

Digital Electronics (25ECH-101)

Compiled by : Subhayu





Syllabus for Unit 1

Digital Electronics (25ECH-101)

- **Introduction to Digital Electronics and Logic Design**

- Basic introduction to digital electronics
- Need of digital electronics
- Difference between analog and digital electronics
- Introduction to logic gates
- Validation of logic gates using truth tables
- Implementation of logic gates using universal gates
- Basic theorems of Boolean algebra
- Proofs and verification of Boolean laws

- **Boolean Algebra and Logic Simplification**

- Sum of Products (SOP) and Product of Sums (POS)
- Minterms and maxterms
- Relations between minterms and maxterms
- Canonical forms
- Simplification using Karnaugh maps (K-maps)
- K-map simplification with don't care conditions
- Implementation of Boolean expressions using basic logic gates

- **Fundamentals of Binary Arithmetic and Code Conversions**

- Number systems and their types
- Binary arithmetic operations
- Conversion of binary, octal and hexadecimal to decimal
- Conversion from decimal to binary, octal and hexadecimal
- Conversion from binary to other number systems
- Complements: binary, octal, hexadecimal, Gray code, BCD code, Excess-3 code
- 1's and 2's complement
- Addition and subtraction using 1's and 2's complement

Chapter 1

Introduction to Digital Electronics and Logic Design

Digital Electronics and Logic Design forms the backbone of modern computing systems, communication devices, and embedded technologies. From smartphones and laptops to traffic control systems and medical instruments, digital electronics governs how information is processed, stored, and transmitted in a reliable and efficient manner.

This chapter builds a strong conceptual foundation by starting from the basics, explaining *why* digital electronics is needed, how it differs from analog electronics, and how logic gates act as the fundamental building blocks of all digital systems.

1.1 Basic Introduction to Digital Electronics

Digital Electronics is a branch of electronics that deals with circuits and systems which operate using *discrete signal levels* rather than continuous signals. These discrete levels are usually represented using binary values:

$$0 \text{ (LOW / FALSE)} \quad \text{and} \quad 1 \text{ (HIGH / TRUE)}$$

Unlike analog electronics, where signals vary smoothly over a range of values, digital electronics relies on clearly distinguishable states. This property makes digital systems highly reliable and less susceptible to noise.

1.1.1 Digital Signals

A **digital signal** can take only a finite number of values, typically two.

- Logic 0: Usually represented by 0 V
- Logic 1: Usually represented by 5 V or 3.3 V (depending on technology)



Information is encoded using combinations of 0s and 1s, forming the binary number system.

Key Insight

Any complex digital system, including processors and memory units, ultimately operates by manipulating binary digits (bits).

1.1.2 Applications of Digital Electronics

Digital electronics is used extensively in:

- Computers and laptops
- Mobile phones and tablets
- Digital watches and calculators
- Communication systems
- Control systems and automation

1.2 Need of Digital Electronics

The shift from analog to digital electronics occurred due to several practical advantages offered by digital systems.

1.2.1 Accuracy and Reliability

Digital systems provide consistent outputs because small variations (noise) do not usually affect the interpretation of 0s and 1s.

Why Digital Works Better

If a signal meant to be logic 1 fluctuates slightly due to noise, it is still recognized as 1 as long as it remains above a threshold voltage.

1.2.2 Ease of Storage and Processing

Digital data can be:

- Stored easily in memory devices
- Processed using algorithms
- Transmitted over long distances without significant degradation



1.2.3 Programmability and Scalability

Digital systems are highly programmable. The same hardware can perform different tasks simply by changing the program.

Example: A microcontroller can be programmed to control a washing machine today and a traffic light tomorrow.

1.2.4 Cost and Integration

With advancements in VLSI (Very Large Scale Integration), millions of logic gates can be placed on a single chip, reducing cost and size while increasing performance.

1.3 Difference Between Analog and Digital Electronics

Understanding the difference between analog and digital electronics is crucial before diving deeper into logic design.

1.3.1 Analog Electronics

Analog electronics deals with signals that vary continuously with time.

Examples:

- Audio signals from a microphone
- Temperature variations
- Voltage across a potentiometer

1.3.2 Digital Electronics

Digital electronics deals with signals that have discrete levels.

Examples:

- Binary data in computers
- Digital clocks
- On/Off control systems



1.3.3 Comparison Table

Parameter	Analog	Digital
Signal type	Continuous	Discrete
Noise immunity	Low	High
Accuracy	Limited	High
Storage	Difficult	Easy
Processing	Complex	Simple and fast

Common Doubt Clarified

Digital systems are not always more “accurate” than analog systems, but they are more **consistent and reliable** under noisy conditions.

1.4 Introduction to Logic Gates

Logic gates are the fundamental building blocks of digital electronics. A **logic gate** is an electronic circuit that performs a logical operation on one or more binary inputs to produce a single binary output.

1.4.1 Basic Logic Gates

The basic logic gates include:

- AND Gate
- OR Gate
- NOT Gate

These gates implement Boolean logic.

1.4.2 AND Gate

Definition: The AND gate produces an output of 1 only when *all* inputs are 1.

$$Y = A \cdot B$$

Truth Table:

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



Solved Example: If $A = 1$ and $B = 0$, then:

$$Y = A \cdot B = 1 \cdot 0 = 0$$

1.4.3 OR Gate

Definition: The OR gate produces an output of 1 if *at least one* input is 1.

$$Y = A + B$$

Truth Table:

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

1.4.4 NOT Gate

Definition: The NOT gate (also called an inverter) produces the complement of the input.

$$Y = \overline{A}$$

Truth Table:

A	Y
0	1
1	0

Remember This

Every complex digital circuit can be built using combinations of simple logic gates.

1.4.5 Real-Life Analogy

- AND Gate: A car starts only if the key is turned **and** the clutch is pressed.
- OR Gate: A room light turns on if switch A **or** switch B is pressed.
- NOT Gate: A motion sensor light turns OFF when motion is detected.

This completes the foundational understanding required to confidently move forward into Boolean algebra, combinational circuits, and sequential logic.



1.5 Validation of Logic Gates Using Truth Tables

A **truth table** is a systematic tabular representation that shows all possible input combinations for a logic gate or Boolean expression along with their corresponding outputs. Truth tables serve as the most fundamental tool to *validate*, *analyze*, and *verify* the behavior of logic gates.

1.5.1 Why Truth Tables Are Important

Truth tables help in:

- Verifying the correctness of a logic gate
- Understanding logical relationships between inputs and outputs
- Debugging digital circuits
- Comparing different Boolean expressions

Conceptual Clarity

If two logic circuits or Boolean expressions produce identical truth tables, they are logically equivalent.

1.5.2 General Rules for Constructing Truth Tables

- For n inputs, the truth table contains 2^n rows
- Each input variable takes values 0 and 1 in a binary counting manner
- Output is evaluated for each input combination

1.5.3 Validation Example: AND Gate

Boolean Expression:

$$Y = A \cdot B$$

Truth Table:

A	B	$Y = A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

Conclusion: The output is HIGH only when both inputs are HIGH, validating the AND gate operation.



1.5.4 Validation Example: Expression

Validate the expression:

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

Truth Table:

A	B	Y
0	0	1
0	1	0
1	0	1
1	1	1

Doubt Clarification

Truth tables do not depend on circuit implementation. They depend only on logical relationships.

1.6 Implementation of Logic Gates Using Universal Gates

Certain logic gates are termed **universal gates** because they can be used to implement any Boolean function or logic gate.

1.6.1 Universal Gates

The two universal gates are:

- NAND Gate
- NOR Gate

Why Universal Gates Matter

Using only one type of gate simplifies IC fabrication and reduces hardware complexity.

1.6.2 Implementation Using NAND Gate

NOT Gate Using NAND

$$Y = \overline{A} = A \text{ NAND } A$$



AND Gate Using NAND

$$Y = (A \text{ NAND } B)' = A \cdot B$$

Explanation: First NAND gives $(A \cdot B)'$, applying NAND again acts as inversion.

OR Gate Using NAND

Using De Morgan's Theorem:

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

Steps:

- Invert A and B using NAND
- NAND the inverted inputs

1.6.3 Implementation Using NOR Gate

NOT Gate Using NOR

$$Y = \overline{A} = A \text{ NOR } A$$

OR Gate Using NOR

$$Y = (A \text{ NOR } B)' = A + B$$

AND Gate Using NOR

Using De Morgan's Theorem:

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

Exam Tip

Questions frequently ask to implement AND/OR using only NAND or only NOR gates.

1.7 Basic Theorems of Boolean Algebra

Boolean algebra is a mathematical system used to analyze and simplify digital logic circuits.

1.7.1 Fundamental Boolean Laws

Identity Laws

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



Null Laws

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

Idempotent Laws

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

Complement Laws

$$A + \bar{A} = 1 \quad A \cdot \bar{A} = 0$$

Commutative Laws

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

Associative Laws

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

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

Distributive Laws

$$A(B + C) = AB + AC$$

$$A + BC = (A + B)(A + C)$$

Important

Boolean distributive laws differ from conventional algebra — both forms are valid.

1.8 Proofs and Verification of Boolean Laws

Boolean laws can be verified using:

- Algebraic manipulation
- Truth tables

1.8.1 Proof Using Algebraic Method

Prove:

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

Proof:

$$A + A\bar{B} = A(1 + \bar{B})$$



$$= A \cdot 1$$

$$= A$$

1.8.2 Proof Using Truth Table

Verify:

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

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

Conclusion: Both sides give identical outputs, hence the law is verified.

1.8.3 De Morgan's Theorems

Theorems:

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

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

1.8.4 Verification Example

Verify:

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

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

Final Takeaway

Boolean laws enable simplification of logic circuits, reducing gate count, cost, and power consumption.

Chapter 2

Boolean Algebra and Logic Simplification

Boolean Algebra provides a systematic and mathematical framework to represent, analyze, and simplify digital logic circuits. Logic simplification is crucial because simpler circuits require fewer gates, consume less power, occupy less hardware area, and are more reliable.

This chapter focuses on standardized representations of Boolean functions—namely SOP and POS forms—along with the foundational concepts of minterms, maxterms, their interrelations, and canonical representations.

2.1 Sum of Products (SOP) and Product of Sums (POS)

Boolean expressions can be represented in multiple algebraic forms. Among them, **Sum of Products (SOP)** and **Product of Sums (POS)** are the most widely used and standardized forms.

2.1.1 Sum of Products (SOP)

Definition: A Boolean expression is said to be in **Sum of Products (SOP)** form if it consists of:

- One or more *product terms* (AND terms)
- All product terms combined using OR operation

$$Y = AB + \overline{A}C + BC$$



Here, AB , $\overline{A}C$, and BC are product terms, and their sum represents the SOP expression.

Key Observation

In SOP form, AND operations are performed first, followed by OR operations.

2.1.2 Example of SOP

Given Expression:

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

This is already in SOP form since it is a sum (OR) of product (AND) terms.

2.1.3 Product of Sums (POS)

Definition: A Boolean expression is said to be in **Product of Sums (POS)** form if it consists of:

- One or more *sum terms* (OR terms)
- All sum terms combined using AND operation

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

Conceptual Clarity

In POS form, OR operations are performed first, followed by AND operations.

2.1.4 Comparison Between SOP and POS

Parameter	SOP	POS
Structure	OR of AND terms	AND of OR terms
Gate realization	AND-OR	OR-AND
Used with	Minterms	Maxterms

2.2 Minterms and Maxterms

Minterms and maxterms are standardized building blocks used to express Boolean functions in canonical form.



2.2.1 Minterms

Definition: A **minterm** is a product (AND) term that includes *all input variables exactly once*, either in complemented or uncomplemented form.

Example: For variables A and B , possible minterms are:

$$\overline{A}\overline{B}, \quad \overline{A}B, \quad A\overline{B}, \quad AB$$

Each minterm corresponds to exactly *one row* of the truth table where the function output is 1.

2.2.2 Minterm Notation

Minterms are denoted using m_i , where i is the decimal equivalent of the binary input combination.

Example:

$$AB = 10_2 = m_2$$

2.2.3 Maxterms

Definition: A **maxterm** is a sum (OR) term that includes *all input variables exactly once*, either in complemented or uncomplemented form.

Example:

$$(A + B), \quad (A + \overline{B}), \quad (\overline{A} + B), \quad (\overline{A} + \overline{B})$$

Each maxterm corresponds to exactly *one row* of the truth table where the function output is 0.

2.2.4 Maxterm Notation

Maxterms are denoted using M_i , where i is the decimal equivalent of the binary input combination that produces output 0.

Doubt Clarification

Minterms focus on output 1 rows, while maxterms focus on output 0 rows of the truth table.

2.3 Relations Between Minterms and Maxterms

Minterms and maxterms are closely related and complementary in nature.



2.3.1 Key Relationship

For a Boolean function $F(A, B, \dots)$:

$$F = \sum m(i) \iff F = \prod M(j)$$

where:

- i represents indices where output is 1
- j represents indices where output is 0

2.3.2 Complementary Nature

$$m_i = \overline{M_i}$$

This means:

- The complement of a minterm is the corresponding maxterm
- The complement of a maxterm is the corresponding minterm

2.3.3 Solved Example

Given:

$$F(A, B) = \sum m(1, 3)$$

Step 1: List all possible indices

$$0, 1, 2, 3$$

Step 2: Identify maxterms

$$F(A, B) = \prod M(0, 2)$$

Exam Insight

If minterms are given, maxterms can be found by excluding those indices from the complete set.

2.4 Canonical Forms

A Boolean expression is said to be in **canonical form** if it is expressed using either:

- Only minterms (canonical SOP)



- Only maxterms (canonical POS)

2.4.1 Canonical SOP Form

Definition: A Boolean function expressed as a sum of all its minterms is said to be in **canonical SOP form**.

Example:

$$F(A, B) = \sum m(1, 2)$$

Expanded form:

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

2.4.2 Canonical POS Form

Definition: A Boolean function expressed as a product of all its maxterms is said to be in **canonical POS form**.

Example:

$$F(A, B) = \prod M(0, 3)$$

Expanded form:

$$F = (A + B)(\bar{A} + \bar{B})$$

2.4.3 Conversion to Canonical Form

Rule: Every term must contain all variables.

Example: Convert to canonical SOP

$$F = A + B$$

Solution:

$$\begin{aligned} F &= A(B + \bar{B}) + B(A + \bar{A}) \\ &= AB + A\bar{B} + \bar{A}B \end{aligned}$$

Final Takeaway

Canonical forms provide a unique and systematic representation of Boolean functions, forming the foundation for Karnaugh Maps and digital circuit optimization.

2.5 Simplification Using Karnaugh Maps (K-maps)

As the number of variables in a Boolean function increases, algebraic simplification becomes lengthy and error-prone. **Karnaugh Maps (K-maps)** provide a visual and sys-



tematic technique to simplify Boolean expressions by exploiting adjacency of minterms or maxterms.

2.5.1 What is a Karnaugh Map?

A **Karnaugh Map** is a graphical representation of a truth table, arranged in such a way that:

- Adjacent cells differ by only one variable (Gray code ordering)
- Simplification can be done by grouping adjacent 1s (for SOP)
- Larger groups lead to simpler expressions

Core Idea

K-map simplification is based on the principle that adjacent minterms differ in only one literal, which can be eliminated.

2.5.2 Types of K-maps

- 2-variable K-map: 4 cells
- 3-variable K-map: 8 cells
- 4-variable K-map: 16 cells

2.5.3 2-Variable K-map

For variables A and B :

$A \backslash B$	0	1
0	m_0	m_1
1	m_2	m_3

2.5.4 Simplification Rules

- Group only adjacent 1s (horizontal, vertical, or wrap-around)
- Group sizes must be powers of 2: 1, 2, 4, 8, ...
- Each group eliminates the variable that changes within the group
- Larger groups give simpler expressions



2.5.5 Solved Example (2-Variable)

Given:

$$F(A, B) = \sum m(1, 3)$$

K-map Filling:

$A \setminus B$	0	1
0	0	1
1	0	1

Grouping: The two 1s in column $B = 1$ are grouped.

Simplified Expression:

$$F = B$$

Interpretation

Variable A changes within the group, hence it is eliminated.

2.5.6 3-Variable K-map

For variables A , B , and C :

$A \setminus BC$	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

2.5.7 Solved Example (3-Variable)

Given:

$$F(A, B, C) = \sum m(1, 3, 5, 7)$$

Observation: All minterms correspond to $C = 1$.

Simplified Expression:

$$F = C$$

2.6 K-map Simplification with Don't Care Conditions

In many practical digital systems, certain input combinations never occur or their output is irrelevant. These cases are called **don't care conditions**.



2.6.1 Don't Care Conditions

Definition: A don't care condition is an input combination for which the output can be either 0 or 1 without affecting system operation.

- Represented using symbol X
- Can be treated as 1 or 0 for simplification

Why Use Don't Cares?

Don't care conditions help form larger groups, resulting in simpler Boolean expressions.

2.6.2 Solved Example with Don't Care

Given:

$$F(A, B, C) = \sum m(1, 3, 7) + d(5)$$

K-map Filling:

$A \backslash BC$	00	01	11	10
0	0	1	1	0
1	0	X	1	0

Grouping:

- Use X to form a group of four

Simplified Expression:

$$F = C$$

Important Note

Don't care terms must be used only if they help simplification; otherwise, they can be ignored.

2.7 Implementation of Boolean Expressions Using Basic Logic Gates

Once a Boolean expression is simplified, it must be implemented using logic gates to form a digital circuit.



2.7.1 Gate-Level Implementation

Boolean expressions are implemented using:

- AND gates
- OR gates
- NOT gates

2.7.2 Implementation of SOP Expression

Example:

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

Steps:

- Use NOT gate to obtain $\overline{B}$
- AND gate for $A\overline{B}$
- AND gate for BC
- OR gate to sum the product terms

2.7.3 Implementation of POS Expression

Example:

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

Steps:

- Use NOT gate to obtain $\overline{A}$
- OR gates for sum terms
- AND gate to combine sum terms

2.7.4 Design Flow Summary

1. Write the truth table
2. Obtain Boolean expression
3. Simplify using K-map
4. Implement using logic gates

Chapter 3

Fundamentals of Binary Arithmetic and Code Conversions

Digital systems operate internally using numbers, but not in the decimal form humans are naturally comfortable with. Instead, they rely on *binary-based number systems* that align with the physical realities of electronic hardware (ON/OFF, HIGH/LOW).

This chapter builds a strong numerical foundation by explaining different number systems, binary arithmetic operations, and systematic conversion techniques between decimal, binary, octal, and hexadecimal systems.

3.1 Number Systems and Their Types

A **number system** is a method of representing numbers using a set of symbols (digits) and a base (radix).

3.1.1 Key Terms

- **Radix (Base):** Total number of unique digits including zero
- **Positional Value:** Each digit's value depends on its position

Core Concept

All number systems used in digital electronics are positional number systems.

3.1.2 Types of Number Systems

Decimal Number System

- Base (radix): 10
- Digits used: 0 to 9



Example:

$$(347)_{10} = 3 \times 10^2 + 4 \times 10^1 + 7 \times 10^0$$

Binary Number System

- Base: 2
- Digits used: 0 and 1

Example:

$$(1011)_2$$

Binary is the fundamental number system used in digital electronics due to its compatibility with electronic switches.

Octal Number System

- Base: 8
- Digits used: 0 to 7

Hexadecimal Number System

- Base: 16
- Digits used: 0–9 and A–F

Hex Digit	Decimal Value
A	10
B	11
C	12
D	13
E	14
F	15

Why Octal and Hex?

Octal and hexadecimal provide compact representations of long binary numbers.

3.2 Binary Arithmetic Operations

Binary arithmetic forms the basis of all computations performed by digital systems.



3.2.1 Binary Addition

A	B	Sum
0	0	0
0	1	1
1	0	1
1	1	10

Solved Example

$$(1011)_2 + (0101)_2$$

$$\begin{array}{r} 1011 \\ +0101 \\ \hline 10000 \end{array}$$

3.2.2 Binary Subtraction

A	B	Difference
0	0	0
1	0	1
1	1	0
0	1	1 (borrow)

3.2.3 Binary Multiplication

Binary multiplication is simpler than decimal multiplication.

Rules:

$$0 \times 0 = 0, \quad 0 \times 1 = 0, \quad 1 \times 0 = 0, \quad 1 \times 1 = 1$$

Solved Example

$$(101)_2 \times (11)_2$$

$$\begin{array}{r} 101 \\ \times 11 \\ \hline 101 \\ +1010 \\ \hline 1111 \end{array}$$

3.2.4 Binary Division

Binary division follows the same logic as decimal division but uses binary subtraction.

**Exam Hint**

Binary addition and subtraction are frequently used to design arithmetic circuits like adders and subtractors.

3.3 Conversion of Binary, Octal, and Hexadecimal to Decimal

3.3.1 Binary to Decimal Conversion

Method: Multiply each bit by 2^{position} and sum the results.

Solved Example

Convert $(1011)_2$ to decimal.

$$\begin{aligned} &= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\ &= 8 + 0 + 2 + 1 = 11 \end{aligned}$$

3.3.2 Octal to Decimal Conversion

Method: Multiply each digit by 8^{position} .

Solved Example

$$\begin{aligned} (257)_8 &= 2 \times 8^2 + 5 \times 8^1 + 7 \times 8^0 \\ &= 128 + 40 + 7 = 175 \end{aligned}$$

3.3.3 Hexadecimal to Decimal Conversion

Method: Multiply each digit by 16^{position} .

Solved Example

$$\begin{aligned} (2F)_{16} &= 2 \times 16^1 + 15 \times 16^0 \\ &= 32 + 15 = 47 \end{aligned}$$



3.4 Conversion from Decimal to Binary, Octal, and Hexadecimal

3.4.1 Decimal to Binary Conversion

Method: Repeatedly divide the decimal number by 2 and note remainders.

Solved Example

Convert $(13)_{10}$ to binary.

$$13 \div 2 = 6 \text{ R } 1$$

$$6 \div 2 = 3 \text{ R } 0$$

$$3 \div 2 = 1 \text{ R } 1$$

$$1 \div 2 = 0 \text{ R } 1$$

$$(13)_{10} = (1101)_2$$

3.4.2 Decimal to Octal Conversion

Method: Repeated division by 8.

Solved Example

$$(83)_{10} = (123)_8$$

3.4.3 Decimal to Hexadecimal Conversion

Method: Repeated division by 16.

Solved Example

$$(254)_{10} = (FE)_{16}$$

Final Takeaway

Mastery of number systems and conversions is essential for understanding data representation, memory addressing, and arithmetic operations in digital systems.



3.5 Conversion from Binary to Other Number Systems

Binary is the core number system used internally by digital hardware. However, for human readability and compact representation, binary numbers are often converted to octal, hexadecimal, or coded number systems.

3.5.1 Binary to Decimal

This conversion has already been discussed earlier and serves as the foundation for understanding other conversions.

Reminder Method:

$$(10101)_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

3.5.2 Binary to Octal

Method:

- Group binary digits in sets of three starting from the LSB
- Add leading zeros if required
- Convert each group to its octal equivalent

Solved Example

Convert $(1101011)_2$ to octal.

$$001 \ 101 \ 011$$

$$(001)_2 = 1, \quad (101)_2 = 5, \quad (011)_2 = 3$$

$$(1101011)_2 = (153)_8$$

Why Group by 3?

Since $8 = 2^3$, three binary bits correspond to one octal digit.

3.5.3 Binary to Hexadecimal

Method:



- Group binary digits in sets of four from the LSB
- Add leading zeros if required
- Convert each group to its hexadecimal equivalent

Solved Example

Convert $(11101101)_2$ to hexadecimal.

1110 1101

$$(1110)_2 = E, \quad (1101)_2 = D$$

$$(11101101)_2 = (ED)_{16}$$

3.6 Complements and Coded Number Systems

Complements and codes play a vital role in simplifying arithmetic operations and representing data in digital systems.

3.6.1 Complements

A **complement** of a number system is a representation that helps in performing subtraction using addition.

Binary Complement

Binary system uses:

- 1's complement
- 2's complement

Octal Complement

- 7's complement
- 8's complement



Hexadecimal Complement

- 15's complement
- 16's complement

3.6.2 Gray Code

Definition: Gray code is a *unit-distance code* in which two successive values differ by only one bit.

Example:

Decimal	Gray Code
0	000
1	001
2	011
3	010
4	110

Why Gray Code?

Used in rotary encoders to avoid errors due to bit changes.

3.6.3 Binary Coded Decimal (BCD)

Definition: BCD represents each decimal digit using its 4-bit binary equivalent.

Example:

$$(59)_{10} = 0101 \ 1001 \text{ (BCD)}$$

3.6.4 Excess-3 Code

Definition: Excess-3 is a non-weighted code obtained by adding 3 to each decimal digit and then converting it to binary.

Solved Example

Convert decimal 47 to Excess-3 code.

$$4 + 3 = 7 \Rightarrow 0111$$

$$7 + 3 = 10 \Rightarrow 1010$$

$$(47)_{10} = 0111 \ 1010 \text{ (Excess-3)}$$



3.7 1's and 2's Complement

3.7.1 1's Complement

Definition: 1's complement of a binary number is obtained by inverting all bits.

Example:

$$(10110)_2 \Rightarrow (01001)_2$$

3.7.2 2's Complement

Definition: 2's complement is obtained by adding 1 to the 1's complement of a binary number.

Example:

$$\begin{aligned}(10110)_2 \\ 1's \text{ complement} &= 01001 \\ 2's \text{ complement} &= 01010\end{aligned}$$

Why 2's Complement?

2's complement allows representation of negative numbers and simplifies subtraction.

3.8 Addition and Subtraction Using 1's and 2's Complement

3.8.1 Subtraction Using 1's Complement

Steps:

1. Take 1's complement of subtrahend
2. Add it to the minuend
3. If carry is generated, add it back to the result (end-around carry)

Solved Example

$$(1101)_2 - (0101)_2$$

$$1's \text{ complement of } 0101 = 1010$$



$$1101 + 1010 = 10111$$

$$\text{End-around carry: } 0111 + 1 = 1000$$

3.8.2 Subtraction Using 2's Complement

Steps:

1. Take 2's complement of subtrahend
2. Add it to the minuend
3. Ignore carry if generated

Solved Example

$$(1010)_2 - (0011)_2$$

$$\text{2's complement of } 0011 = 1101$$

$$1010 + 1101 = 10111$$

$$\text{Ignoring carry: } 0111$$

$$\Rightarrow (1010)_2 - (0011)_2 = (0111)_2$$

Final Confidence Note

Complement-based arithmetic is the foundation of arithmetic logic units (ALUs) in modern processors.



Mid Semester Test 1 – Sample Paper

Digital Electronics

Total Marks : 20

Time Allotted : 1 hour

Instructions:

- The question paper consists of 2 sections: Section A and Section B.
- Section A has 5 questions of 2 marks each and Section B has 2 questions of 5 marks each.

Section A ($5 \times 2 = 10$ marks)

1. Define Digital Electronics. State any two advantages of digital systems over analog systems.
2. Write the truth table of a two-input NAND gate and justify why it is called a universal gate.
3. What is a minterm? How many minterms are possible for a Boolean function with three variables?
4. Convert the binary number $(101101)_2$ into its decimal equivalent.
5. What is 2's complement? Mention one practical advantage of using 2's complement representation.

Section B ($2 \times 5 = 10$ marks)

6. (a) Explain the difference between analog and digital electronics with suitable examples.
(b) State and prove any two basic theorems of Boolean algebra using algebraic method or truth tables.
7. (a) Simplify the following Boolean expression using Karnaugh Map:

$$F(A, B, C) = \sum m(1, 3, 5, 7)$$

- (b) Implement the simplified expression using basic logic gates and draw the logic diagram.

Closing Note

I sincerely hope that these notes and the preparatory questions included alongside them prove helpful in strengthening your understanding and boosting your confidence for the examination you are preparing for.

As you revise, try not to rush through the content—pause at examples, work out the numericals on your own, and revisit sections that feel slightly uncomfortable. That discomfort usually signals learning in progress. Use these notes as a guide and a companion, not as something to be memorized line by line.

Wishing you the very best for your exams—may your preparation feel clear, structured, and calm when it matters most.

For future updates, refinements, and additional resources, keep checking **Noteify**:

<https://subhayu18.github.io/Noteify/>

If you have any doubts, suggestions, or simply wish to connect, feel free to reach out to me on LinkedIn:

<https://www.linkedin.com/in/subhayu-mukherjee-146421294/>

You can also use the direct contact options provided on the Noteify website itself.

Regards,
Subhayu