MAA OMWATI DEGREE COLLEGE HASSANPUR

(PALWAL)

NOTES

BCA 2nd SEM

DIGITAL LOGIC DESIGN

UNIT-1

1. Digital Systems

Digital systems are systems that process discrete (binary) values, unlike analog systems that process continuous signals. These systems are built using digital circuits and perform operations on binary numbers.

---

2. Digital Signals

Digital signals represent information using binary format (0 and 1). These signals have two distinct levels:

- **LOW (0)**
- **HIGH (1)**

They are less susceptible to noise and can be easily regenerated.

Example: A square wave that alternates between 0V and 5V.

---

3. Digital Waveforms

A digital waveform is a graphical representation of digital signal levels over time. It typically consists of alternating HIGH and LOW states.

---

4. Digital Computers

Digital computers are electronic machines that perform calculations and logical operations using digital signals. Components include:

- Central Processing Unit (CPU)
- Memory (RAM/ROM)

- Input/Output Devices

---

5. Digital Integrated Circuits

These are compact microelectronic devices containing thousands/millions of digital logic gates fabricated on a semiconductor.

- **SSI (Small Scale Integration)**
- **MSI (Medium Scale Integration)**
- **LSI (Large Scale Integration)**
- **VLSI (Very Large Scale Integration)**

---

6. Number Systems

Systems used to represent numerical values. Common ones include:

- **Decimal (Base 10)**
- **Binary (Base 2)**
- **Octal (Base 8)**
- **Hexadecimal (Base 16)**

---

7. Binary Number System

Uses only two symbols: 0 and 1. It is the foundation of all digital electronics.

Example: Binary: 1011 = Decimal: 11

---

8. Octal and Hexadecimal Systems

- **Octal** uses base 8 (0 to 7)

    - Example: (17)_8 = (15)_10
- **Hexadecimal** uses base 16 (0 to 9, A to F)
    - Example: (1F)_16 = (31)_10

---

9. Number Base Conversions

- Binary to Decimal: Multiply each bit by 2^position
- Decimal to Binary: Divide by 2, record remainder

Example: (13)_10 = (1101)_2

---

10. Complements

Used in subtraction and representation:

- **1’s Complement:** Invert all bits
- **2’s Complement:** 1’s complement + 1

Example: 2’s complement of 0101 = 1011

---

11. Signed Binary Numbers

Used to represent both positive and negative numbers:

- **Sign-Magnitude**
- **1’s Complement**
- **2’s Complement** (most common)

---

12. Binary Codes

- **BCD (Binary-Coded Decimal)**
- **Gray Code**

- **ASCII (American Standard Code for Information Interchange)**

---

13. Error Detection and Correction Codes

- **Parity Bit (even/odd)**
- **Hamming Code (error correction)**

---

14. Boolean Algebra

A mathematical structure to represent and manipulate logical expressions using:

- **AND** (.)
- **OR** (+)
- **NOT** (¯)

---

15. Axiomatic Definition, Theorems and Properties

- Identity Law: A + 0 = A, A.1 = A
- Null Law: A + 1 = 1, A.0 = 0
- Idempotent Law: A + A = A
- Involution: (A¯)¯ = A
- DeMorgan's Theorem

---

16. Boolean Functions

Functions written using Boolean operators. **Example:** F(A, B, C) = A.B + B.C

---

17. Canonical Standard Forms

- **Sum of Minterms (SOP)**
- **Product of Maxterms (POS)**

---

18. SOP and POS Forms

- SOP: Sum of ANDed terms
  - Example: $\bar{A}B + A\bar{B}$
- POS: Product of ORed terms
  - Example: $(A + B)(A + C)$

---

19. Digital Logic Gates

NOT Gate

- Inverts the input
- Truth Table:

A	**$Y=\bar{A}$**
0	1
1	0

OR Gate

- Output is 1 if any input is 1

A	B	$Y=A+B$
0	0	0
0	1	1
1	0	1

1 1 1

AND Gate

- Output is 1 only if all inputs are 1

NOR Gate = NOT(OR)

NAND Gate = NOT(AND)

XOR Gate = Output is 1 if inputs differ

XNOR Gate = Output is 1 if inputs are same

---

20. Universal Gates

- **NAND and NOR** are universal gates
- Any logic circuit can be built using only NAND or NOR gates

Example:

- NOT using NAND:
  - A NAND A = NOT A
- AND using NAND:
  - (A NAND B) NAND (A NAND B) = A AND B

Unit-2

1. Gate-Level Minimization

Gate-level minimization refers to the process of reducing the number of logic gates or the complexity of a digital circuit without changing its function. The aim is to design efficient digital circuits with minimal hardware, cost, power consumption, and propagation delay.

When you design digital systems using logic gates (AND, OR, NOT, etc.), you often start with a Boolean expression derived from the truth table. These Boolean expressions may not be optimal and could lead to complex circuits if implemented directly. Minimization techniques help simplify these expressions before converting them to physical circuits.

Goals of Gate-Level Minimization:

- Reduce the number of gates used.
- Reduce the number of inputs to each gate.
- Use less power and occupy less space.
- Increase speed and reliability.

---

2. Karnaugh Map (K-map) Method

The **Karnaugh map (K-map)** is a graphical tool used for simplifying Boolean expressions and logic functions. It is a visual representation of truth tables and allows easy identification of patterns (like adjacent 1s or 0s) that can be grouped to simplify expressions.

Characteristics:

- Each cell represents a minterm (for SOP) or maxterm (for POS).

- The map is constructed such that adjacent cells differ by only one variable (this adjacency helps identify simplifications).
- Variables are arranged using **Gray code** so that only one variable changes between adjacent cells.

K-map Sizes:

- **2-variable K-map:** 4 cells (2^2)
- **3-variable K-map:** 8 cells (2^3)
- **4-variable K-map:** 16 cells (2^4)
- **5-variable and 6-variable maps** are possible but hard to draw and are usually solved using software or tabular methods like Quine–McCluskey.

Simplification Using K-map:

1. Fill the map with 1s (for SOP) or 0s (for POS) based on the function.
2. Identify groups of **1, 2, 4, 8,...** adjacent 1s (SOP) or 0s (POS).
3. Groups must be rectangular and contain 2^n cells.
4. Write the simplified expression by eliminating the variables that stay constant within each group.

---

3. Simplification: Algebra Postulates and Canonical Forms

Boolean Algebra Postulates and Theorems:

Boolean algebra provides a set of rules for simplifying expressions algebraically.

Basic Laws/Postulates:

- **Identity Law:** $A + 0 = A$, $A \cdot 1 = A$
- **Null Law:** $A + 1 = 1$, $A \cdot 0 = 0$

- **Idempotent Law:** A + A = A, A · A = A
- **Complement Law:** A + A' = 1, A · A' = 0
- **Involution Law:** (A')' = A

Algebraic Simplification Techniques: Use distributive, commutative, associative, DeMorgan's theorems, and absorption laws to simplify expressions step by step.

Canonical Forms:

These are standardized ways of writing Boolean expressions.

Sum of Products (SOP):

- Each term is a product (AND) of literals.
- The expression is a sum (OR) of product terms.
- E.g., F = A'B'C + AB'C + ABC

Product of Sums (POS):

- Each term is a sum (OR) of literals.
- The expression is a product (AND) of sum terms.
- E.g., F = (A + B + C')(A' + B + C)

These forms are useful for input into K-maps and for logic circuit design.

---

4. Prime Implicants: Types, Determination and Selection

Implicant:

An implicant is a product term in SOP that covers one or more minterms in the function.

Prime Implicant:

A **prime implicant** is an implicant that cannot be combined with another to eliminate a variable (i.e., it is not a subset of a larger implicant).

Types of Implicants:

- **Essential Prime Implicants:** Prime implicants that cover at least one minterm not covered by any other prime implicant.
- **Non-Essential Prime Implicants:** Prime implicants that cover minterms already covered by other implicants.

Determination of Prime Implicants (via K-map):

1. Create the K-map and fill in 1s for minterms.
2. Group adjacent 1s into rectangles (of 2^n).
3. Each group corresponds to a prime implicant.
4. Continue grouping until all 1s are covered with the fewest and largest possible groups.

Selection of Prime Implicants:

- Always include **essential prime implicants** in the final expression.
- Choose the smallest combination of remaining (non-essential) prime implicants that cover all other minterms.

---

5. Don’t Care Conditions

Don't care conditions represent input combinations that **will never occur** or whose output doesn’t affect the operation. They are typically marked as **X** or **d**.

Usage in Minimization:

- In **K-maps**, don’t cares can be used as **1 or 0**, whichever helps form larger groups.

- They allow greater flexibility during minimization and help reduce the number of gates further.

Example:

In a circuit with 3-bit inputs where only inputs from 000 to 101 are valid, the inputs 110 and 111 are "don't care." They can be used to form larger groups during K-map simplification.

---

6. NAND and NOR Implementation

Logic gates like **NAND** and **NOR** are called **universal gates** because any Boolean function can be implemented using only NAND or only NOR gates.

Why Use NAND/NOR:

- Economical: Most ICs use NAND gates for cost and speed.
- Simplifies hardware manufacturing.
- Reduces the number of different gate types needed.

NAND Implementation:

1. Start with the simplified Boolean expression.
2. Convert AND-OR form to NAND-NAND using **double negation**.
3. Apply DeMorgan's theorem to push negation inward.

NOR Implementation:

1. Similar to NAND implementation.
2. Expression should be in OR-AND form (POS).
3. Use **double negation** and DeMorgan's laws for transformation.

Example – NAND:

Given $F = A + B$

- $F = ((A')\cdot(B'))'$ → uses a NAND gate to realize OR logic.

Example – NOR:

Given $F = A \cdot B$

- $F = ((A + B)')'$ → uses a NOR gate to realize AND logic.

Unit-3

1. Combinational Circuits: Introduction

Combinational circuits are a type of digital circuit whose output is determined **solely by the current input values**. There is **no memory or feedback** in these circuits, meaning they do not store previous input or output values.

Unlike **sequential circuits**, which depend on both current inputs and past states (using memory elements like flip-flops), combinational circuits are **stateless** and produce outputs instantaneously (neglecting gate delay).

Real-World Analogy:

Think of a basic calculator that adds two numbers. As soon as you input values and press '+', the result is calculated directly this is how a combinational circuit works.

---

2. Characteristics of Combinational Circuits

1. **No Memory:** Output depends only on the present input values.
2. **No Feedback Path:** Output is not fed back into the input.
3. **Instantaneous Output:** The output changes as soon as input changes (idealized—real circuits have small gate delays).
4. **Deterministic Behavior:** For a given input, the output is always the same.
5. **Use of Logic Gates:** Built using basic gates (AND, OR, NOT) or universal gates (NAND, NOR).
6. **Speed:** Generally faster than sequential circuits due to absence of clock and memory.

7. **Simplification Possible:** Boolean algebra and Karnaugh maps can be used to reduce circuit complexity.

---

3. Designing Principles of Combinational Circuits

Designing combinational circuits involves several systematic steps:

Step 1: Problem Specification

- Define what the circuit is supposed to do.
- Understand inputs and expected outputs.

Step 2: Determine the Number of Inputs and Outputs

- Identify how many bits or variables are involved.
- Label inputs and outputs clearly.

Step 3: Construct a Truth Table

- List all possible input combinations.
- Define output for each input set.

Step 4: Derive Boolean Expressions

- Use SOP (Sum of Products) or POS (Product of Sums) based on the truth table.

Step 5: Simplify the Boolean Expressions

- Apply Boolean algebra rules or Karnaugh map simplification.
- Aim for minimal logic gate usage.

Step 6: Implement the Circuit

- Draw the logic diagram using gates.
- Verify the circuit behavior.

---

4. Binary Adder Circuits

Binary adders perform addition of binary numbers and are crucial in arithmetic and logic units (ALUs) of processors.

4.1 Half-Adder

Function: Adds two single-bit binary numbers (A and B).

Inputs: A, B
Outputs: Sum (S), Carry (C)

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- **Sum = A ⊕ B** (XOR)
- **Carry = A · B**

4.2 Full-Adder

Function: Adds three bits – two operands and a carry-in.

Inputs: A, B, Cin (Carry-in)
Outputs: Sum, Cout (Carry-out)

A	B	Cin	Sum	Cout
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1

A	B	Cin	Sum	Cout
1	1	1	1	1

- **Sum = A ⊕ B ⊕ Cin**
- **Cout = AB + BCin + ACin**

A full-adder can be made using **two half-adders and an OR gate**.

---

5. Subtractor Circuits

Subtractor circuits perform binary subtraction.

5.1 Half-Subtractor

Function: Subtracts one bit from another.

Inputs: A (minuend), B (subtrahend)
Outputs: Difference, Borrow

A	B	Diff	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

- **Difference = A ⊕ B**
- **Borrow = A' · B**

5.2 Full-Subtractor

Function: Subtracts B and Bin (Borrow-in) from A.

Inputs: A, B, Bin
Outputs: Difference, Bout

- **Difference = A ⊕ B ⊕ Bin**

- **Borrow = A'B + A'(Bin) + BBin**

---

6. Parallel Binary Adder/Subtractor

These circuits allow addition or subtraction of multi-bit binary numbers by connecting multiple full-adders/subtractors in sequence.

4-bit Parallel Adder:

- Four full-adders are cascaded.
- Carry-out of one is carry-in of the next.
- Used in ALUs.

4-bit Adder/Subtractor Combo:

- Uses XOR gates and a control line.
- Control = 0: performs addition.
- Control = 1: performs subtraction (B is XORed with 1s = two's complement subtraction).

---

7. Binary Multiplier

Performs binary multiplication, which is a series of AND and shift-add operations.

Example:

Multiply 1010 (10) × 1101 (13)

Use partial products like in decimal multiplication and add them.

Implementation:

- Use **AND gates** to generate partial products.
- Use **adders** to sum the shifted partial products.

More complex versions use:

- **Array Multipliers**
- **Booth's Multipliers** for signed numbers.

---

8. Comparators

Comparators are circuits that compare two binary numbers and determine their relationship.

Outputs:

- A > B
- A = B
- A < B

1-bit Comparator:

- A = B: A'B' + AB
- A > B: A · B'
- A < B: A' · B

Multi-bit Comparator:

Built using multiple 1-bit comparators and cascading logic.

Used in:

- Digital counters
- Sorting algorithms
- Digital decision-making systems

---

9. Multiplexers (MUX)

A **multiplexer** selects one of many input lines and routes it to a single output.

Structure:

- Inputs: 2^n data inputs
- Select lines: n
- 1 Output

Example: 4-to-1 MUX

- Inputs: I_0 to I_3
- Select: S_0, S_1
- Output: Y = S_1S_0 used to choose which input appears on output.

Application:

- Data routing
- ALUs
- Control systems

---

10. De-Multiplexers (DEMUX)

Opposite of MUX: takes one input and routes it to one of many outputs based on select lines.

Structure:

- 1 Input

- n Select Lines
- 2^n Outputs

Example: 1-to-4 DEMUX

- Routes input to one of four outputs.

Used in:

- Memory addressing
- Data distribution
- Signal demultiplexing

---

11. Encoders

Encoders take **2^n inputs** and generate an **n-bit binary output**. It performs the reverse operation of a decoder.

Example: 8-to-3 Encoder

- 8 input lines (only one active at a time)
- Output: 3-bit binary code corresponding to the active input.

Applications:

- Keyboard encoding
- Interrupt request lines

---

12. Decoders

Decoders take an n-bit input and activate **one of 2^n outputs**. It is the opposite of an encoder.

Example: 3-to-8 Decoder

- Inputs: A2, A1, A0
- Outputs: D_0 to D_7
- Only one output is high based on input.

Applications:

- Memory address decoding
- Display drivers

Unit-4

1. Sequential Circuits: Introduction

Sequential circuits are a class of digital circuits whose output depends on both the **present inputs** and the **past history (state)** of the system. This is in contrast to **combinational circuits**, where the output is solely a function of the current inputs.

Sequential circuits incorporate **memory elements** (like flip-flops or latches) that store information about the past input states. Because of this, sequential circuits can perform **stateful operations**, making them the backbone of devices like processors, timers, and digital clocks.

---

2. Characteristics of Sequential Circuits

1. **Memory Capability:** Stores previous inputs using flip-flops or latches.
2. **Time Dependency:** Output depends on the sequence of inputs over time.
3. **Feedback Path:** Part of the output is fed back to inputs, creating loops that store states.
4. **Clock Signal (for synchronous circuits):** Regulates when state transitions occur.
5. **Finite State:** Can be described using finite state machines (FSMs).
6. **Types:** Divided into:
    - **Synchronous Sequential Circuits** – state changes triggered by a clock pulse.
    - **Asynchronous Sequential Circuits** – state changes occur immediately upon input change, without clocking.

3. Latches

A **latch** is a basic memory device that stores one bit of data. It is **level-triggered**, meaning its output changes as long as the control signal (enable) is active.

SR Latch (Set-Reset Latch):

- Constructed using two NOR or NAND gates cross-connected.
- **Inputs:** S (Set), R (Reset)
- **Outputs:** Q and Q’ (complement)

S	**R**	**Q (Next State)**
0	0	No change
0	1	Reset (Q = 0)
1	0	Set (Q = 1)
1	1	Invalid (if NOR)

- Used where data needs to be held until explicitly changed.
- **Problem:** Unstable when both S and R are 1 in NOR-based latches.

4. Flip-Flops: Introduction

Flip-flops are edge-triggered memory elements, unlike latches which are level-triggered. Flip-flops store **one bit** and change state only during a **specific clock transition** (rising or falling edge).

Key Terms:

- **Edge-triggered:** Responds only to the transition (0→1 or 1→0) of the clock.

- **Clock (CLK):** Synchronization signal for controlled state changes.
- **Preset & Clear:** Used to directly set/reset the output without waiting for clock.

4.1 S-R Flip-Flop (Set-Reset)

Same as SR latch but synchronized with a **clock**.

S	R	Q (Next State)
0	0	No change
0	1	Reset
1	0	Set
1	1	Invalid

- Rarely used in practical circuits due to the invalid state.

4.2 J-K Flip-Flop

Eliminates the invalid state problem of the SR flip-flop by toggling the output when both inputs are high.

J	K	Q (Next State)
0	0	No change
0	1	Reset (Q = 0)
1	0	Set (Q = 1)
1	1	Toggle Q

- **Edge-triggered**
- Versatile and widely used.

4.3 D Flip-Flop (Data or Delay Flip-Flop)

Simplified version of SR flip-flop that avoids the invalid state by having only one input.

- **Q (next) = D**
- Copies D to output at clock edge.

D	CLK	Q
0	↑	0
1	↑	1

- Very common in **registers and memory circuits**.

4.4 T Flip-Flop (Toggle)

- Toggles the output on every clock edge when T = 1.

T	Q (Next)
0	No change
1	Toggle Q

- Built from J-K flip-flop with J = K = T.
- Used in **counters**.

4.5 Master-Slave Flip-Flop

A flip-flop consisting of two stages:

- **Master** – triggered by the clock pulse.
- **Slave** – triggered by the **inverted** clock.

This configuration avoids race conditions. The input is latched in the **master** and transferred to the **slave** only when the clock falls.

Used where precise edge-timing is crucial.

---

5. Registers

Registers are groups of flip-flops used to store multi-bit binary data. Each flip-flop holds one bit.

Types of Registers:

5.1 Shift Registers

Used to shift data left or right.

- **Serial-In Serial-Out (SISO)**
- **Serial-In Parallel-Out (SIPO)**
- **Parallel-In Serial-Out (PISO)**
- **Parallel-In Parallel-Out (PIPO)**

Applications:

- Data serialization/deserialization
- Communication systems
- Delay lines
- Multiplication/division by 2 (using shifts)

---

6. Counters

Counters are sequential circuits that go through a predefined sequence of states (usually binary).

6.1 Asynchronous Counters (Ripple Counters)

- Flip-flops are triggered **one after another** (ripple effect).
- First flip-flop gets clock, others are clocked by previous flip-flop's output.

Drawbacks:

- Slow due to propagation delays.
- Difficult to design for complex sequences.

Example: 4-bit asynchronous counter (counts 0–15)

6.2 Synchronous Counters

- All flip-flops are triggered **simultaneously** by the same clock.
- No ripple effect; faster and more reliable.

More complex to design, but better suited for high-speed operations.

6.3 Modulo-N Counters

- A counter that cycles through N states before resetting.
- **Mod-10 Counter (Decade Counter):** Counts from 0 to 9, then resets.
- Can be built using flip-flops and reset logic.

6.4 Up-Down Counters

- Can count in both **incrementing (up)** and **decrementing (down)** directions.
- Controlled by a mode input.
  - Mode = 1 → Count up
  - Mode = 0 → Count down

Application:

- Elevator systems
- Scoreboards
- Multi-functional digital systems