overline{A} + B \cdot (A + \overline{B})} = (A \cdot \overline{B}) \cdot (A + \overline{B})$ (De Morgan's Theorem)

$= A \cdot \overline{B} \cdot A + A \cdot \overline{B} \cdot \overline{B}$ (Distributive Law)

$= A \cdot \overline{B} + A \cdot \overline{B}$ (Idempotent Law)

$= A \cdot \overline{B}$ (Identity Law)

3.4. Creating Logic Diagrams

The logic diagrams depict the working of a digital circuit through symbols that represent to its individual logic gates. To create a logic diagram:

- Find out the logic gates needed for the Boolean function.
- Arrange the gates to perform the operations as defined by the function of the
- circuit.
- Connect the inputs and the output of the gates correctly.

To summarize, knowledge of Boolean algebra and logic gates is crucial when it comes to the creation and study of digital circuits. If students understand those concepts, they can build efficient and effective digital systems.

3.5. Application of Digital Logic

Digital logic is an essential aspect for the functioning of several modern electronic systems, such as computers, smart phones, and other digital gadgets. Digital logic optimize in many ways in order to create and enhance circuits meant to perform various tasks. Two important applications of digital logic are the design of adder circuits and the use of Karnaugh maps for function simplification.

3.5.1 Half-adder and Full-adder Circuits

Adder circuits are widely used in the digital circuits to perform arithmetic calculations. There are two general forms of adder circuits known as half-adders and full adders.

3.5.1.1 Half-adder Circuits

A half adder is a basic circuitry unit that performs addition of two single-bit binary digits. It has two inputs, usually denoted as A and B, and two outputs: the sum (S) and the carry (C).

Truth Table for Half-adder:

A	B	Sum (S)	Carry (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Table 3.6: Truth Table for Half-adder

Boolean Expressions for Half-adder:

$$S = A \oplus B$$

$$C = A \cdot B$$

In this case the symbol ⊕ represents the XOR operation. The sum output is high when only one of the inputs is high, while the carry output is high when both inputs are high.

Boolean Expressions:

Sum = $A \oplus B$

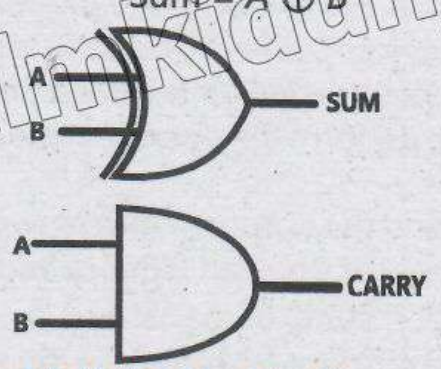


Table 3.8: Half-Adder Circuit

3.5.1.2 Full-adder Circuits

A full-adder is a more complex circuit that adds three single-bit binary numbers: two bits that belong to the sum and a carry bit from a previous addition. It has three inputs, denoted as A, B, and C_{in} (carry input), and two outputs: called the sum (S) and the carry (Cout) with both being integer values.

A	B	C_{in}	Sum (S)	*Carry* (C_{out})
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 3.7: Truth Table for Full-adder

Boolean Expressions:

Sum = A⊕B⊕Cin

Carry= (A.B) + (Cin . (A⊕B))

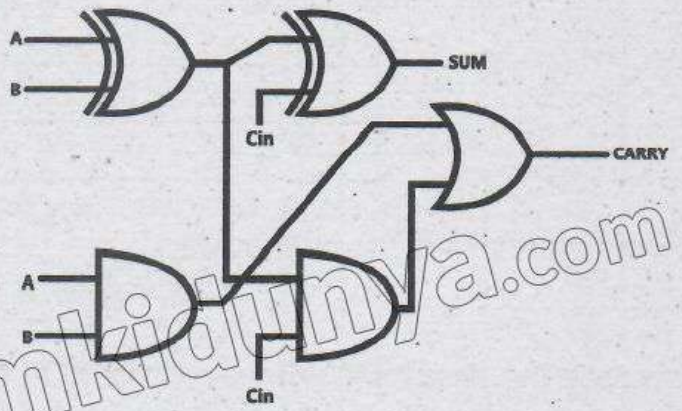


Table 3.9: Half-Adder Circuit

The sum output is high if the number of high inputs is odd whereas the carry output is high if the number of high inputs is at least 2.

3.5.2 Karnaugh Map(K-Map)

A Karnaugh map (K-map) is a graphical representation which can be used to solve Boolean algebra expressions and minimize a logic function where algebraic computations are not employed. It is a technique in which the truth value of Boolean function is plotted to enable the identification of patterns and to perform term combining for simplification.

Minterm	Variables Combination	Minterm Expression
m0	A= 0, B= 0,C= 0	$\overline{ABC}$
m1	A= 0, B= 0,C= 1	$\overline{AB}\ C$
m2	A= 0, B= 1,C= 0	$\overline{A}\,B\,\overline{C}$
m3	A= 0, B= 1,C= 1	$\overline{A}\,B\,C$
m4	A= 1, B= 0,C= 0	$A\,\overline{BC}$
m5	A= 1, B= 0,C= 1	$A\,\overline{B}\,C$
m6	A= 1, B= 1,C= 0	$A\,B\,\overline{C}$
m7	A= 1, B= 1,C= 1	$A\,B\,C$

Table 3.8: Possible Minterms for A,B and C

3.5.2.1 Structure of Karnaugh Maps

A K-map is a matrix where each square is a cell, which corresponds to a positioned combination. These cells are filled with '1' or '0' in reference to the truth table of the Boolean function. The size of the K-map depends on the number of variables:

- 2 Variables: 2x2 grid
- 3 Variables: 2x4 grid
- 4 Variables: 4x4 grid least
- 5 Variables: 4x8 grid (less common for manual simplification)

Every cell in the K-map represents a minterm, and the cells in each row of the K-map differ by only one bit at any particular position, following the gray code sequence.

3.5.2.2 Minterms in Boolean Algebra

In Boolean algebra, a minterm is a particular product term whereby every variable of the function is present in either 1 its true form or its complement. Each minterm corresponds to one and only one set of variable values that makes the Boolean function equal to true or 1.

Minterm Notation For a Boolean function with variables A, B and C:
The minterm where A = 1, B = 0 and C = 1 is written as $A\overline{B}C$.
Consider a Boolean function F(A,B,C). The possible minterms for this function are:
Possible Minterms for A,B,C

3.5.2.3 Creating Karnaugh Maps

To create a K-map, follow these steps:

1. Create a grid based on the number of variables that exists in the system.
2. Let us complete the grid using the output values in the truth table.
3. Arrange the 1s in the grid in the largest possible groups of size 1, 2, 4, 8 and so on. Every group must have one or more 1s, must be a power of two, and they must be in a continuous rows or columns.

Example: Simplifying a Boolean Expression with a K-map

To simplify the Boolean expression $A \cdot \overline{B} + \overline{A} \cdot B + A \cdot B$ using a Karnaugh map (K-map):

1. Expression: $A \cdot \overline{B} + \overline{A} \cdot B + A \cdot B$

Step 1: Draw the K-map Grid

For two variables *A* and *B*:

	$\overline{B} = 0$	$B = 1$
$\overline{A} = 0$	0	1
$A = 1$	1	1

Step 2: Fill in the K-map

Determine the output for each combination of *A* and *B* based on the expression:

- For $A = 0$ and $B = 0$: $\overline{F} = \overline{A} \cdot B + A \cdot B + A \cdot B =$ $1 \cdot 0 + 0 \cdot 0 + 0 \cdot 0 = 0$
- For $A = 0$ and $B = 1$: $F = A \cdot \overline{B} + \overline{A} \cdot B + A \cdot B =$ $0 \cdot 0 + 1 \cdot 1 + 0 \cdot 1 = 1$
- For $A = 1$ and $B = 0$: $F = A \cdot \overline{B} + \overline{A} \cdot B + A \cdot B =$ $1 \cdot 1 + 0 \cdot 0 + 1 \cdot 0 = 1$
- For $A = 1$ and $B = 1$: $F = A \cdot \overline{B} + \overline{A} \cdot B + A \cdot B =$ $1 \cdot 0 + 0 \cdot 1 + 1 \cdot 1 = 1$

Step 3: Group the 1s in the K-map

Group adjacent 1s to simplify the expression

	$\overline{B} = 0$	$B = 1$
$\overline{A} = 0$	0	1
$A = 1$	1	1

From the K-map, we can form the groups:

1. Group of two 1s in the second column: $A.B + A.B$
   $A.B + A.B = (A + A).B = 1.B = B$
2. Group of two 1s in the second row: $A.B + A.B$
   $A.B + A.B = (B + B).A = 1.A = A$

Final Simplified Expression

$$F(A, B) = B + A$$

Practical Usage:

Karnaugh maps are extensively used in digital circuit design to minimize the number of gates needed for a given function. This leads to circuits that are faster, cheaper, and consume less power.

Class Activity

Activity: Construct a digital circuit that includes both half-adders and full-adders to add two 4-bit binary numbers. Create the truth tables, Boolean expressions, and circuit diagrams for each step.

Summary

- Digital systems are the basis of the present-day electronics and computing. They process digital data in form of '0' and '1'.
- Analog signals are continuous time varying signal.
- ADC (Analog to Digital Converter) is the process of converting the continuous signals into discrete signals that can be processed by digital devices for example computers and smart phones.
- DAC (Digital to Analog Converter) converts the digital signal back to the analog signal.
- Digital logic is the basis of all digital systems. This is the technique we use to process digital information in the form of binary numbers.
- Boolean algebra is a sub-discipline of mathematics based on operations involving binary variables.
- In the case of AND operation the output is 1 only when both input values are 1. Otherwise, the output is 0.
- In an OR gate, the result is 0 only when both the input values are 0. Otherwise, the output is 1.
- The NOT operation the simplest logical operation in Boolean algebra, which accept a single binary inputs and gives its opposite as the outputs.
- Boolean functions are mathematical expressions that represent logical operations involving binary variables.
- A crucial element of digital circuit design is the logic diagram, which represents the structure of the circuit by showing connections between

logic gates.

- Adder circuits are widely used in the digital electronic systems with the principal application in arithmetic operations.
- A half-adder is a digital circuit used to compute the addition of two single-bit binary numbers.
- A full-adder is a more complex circuit that adds three single-bit numbers; two main bits and a carry bit from a previous addition.
- A Karnaugh map (K-map) is a graphic aid that is employed in simplification of Boolean expressions and minimizing logic functions without the used for complex algebraic operations.
- A minterm in Boolean algebra is a specific product (AND) form of a Boolean expression that includes all of the function's variables, either in their normal or complemented form.

Multiple- Choice Questions (MCQs)

1. Which of the following Boolean expressions represents the OR operation?
   (a) $A \cdot B$ (b) $A + B$ (c) $\bar{A}$ (d) $A \oplus B$
2. What is the dual of the Boolean expression $A \cdot 0 = 0$?
   (a) $A + 1 = 1$ (b) $A + 0 = A$ (c) $A \cdot 1 = A$ (d) $A \cdot 0 = 0$
3. Which logic gate outputs true only if both inputs are true?
   (a) OR gate (b) AND gate (c) XOR gate (d) NOT gate
4. In a half-adder circuit, the carry is generated by which operation?
   (a) XOR operation (b) AND operation
   (c) OR operation (d) NOT operation
5. What is the decimal equivalent of the binary number 1101?
   (a) 11 (b) 12 (c) 13 (d) 14

Short Questions

1. Define a Boolean function and give an example.
2. What is the significance of the truth table in digital logic?
3. Explain the difference between analog and digital signals.
4. Describe the function of a NOT gate with its truth table.
5. What is the purpose of a Karnaugh map in simplifying Boolean expressions?

Long Questions

1. Explain the usage of Boolean functions in computers.
2. Describe how to construct a truth table for a Boolean expression with an example.

3. Describe the concept of duality in Boolean algebra and provide an example to illustrate it.
4. Compare and contrast half-adders and full-adders, including their truth tables, Boolean expressions, and circuit diagrams.
5. How do Karnaugh maps simplify Boolean expressions? Provide a detailed example with steps.
6. Design a 4-bit binary adder using both half-adders and full-adders. Explain each step with truth tables, Boolean expressions, and circuit diagrams.
7. Simplify the following Boolean function using Boolean algebra rules:

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

8. Use De Morgan's laws to simplify the following function:

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

9. Simplify the following expressions

(a) $\overline{A + B} \cdot (A + B)$

(b) $(A + \overline{B}) \cdot (\overline{A} + B)$

(c) $A + \overline{A} \cdot (\overline{B} + C)$

(d) $\overline{A \cdot B} + A \cdot B$

(e) $(A \cdot B) + (\overline{A} \cdot \overline{B})$