De (4321102) - Problem Solver

Antigravity AI

June 4, 2026

De

First Edition: 2026

Author: Antigravity AI
Website: milav.in
YouTube: @milav.dabgar

To my students,
who constantly inspire me to find new analogies,
break down complex theories, and make learning
an engineered science.

Contents

Preface

About this Book

Welcome to **Computer Networks**! This textbook is meticulously designed to serve as a comprehensive problem solver and study companion. It provides a robust and intuitive foundation in the core principles of this subject, bridging the gap between theoretical models and practical applications. A unique feature of this book is its focus on decoding complex concepts using clear, step-by-step methodologies and real-world analogies.

Target Audience

This book is intended for students and professionals looking to master the concepts of Computer Networks. Whether you are revising for exams, seeking to solidify your fundamental understanding, or looking for practical examples to clarify abstract concepts, this book has been structured to support your learning journey. It serves as an accessible guide designed to remove the fear of complex topics, replacing it with an appreciation for the underlying mechanics.

Scope and Coverage

The coverage is divided logically to follow standard academic curriculums. Each chapter focuses on key topics, breaking them down into digestible sections:

- **Theoretical Insights** – Core concepts explained intuitively.
- **Method Blocks** – Standardized, step-by-step procedures for solving specific types of problems.
- **Worked Examples** – Fully solved problems that apply the methods in practice.

We hope this resource proves invaluable in your studies and helps you achieve excellence in **Computer Networks**. Happy learning!

Acknowledgments

Writing a comprehensive book that connects the fundamental forces of Chemistry with practical engineering applications requires more than just academic knowledge—it requires a community of support. I would like to express my sincere gratitude to everyone who contributed to the realization of this project.

Specially, I extend my heartfelt gratitude to the **Department of Technical Education (DTE), Government of Gujarat**, and **Government Polytechnic, Palanpur**, for providing an environment that fosters innovation, academic rigor, and continuous professional development.

I am deeply thankful to my family for their unwavering patience and support during the countless hours spent researching, formatting, and refining these chapters.

Finally, my deepest appreciation goes to my students. Your curiosity, challenging questions, and continuous feedback are the true driving force behind the unique analogies and streamlined structure of this book. This textbook was engineered for you.

Antigravity AI

About the Author

Milav Jayeshkumar Dabgar is an engineering educator and R&D professional with over 9 years of experience in electronics hardware development, embedded systems, AI/ML, and full-stack software engineering. He currently serves as a **Lecturer (Class - II) & Innovation Leader** at Government Polytechnic, Palanpur, under the Education Department of the Government of Gujarat.

Milav holds a Master of Engineering (ME) in Communication Systems from L.D. College of Engineering and a Bachelor of Engineering (BE) in Electronics & Communication. He is also currently pursuing a **BS in Data Science and Applications from IIT Madras**, where he has completed both the **Diploma in Programming** and the **Diploma in Data Science** with distinction.

Most recently, he completed the **AICTE QIP PG Certificate Program in Deep Learning from SVNIT Surat** with a **perfect 10/10 CGPA**, topping his batch.

A dedicated mentor, Milav has guided numerous student teams in securing over **INR 45 lakhs** in innovation funding, including INR 25 lakhs from Shark Tank India and INR 20 lakhs through iHub Gujarat, resulting in two student patents. He is recognized as an **NPTEL Evangelist** and **NPTEL Discipline Star** in Computer Science, reflecting his commitment to continuous learning and academic excellence.

His technical expertise spans from embedded systems (8051, STM32, Arduino) to modern web technologies (Next.js, Python, PostgreSQL) and Data Science (TensorFlow, PyTorch). Through this textbook, he aims to simplify complex Engineering Chemistry concepts by integrating relatable, real-world analogies drawn from modern tech and engineering landscapes.

Milav is also an active content creator, running a popular **YouTube channel** (@milav.dabgar) where he shares technical tutorials and academic resources. He maintains a personal blog and resource hub at milav.in and can be reached for professional collaborations on LinkedIn.

CHAPTER 1

Number Systems and Codes

1.1 Interpret Various Number Systems

Methodology:

Number Systems Overview A number system is defined by its base (or radix) which specifies the total number of distinct symbols used.

1. **Decimal System** (Base 10): Uses 10 symbols (0 to 9).
2. **Binary System** (Base 2): Uses 2 symbols (0 and 1).
3. **Octal System** (Base 8): Uses 8 symbols (0 to 7).
4. **Hexadecimal System** (Base 16): Uses 16 symbols (0 to 9 and A to F where $A = 10$ $B = 11$ $C = 12$ $D = 13$ $E = 14$ $F = 15$).

1.2 Convert Between Number Systems

Methodology:

Decimal to Any Base Conversion To convert a decimal number to another base r:

1. **Integer Part**: Successively divide the decimal integer by the base r until the quotient is 0. Record the remainders. Read the remainders from bottom to top (last remainder is the Most Significant Digit).
2. **Fractional Part**: Successively multiply the decimal fraction by the base r. Record the integer parts generated. Read the integers from top to bottom (first integer is the Most Significant Digit).

Worked Example:

Decimal to Binary Convert $(25.375)_{10}$ to Binary.
Solution: Step 1: Convert integer part (25).

Division	Quotient	Remainder
$25 \div 2$	12	1 (LSB)
$12 \div 2$	6	0
$6 \div 2$	3	0
$3 \div 2$	1	1
$1 \div 2$	0	1 (MSB)

Reading bottom to top: $(25)_{10} = (11001)_2$.
Step 2: Convert fractional part (0.375).

Multiplication	Result	Integer Part
0.375×2	0.750	0 (MSB)
0.750×2	1.500	1
0.500×2	1.000	1 (LSB)

Reading top to bottom: $(0.375)_{10} = (0.011)_2$.
Final Answer: $(25.375)_{10} = (11001.011)_2$

Worked Example:

Decimal to Hexadecimal Convert $(450)_{10}$ to Hexadecimal.
Solution: Step 1: Repeatedly divide by 16.

Division	Quotient	Remainder
$450 \div 16$	28	2
$28 \div 16$	1	12 (which is C)
$1 \div 16$	0	1

Reading from bottom to top: 1, C, 2.
Final Answer: $(450)_{10} = (1C2)_{16}$

Methodology:

Any Base to Decimal Conversion To convert a number from base r to decimal multiply each digit by its positional weight (r^n) and sum the results. For a number $d_2d_1d_0.d_{-1}d_{-2}$ in base r:

$$\text{Decimal Value} = d_2 \times r^2 + d_1 \times r^1 + d_0 \times r^0 + d_{-1} \times r^{-1} + d_{-2} \times r^{-2}$$

Worked Example:

Binary to Decimal Convert $(1101.101)_2$ to Decimal.
Solution: Multiply each bit by powers of 2.

$$\begin{aligned}(1101.101)_2 &= 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \\ &= 8 + 4 + 0 + 1 + 0.5 + 0 + 0.125 \\ &= 13.625\end{aligned}$$

Final Answer: $(1101.101)_2 = (13.625)_{10}$

Methodology:

Binary and Octal Interconversion

1. **Binary to Octal**: Group the binary bits into sets of 3 starting from the decimal point. Add leading or trailing zeros if necessary. Replace each 3-bit group with its octal equivalent.
2. **Octal to Binary**: Replace each octal digit with its 3-bit binary equivalent.

Worked Example:

Binary to Octal and Octal to Binary 1. Convert $(1011010.11)_2$ to Octal. 2. Convert $(62.4)_8$ to Binary.
Solution for 1: Group bits into threes from the binary point: Integer part: 001 011 010 (added two leading zeros) Fraction part: 110 (added one trailing zero) Convert each group: 001 $\rightarrow$ 1, 011 $\rightarrow$ 3, 010 $\rightarrow$ 2, 110 $\rightarrow$ 6. **Answer:** $(132.6)_8$
Solution for 2: Convert each octal digit to 3 bits: 6 $\rightarrow$ 110 2 $\rightarrow$ 010 4 $\rightarrow$ 100 Combine them: 110010.100

Answer: $(110010.1)_2$

Methodology:

Binary and Hexadecimal Interconversion

1. **Binary to Hexadecimal**: Group the binary bits into sets of 4 starting from the decimal point. Pad with zeros if needed. Replace each 4-bit group with its hex equivalent.
2. **Hexadecimal to Binary**: Replace each hex digit with its 4-bit binary equivalent.

Worked Example:

Hexadecimal to Binary Convert $(2A.C)_{16}$ to Binary.
Solution: Convert each digit to 4 bits: $2 \rightarrow 0010$ $A(10) \rightarrow 1010$ $C(12) \rightarrow 1100$ Combine them: 00101010.1100
Final Answer: $(2A.C)_{16} = (101010.11)_2$

Methodology:

Octal and Hexadecimal Interconversion To convert between Octal and Hexadecimal use Binary as an intermediate step.

1. **Octal to Hexadecimal**: Convert Octal $\rightarrow$ Binary $\rightarrow$ Hexadecimal.
2. **Hexadecimal to Octal**: Convert Hexadecimal $\rightarrow$ Binary $\rightarrow$ Octal.

Worked Example:

Octal to Hexadecimal Convert $(345)_8$ to Hexadecimal.
Solution: Step 1: Octal to Binary $3 \rightarrow 011$ $4 \rightarrow 100$ $5 \rightarrow 101$ Binary: $(011100101)_2$
Step 2: Binary to Hexadecimal Group into fours from right to left: 0000 1110 0101 $0 \rightarrow 0$, $1110 \rightarrow E$, $0101 \rightarrow 5$
Final Answer: $(345)_8 = (E5)_{16}$

1.3 Arithmetic Operations on Binary Numbers

Methodology:

Binary Addition and Subtraction **Addition Rules:** $0+0=0$ $0+1=1$ $1+0=1$ $1+1=0$ (carry 1) $1+1+1=1$ (carry 1)
Subtraction Rules: $0-0=0$ $1-0=1$ $1-1=0$ $0-1=1$ (borrow 1 from next higher bit)

Worked Example:

Binary Addition Add 1101_2 and 1011_2.
Solution:

$$\begin{array}{cccccl} & 1 & 1 & 1 & & \leftarrow \text{Carry} \\ & 1 & 1 & 0 & 1 & \\ + & 1 & 0 & 1 & 1 & \\ \hline 1 & 1 & 0 & 0 & 0 & \end{array}$$

Final Answer: $(11000)_2$

Worked Example:

Binary Subtraction Subtract 101_2 from 1100_2.
Solution:

$$\begin{array}{rcccc} & 0 & 10 & 1 & 10 \quad \leftarrow \text{Borrow} \\ & 1 & 1 & 0 & 0 \\ - & & 1 & 0 & 1 \\ \hline & 0 & 1 & 1 & 1 \end{array}$$

Final Answer: $(111)_2$

Methodology:

Binary Multiplication and Division **Multiplication Rules:** $0 \times 0 = 0 \quad 0 \times 1 = 0 \quad 1 \times 0 = 0 \quad 1 \times 1 = 1$. Multiply similarly to decimal multiplication.
Division Rules: Perform long division using binary subtraction.

Worked Example:

Binary Multiplication Multiply 1011_2 by 101_2.
Solution:

$$\begin{array}{r} 1011 \\ \times 101 \\ \hline 1011 \\ 0000\times \\ 1011\times\times \\ \hline 110111 \end{array}$$

Final Answer: $(110111)_2$

Worked Example:

Binary Division Divide 11110_2 by 110_2.
Solution:

$$\begin{array}{r|l} 110 & 11110 \\ & 110 \\ \hline & 00110 \\ & 00110 \\ \hline & 00000 \end{array}$$

Final Answer: Quotient is 101_2, Remainder is 0_2.

Methodology:

Subtraction using Ones Complement To compute $A - B$ using 1's complement:

1. Make the number of bits in A and B equal by padding zeros to the left.
2. Find the 1's complement of the subtrahend B (invert all bits: $0 \to 1$ and $1 \to 0$).
3. Add the 1's complement of B to A.
4. Check the end-around carry:
   - If there is a carry add 1 to the LSB of the result. The answer is positive.
   - If there is no carry the answer is negative. Take the 1's complement of the result and attach a negative sign.

Worked Example:

Ones Complement Subtraction with Carry Subtract 1010_2 from 1101_2 using 1's complement.
Solution: $A = 1101$, $B = 1010$. Step 1: 1's complement of $B = 0101$. Step 2: Add A and 1's complement of B:

$$\begin{array}{cccccl} & 1 & 1 & 0 & 1 & \\ + & 0 & 1 & 0 & 1 & \\ \hline \mathbf{1} & 0 & 0 & 1 & 0 & \leftarrow \text{Carry exists} \end{array}$$

Step 3: Add carry to the LSB.

$$\begin{array}{cccccl} & & 0 & 0 & 1 & 0 \\ + & & & & & 1 \\ \hline & & 0 & 0 & 1 & 1 \end{array}$$

Final Answer: $+0011_2$

Methodology:

Subtraction using Twos Complement To compute $A - B$ using 2's complement:

1. Make the number of bits in A and B equal.
2. Find the 2's complement of the subtrahend B (invert all bits and add 1 to the LSB).
3. Add the 2's complement of B to A.
4. Check the carry:
   - If there is a carry **discard** it. The answer is positive.
   - If there is no carry the answer is negative. Take the 2's complement of the result and attach a negative sign.

Worked Example:

Twos Complement Subtraction without Carry Subtract 1101_2 from 1000_2 using 2's complement.
Solution: $A = 1000$, $B = 1101$. Step 1: 2's complement of B. 1's complement is 0010. Add $1 \rightarrow 0011$. Step 2: Add A and 2's complement of B:

$$\begin{array}{cccccl} & 1 & 0 & 0 & 0 & \\ + & 0 & 0 & 1 & 1 & \\ \hline & 1 & 0 & 1 & 1 & \end{array}$$

Step 3: No carry is generated. Thus the result is negative. Step 4: Take the 2's complement of 1011. 1's complement is 0100. Add $1 \rightarrow 0101$.
Final Answer: -0101_2

1.4 Interpret the Binary Codes

Methodology:

Binary Coded Decimal Code BCD (Binary Coded Decimal) or 8421 code expresses each decimal digit as a 4-bit binary sequence.

1. **Decimal to BCD**: Replace each decimal digit with its 4-bit binary equivalent.
2. **BCD to Decimal**: Group the BCD bits into fours from right to left. Replace each 4-bit group with its decimal digit.

Note: BCD only uses binary sequences from 0000 to 1001 (0 to 9). Sequences 1010 to 1111 are invalid.

Worked Example:

Decimal to BCD Conversion Convert 849_{10} to BCD code.
Solution: Replace each decimal digit with its 4-bit binary value. $8 \rightarrow 1000$ $4 \rightarrow 0100$ $9 \rightarrow 1001$
Final Answer: $(1000\ 0100\ 1001)_{\text{BCD}}$

Methodology:

Excess 3 Code Excess-3 code is a non-weighted BCD code.

1. **Decimal to Excess-3**: Add 3 to each decimal digit and then convert each resulting sum into a 4-bit binary number.
2. **Excess-3 to Decimal**: Subtract 3 (0011) from each 4-bit group and replace with the equivalent decimal digit.

Worked Example:

Decimal to Excess 3 Conversion Convert 27_{10} to Excess-3 code.
Solution: Add 3 to each digit: $2 + 3 = 5$ $7 + 3 = 10$
Convert each sum to a 4-bit binary code: $5 \rightarrow 0101$ $10 \rightarrow 1010$
Final Answer: $(0101\ 1010)_{\text{Excess-3}}$

Methodology:

Gray Code Gray code is an unweighted code where successive numbers differ by only one bit.

1. **Binary to Gray Code**:
   - The MSB of the Gray code is identical to the MSB of the binary number.
   - Each subsequent Gray code bit is obtained by XORing the current binary bit with the previous binary bit.
2. **Gray Code to Binary**:
   - The MSB of the binary number is identical to the MSB of the Gray code.
   - Each subsequent binary bit is obtained by XORing the current Gray code bit with the previously obtained binary bit.

Worked Example:

Binary to Gray Code Conversion Convert Binary 1011 to Gray code.
Solution: Let binary number be $B_3B_2B_1B_0 = 1011$. Gray code $G_3G_2G_1G_0$: $G_3 = B_3 = 1$ $G_2 = B_3 \oplus B_2 = 1 \oplus 0 = 1$ $G_1 = B_2 \oplus B_1 = 0 \oplus 1 = 1$ $G_0 = B_1 \oplus B_0 = 1 \oplus 1 = 0$
Final Answer: Gray code is 1110.

Worked Example:

Gray Code to Binary Conversion Convert Gray code 1101 to Binary.
Solution: Let Gray code be $G_3G_2G_1G_0 = 1101$. Binary number $B_3B_2B_1B_0$: $B_3 = G_3 = 1$ $B_2 = B_3 \oplus G_2 = 1 \oplus 1 = 0$ $B_1 = B_2 \oplus G_1 = 0 \oplus 0 = 0$ $B_0 = B_1 \oplus G_0 = 0 \oplus 1 = 1$
Final Answer: Binary number is 1001.

CHAPTER 2

Boolean Algebra and Logic Gates

2.1 Functions of Logic Gates

Methodology:

Basic Logic Gates and Operations Logic gates are the basic building blocks of any digital system. They operate on one or more binary inputs to produce a single binary output.

1. **AND Gate**: Output is 1 only if all inputs are 1. Expression: $Y = A \cdot B$
2. **OR Gate**: Output is 1 if any input is 1. Expression: $Y = A + B$
3. **NOT Gate (Inverter)**: Output is the complement of the input. Expression: $Y = \bar{A}$
4. **NAND Gate**: Complement of AND. Output is 0 only if all inputs are 1. Expression: $Y = \overline{A \cdot B}$
5. **NOR Gate**: Complement of OR. Output is 0 if any input is 1. Expression: $Y = \overline{A + B}$
6. **EX-OR (Exclusive-OR) Gate**: Output is 1 if inputs are different. Expression: $Y = A \oplus B = A\bar{B} + \bar{A}B$
7. **EX-NOR (Exclusive-NOR) Gate**: Output is 1 if inputs are the same. Expression: $Y = A \odot B = AB + \bar{A}\bar{B}$

Worked Example:

Determining Truth Table from Logic Expression Determine the truth table for the Boolean function $Y = A\bar{B} + C$.

Step 1: Identify the number of inputs. There are 3 variables (A, B and C) so the truth table will have $2^3 = 8$ rows.

Step 2: Construct the table and compute intermediate terms. Compute $\bar{B}$, then $A\bar{B}$, and finally $Y = A\bar{B} + C$.

A	B	C	$\bar{B}$	$A\bar{B}$	$Y = A\bar{B} + C$
0	0	0	1	0	0
0	0	1	1	0	1
0	1	0	0	0	0
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	0	0	0
1	1	1	0	0	1

2.2 Logic Operations and Theorems of Boolean Algebra

Methodology:

Boolean Algebra Theorems and Laws Boolean algebra uses laws and theorems to manipulate logic expressions.

1. **Identity Law**: $A + 0 = A$, $A \cdot 1 = A$
2. **Null Law**: $A + 1 = 1$, $A \cdot 0 = 0$
3. **Idempotent Law**: $A + A = A$, $A \cdot A = A$
4. **Complement Law**: $A + \bar{A} = 1$, $A \cdot \bar{A} = 0$
5. **Involution Law**: $\overline{\bar{A}} = A$
6. **Commutative Law**: $A + B = B + A$, $A \cdot B = B \cdot A$
7. **Associative Law**: $(A + B) + C = A + (B + C)$, $(A \cdot B) \cdot C = A \cdot (B \cdot C)$
8. **Distributive Law**: $A \cdot (B + C) = A \cdot B + A \cdot C$, $A + (B \cdot C) = (A + B) \cdot (A + C)$
9. **Absorption Law**: $A + A \cdot B = A$, $A \cdot (A + B) = A$
10. **De Morgans Theorems**:
    - Theorem 1: $\overline{A + B} = \bar{A} \cdot \bar{B}$
    - Theorem 2: $\overline{A \cdot B} = \bar{A} + \bar{B}$

Worked Example:

Proving Boolean Identities Prove that $A + \bar{A}B = A + B$.
Step 1: Apply the Distributive Law. Using the rule $X + YZ = (X + Y)(X + Z)$:

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

Step 2: Apply the Complement Law. Since $A + \bar{A} = 1$:

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

Step 3: Apply the Identity Law.

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

Thus $A + \bar{A}B = A + B$ is proved.

Worked Example:

Applying De Morgans Theorems Find the complement of the function $F = (A + \bar{B})(C + D)$ and simplify.
Step 1: Take the complement of the entire function.

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

Step 2: Apply De Morgans Theorem for products ($\overline{XY} = \bar{X} + \bar{Y}$).

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

Step 3: Apply De Morgans Theorem for sums ($\overline{X + Y} = \bar{X} \cdot \bar{Y}$).

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

Step 4: Apply Involution Law ($\overline{\bar{B}} = B$).

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

2.3 Boolean Functions and Implementation using Logic Gates

Methodology:

Implementing Boolean Functions To implement a given Boolean function using logic gates:

1. Simplify the Boolean expression if required.
2. Identify the basic logic operations (AND OR NOT) required.
3. Draw input lines for each variable and its complement if needed.
4. Connect inputs to AND and OR gates according to the expression's terms.
5. Combine the outputs of intermediate gates to obtain the final output.

Worked Example:

Implementing with Basic Gates Implement the Boolean function $F = A\bar{B} + \bar{A}B$ using basic logic gates.
Step 1: Identify operations. The function requires two AND operations ($A \cdot \bar{B}$ and $\bar{A} \cdot B$) one OR operation to add them and NOT operations to get $\bar{A}$ and $\bar{B}$.
Step 2: List the logic gates needed.

- Two NOT gates (for $\bar{A}$ and $\bar{B}$)
- Two 2-input AND gates
- One 2-input OR gate

Step 3: Connect gates.

- Connect A directly to one input of the first AND gate and B through a NOT gate to the other input. Output is $A\bar{B}$.
- Connect A through a NOT gate to one input of the second AND gate and B directly to the other. Output is $\bar{A}B$.
- Connect the outputs of both AND gates to the inputs of the OR gate.
- Final output is $F = A\bar{B} + \bar{A}B$.

Methodology:

Universal Gates Implementation NAND and NOR gates are universal gates because any boolean function can be implemented using only NAND or only NOR gates. To convert an AND-OR logic circuit to NAND-NAND logic:

1. Implement the function in Sum of Products (SOP) form using AND and OR gates.
2. Add inversions at the outputs of AND gates and inputs of OR gates.
3. Ensure that any added inversion is compensated by another inversion.
4. Replace the bubbled-OR gates with NAND gates.

Worked Example:

Implementing using only NAND Gates Implement $Y = AB + CD$ using only NAND gates.
Step 1: Write function in SOP form. $Y = AB + CD$. This requires two AND gates and one OR gate.

Step 2: Apply double complement.

$$Y = \overline{\overline{AB + CD}}$$

Step 3: Apply De Morgans theorem to the inner complement.

$$Y = \overline{\overline{AB} \cdot \overline{CD}}$$

Step 4: Map to NAND gates.

- Gate 1: A 2-input NAND gate for $\overline{AB}$ with inputs A and B.
- Gate 2: A 2-input NAND gate for $\overline{CD}$ with inputs C and D.
- Gate 3: A 2-input NAND gate taking outputs of Gate 1 and Gate 2 as inputs to produce Y.

2.4 Simplification of Boolean Functions

Methodology:

Algebraic Simplification To simplify expressions algebraically repeatedly apply Boolean laws to reduce the number of terms and variables.

1. Expand expressions using Distributive laws.
2. Group common terms and factor them out.
3. Apply laws like $A + \bar{A} = 1$ or $A + 1 = 1$ to eliminate variables.
4. Apply Absorption laws ($A + AB = A$) to remove redundant terms.

Worked Example:

Simplifying using Algebraic Methods Simplify the Boolean function $Z = \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}C + ABC$.
Step 1: Group terms with common variables. Group the first two terms and the last two terms:

$$Z = \bar{A}C(\bar{B} + B) + AC(\bar{B} + B)$$

Step 2: Apply Complement Law ($X + \bar{X} = 1$).

$$Z = \bar{A}C(1) + AC(1) = \bar{A}C + AC$$

Step 3: Factor out the common term C.

$$Z = C(\bar{A} + A)$$

Step 4: Apply Complement Law again.

$$Z = C(1) = C$$

The simplified expression is $Z = C$.

Methodology:

Simplification using Karnaugh Maps A Karnaugh Map (K-map) is a graphical tool for simplifying Boolean expressions systematically.

1. Construct a grid of squares where the number of squares is 2^n (n is number of variables).
2. Label the rows and columns using Gray code (00, 01, 11, 10).

3. Plot 1s in the squares corresponding to the minterms of the Sum of Products (SOP) expression.
4. Group adjacent 1s in powers of 2 (octets quads pairs). Groups can wrap around the edges.
5. For each group write down the variables that remain constant within the group.
6. Sum the terms found in step 5 to get the simplified SOP expression.

Worked Example:

Simplifying a 3 Variable K-map Simplify the Boolean function given by minterms: $F(A, B, C) = \sum m(0, 1, 2, 4, 5, 6)$.
Step 1: Construct a 3-variable K-map (8 cells). Columns: BC (00, 01, 11, 10). Rows: A (0, 1).
Step 2: Plot the 1s for minterms 0 1 2 4 5 6.

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

Step 3: Make groups of adjacent 1s.

- Group 1 (Quad): Cells (0, 1, 4, 5). In these cells C changes (0 to 1) A changes (0 to 1) but B is constant at 0. So term is $\bar{B}$.
- Group 2 (Quad): Cells (0, 2, 4, 6) mapping edges. In these cells B changes (0 to 1) A changes (0 to 1) but C is constant at 0. So term is $\bar{C}$.

Step 4: Sum the terms. The simplified function is $F = \bar{B} + \bar{C}$.

Worked Example:

Simplifying a 4 Variable K-map Simplify the Boolean function $F(A, B, C, D) = \sum m(1, 3, 5, 7, 9, 11, 13, 15)$.
Step 1: Construct a 4-variable K-map (16 cells). Columns: CD (00, 01, 11, 10). Rows: AB (00, 01, 11, 10).
Step 2: Plot 1s for the given minterms.

$AB\backslash CD$	00	01	11	10
00	0	1 (1)	1 (3)	0
01	0	1 (5)	1 (7)	0
11	0	1 (13)	1 (15)	0
10	0	1 (9)	1 (11)	0

Step 3: Make groups of adjacent 1s. All eight 1s can be grouped together into a single Octet (8 cells). The octet covers all rows (AB change completely) so A and B are eliminated. In columns 01 and 11 C changes from 0 to 1 but D is constant at 1.
Step 4: Determine the simplified term. The only variable that does not change is D (it is 1). Thus $F = D$.

Worked Example:

Simplifying K-map with Dont Cares Simplify the function $F(A, B, C, D) = \sum m(0, 1, 2, 8, 9) + d(10, 11)$ where d indicates don't care conditions.
Step 1: Plot 1s for minterms and X for don't cares.

$AB\backslash CD$	00	01	11	10
00	1 (0)	1 (1)	0	1 (2)
01	0	0	0	0
11	0	0	0	0
10	1 (8)	1 (9)	X (11)	X (10)

Step 2: Group the 1s, using Xs if they help make larger groups. Do not group Xs if they don't help cover uncovered 1s.

- Group 1 (Quad): Cells (0, 1, 8, 9) wrapping top-to-bottom. Rows 00 and 10 (B is constant at 0). Columns 00 and 01 (C is constant at 0). Term is $\bar{B}\bar{C}$. This covers 1s at 0, 1, 8, 9.
- Group 2 (Quad): Cells (0, 2, 8, 10) wrapping all four corners. Use X at 10 to help form a quad to cover the 1 at cell 2. Rows 00 and 10 (B is constant at 0). Columns 00 and 10 (D is constant at 0). Term is $\bar{B}\bar{D}$.

Note that we do not need to group the X at cell 11 and we do not form a group of (8, 9, 10, 11) because the 1s at 8 and 9 are already covered by Group 1.
Step 3: Sum the essential terms. $F = \bar{B}\bar{C} + \bar{B}\bar{D}$

CHAPTER 3

Combinational Logic Circuits

3.1 Function and Design of Combinational Circuits

Combinational logic circuits are digital circuits whose outputs at any given time are determined solely by the current inputs. There is no memory or feedback involved in these circuits.

Methodology:

Combinational Logic Design Procedure The systematic design of a combinational logic circuit involves the following computational steps:

1. **Understand the Problem**: Identify the number of input variables and output variables from the problem specification.
2. **Assign Variables**: Assign letter symbols to the input and output variables.
3. **Truth Table**: Construct a truth table that defines the required relationship between the inputs and outputs.
4. **Boolean Expression**: Obtain the simplified Boolean expression for each output using Boolean algebra or Karnaugh Maps (K-Maps).
5. **Logic Diagram**: Draw the logic diagram using standard logic gates.

Worked Example:

Design a Half Adder Circuit Design a combinational circuit that performs the arithmetic addition of two 1-bit numbers.
Step 1: Identify Inputs and Outputs Two 1-bit inputs are required. The addition of two bits can result in a maximum of '10' (decimal 2) which requires two output bits: a Sum and a Carry.

- Inputs: A and B
- Outputs: Sum (S) and Carry (C)

Step 2: Construct the Truth Table

Inputs		**Outputs**	
A	B	Carry (C)	Sum (S)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Step 3: Derive Simplified Boolean Expressions From the truth table: Sum S is 1 when inputs are different. $S = \bar{A}B + A\bar{B} = A \oplus B$ (XOR operation)
Carry C is 1 only when both inputs are 1. $C = AB$ (AND operation)
Step 4: Logic Implementation The circuit requires one XOR gate for the Sum and one AND gate for the Carry.

3.2 Implementation of Arithmetic Combinational Circuits

Methodology:

Full Adder Logic Implementation A full adder adds three 1-bit numbers (two significant bits and a previous carry) and outputs a Sum and a Carry.

1. Define inputs A, B and C_{in}.
2. Define outputs Sum (S) and Carry out (C_{out}).
3. Compute Sum: $S = A \oplus B \oplus C_{in}$
4. Compute Carry: $C_{out} = AB + BC_{in} + AC_{in}$

Worked Example:

Design a Full Adder Circuit Implement a full adder using K-Maps to derive the simplified Boolean expressions.
Step 1: Truth Table

A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Step 2: K-Map for Sum (S) The minterms for S are m_1, m_2, m_4 and m_7. No grouping is possible.
$S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C_{in}} + A\bar{B}\bar{C_{in}} + ABC_{in}$
$S = C_{in}(\bar{A}\bar{B} + AB) + \bar{C_{in}}(\bar{A}B + A\bar{B})$
$S = C_{in}\overline{(A \oplus B)} + \bar{C_{in}}(A \oplus B)$
$S = A \oplus B \oplus C_{in}$
Step 3: K-Map for Carry (C_{out}) The minterms for C_{out} are m_3, m_5, m_6 and m_7. Grouping pairs yields:
$C_{out} = AB + BC_{in} + AC_{in}$

3.3 Multiplexers and Decoders

Methodology:

Multiplexer Circuit Design A Multiplexer (MUX) is a data selector that routes one of many input signals to a single output.

1. Determine the number of select lines: For 2^n input lines, exactly n selection lines are required.
2. Formulate the Boolean function: The output is determined by $Y = \sum(I_k \cdot m_k)$ where I_k is the data input and m_k is the corresponding minterm of the selection variables.

Worked Example:

Implement a Boolean Function using a MUX Implement the Boolean function $F(ABC) = \Sigma m(1, 3, 5, 6)$ using an 8-to-1 Multiplexer.
Step 1: Identify MUX Specifications The function has 3 variables (A, B, C). An 8-to-1 MUX has 3 select lines (S_2, S_1, S_0) and 8 data inputs (D_0 to D_7). Let $S_2 = A$, $S_1 = B$, and $S_0 = C$.

Step 2: Assign Data Inputs The function output is 1 for minterms 1, 3, 5 and 6. Therefore, tie the corresponding data inputs to logic 1 (High). Tie the remaining inputs to logic 0 (Low).

- Inputs $D_1 = 1$, $D_3 = 1$, $D_5 = 1$, $D_6 = 1$
- Inputs $D_0 = 0$, $D_2 = 0$, $D_4 = 0$, $D_7 = 0$

Step 3: Verification When $ABC = 101$ (minterm 5), the MUX selects D_5. Since D_5 is hardwired to 1, the output $Y = 1$. This perfectly mirrors the required Boolean function.

3.4 Code Converter Circuits

Methodology:

Binary to Gray Code Conversion Steps Gray code is an unweighted code where successive values differ by only one bit. To convert an n-bit binary number $B_{n-1}B_{n-2}...B_0$ to Gray code $G_{n-1}G_{n-2}...G_0$:

1. The most significant bit (MSB) of the Gray code equals the MSB of the binary number: $G_{n-1} = B_{n-1}$.
2. Each remaining Gray code bit is the XOR sum of the corresponding binary bit and the adjacent higher-order binary bit: $G_i = B_i \oplus B_{i+1}$ (for $i = 0$ to $n - 2$).

Worked Example:

Design a 4-bit Binary to Gray Converter Design a combinational logic circuit to convert a 4-bit binary number ($B_3B_2B_1B_0$) to a 4-bit Gray code ($G_3G_2G_1G_0$).
Step 1: Apply the Conversion Algorithm Using the conversion rules:

- $G_3 = B_3$ (MSB remains the same)
- $G_2 = B_3 \oplus B_2$
- $G_1 = B_2 \oplus B_1$
- $G_0 = B_1 \oplus B_0$

Step 2: Construct the Truth Table to Verify

Binary Input				Gray Code Output			
B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	1	0	0	0	1	1	0
1	0	0	0	1	1	0	0

Step 3: Circuit Implementation The logic circuit is formed by feeding the binary inputs into three 2-input XOR gates as governed by the derived Boolean equations.

Methodology:

BCD to Excess 3 Conversion Method Binary Coded Decimal (BCD) represents decimal digits 0-9 using 4-bit binary. Excess-3 code is derived by adding 3 (binary 0011) to each BCD digit.

1. List the truth table with 4 BCD input bits (A, B, C, D) and 4 Excess-3 output bits (W, X, Y, Z).
2. Inputs representing decimal 10 to 15 are invalid BCD codes and are treated as "Don't Care" (X) conditions.

3. Solve K-Maps for each output variable W, X, Y, Z using the Don't Care states to simplify the equations.

Worked Example:

Design BCD to Excess 3 Converter Find the simplified Boolean expressions for a BCD to Excess-3 code converter.

Step 1: Truth Table Definition

BCD Input				**Excess-3 Output**			
A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

Note: Inputs 1010 to 1111 are Don't Cares (X).

Step 2: K-Map Simplification Using K-maps and including the don't cares for optimization yields:

- $Z = \bar{D}$
- $Y = CD + \bar{C}\bar{D} = \overline{C \oplus D}$
- $X = \bar{B}C + \bar{B}D + B\bar{C}\bar{D} = \bar{B}(C + D) + B\overline{(C + D)} = B \oplus (C + D)$
- $W = A + BC + BD = A + B(C + D)$

CHAPTER 4

Sequential Logic Circuits

Sequential logic circuits differ from combinational circuits because their outputs depend not only on the present inputs but also on the past inputs (i.e., the present state). This requires memory elements, primarily flip-flops, which are driven by clock signals. This chapter focuses on the computational methods for analyzing and designing flip-flops, shift registers, and counters.

4.1 Flip-Flops and Their Conversions

Flip-flops are the basic memory elements in digital systems. Each type has a specific truth table (which defines the next state based on current inputs and current state) and an excitation table (which defines the required inputs to achieve a specific state transition).

Methodology:

Flip-Flop Characteristic and Excitation Tables Understanding the characteristic equations and excitation tables is the foundational step for all sequential circuit design. Let Q_n be the present state and Q_{n+1} be the next state.

1. SR (Set-Reset) Flip-Flop

- **Characteristic Equation:** $Q_{n+1} = S + R'Q_n$ (with condition $SR = 0$)
- **Excitation Table:**

Q_n	Q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

2. JK Flip-Flop

- **Characteristic Equation:** $Q_{n+1} = JQ'_n + K'Q_n$
- **Excitation Table:**

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

3. D (Data) Flip-Flop

- **Characteristic Equation:** $Q_{n+1} = D$
- **Excitation Table:**

Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

4. T (Toggle) Flip-Flop

- **Characteristic Equation:** $Q_{n+1} = T \oplus Q_n$
- **Excitation Table:**

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Worked Example:

Conversion of SR Flip-Flop to JK Flip-Flop **Problem:** Design a JK flip-flop using an SR flip-flop and basic logic gates.
Solution: Step 1: Create the Conversion Table. Write the truth table for the target flip-flop (JK) and append the required inputs for the available flip-flop (SR) using its excitation table.

Target (JK) Inputs/State			Next State	Required (SR) Inputs	
J	K	Q_n	Q_{n+1}	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

Step 2: Simplify Boolean Expressions using K-maps. Extract minterms for S and R in terms of variables J, K, and Q_n.

- For S: Minterms are at indices 4, 6. Don't cares (X) are at 1, 5. Solving the K-map gives: $S = JQ_n'$
- For R: Minterms are at indices 3, 7. Don't cares (X) are at 0, 2. Solving the K-map gives: $R = KQ_n$

Step 3: Implementation. To convert an SR flip-flop into a JK flip-flop, connect the inputs as follows: Drive the S terminal with an AND gate taking J and the inverted output $\bar{Q}$. Drive the R terminal with an AND gate taking K and the non-inverted output Q.

4.2 Shift Registers

A shift register consists of a cascade of flip-flops sharing the same clock, wherein the output of one flip-flop is connected to the data input of the next. They are primarily used for data storage and data transfer (Serial-In Serial-Out, Serial-In Parallel-Out, etc.).

Methodology:

Shift Register State Transitions To track the contents of an N-bit shift register over successive clock pulses, use the following algorithm:
Step 1: Identify Initial State. Write down the initial contents of the register's flip-flops, typically from Q_{N-1} (Most Significant Bit) down to Q_0 (Least Significant Bit).
Step 2: Apply Shift Operation per Clock Pulse.

- **For Right Shift:** Each bit shifts to the right ($Q_i \leftarrow Q_{i+1}$). The MSB (Q_{N-1}) receives the new input data bit, and the LSB (Q_0) is lost or outputted.
- **For Left Shift:** Each bit shifts to the left ($Q_{i+1} \leftarrow Q_i$). The LSB (Q_0) receives the new input data bit, and the MSB (Q_{N-1}) is lost or outputted.

Step 3: Record Next State. Update the state vector according to the chosen operation after each active clock edge.

Worked Example:

Four Bit Shift Register Data Sequence **Problem:** A 4-bit Serial-In Serial-Out (SISO) right-shift register is initially cleared (containing 0000). A data sequence of 1011 is applied serially to the input. Determine the state of the shift register after each of the first four clock pulses. (Assume the LSB of the input string enters first, so the input sequence in time order is 1, 1, 0, 1).
Solution: Let the flip-flops be Q_3, Q_2, Q_1, Q_0. The serial input is fed into Q_3. Input sequence (Time 1 to Time 4): $D_{in} = 1, 1, 0, 1$.
Initial State (Pulse 0): $Q_3 = 0, Q_2 = 0, Q_1 = 0, Q_0 = 0$
After Pulse 1: $D_{in} = 1$ is applied. Shift bits right. $Q_3 \leftarrow 1$ $Q_2 \leftarrow Q_3$ (which was 0) $Q_1 \leftarrow Q_2$ (which was 0) $Q_0 \leftarrow Q_1$ (which was 0) **State 1:** 1000
After Pulse 2: $D_{in} = 1$ is applied. $Q_3 \leftarrow 1$, $Q_2 \leftarrow 1$, $Q_1 \leftarrow 0$, $Q_0 \leftarrow 0$ **State 2:** 1100
After Pulse 3: $D_{in} = 0$ is applied. $Q_3 \leftarrow 0$, $Q_2 \leftarrow 1$, $Q_1 \leftarrow 1$, $Q_0 \leftarrow 0$ **State 3:** 0110
After Pulse 4: $D_{in} = 1$ is applied. $Q_3 \leftarrow 1$, $Q_2 \leftarrow 0$, $Q_1 \leftarrow 1$, $Q_0 \leftarrow 1$ **State 4:** 1011
The final state of the register is 1011.

4.3 Counters

Counters are sequential circuits that go through a prescribed sequence of states upon the application of input pulses. They can be broadly classified as Synchronous (all flip-flops clocked simultaneously) and Asynchronous (ripple counters, where clock propagates through flip-flops).

4.3.1 Synchronous Counters

Methodology:

Synchronous Counter Design Algorithm **Step 1: State Diagram Formulation.** Determine the number of flip-flops required (n) for N states, where $2^n \geq N$. Draw the state transition diagram indicating the sequence of counts.
Step 2: Excitation Table Construction. Create a table with columns for Present State, Next State, and the required Flip-Flop Inputs. Fill the required inputs by cross-referencing the state transitions with the chosen flip-flop's excitation table.
Step 3: Logic Simplification. Extract the Boolean function for each flip-flop input in terms of the Present State variables using Karnaugh Maps (K-maps).
Step 4: Circuit Implementation. Draw the logic diagram by connecting the common clock to all flip-flops and implementing the simplified boolean expressions using logic gates at the flip-flop inputs.

Worked Example:

Design of a Modulo 3 Synchronous Counter **Problem:** Design a Mod-3 synchronous UP counter using T flip-flops. The counter should count $00 \rightarrow 01 \rightarrow 10 \rightarrow 00$.
Solution: Step 1: States and Flip-flops. A Mod-3 counter requires 2 flip-flops ($2^2 = 4 \geq 3$). Let states be Q_1, Q_0. Sequence: $00 \rightarrow 01 \rightarrow 10 \rightarrow 00$. The state 11 is an unused/don't-care state.
Step 2: Excitation Table. Using the T flip-flop excitation table ($T = Q_n \oplus Q_{n+1}$):

Present State		Next State		T Inputs	
Q_1	Q_0	$Q_1(next)$	$Q_0(next)$	T_1	T_0
0	0	0	1	0	1
0	1	1	0	1	1
1	0	0	0	1	0
1	1	X	X	X	X

Step 3: Logic Simplification. For T_1: Minterm is at 1. Don't care at 3. K-map cells: (0,0)=0, (0,1)=1, (1,0)=1, (1,1)=X. From K-map, grouping (0,1) and (1,1) gives $T_1 = Q_0$. Grouping (1,0) and (1,1) gives $T_1 = Q_1$. Thus, $T_1 = Q_1 + Q_0$.
For T_0: Minterms are at 0, 1. Don't care at 3. K-map cells: (0,0)=1, (0,1)=1, (1,0)=0, (1,1)=X. From K-map, grouping (0,0) and (0,1) gives $T_0 = \overline{Q_1}$.
Step 4: Resulting Equations. $T_1 = Q_1 + Q_0$
$T_0 = \overline{Q_1}$
These equations dictate how the logic gates will be wired to the T inputs of the two flip-flops. Both flip-flops receive the same external clock signal.

4.3.2 Asynchronous (Ripple) Counters

In asynchronous counters, only the first flip-flop is driven by the external clock. Subsequent flip-flops are clocked by the output of the preceding flip-flop.

Methodology:

Asynchronous Modulo N Counter Design **Step 1: Determine Flip-Flops.** Calculate n such that $2^n \geq N$. Connect n flip-flops in a ripple configuration (typically using JK flip-flops set to toggle mode, $J = 1, K = 1$).
Step 2: Identify Reset Condition. Determine the binary representation of the state N. This state should not be visible in a Mod-N counter; it must immediately reset the counter to 0.
Step 3: Design the Reset Logic. Identify all flip-flops that output a logic '1' for state N. Connect the non-inverted outputs (Q) of these specific flip-flops to the inputs of a NAND gate.
Step 4: Connect Reset. Connect the output of the NAND gate to the active-low $\overline{CLR}$ (clear) inputs of all flip-flops in the counter.

Worked Example:

Design of a Modulo 6 Asynchronous Counter **Problem:** Design a Mod-6 asynchronous UP counter. Show the required logic for the clearing operation.
Solution: Step 1: Flip-Flops Required. We need n flip-flops such that $2^n \geq 6$. Thus, $n = 3$ flip-flops (Q_2, Q_1, Q_0) are required. They are connected in ripple fashion: clock to Q_0, Q_0 to clock of Q_1, Q_1 to clock of Q_2.
Step 2: Reset Condition. The Mod-6 counter counts from 0 (000) to 5 (101). The counter must reset as soon as it attempts to enter state 6. State 6 in binary is $Q_2 = 1, Q_1 = 1, Q_0 = 0$.
Step 3: Reset Logic. In state 6, the flip-flops with logic '1' are Q_2 and Q_1. Therefore, a NAND gate is required with inputs connected to Q_2 and Q_1. The boolean expression for the NAND gate output is $Y = \overline{Q_2 \cdot Q_1}$.
Step 4: Operational Flow. When the count transitions from 5 (101) to 6 (110), the inputs to the NAND gate (Q_2, Q_1) both become HIGH. The NAND gate output drops to LOW (0). Because this output is connected to the active-low $\overline{CLR}$ pins of all three flip-flops, the entire register is asynchronously forced back

to 000 in a matter of nanoseconds.

CHAPTER 5

Introduction to Memories and Logic Families

5.1 Semiconductor Memories: RAM and ROM

Semiconductor memories are digital circuits designed to store data. They are classified into Random Access Memory (RAM) for volatile read/write operations and Read-Only Memory (ROM) for non-volatile storage.

5.1.1 Types of RAM

- **SRAM (Static RAM):** Uses flip-flops to store each bit. Faster, consumes more power, and does not require periodic refreshing.
- **DRAM (Dynamic RAM):** Uses a capacitor and a transistor for each bit. Slower, highly dense, and requires continuous refreshing to prevent data loss.

5.1.2 Types of ROM

- **PROM (Programmable ROM):** Can be programmed only once by the user by blowing internal fuses.
- **EPROM (Erasable PROM):** Can be erased by exposing the chip to Ultraviolet (UV) light for a specific duration, allowing reprogramming.
- **EEPROM (Electrically Erasable PROM):** Can be erased and reprogrammed electrically block-by-block or byte-by-byte without removing it from the circuit.

Methodology:

Memory Capacity and Address Line Calculation The capacity of a memory chip dictates how many data bits it can store and is typically given as $W \times B$, where W is the number of words (addressable locations) and B is the number of bits per word.

1. **Identify memory size:** Let the memory capacity be expressed as $2^N \times M$.
2. **Determine Address Lines (N):** The number of address lines required to access 2^N unique memory locations is precisely N.
3. **Determine Data Lines (M):** The number of bits in each memory word dictates the number of data lines M.
4. **Total Capacity:** Total storage in bits is the product of words and bits per word ($Capacity = 2^N \times M$). To convert to bytes divide by 8.

Worked Example:

Calculate Address and Data Lines for a Memory Chip A digital system requires a RAM chip with a specified capacity of 16K × 8. Determine the number of address lines data lines and the total capacity in bytes.
Step 1: Determine Data Lines
The format given is Words × Bits per Word.

Data lines (M) = 8.
Step 2: Determine Address Lines
The number of words is 16K.
Since 1K = 1024 = 2^{10}, we have 16K = $16 \times 2^{10} = 2^4 \times 2^{10} = 2^{14}$.
The number of unique addressable locations is 2^{14}.
Therefore Address lines (N) = 14.
Step 3: Calculate Total Capacity
Total bits = $16,384 \times 8 = 131,072$ bits.
Since 1 byte = 8 bits the capacity is precisely 16KB.

5.2 Memory Expansion

Often available memory chips do not have the required word size or capacity. Multiple chips can be cascaded to expand either the word length or the number of words.

Methodology:

Expanding Memory Capacity and Word Size To build a target memory of size $W_T \times B_T$ using available chips of size $W_A \times B_A$:

1. **Word Size Expansion:** If $B_T > B_A$, connect multiple chips in parallel. Address lines and Chip Select (CS) lines are tied together. Data lines are split across chips. Number of chips needed = B_T/B_A.
2. **Capacity Expansion:** If $W_T > W_A$, connect chips in series logic. The lower address lines are tied together. Higher-order address lines are passed through a decoder to drive the Chip Select (CS) of each chip. Number of chips needed = W_T/W_A.
3. **Total Chips Required:** Compute total chips as $(W_T/W_A) \times (B_T/B_A)$.

Worked Example:

Design a 4KB Memory using 1KB Chips Design a 4K × 8 memory system using commercially available 1K × 8 memory chips. Determine the number of chips and the required decoder.
Step 1: Calculate Total Chips Required
Target Memory = 4K × 8
Available Chip = 1K × 8
Number of chips = $\frac{4K \times 8}{1K \times 8}$ = 4 chips.
Step 2: Determine Address Lines
For the 1K × 8 chip: 1K = 2^{10}, so 10 address lines (A_0 to A_9) are connected to all 4 chips in parallel.
For the target 4K × 8 memory: 4K = 2^{12}, requiring 12 address lines overall (A_0 to A_{11}).
Step 3: Decoder Design
The remaining higher-order address lines (A_{10} and A_{11}) must be used to select one of the 4 chips.
We use a 2-to-4 line decoder. A_{10} and A_{11} act as inputs to the decoder and the 4 outputs are connected to the Chip Select (CS) pins of Chip 0, Chip 1, Chip 2, and Chip 3 respectively.

5.3 Logic Family Definitions and Parameters

Digital logic families (such as TTL CMOS and ECL) are evaluated based on standard electrical characteristics.

Methodology:

Calculation of Logic Family Parameters

1. **Fan-in:** The maximum number of inputs a logic gate can handle without exceeding voltage/current specifications.

2. **Fan-out:** The maximum number of standard inputs of the same logic family that a gate output can drive safely. Calculated as the minimum of the High-state and Low-state fan-out:

$$\text{Fan-out} = \min\left(\frac{I_{OH}}{I_{IH}}, \frac{I_{OL}}{I_{IL}}\right)$$

where I_{OH} and I_{OL} are output currents and I_{IH} and I_{IL} are input currents.

3. **Noise Margin:** The maximum noise voltage that can be added to an input signal before causing an erroneous output state.

$$NM_{HIGH} = V_{OH(min)} - V_{IH(min)}$$
$$NM_{LOW} = V_{IL(max)} - V_{OL(max)}$$

4. **Propagation Delay (t_{pd}):** The average time required for a signal to transition through a gate.

$$t_{pd} = \frac{t_{phl} + t_{plh}}{2}$$

5. **Power Dissipation (P_D):** The power consumed by the gate, typically calculated as $V_{CC} \times I_{CC(avg)}$.

6. **Figure of Merit (FOM):** The Speed-Power Product (SPP) evaluates overall efficiency.

$$\text{FOM (Joules)} = t_{pd} \times P_D$$

Worked Example:

Calculate Fan out and Noise Margin for a TTL Gate A standard TTL logic gate has the following specifications: $I_{OH} = 400\mu$A, $I_{IH} = 40\mu$A, $I_{OL} = 16$mA, $I_{IL} = 1.6$mA.
$V_{OH} = 2.4$V, $V_{IH} = 2.0$V, $V_{OL} = 0.4$V, $V_{IL} = 0.8$V.
Calculate the Fan-out and the Noise Margins.
Step 1: Calculate Fan-out
High-state Fan-out: $\frac{I_{OH}}{I_{IH}} = \frac{400\mu\text{A}}{40\mu\text{A}} = 10$.
Low-state Fan-out: $\frac{I_{OL}}{I_{IL}} = \frac{16\text{mA}}{1.6\text{mA}} = 10$.
Overall Fan-out $= \min(10, 10) = 10$.
Step 2: Calculate Noise Margins
High-state Noise Margin (NM_{HIGH}): $V_{OH} - V_{IH} = 2.4\text{V} - 2.0\text{V} = 0.4\text{V}$.
Low-state Noise Margin (NM_{LOW}): $V_{IL} - V_{OL} = 0.8\text{V} - 0.4\text{V} = 0.4\text{V}$.
Overall Noise Margin $= 0.4$V.

Worked Example:

Calculate Figure of Merit for an IC A CMOS logic gate operates with an average propagation delay of 10ns and dissipates 2.5mW of power. Calculate its Figure of Merit (Speed-Power Product).
Step 1: Convert units to standard SI
Propagation Delay (t_{pd}) $= 10 \times 10^{-9}$s.
Power Dissipation (P_D) $= 2.5 \times 10^{-3}$W.
Step 2: Calculate Figure of Merit
FOM $= t_{pd} \times P_D$
FOM $= (10 \times 10^{-9}) \times (2.5 \times 10^{-3})$
FOM $= 25 \times 10^{-12}$J $= 25$pJ (picojoules).

5.4 Comparison of TTL CMOS and ECL

The most prevalent logic families differ mainly in the components used (Bipolar Junction Transistors vs MOSFETs) and their approach to saturation.

Parameter	TTL	CMOS	ECL
Basic Component	BJT (Saturated)	MOSFET	BJT (Non-saturated)
Speed (Delay)	Medium (10 ns)	Slow to Medium	Very Fast (1-2 ns)
Power Dissipation	Medium (10 mW)	Very Low (0.01 mW static)	High (40-50 mW)
Fan-out	Typically 10	Very High (50+)	About 25
Noise Margin	Poor (0.4V)	Excellent (approx 45% of V_{DD})	Poor (0.2V)
Packing Density	Low	Very High	Low

5.5 E-waste Management of ICs

Integrated circuits contain various hazardous heavy metals (lead, cadmium) and valuable precious metals (gold, silver, copper). Proper E-waste management focuses on safely dismantling, separating, and chemically recovering these elements.

Methodology:

Algorithm for IC E waste Recovery To computationally evaluate the economic viability and material yields of IC recycling:

1. **Total Batch Mass (M_{batch}):** Determine the total weight of discarded ICs collected.
2. **Fractional Composition (f_i):** Identify the known fraction of specific metals (e.g., Gold f_{Au}, Copper f_{Cu}) per unit mass of the ICs.
3. **Recovery Efficiency (η_i):** Standard hydrometallurgical or pyrometallurgical recovery methods are not 100% efficient. Note the efficiency constant η_i for each metal.
4. **Yield Calculation:** The recovered mass of metal i is calculated as:

$$M_{recovered\_i} = M_{batch} \times f_i \times \eta_i$$

5. **Value Assessment:** Multiply the recovered mass by the current market price per unit mass to determine gross recovery value.

Worked Example:

Estimate Gold Recovery from E waste ICs An E-waste recycling facility processes 500kg of discarded microprocessor ICs. The average gold content in this specific batch is 250g per ton of ICs (1ton = 1000kg). The chemical leaching process yields a recovery efficiency of 85%. Calculate the total mass of gold recovered.
Step 1: Identify Batch Mass and Composition
Batch mass (M_{batch}) = 500kg.
Fraction of gold (f_{Au}) = 250g/1000kg = 0.25g/kg.
Step 2: Calculate Theoretical Maximum Gold
Max Gold = $M_{batch} \times f_{Au}$ = 500kg × 0.25g/kg = 125g.
Step 3: Apply Recovery Efficiency
Efficiency (η_{Au}) = 0.85.
Recovered Mass = 125g × 0.85 = 106.25g.
The facility will successfully recover 106.25g of gold from the batch.

Formulae

Appendix: Key Formulae

This appendix provides a quick reference to the key formulae and mathematical relationships discussed in this book.

Number Systems

- **Polynomial Expansion (Binary to Decimal):**
  The decimal equivalent of a binary number is found by multiplying each bit by its positional weight and summing them:
  $$(N)_2 = \sum_{i=-m}^{n} b_i 2^i$$
  where b_i is the bit at position i.

Boolean Algebra

- **Identity Law:**
  $$A \cdot 1 = A \quad \text{and} \quad A + 0 = A$$
- **Complementarity Law:**
  $$A + \bar{A} = 1 \quad \text{and} \quad A \cdot \bar{A} = 0$$
- **Absorption Law:**
  $$A + A \cdot B = A \quad \text{and} \quad A \cdot (A + B) = A$$
- **Distributive Law:**
  $$A + (B \cdot C) = (A + B) \cdot (A + C)$$
- **De Morgan's Theorems:**
  $$\overline{A + B} = \bar{A} \cdot \bar{B}$$
  $$\overline{A \cdot B} = \bar{A} + \bar{B}$$

Combinational Logic

- **Binary to Gray Code Conversion:** The MSB of Gray code is the same as the MSB of Binary. For subsequent bits:
  $$G_i = B_i \oplus B_{i+1}$$
- **Multiplexer Output:** For a multiplexer with inputs I_k and select line minterms m_k:
  $$Y = \sum (I_k \cdot m_k)$$

Sequential Logic

- **SR Flip-Flop Constraint:**
$$S \cdot R = 0$$
- **JK Flip-Flop Characteristic Equation:**
$$Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n$$
- **T Flip-Flop Excitation Equation:**
$$T = Q_n \oplus Q_{n+1}$$

Logic Families

- **Noise Margin (LOW):**
$$NM_{LOW} = V_{IL(max)} - V_{OL(max)}$$
- **Noise Margin (HIGH):**
$$NM_{HIGH} = V_{OH(min)} - V_{IH(min)}$$
- **Fan-Out (HIGH State):**
$$FO_{HIGH} = \frac{I_{OH(max)}}{I_{IH(max)}}$$
- **Fan-Out (LOW State):**
$$FO_{LOW} = \frac{I_{OL(max)}}{I_{IL(max)}}$$

Memory and Digital Storage

- **Memory Capacity Calculation:** For a memory with N_A address lines and a word size of D bits, the total capacity is:
$$\text{Capacity} = 2^{N_A} \times D \text{ bits}$$
- **Common Units of Storage:**
$$\begin{aligned} 1\text{K (Kilo)} &= 1024 = 2^{10} \\ 1\text{M (Mega)} &= 1024\text{K} = 2^{20} \\ 1\text{G (Giga)} &= 1024\text{M} = 2^{30} \end{aligned}$$