

Digital System Design using Verilog

BEC302

CREDITS: 4

Pradeep Kumar Y
Assistant Professor
Department of ECE
ATME College of Engineering
Mysuru-570028



Course Objectives:

- 1. Apply** Boolean algebra principles to simplify logic functions using Karnaugh Maps (K-Maps) and the Quine–McCluskey minimization technique.
- 2. Analyze** the operation and behavior of combinational logic circuits and standard logic building blocks.
- 3. Design** combinational digital systems using logic gates, multiplexers, decoders, and arithmetic circuits.
- 4. Analyze and design** synchronous and asynchronous sequential circuits using SR, JK, D, and T flip-flops.
- 5. Model, simulate, and verify** combinational and sequential digital circuits using the Verilog Hardware Description Language (HDL).

Module 1: Principles of Combinational Logic

1. Definition of combinational logic
2. Canonical forms: SOP & POS
3. Switching equations from truth tables
4. Karnaugh Maps (up to 4 variables)
5. Quine-McCluskey Minimization Technique
6. Handling Don't Care terms in Quine-McCluskey

Module 2: Logic Design with MSI Components & PLDs

1. Binary Adders and Subtractors
2. Comparators
3. Decoders and Encoders
4. Multiplexers
5. Introduction to Programmable Logic Devices (PLDs)

Module 3: Flip-Flops and Applications

1. Master-Slave Flip-Flops (Pulse-Triggered)
2. SR and JK Flip-Flops
3. Characteristic Equations
4. Registers
5. Binary Ripple Counters
6. Synchronous Binary Counters
7. Shift Register-based Counters
8. Design of Synchronous Mod-n Counters using T, JK, D, and SR Flip-Flops

Module 4: Introduction to Verilog & Dataflow Modeling

1. Structure of a Verilog Module
2. Operators and Data Types
3. Styles of Description (Behavioral, Dataflow, Structural)
4. Highlights of Dataflow Modeling
5. Structure of Dataflow Description

Module 5: Behavioral & Structural Modeling in Verilog

Behavioral Modeling:

1. Structure
2. Variable Assignment
3. Sequential & Loop Statements
4. Behavioral Modeling of Multiplexers (2:1, 4:1, 8:1)

Structural Modeling:

1. Highlights & Organization
2. Structural Description of Ripple Carry Adder

IPCC Practical Component – Digital System Design using Verilog

1. Simplify the given Boolean expressions and realize using Verilog program
2. Realize Adder/Subtractor (Full/Half) circuits using Verilog dataflow description
3. Realize 4-bit ALU using Verilog program
4. Realize the following Code Converters using Verilog behavioral description:
 - a) Gray to Binary and vice versa
 - b) Binary to Excess-3 and vice versa
5. Realize using Verilog behavioral description:
 - a) 8:1 Multiplexer
 - b) 8:3 Encoder
 - c) Priority Encoder

IPCC Practical Component – Digital System Design using Verilog

6. Realize using Verilog behavioral description:

- a) 1:8 Demultiplexer
- b) 3:8 Decoder
- c) 2-bit Comparator

7. Realize Flip-Flops using Verilog behavioral description:

- a) JK type
- b) b) SR type
- c) c)T type
- d) d) D type

8. Realize Counters (Up/Down – BCD and Binary) using Verilog behavioral description

9. Verilog program to interface a stepper motor to FPGA/CPLD and rotate the motor in the specified direction (by N steps)

10 Verilog programs to interface switches and LEDs to FPGA/CPLD and demonstrate output

IPCC Practical Component – Digital System Design using Verilog

Open Ended Experiments

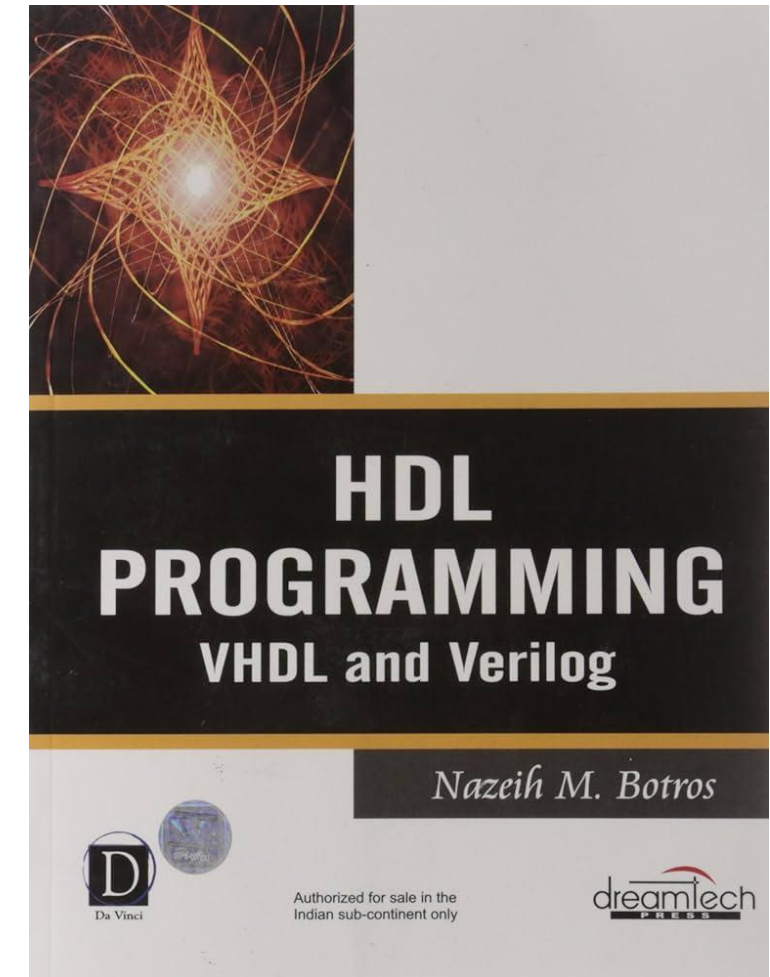
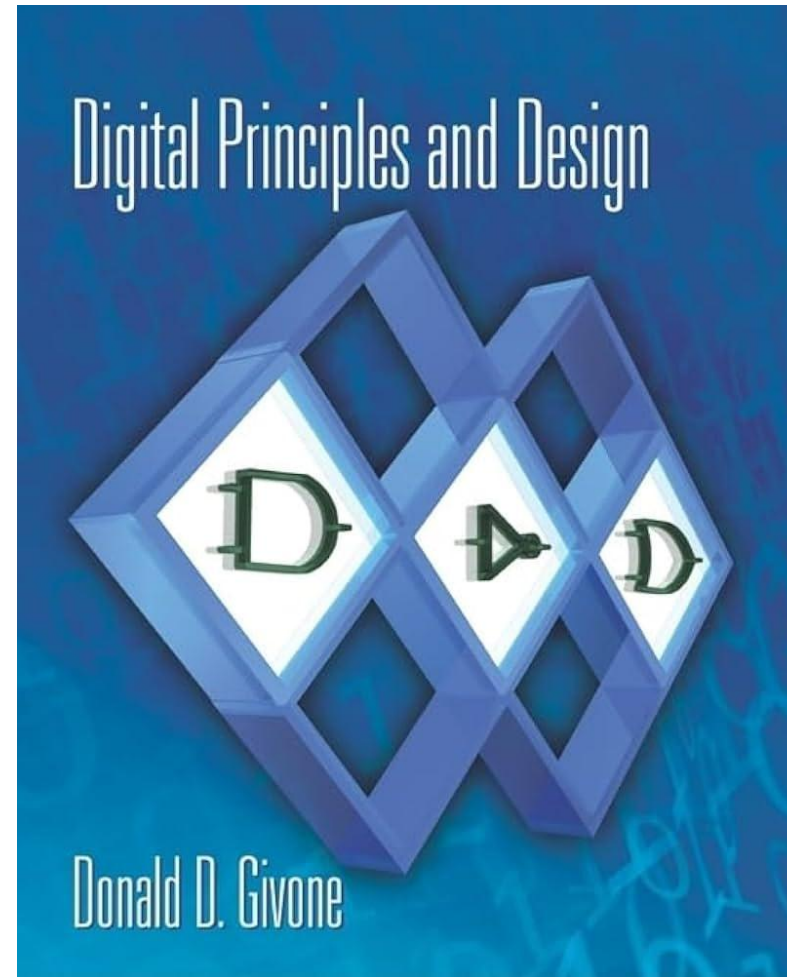
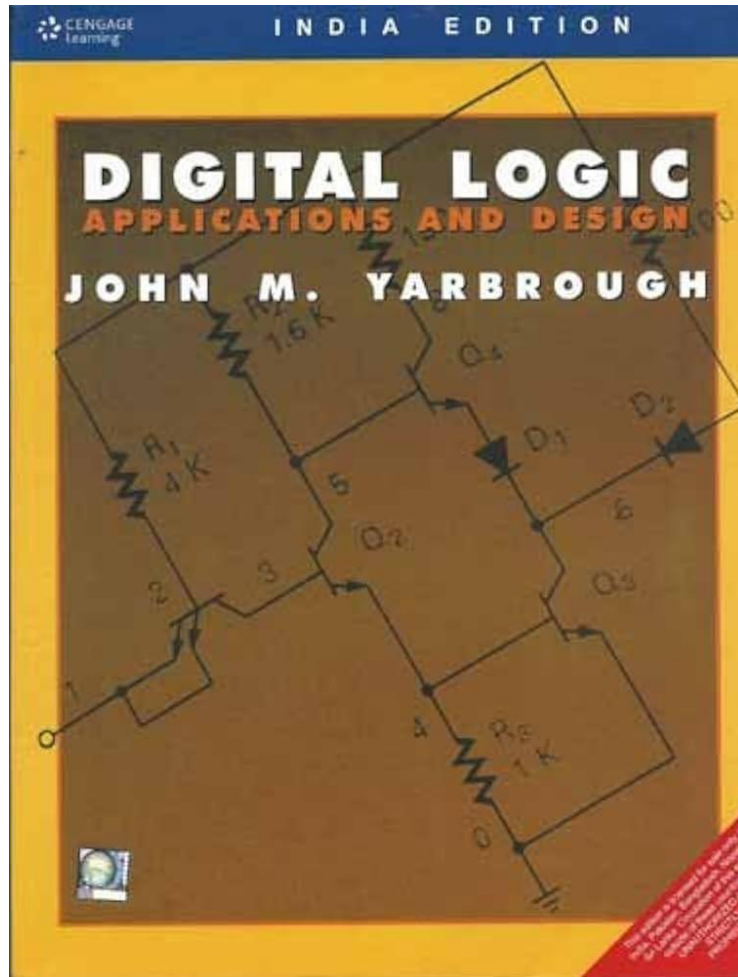
- 1 Design and Optimization of a Combinational Logic System
- 2 Configurable Arithmetic Logic Unit (ALU) Design
- 3 Design of a Finite State Machine (FSM) for a Real-Time Application
- 4 Sequential Circuit Design Using Flip-Flops
- 5 Memory-Based System Design Using Verilog Concept



Course Outcomes

- 1. Apply** Boolean algebra principles to simplify logic functions using Karnaugh Maps (K-Maps) and the Quine–McCluskey minimization technique..
- 2. Analyze** the operation and behavior of combinational logic circuits and standard logic building blocks.
- 3. Design** combinational digital systems using logic gates, multiplexers, decoders, and arithmetic circuits
- 4. Analyze and design** synchronous and asynchronous sequential circuits using SR, JK, D, and T flip-flops
- 5. Model, simulate, and verify** combinational and sequential digital circuits using the Verilog Hardware Description Language (HDL).

Learning Resources



Learners enrolled: 6804 | Exam registration: 1575**ABOUT THE COURSE:**

Digital Design is a fundamental course for developing large VLSI designs. This course helps student to understand the internal logic of various combinational units that is needed to develop large VLSI design. The course also introduces the sequential components, clocks and concepts of register transfer level design development process. In this course, we not only introduce the core concepts of digital design, we also introduce hardware description language Verilog. In each module, we will discuss how to implement all fundamental blocks in Verilog. Therefore, this course will help students to understand the internal details of fundamental blocks of digital circuits and also their implementation details.



NOC: Digital System Design, IIT Ropar

Prof. Neeraj Goel

[About Course](#)
[Downloads](#)
[TA List](#)
[Statistics](#)
[Toppers List](#)
[Certificate Type](#)
[Feedback](#)
[Syllabus](#)



Introduction - Digital System Design

Digital System Design

Watch on  YouTube



Watch later Share

Course Duration :
Jan-Apr 2025

Enrollment :
2024-11-15 to 2025-01-27

Exam Registration :
2024-12-16 to 2025-02-14

Exam Date :
2025-04-27

Course Abstract

Digital system design course focuses on design digital system from scratch. The course focuses on designing combinational and sequential building blocks, using these building blocks to design bigger digital systems. During this course we also learn how to use Verilog to design/model a digital system.

Module 1: Principles of Combinational Logic

1. Definition of combinational logic
2. Canonical forms: SOP & POS
3. Switching equations from truth tables
4. Karnaugh Maps (up to 4 variables)
5. Quine-McCluskey Minimization Technique
6. Handling Don't Care terms in Quine-McCluskey

INTRODUCTION

Definition of Digital System

- A **digital system** is a system that processes information represented in **binary form (0 and 1)**.
- Uses **digital circuits** built from logic gates.
- Performs computation, data processing, and control operations.
- Examples:
 - Computers
 - Calculators
 - Digital watches
 - Communication systems



INTRODUCTION

Types of Digital Systems

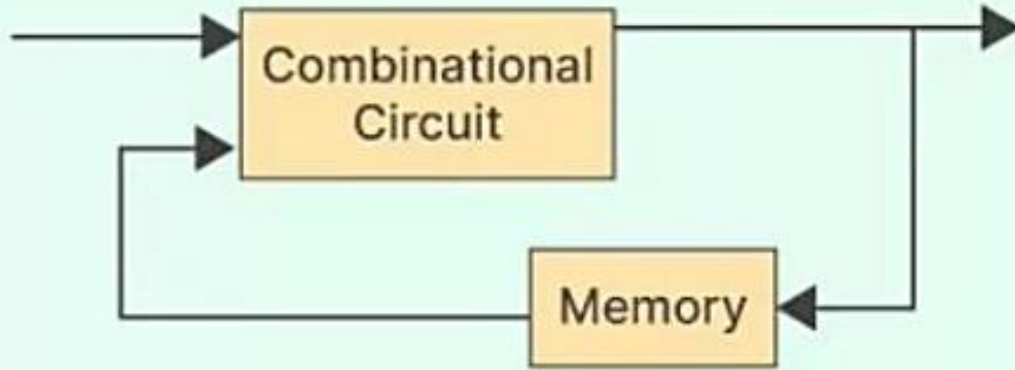
1. Combinational Systems

1. Output depends only on **present input values**.
2. No memory elements.
3. Examples: Adders, Decoders, Multiplexers.

2. Sequential Systems

1. Output depends on **present input + past states (memory)**.
2. Requires memory elements like flip-flops, registers.
3. Examples: Counters, Shift registers, State machines.

INTRODUCTION

Combinational Circuit	Sequential Circuit
Output only depends on the present input	Output depends on present input & past output
Memory element is absent	Memory element is present
No clock signal is applied	Clock signal is required
	
Example - Half Adder, Full Adder, Multiplexer	Example - Flipflop, Counters, Registers

1. Definition of combinational logic

- Combines basic logic gates to perform desired functions.
- Circuit **without feedback** from output to input.

Characteristics

- Output depends **only on input combinations**.
- Any input change ($0 \rightarrow 1$ or $1 \rightarrow 0$) immediately affects output.
- **No memory, no timing, no feedback loops.**
- Purely functional circuits.

Examples of Combinational Logic Circuits

Adders & Subtractors, Decoders & Encoders, Multiplexers (MUX), Display Drivers

1. Definition of combinational logic

Representations of Combinational Logic

☐ Boolean Algebra

Algebraic expression using logic variables.

Each variable is **True (1)** or **False (0)**.

☐ Truth Table

Tabular form.

Shows all possible input combinations and corresponding outputs.

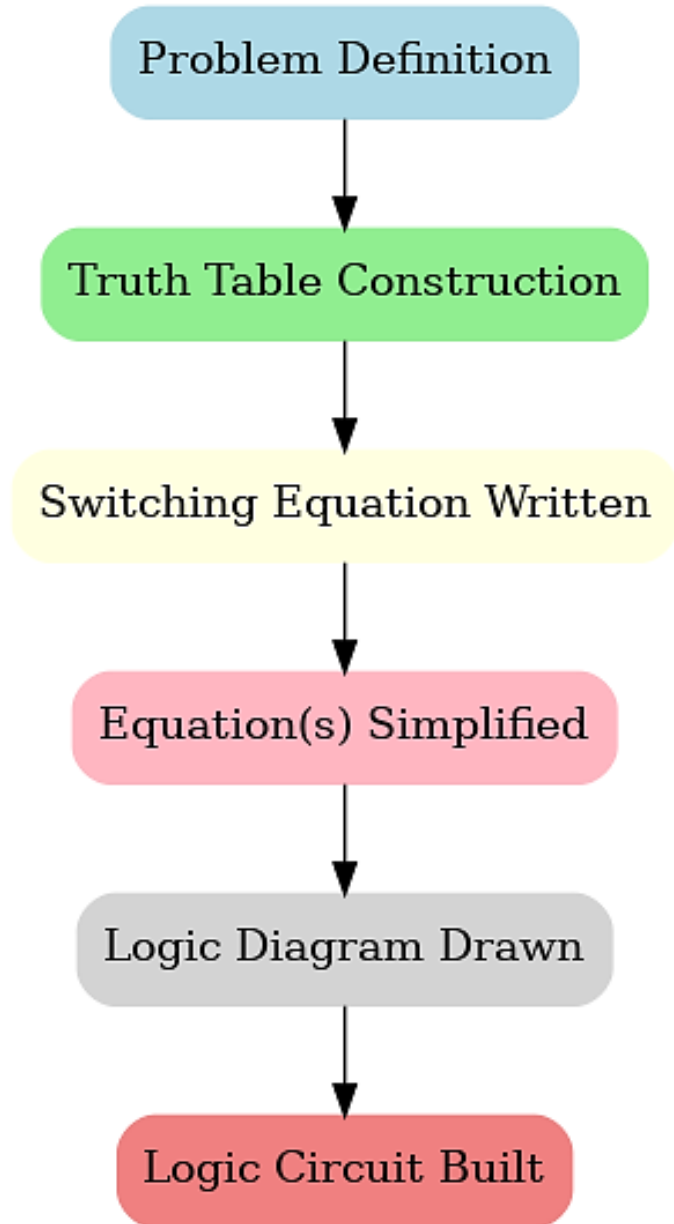
☐ Logic Diagram

Graphical circuit representation.

Uses standard symbols for gates and connections.

1. Definition of combinational logic

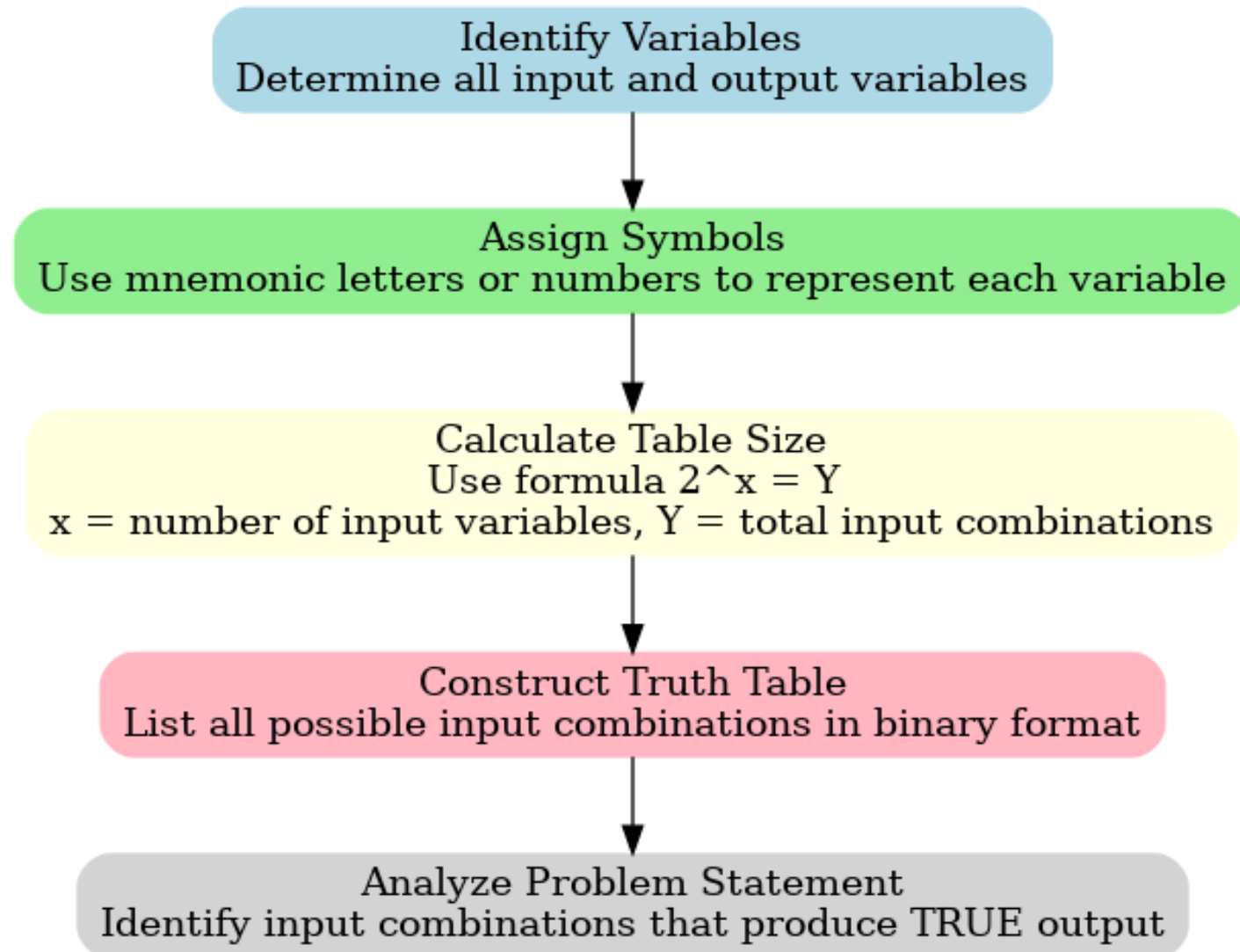
General Logic Design Sequence



1. Definition of combinational logic

Problem Statement to Truth Tables

Design a combinational logic truth table so that an output is generated indicating when a majority of 4 inputs are true.



1. Definition of combinational logic

Deriving Switching Equations

a	b	c	d	y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

1. Definition of combinational logic

1. Literal: A Boolean variable or its complement.

Example: For variable $X \rightarrow X$ and X' are literals

2. Product Term: A literal or logical AND of multiple literals.

Examples with variables X, Y, Z :

XY

XYZ'

$XY'Z$

3. Sum Term: A literal or logical OR of multiple literals.

Examples with variables X, Y, Z :

$X + Y$

$X + Y + Z'$

$X + Y' + Z$

1. Definition of combinational logic

4. Sum of Products (SOP)

Logical or of multiple **product terms**.

Each product term = AND of literals.

Example: $XYZ + XY' + XYZ'$

5. Product of Sums (POS)

Logical AND of multiple sum terms.

Each sum term = OR of literals.

Example: $(X+Y+Z)(X+Y')(X+Y+Z')$

1. Definition of combinational logic

4. Sum of Products (SOP)

Logical **OR** of multiple **product terms**.

Each product term = AND of literals.

Example: $XYZ + XY' + XYZ'$

5. Product of Sums (POS)

Logical **AND** of multiple **sum terms**.

Each sum term = OR of literals.

Example: $(X+Y+Z)(X+Y')(X+Y+Z')$

1. Definition of combinational logic

6. Minterm: Special case of product (AND) term.

Contains **all input variables** (each literal appears once).

Example for X, Y, Z: **$X'Y'Z$, $XY'Z'$**

7. Maxterm: Special case of sum (OR) term.

Contains **all input variables** (each literal appears once).

Example for X, Y, Z: **$(X + Y + Z)$, $(X + Y' + Z')$**

1. Definition of combinational logic

Minterm and Maxterm designation for three variables

a	b	c	Minterm (m)	Maxterm (M)
0	0	0	$m_0 = a'b'c'$	$M_0 = a + b + c$
0	0	1	$m_1 = a'b'c$	$M_1 = a + b + c'$
0	1	0	$m_2 = a'bc'$	$M_2 = a + b' + c$
0	1	1	$m_3 = a'bc$	$M_3 = a + b' + c'$
1	0	0	$m_4 = ab'c'$	$M_4 = a' + b + c$
1	0	1	$m_5 = ab'c$	$M_5 = a' + b + c'$
1	1	0	$m_6 = abc'$	$M_6 = a' + b' + c$
1	1	1	$m_7 = abc$	$M_7 = a' + b' + c'$

1. Definition of combinational logic

Problem Statement to Truth Tables

Design a combinational logic truth table so that an output is generated indicating when a majority of 4 inputs are true.

a	b	c	d	y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

2. Canonical form

- ❑ The word *canonical* refers to a “standard” or “rule-based” form.
- ❑ In switching equations, **canonical form** means that **each term must contain all the available input variables** (either in complemented or uncomplemented form).
- ❑ There are **two canonical forms** commonly used:
 1. **Sum of Minterms (SOP canonical form)**
 2. **Product of Maxterms (POS canonical form)**
- ❑ Canonical forms are usually not simplified. In fact, they are the *opposite* of simplification since they include redundancies.
- ❑ Provide a **unique standard form**, which helps in analysis, design, and truth table verification.
- ❑ Simplification is then applied later to reduce hardware cost.

Canonical SOP Form (Sum of Minterms)

- Steps to convert any SOP expression into canonical form:

1. Identify missing variable(s) in each product (AND) term.

Example: for variables x, y, z , the term xy is missing z .

2. Introduce the missing variable with its complement using the identity:

$$z + z' = 1$$

Therefore $xy = xy(z + z')$

3. Distribute to expand the term:

$$xy(z + z') = xyz + xyz'$$

4. Remove repeated minterms if any

Canonical POS Form (Product of Maxterms)

Steps to convert any POS expression into canonical form:

1. Identify missing variable(s) in each sum (OR) term.

Example: for variables x, y, z , the term $x + y$ is missing z .

2. Introduce the missing variable with its complement using the identity:

$$z \cdot z' = 0$$

$$\text{Therefore } (x + y) = (x + y + z)(x + y + z')$$

- Because OR with zero doesn't change the expression.

3. Expand by applying distribution if necessary.

4. Remove repeated Maxterms if any

a. $P = f(a, b, c) = ab' + ac' + bc$

b. $G = f(w, x, y, z) = w'x + yz'$

c. $T = f(a, b, c) = (a + b')(b' + c)$

d. $J(A, B, C, D) = (A + B' + C)(A' + D)$

3. Generating Switching Equations from truth table:

Problem Statement to Truth Tables

Design a combinational logic truth table so that an output is generated indicating when a majority of 4 inputs are true.

a	b	c	d	y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

3. Generating Switching Equations from truth table:

- ❑ The Sign **m** or Σ indicates summations and stands for the canonical SOP.
- ❑ The Sign **M** or Π indicates summations and stands for the canonical POS

Express the following SOP and POS equations in a minterm and maxterm list (shorthand decimal notation) form:

1. $H = f(A, B, C) = A'BC + A'B'C + ABC$

2. $G = f(w, x, y, z) = wxyz' + wx'yz' + w'xyz' + w'x'yz'$

3. $T = f(a, b, c) = (a + b' + c)(a + b' + c')(a' + b' + c)$

4. $J = f(A, B, C, D) = (A + B' + C + D)(A + B' + C + D')(A' + B + C + D)(A' + B' + C + D)(A' + B + C' + D)(A' + B' + C' + D)$

3. Generating Switching Equations from truth table:

x	y	z	A	A'
0	0	0	0	1
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	0	1
1	1	1	0	1

4. Karnaugh Maps (K-Map)

- ☐ A Tool for Simplifying Boolean Expressions
- ☐ Used to minimize Boolean expressions in digital circuits
- ☐ Applicable for 2, 3, 4, and 5 variables (practically up to 6 variables)
- ☐ Provides simplified expressions without applying Boolean algebra theorems manually
- ☐ Two forms:
 - ☐ **Sum of Products (SOP)**
 - ☐ **Product of Sums (POS)**

4. Karnaugh Maps (K-Map)

Why K-Map?

Truth table → shows complete information, but not easy for simplification

Boolean algebra → may be lengthy, requires applying rules step by step

K-Map → Simple, visual method for minimization

Reduces number of logic gates in circuits → saves cost, space, and power

4. Karnaugh Maps (K-Map)

K-Map Basics

K-Map is a grid-like representation of truth table values

Each cell corresponds to a **minterm (for SOP)** or **maxterm (for POS)**

The arrangement follows **Gray Code sequence** → only one variable changes between adjacent cells

Grid sizes:

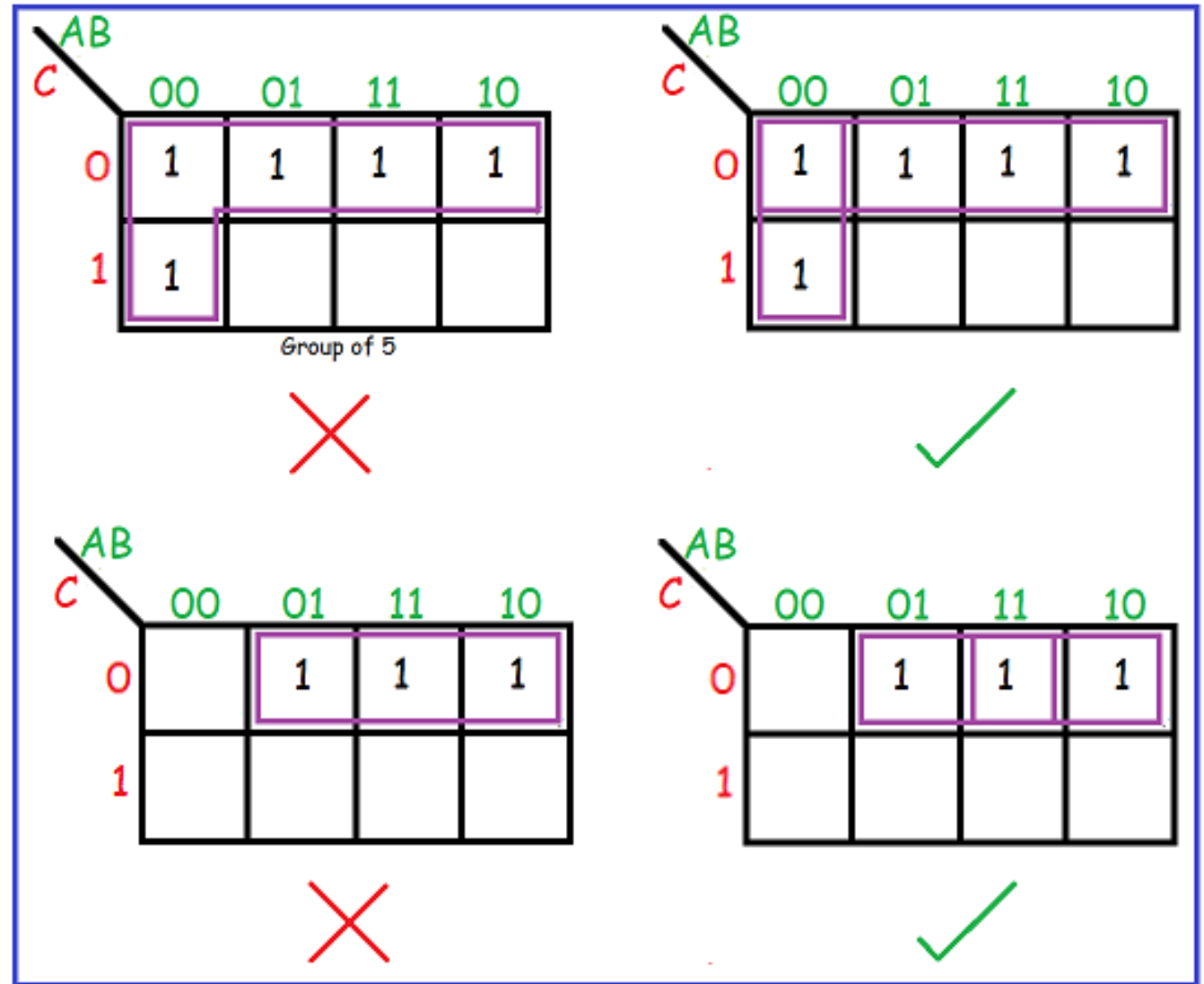
2-variable → 2×2

3-variable → 2×4

4-variable → 4×4

5-variable → 4×8

4. Karnaugh Maps (K-Map)



4. Karnaugh Maps (K-Map)

Find all Prime Implicants (PIs) and Essential Prime Implicants (EPIs) and then give the simplified expression.

$$Y = f(a, b, c) = \sum (0, 1, 4, 5)$$

$$P = f(a, b, c) = \sum (0, 2, 4, 6)$$

$$Q = f(a, b, c) = \sum (0, 2, 3, 7)$$

$$R = f(a, b, c) = \sum (1, 3, 6, 7)$$

$$T = f(a, b, c) = \sum (1, 2, 3, 6, 7)$$

$$J = f(x, y, z) = \sum (0, 2, 3, 4, 5, 7)$$

4. Karnaugh Maps (K-Map)

Simplify the following four variable equations using K-Map

a. $K = f(a, b, c, d) = \sum (0, 1, 4, 5, 9, 11, 13, 15)$

b. $L = f(a, b, c, d) = \sum (0, 2, 5, 7, 8, 10, 13, 15)$

c. $P = f(r, s, t, u) = \sum (1, 3, 4, 6, 9, 11, 12, 14)$

d. $D = f(w, x, y, z) = \sum (5, 7, 8, 9, 13)$

e. $P = f(a, b, c, d) = \sum (0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

f. $S = f(a, b, c, d) = \sum (1, 3, 4, 5, 7, 8, 9, 11, 15)$

g. $U = f(w, x, y, z) = \sum (1, 5, 7, 8, 9, 10, 11, 13, 15)$

4. Karnaugh Maps (K-Map)

Rules for Grouping in K-Map

- ☐ Groups must contain **2^n cells** (1, 2, 4, 8, ...).
- ☐ Group cells only if they contain **1s (for SOP)** or **0s (for POS)**.
- ☐ Groups should be made as **large as possible** to achieve maximum simplification.
- ☐ Each group must form a **rectangle** (or square) shape.
- ☐ Grouping can be done **horizontally or vertically**, but **not diagonally**.
- ☐ Groups may **wrap around** the edges of the map.
- ☐ Each "1" (or "0") should be included in **at least one group**.
- ☐ Overlapping of groups is **allowed** if it leads to simplification.
- ☐ Larger groups give **simpler expressions**, so prioritize them.

4. Karnaugh Maps (K-Map)

Any single minterm or permitted group of minterms is called an **implicant** of an output function

Prime Implicants (PIs)

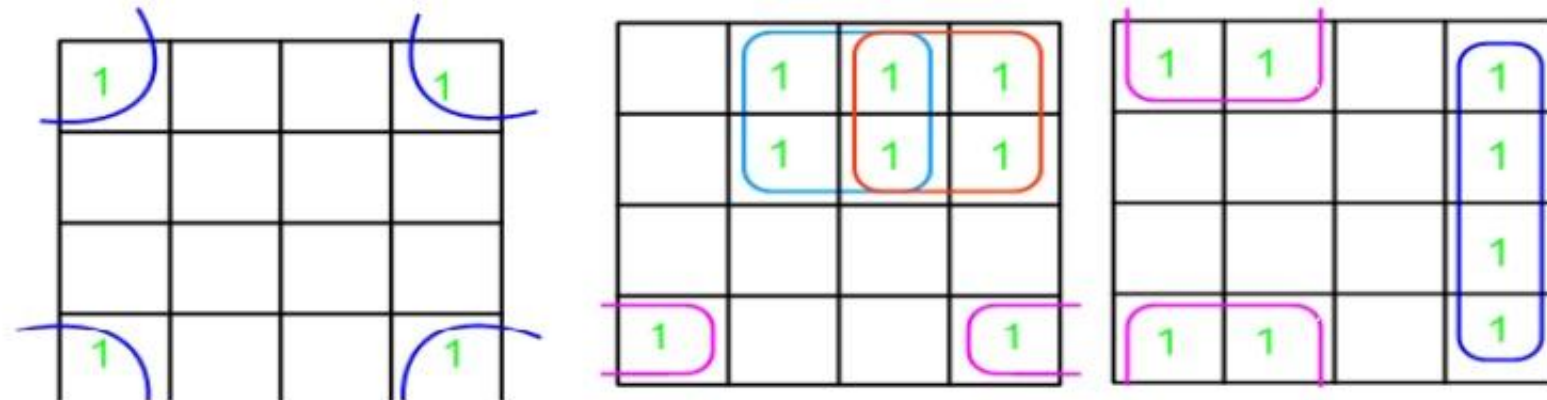
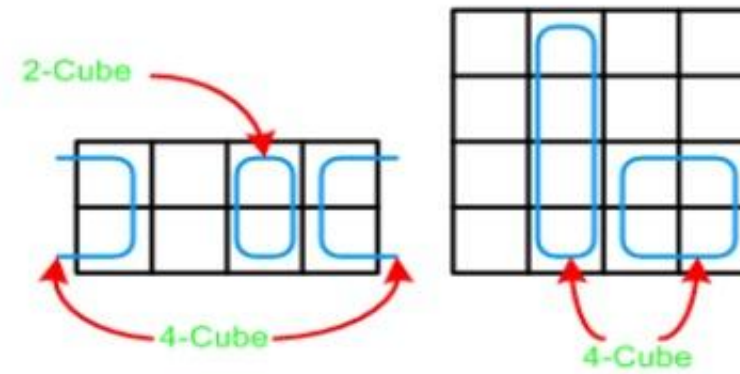
- ☐ A **prime implicant** is a **maximal group of 1s** (for SOP) or **0s** (for POS).
- ☐ It cannot be combined into a larger valid group.
- ☐ All groups formed in K-map are potential prime implicants.
- ☐ Larger groups are always preferred since they reduce variables.
- ☐ A K-map may have multiple prime implicants.

4. Karnaugh Maps (K-Map)

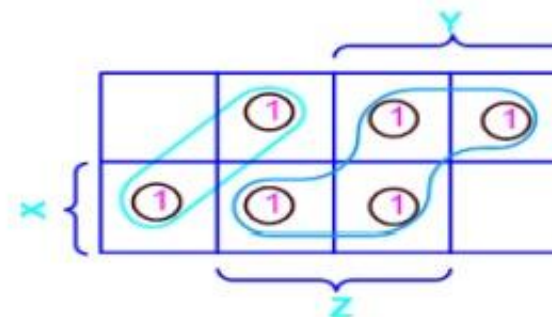
Essential Prime Implicants (EPIs)

- ☐ A prime implicant is **essential** if it covers at least one **1 (minterm)** that is not covered by any other group.
- ☐ EPIs are **necessary** to include in the final simplified expression.
- ☐ Non-essential PIs may still be included for covering uncovered minterms.
- ☐ The set of EPIs ensures **minimum expression** for the Boolean function.

4. Karnaugh Maps (K-



Example of invalid groups



4. Karnaugh Maps (K-Map)

Incompletely Specified Functions (Don't Care Terms)

- A Boolean function is **completely specified** if the output is defined for **every input combination**.
- A function is **incompletely specified** if output is **not defined** for certain input combinations.
- This situation usually arises when:
 - Some input combinations **never occur in practice**.
 - Certain input cases are **irrelevant to system operation**.
- The **undefined minterms or maxterms** are called **Don't Care Terms (X)**.
- Don't care terms can be treated as **0 or 1** during simplification.
- They provide **flexibility** in grouping K-map cells.
- Using don't cares helps achieve a **simpler minimized expression**.

4. Karnaugh Maps (K-Map)

- Example: BCD codes (valid 0000–1001) → combinations 1010–1111 are **don't cares**.

B3	B2	B1	B0	E3	E2	E1	E0
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
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

4. Karnaugh Maps (K-Map)

Simplify Maxterm Equations:

Loading and grouping Maxterms is exactly same as loading and grouping minterms, except that 0s are loaded into the map and grouped to form the PI and EPI groups. Reading the map is, however different.

Simplify the Boolean functions using K- Map technique.

$$J = f(x, y, z) = \prod (0, 3, 4, 7)$$

$$G = f(a, b, c, d) = \prod (0, 4, 5, 7, 8, 9, 11, 12, 13, 15)$$

$$W = f(w, x, y, z) = \prod (1, 3, 8, 10, 12, 13, 14, 15)$$

4. Karnaugh Maps (K-Map)

Simplify Maxterm Equations:

Loading and grouping Maxterms is exactly same as loading and grouping minterms, except that 0s are loaded into the map and grouped to form the PI and EPI groups. Reading the map is, however different.

Simplify the Boolean functions using K- Map technique.

$$J = f(x, y, z) = \prod (0, 3, 4, 7)$$

$$G = f(a, b, c, d) = \prod (0, 4, 5, 7, 8, 9, 11, 12, 13, 15)$$

$$W = f(w, x, y, z) = \prod (1, 3, 8, 10, 12, 13, 14, 15)$$

5. Quine-McCluskey Minimization Technique (QM Technique)

- Boolean minimization is essential to simplify logic circuits.
- Quine–McCluskey method = **Tabular method of minimization**.
- Works systematically using binary comparisons.
- Suitable for manual minimization of small functions & computer implementation for larger ones.

5. Quine-McCluskey Minimization Technique (QM Technique)

Steps in Quine–McCluskey Method

1. **List minterms** in binary form.
2. **Group minterms** by the number of 1's (0's, 1's, 2's ...).
3. **Compare adjacent groups** to combine terms differing in only one bit → replace differing bit with “–”.
4. **Repeat grouping & combining** until no further reduction possible.
5. **Mark unused terms** → Prime Implicants.
6. **Construct Prime Implicant (PI) Chart** with minterms vs PIs.
7. **Select Essential Prime Implicants** that cover unique minterms.
8. **Choose minimal cover** for the remaining minterms.

5. Quine-McCluskey Minimization Technique (QM Technique)

Simplify the following using the Quine – McCluskey minimization technique

$$Y = f(a, b, c, d) = \sum (0, 1, 2, 3, 6, 7, 8, 9, 14, 15)$$

$$P = f(w, x, y, z) = \sum (1, 3, 4, 5, 7, 8, 15) \quad .$$