

## MODULE 1: Introduction

### Structure

- 1.1 Introduction: Embedded Systems and general purpose computer systems,
- 1.2 History, classifications, applications and purpose of embedded systems (Chapter 1 – Text 1)
- 1.3 Core of Embedded Systems : Microprocessors and microcontrollers,
- 1.4 RISC and CISC controllers,
- 1.5 Big endian and Little endian processors,
- 1.6 Application specific ICs,
- 1.7 Programmable logic devices,
- 1.8 COTS, sensors and actuators,
- 1.9 Communication interface, Embedded firmware,
- 1.10 other system components, PCB and passive components (Chapter 2 – Text 1)

### Objectives

#### Learning Objectives

##### 1. Understand Embedded Systems Basics

- Differentiate embedded vs. general-purpose computers.
- Learn history, classification, and applications.

##### 2. Explore Core Components & Architectures

- Study microprocessors, microcontrollers, RISC & CISC, endianness.
- Understand ASICs, PLDs, and COTS components.

##### 3. Analyze Hardware & Communication

- Learn about sensors, actuators, firmware, and PCBs.
- Understand communication interfaces and passive components.

## 1.1 Embedded Systems and General purpose computer systems

### Embedded Systems Definition

An embedded system is an electronic/electro-mechanical system designed to perform a specific function and is a combination of both hardware and firmware (software).




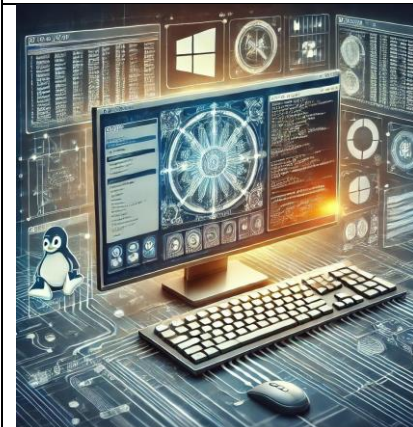

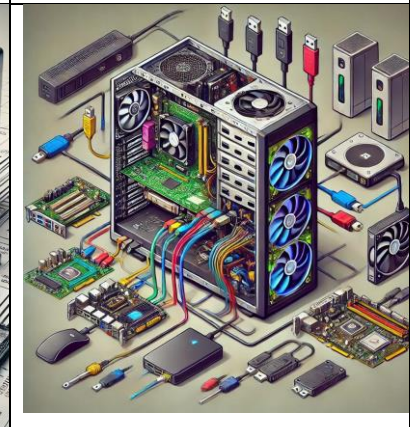


#### Need:

Embedded systems are becoming an inevitable part of any product or equipment in all fields including household appliances, telecommunications, medical equipment, industrial control, consumer products, etc.

### Comparison Between Embedded Systems and General-Purpose Computing Systems

Feature	Embedded System	General Purpose Computing System
<b>Purpose</b>	Designed for a specific function or task	Designed for multiple applications and tasks
<b>Operating System</b>	Fixed or custom OS, often non-modifiable	User can install or change OS (Windows, Linux, etc.)
<b>Hardware Flexibility</b>	Limited hardware customization	Highly flexible, supports multiple peripherals
<b>User Applications</b>	Cannot install third-party applications	Users can install and run multiple applications
<b>Interfaces</b>	Minimal, task-specific interfaces (e.g., IR remote)	Multiple interfaces (USB, Bluetooth, Wi-Fi, Ethernet, etc.)
<b>Computational Requirements</b>	Real-time processing with strict deadlines	General-purpose processing, not always real-time
<b>Power Efficiency</b>	Optimized for lower power consumption	Higher power consumption due to multitasking capability
<b>Memory Availability</b>	Limited and optimized for function	Large memory (RAM, HDD/SSD) for multiple applications
<b>Software Modification</b>	Not user-modifiable	Users can modify, update, and install new software
<b>Functionality Expansion</b>	Fixed function, cannot be easily repurposed	Can be repurposed for various tasks
<b>Example Devices</b>	DVD Player, Washing Machine, Car ECU	Desktop PC, Laptop, Palmtop, Workstation

## Comparison Between Embedded Systems and General Purpose Computing Systems

		
<p>Purpose of an Embedded System: A washing machine with predefined functions.</p>	<p>Purpose of a General-Purpose Computing System: A laptop running multiple applications.</p>	<p>Operating System in an Embedded System: A digital thermostat with a fixed interface.</p>
		
<p>Operating System in a General-Purpose Computing System: A desktop PC with a flexible OS.</p>	<p>Hardware Flexibility in an Embedded System: A microwave oven with fixed buttons.</p>	<p>Hardware Flexibility in a General-Purpose Computing System: A desktop PC with multiple peripherals.</p>
		
<p>User Applications in an Embedded System: A smart thermostat with limited options.</p>	<p>User Applications in a General-Purpose Computing System: A smartphone with multiple third-party apps.</p>	

## 1.2 History, classifications, applications and purpose of embedded systems

---

### 1.2.1

#### 1. Classification Based on Evolution (History)

##### 1. First Generation

- 8-bit microprocessors (8085, Z80) and 4-bit microcontrollers.
- Simple hardware, firmware in Assembly language.
- Examples: Digital telephone keypads, stepper motor control units.

##### 2. Second Generation

- 16-bit microprocessors, 8/16-bit microcontrollers.
- More complex instruction sets, some with embedded OS.
- Examples: Data Acquisition Systems, SCADA systems.

##### 3. Third Generation

- 32-bit processors, 16-bit microcontrollers, DSPs, ASICs.
- Instruction pipelining, real-time and general-purpose OS.
- Examples: Robotics, industrial process control, networking.

##### 4. Fourth Generation

- System-on-Chip (SoC), reconfigurable and multi-core processors.
- High-performance, miniaturized, real-time OS.
- Examples: Smartphones, Mobile Internet Devices (MIDs).

#### 2. Classification Based on Complexity and Performance

##### 1. Small-Scale Embedded Systems

- Simple applications, not time-critical.
- Built on low-cost 8/16-bit microprocessors/microcontrollers.
- May or may not have an OS.
- Example: Electronic toys.

##### 2. Medium-Scale Embedded Systems

- Moderately complex hardware and software.
- Uses 16/32-bit microprocessors/microcontrollers or DSPs.
- Usually includes an embedded OS.



### 3. Large-Scale/Complex Embedded Systems

- High-performance, mission-critical applications.
- Uses 32/64-bit RISC processors, RSoC, multi-core processors.
- Includes multiple processors, co-processors, and accelerators.
- Uses high-performance Real-Time Operating Systems (RTOS).
- Examples: Media encoding/decoding, cryptographic processing.

#### 1.2.2 Applications of embedded systems

##### **Embedded Systems in Daily Life**

Embedded systems play a crucial role in modern technology, evolving from early models like the Apollo guidance computer to advanced smart devices. Their applications span multiple domains, including:

1. **Consumer Electronics** – Camcorders, cameras.
2. **Household Appliances** – TVs, DVD players, washing machines, fridges, microwaves.
3. **Home Automation & Security** – Air conditioners, sprinklers, alarms, CCTV cameras.
4. **Automotive Industry** – ABS, engine control, ignition, automatic navigation.
5. **Telecom** – Cell phones, telephone switches, multimedia applications.
6. **Computer Peripherals** – Printers, scanners, fax machines.
7. **Networking** – Routers, switches, hubs, firewalls.
8. **Healthcare** – Scanners, EEG, ECG machines.
9. **Measurement & Instrumentation** – Digital multimeters, CROs, logic analyzers, PLC systems.
10. **Banking & Retail** – ATMs, currency counters, POS systems.
11. **Card Readers** – Barcode scanners, smart card readers, handheld devices.

#### 1.2.3 Purpose of Embedded Systems

Embedded systems perform various specialized tasks in different domains like consumer electronics, healthcare, automotive, and telecommunications. Their functionalities can be categorized into six major roles:

##### **1. Data Collection/Storage/Representation**

- Embedded systems collect and process data from the external environment, which can be in analog or digital form.
- Analog data is converted into digital form using Analog-to-Digital (A/D) converters.
- The collected data may be:

1. Stored for further processing.
  2. Represented visually (LCD, LED) or audibly (buzzers, alarms).
  3. Deleted after processing.
- Examples: Digital multimeters, ECG monitoring devices, digital cameras (store and display images).

## **2. Data Communication**

- Embedded systems enable communication between devices, either through wired (USB, RS-232, TCP/IP) or wireless (Bluetooth, Wi-Fi, ZigBee) methods.
- Data transmission can be analog or digital, with modern systems favoring digital communication.
- Some embedded systems serve as dedicated transmission units, ensuring secure and efficient data exchange.
- Examples: Network routers, switches, telephone systems, home automation devices.

## **3. Data (Signal) Processing**

- Embedded systems are used in applications requiring signal processing, such as audio, video, and electrical signal manipulation.
- They perform functions like speech coding, audio synthesis, and video encoding.
- Example: Digital hearing aids, which amplify and process sound to improve hearing for individuals with impairments.

## **4. Monitoring**

- Designed to observe variables but not control them.
- Sensors gather real-time data, which is displayed for monitoring purposes without altering the system.
- Common in healthcare and industrial applications.
- Examples:
  1. Medical: ECG machines (monitor heartbeat but do not regulate it).
  2. Industrial: Digital storage oscilloscopes (CROs), logic analyzers, and multimeters (monitor voltage, current, etc.).

## **5. Control**

- Embedded systems that regulate processes by taking corrective actions based on sensor input.
- These systems consist of sensors (input), actuators (output), and a control unit that adjusts the output to maintain the desired conditions.

- Example: Air conditioners:
  1. The sensor (thermistor) measures the current temperature.
  2. The user input (desired temperature) is set via a remote control.
  3. The actuator (compressor) adjusts airflow to maintain the desired temperature.

---

### 1.3 Core of Embedded Systems: Microprocessors and microcontrollers

---

#### Microprocessors:

- A **microprocessor** is a CPU on a silicon chip capable of performing arithmetic and logical operations based on predefined instructions.
- Requires additional hardware components like memory, timers, and interrupt controllers for functioning, making it **dependent** and less compact.
- The **evolution of microprocessors** began with Intel's 4004 (4-bit, 1971), followed by Intel 8080, 8085, Zilog Z80, and later 16/32/64-bit processors.
- **Key Features:**
  1. General-purpose usage.
  2. High performance and processing speeds (modern processors reach up to 2.4 GHz).
  3. Based on **Harvard or Von-Neumann architectures** and use either **RISC** or **CISC instruction sets**.
- Used in **high-end markets**, industrial control, and advanced embedded applications.

#### Microcontrollers:

- A **microcontroller** integrates a CPU, RAM, ROM/Flash memory, timers, I/O ports, and interrupt control units into a single chip.
- **Self-contained** and designed specifically for embedded systems, eliminating the need for external components.
- Early examples include TI's **TMS1000 (1974)** and Intel's **8048 (1977)**, leading to the highly popular **Intel 8051** series in the 1980s.
- **Key Features:**
  1. Application-specific or domain-specific design.
  2. Inexpensive, compact, and power-efficient.
  3. Modern microcontrollers (e.g., ARM11, Atmel AVR) offer advanced features like SPI, I2C, USB, and networking.
- Used in consumer electronics, industrial control, IoT, and automotive applications.

### Comparison: Microprocessor vs. Microcontroller

Feature	Microprocessor	Microcontroller
<b>Definition</b>	A silicon chip representing a CPU that performs arithmetic and logical operations.	A highly integrated chip with CPU, RAM, ROM, timers, and I/O ports for embedded systems.
<b>Dependency</b>	Dependent on external hardware (memory, timers, interrupt controllers, etc.).	Self-contained and does not require external hardware for functioning.
<b>Purpose</b>	General-purpose usage in industrial and high-performance computing.	Designed for specific tasks in embedded systems.
<b>I/O Ports</b>	Does not include built-in I/O ports; requires external components like programmable peripheral interface chips.	Includes multiple built-in I/O ports (8, 16, or 32-bit) for direct use.
<b>Applications</b>	Used in computers, servers, and high-performance devices.	Used in IoT, robotics, automotive systems, and consumer electronics.
<b>Architecture</b>	Based on Harvard or Von-Neumann architecture with RISC or CISC instruction sets.	Similar architectures, often domain-specific instruction sets (e.g., AVR for automotive).
<b>Power Efficiency</b>	Less efficient in terms of power consumption.	Optimized for power efficiency and compactness.
<b>Target Market</b>	High-end markets where performance is critical.	Embedded markets, where cost and size are important.
<b>Cost</b>	Expensive due to dependency on external hardware.	Cost-effective, with all required components integrated.

Microcontrollers are ideal for embedded systems due to their compact design, low cost, and energy efficiency, while microprocessors are suited for high-performance, general-purpose applications.

#### 1.4 RISC and CISC controllers,

Feature	RISC (Reduced Instruction Set Computing)	CISC (Complex Instruction Set Computing)
<b>Instruction Set</b>	Lesser number of instructions.	Greater number of instructions.
<b>Instruction Pipelining</b>	Supports instruction pipelining for increased execution speed.	Generally, no instruction pipelining is present.
<b>Instruction Set Orthogonality</b>	Orthogonal instruction set (any instruction can operate on any register or use any addressing mode).	Non-orthogonal instruction set (instructions are instruction-specific regarding registers and addressing modes).
<b>Operations</b>	Operations are performed only	Operations can be performed directly



	on registers; memory operations are limited to load and store instructions.	on registers or memory, depending on the instruction.
<b>Number of Registers</b>	Large number of general-purpose registers are available.	Limited number of general-purpose registers.
<b>Programming</b>	Requires more lines of code as the instructions are simpler and perform smaller operations.	Instructions are like macros in C language, allowing complex operations with a single instruction, reducing the code length.
<b>Instruction Length</b>	Single, fixed-length instructions.	Variable-length instructions.
<b>Hardware Complexity</b>	Uses less silicon and has a lower pin count due to simpler instruction decoding.	Requires more silicon and additional decoder logic to handle complex instruction decoding.
<b>System Architecture</b>	Uses Harvard architecture (separate buses for program and data memory).	Can use either Harvard or Von-Neumann architecture (shared bus for program and data memory).

### Key Differences in Applications

- **RISC** is commonly used in applications requiring high performance and power efficiency, such as **smartphones, tablets, and embedded systems**.
- **CISC** is suitable for applications requiring complex operations, such as **desktop computers, servers, and legacy systems**.

RISC focuses on simplicity and efficiency, while CISC emphasizes complexity and minimizing the number of instructions a programmer needs to write.

### Harvard vs. Von-Neumann Architecture Comparison Table

Feature	Harvard Architecture	Von-Neumann Architecture
<b>Memory Structure</b>	Separate memory for instructions and data	Shared memory for instructions and data
<b>Instruction and Data Fetching</b>	Simultaneous fetching of instructions and data (Pre-fetching)	Sequential fetching of instructions and data
<b>Bus System</b>	Separate buses for instructions and data	Single common bus for instructions and data
<b>Performance</b>	High performance due to parallel execution and pipelining	Lower performance due to sequential fetching
<b>Cost</b>	Comparatively high cost due to separate memory buses	Cheaper due to simpler architecture
<b>Memory Alignment</b>	No memory alignment problems	Memory alignment problems possible
<b>Self-Modifying Code</b>	Does not allow self-modifying code	Allows self-modifying code

Program Memory Corruption Risk	No accidental corruption of program memory	Chances of accidental program memory corruption
--------------------------------	--	---

## 1.5 Big endian and Little endian processors

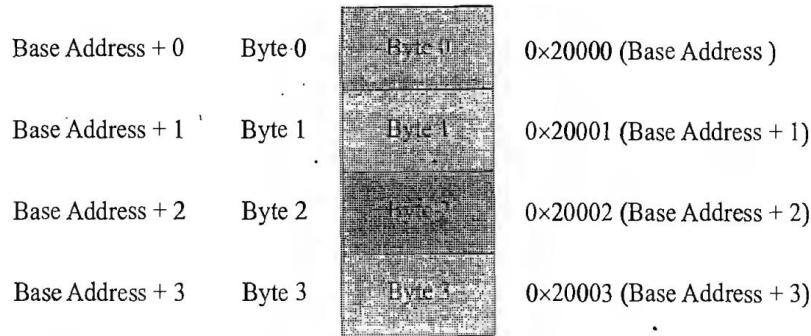
### Endianness in Memory Storage

Endianness defines how multi-byte data is stored in memory by a processor. It determines the order in which bytes are arranged in a system where the word size is greater than one byte.

#### Types of Endianness:

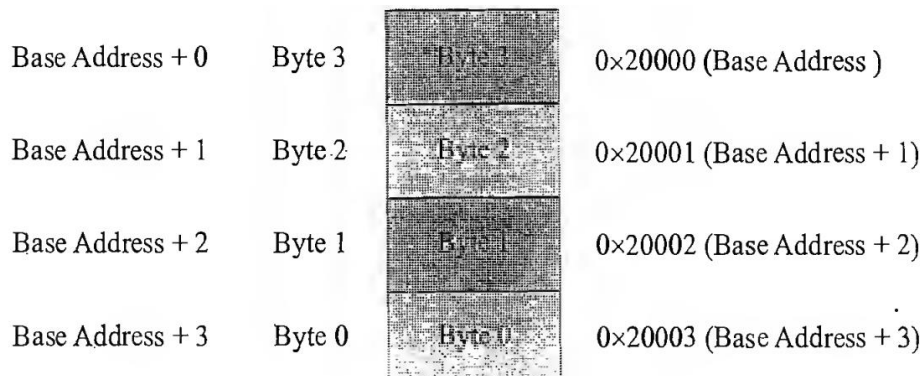
##### 1. Little-Endian:

- The **lower-order byte** is stored at the **lowest memory address**.
- The **higher-order byte** is stored at the **next higher memory address**.
- Example (for a 4-byte integer Byte3 Byte2 Byte1 Byte0):
  - Memory Order: **Byte0** → **Byte1** → **Byte2** → **Byte3** (Lowest to Highest Address).
- Commonly used in **Intel x86** processors.



##### 2. Big-Endian:

- The **higher-order byte** is stored at the **lowest memory address**.
- The **lower-order byte** is stored at the **next higher memory address**.
- Example (for a 4-byte integer Byte3 Byte2 Byte1 Byte0):
  - Memory Order: **Byte3** → **Byte2** → **Byte1** → **Byte0** (Lowest to Highest Address).
- Used in **Motorola, PowerPC, and some network protocols**.



### Key Differences:

- **Little-Endian:** Stores least significant byte first.
- **Big-Endian:** Stores most significant byte first.
- Some architectures, like **ARM and PowerPC**, support both endianness (bi-endian systems).

---

## 1.6 Application specific ICs

---

### 1. Definition:

- ASIC is a **microchip designed for a specific application** rather than general-purpose use.
- Replaces conventional logic chips and integrates multiple functions into a **single chip**.

### 2. Advantages:

- **Reduces system development cost** by integrating multiple functions.
- **Smaller size** enables compact system designs with **high performance**.
- **Optimized for specific tasks**, making them more efficient than general-purpose processors.

### 3. Types of ASICs:

- **Pre-fabricated ASICs:** Designed for **specialized applications**.
- **Custom ASICs:** Built using reusable components for **specific customer needs**.

### 4. Non-Recurring Engineering Charge (NRE):

- **One-time investment** for ASIC fabrication costs.
- **Cost-effective only for large-scale commercial production**.

### 5. Application-Specific Standard Product (ASSP):

- If an ASIC is made publicly available, it is called an **ASSP**.
- Marketed like a general-purpose product but for **specific applications**.
- Example: **ADE7760 Energy Meter ASIC** by Analog Devices.

### 6. Proprietary Nature:

- ASIC designs are usually **confidential** and **not publicly documented**.
- Detailed examples are rarely disclosed due to **legal and intellectual property concerns**.

---

## 1.7 Programmable logic devices,

---

### 1. Definition & Types

- Logic devices perform essential system functions like **data communication, signal processing, and control operations**.

- **Two main categories:**
  - **Fixed Logic Devices:** Permanent circuits, non-changeable once manufactured.
  - **Programmable Logic Devices (PLDs):** Reconfigurable devices with flexible logic capacity and features.

## 2. Advantages of PLDs

- **Fast Development:** No long lead times; immediate testing & deployment.
- **Cost-Effective:** No **Non-Recurring Engineering (NRE)** costs like custom ASICs.
- **Inventory Control:** Order as needed, avoiding surplus or shortage.
- **Field Reprogrammability:** Devices can be updated even after shipping (e.g., firmware updates via the Internet).

## 3. Types of PLDs

- **Field Programmable Gate Arrays (FPGAs):**
  1. High logic density (millions of gates), high performance.
  2. Used in **data processing, storage, telecommunications, digital signal processing**.
  3. Example: **Xilinx Virtex** with 8M system gates, built-in processors (IBM PowerPC), and high-speed interfaces.
- **Complex Programmable Logic Devices (CPLDs):**
  1. Lower logic density (up to 10,000 gates), predictable timing.
  2. Ideal for **control applications, low-power devices, mobile applications** (e.g., Xilinx CoolRunner series).

## 4. PLDs vs Fixed Logic Devices

- **PLDs are flexible** – Designs can be **modified & tested instantly** without new hardware.
- **Fixed logic devices are permanent**, requiring custom manufacturing for design changes.

## 5. FPGA Evolution & Trends

- Older FPGAs: **40 MHz, expensive (\$150+), limited gates**.
- Modern FPGAs: **300 MHz, millions of gates, integrated processors & memory, cost <\$10**.
- **FPGAs are now the preferred solution** for rapid prototyping and high-performance applications.

---

## 1.8 COTS, sensors and actuators,

---

### Commercial Off-the-Shelf (COTS)

- **Definition:** COTS products are **ready-made components** used **as-is** for easy integration into systems.
- **Examples:** Remote control toy car circuits, **high-frequency microwave electronics, analog-to-digital converters, infrared detectors, and TCP/IP plug-in modules** (e.g., WIZnet, Freescale).
- **Advantages:**
  - **Readily available**, reducing development time & cost.
  - **No need for custom design**, firmware is pre-built.

- **Faster time-to-market** for embedded systems.
- **Disadvantages:**
  - **No universal standards**, leading to **vendor lock-in**.
  - **Compatibility issues** between different manufacturers.
  - **Risk of product discontinuation**, affecting long-term design stability.

### Sensors & Actuators

- **Sensor:** A transducer that converts energy from one form to another for measurement or control.
  - Example: **Magnetic Hall Effect Sensor** in a smart running shoe (measures cushion-to-magnet distance).
- **Actuator:** A device that converts signals into physical action (motion).
  - Example: **Micro Stepper Motor** in a smart running shoe (adjusts cushion position).

---

## 1.9 Communication interface, Embedded firmware,

---

### Communication Interfaces

#### 1. Types of Communication Interfaces

- **Onboard Communication Interface:** Connects components within an embedded system (e.g., I2C, SPI, UART, Parallel Bus).
- **External Communication Interface:** Connects the embedded system to other devices (e.g., USB, Ethernet, Wi-Fi, Bluetooth, RF, GPRS).

#### 2. Inter-Integrated Circuit (I2C) Bus

- **Type:** Synchronous, bi-directional, half-duplex, **two-wire serial interface** (SCL & SDA).
- **Developed by:** Philips in the **1980s** for microcontroller-to-peripheral communication.
- **Master-Slave Model:**
  - **Master** controls communication, generates clock, and initiates data transfer.
  - **Slave** responds to master's request.
  - **Multi-master support** allows multiple controllers on the same bus.

#### 3. I2C Communication Sequence

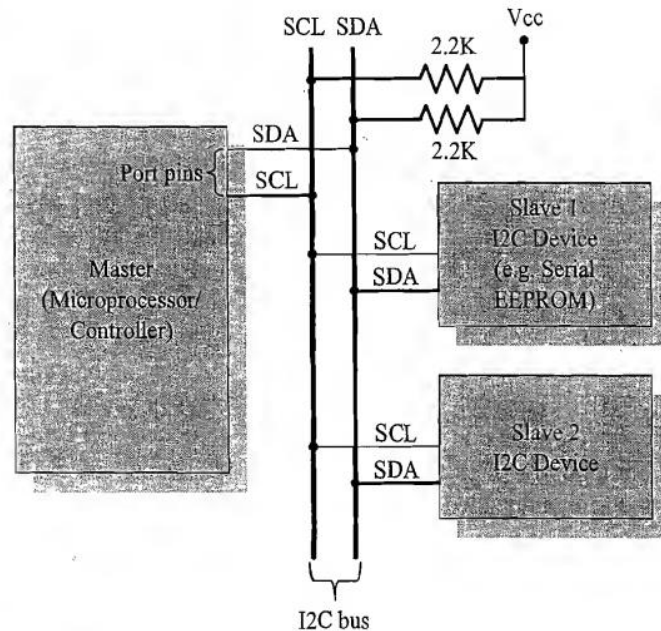
1. **Start Condition:** Master pulls SDA **LOW** while SCL is **HIGH**.
2. **Addressing:** Master sends **7-bit or 10-bit address** of the slave.
3. **Read/Write Bit:** 1 for **Read**, 0 for **Write**.
4. **Acknowledgment:** Slave acknowledges if the address matches.
5. **Data Transfer:**
  - **Write:** Master sends 8-bit data to Slave.
  - **Read:** Slave sends 8-bit data to Master.
6. **Acknowledgment:** Required after each byte transfer.
7. **Stop Condition:** Master releases SDA **HIGH** when SCL is **HIGH**.



#### 4. I2C Data Rates

- **Standard Mode:** Up to 100 kbps.
- **Fast Mode:** Up to 400 kbps.
- **High-Speed Mode:** Up to 3.4 Mbps.

#### I2C Bus Interfacing

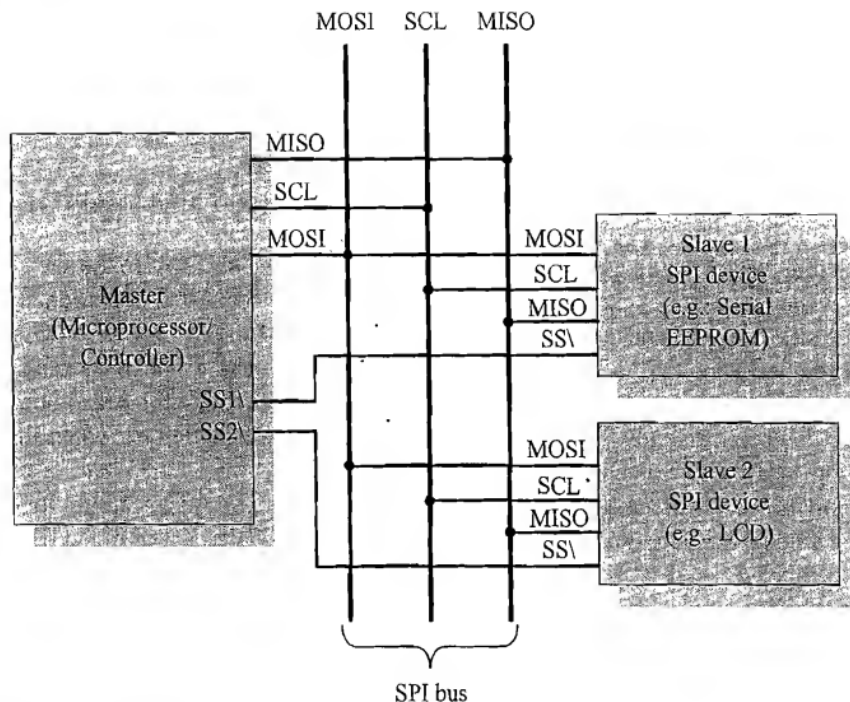


#### I2C Communication Sequence with a Slave Device

- Start Condition:**
  - Master pulls **SDA LOW** while **SCL is HIGH** to signal the start of communication.
- Address Transmission:**
  - Master sends a **7-bit or 10-bit slave address** over the **SDA** line.
  - Clock pulses on **SCL** synchronize the transmission.
- Read/Write Bit:**
  - Master sends **1-bit command**:
    - 0 → **Write to Slave**
    - 1 → **Read from Slave**
- Acknowledgment from Slave:**
  - The addressed slave **acknowledges (ACK)** by pulling **SDA LOW**.
- Data Transfer:**
  - If **writing**, Master sends **8-bit data** to Slave.
  - If **reading**, Slave sends **8-bit data** to Master.
- Acknowledgment after Each Byte:**
  - The receiver (Master or Slave) **sends an ACK** after receiving each byte.
- Stop Condition:**
  - Master pulls **SDA HIGH** while **SCL is HIGH**, signaling end of communication.

### Serial Peripheral Interface (SPI) Bus

- **Definition:** A **synchronous, full-duplex, four-wire serial interface** developed by Motorola.
- **Master-Slave System:**
  - **Single master, multiple slaves** (only one master active at a time).
  - Master **generates clock signal** and selects the slave via **Slave Select (SS)**.



#### **SPI Signal Lines:**

1. **MOSI (Master Out Slave In):** Data from **Master** → **Slave**.
2. **MISO (Master In Slave Out):** Data from **Slave** → **Master**.
3. **SCLK (Serial Clock):** Synchronizes data transfer.
4. **SS (Slave Select):** Activates the selected slave (active LOW).

#### **Working Principle:**

- Uses **Shift Registers** for data transfer.
- Data shifts **simultaneously** between Master & Slave via **MOSI & MISO**.
- Configurable settings: Master/Slave selection, baud rate, clock control, LSB/MSB order.

#### **Comparison with I2C:**

- **Faster** than I2C, ideal for **continuous data transfer (streams)**.
- **No acknowledgment mechanism**, making error detection harder.

### Universal Asynchronous Receiver Transmitter (UART):

- **Definition:** UART is an **asynchronous serial communication protocol** that does not require a clock signal for synchronization.

### Key Features:

#### 1. Data Transmission:

- Data transmission is based on **pre-defined settings** (baud rate, bits/byte, parity, start/stop bits).
- Special bits (Start and Stop) indicate the start and end of data transmission.
- **Parity Bit (optional)**: Used for error detection (1 for odd, 0 for even parity).

#### 2. Operation:

- **Start Bit**: Signals the receiver that data is incoming.
- Receiver polls the line at intervals determined by the baud rate.
- Discards Start, Stop, and Parity bits to form the data byte.

#### 3. Connection:

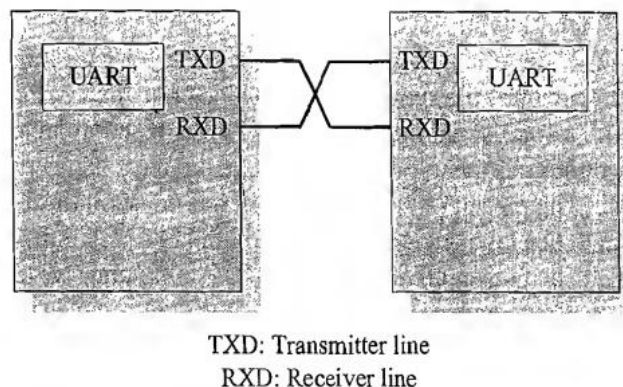
- **Transmit (TX)** line of sender connects to **Receive (RX)** line of receiver.

#### 4. Flow Control:

- UART supports **hardware handshaking** for managing data flow.

#### 5. Implementation:

- Early example: **8250 UART** used in IBM PCs.
- Modern microcontrollers feature **integrated UARTs** for seamless serial communication.



### 1-Wire Interface

- **Definition:** An asynchronous half-duplex communication protocol developed by Maxim (Dallas Semiconductor).
- **Features:**
  - Uses a **single signal wire (DQ)** for communication and power transmission.
  - Follows a **master-slave model** supporting multiple slave devices.
  - Each device has a **unique 64-bit ID** (8-bit family code, 48-bit serial number, 8-bit CRC).

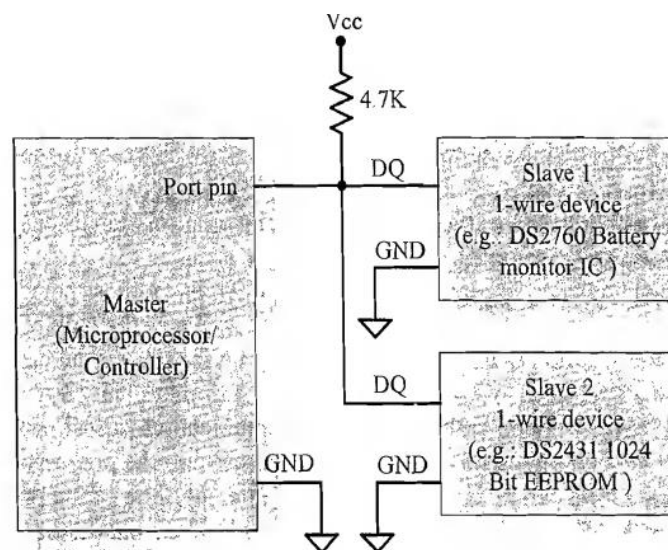
### Sequence of Operation:

1. **Reset Pulse:** Master pulls the bus **LOW** for 480  $\mu$ s to start communication.
2. **Presence Pulse:** Slaves respond with a **LOW pulse** within 60  $\mu$ s.
3. **ROM Command:** Master sends the **64-bit address** of the slave to address specific devices.
4. **Function Command:** Master sends **read/write commands** to interact with the slave's memory or registers.

### 5. Data Transfer: Master initiates **read/write operations**.

#### Communication Details:

- Time slots: **60  $\mu$ s per slot**.
- Writing:
  - Bit '1': Master pulls **LOW** for 1–15  $\mu$ s, then releases.
  - Bit '0': Master pulls **LOW** for 60–120  $\mu$ s.
- Reading:
  - Slave sends:
    - Bit '1': Releases bus.
    - Bit '0': Holds **LOW** for the rest of the time slot.



### Parallel Interface

- **Definition:** On-board parallel interfaces are used for communication with peripheral devices **memory-mapped** to the host processor/controller.

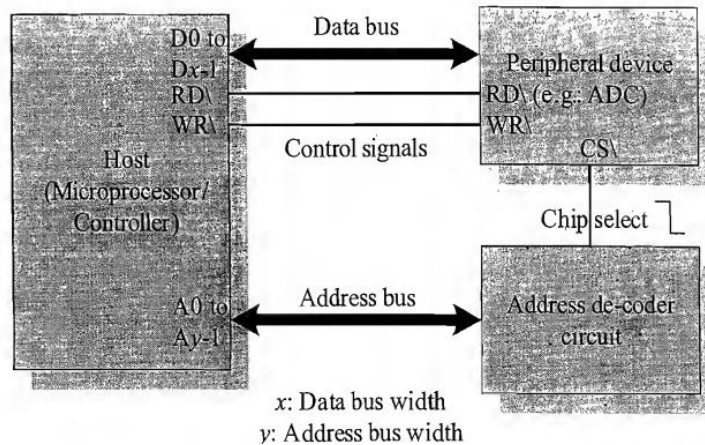
#### Features:

1. **Communication Control:**
  - Controlled by **Read/Write signals** and **Device Select (Chip Select)** signals from the host processor.
  - **Memory Mapping:** Devices are assigned a specific **address range**.
  - **Address Decoder:** Activates the device when the processor selects its address.
2. **Data Transfer:**
  - Direction controlled by **Read (RD)** and **Write (WR)** signals.
  - **Host Initiated:** Communication is always initiated by the host processor.
3. **Interrupt Handling:**
  - Devices can signal the processor using **interrupts** for communication.
  - The device's interrupt line is connected to the host processor's interrupt line.
4. **Bus Width:**

- Matches the **data bus width** of the host processor (e.g., 4-bit, 8-bit, 16-bit, 32-bit, etc.).

#### 5. Timing:

- Parallel communication follows **strict timing protocols** to ensure proper data transfer.



### External Communication Interfaces – RS-232, RS-422, RS-485

#### **RS-232:**

- **Legacy Serial Communication** standard by EIA, supports **point-to-point** communication.
- **Baud Rate:** Up to **19.2 Kbps**, maximum distance **50 ft**.
- Uses **DB-9/DB-25 connectors** and signals logic via **voltage levels** (+3 to +25V for 0, -3 to -25V for 1).
- Popular in industrial legacy applications, less common now due to USB, Bluetooth, etc.
- Requires **level converters** (e.g., **MAX232**) for TTL/CMOS logic compatibility.

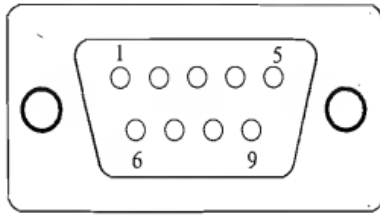
#### **RS-422:**

- Differential data communication, supports up to **100 Kbps** over **400 ft**.
- Allows **multi-drop communication** with **1 transmitter** and up to **10 receivers**.
- Requires RS-232 to RS-422 conversion for use.

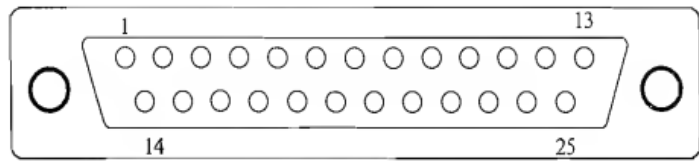
#### **RS-485:**

- Enhanced RS-422, supports **multi-drop communication** with up to **32 transmitters** and **32 receivers**.
- Uses **addressing mechanism** to identify slave devices.
- Ideal for long-distance and multi-device communication in industrial applications.





DB-9



DB-25

### RS-232 Connector Pin Details (DB-9 and DB-25)

#### Key Notes:

- **DB-9 connectors** are more common in modern systems, while **DB-25 connectors** are mostly obsolete.
- For simple data transmission, only **TXD, RXD, and GND** are essential.
- Here are all the pins for **RS-232 connectors (DB-9 and DB-25)**:

Pin Name	Pin Number (DB-9)	Pin Number (DB-25)	Description
<b>TXD</b>	3	2	Transmit Data: Sends serial data.
<b>RXD</b>	2	3	Receive Data: Receives serial data.
<b>RTS (Request to Send)</b>	7	4	Signals readiness to send data.
<b>CTS (Clear to Send)</b>	8	5	Acknowledges readiness to send/receive data.
<b>DSR (Data Set Ready)</b>	6	6	Indicates the device is ready.
<b>DTR (Data Terminal Ready)</b>	4	20	Host signals it is ready to communicate.
<b>DCD (Data Carrier Detect)</b>	1	8	Indicates the presence of a valid carrier signal.
<b>RI (Ring Indicator)</b>	9	22	Indicates an incoming call on a telephone line.
<b>GND (Ground)</b>	5	7	Signal ground: Provides common reference for signal levels.
<b>Secondary TXD (STXD)</b>	-	14	Secondary transmit data for extended configurations.
<b>Secondary RXD (SRXD)</b>	-	16	Secondary receive data for extended configurations.
<b>Secondary RTS (SRTS)</b>	-	19	Secondary request to send signal.
<b>Secondary CTS (SCTS)</b>	-	13	Secondary clear to send signal.
<b>SDCD (Secondary DCD)</b>	-	12	Secondary data carrier detect.
<b>RC (Receiver Clock)</b>	-	17	Timing signal for receiving devices.
<b>TC (Transmission)</b>	-	15	Timing signal for transmitting

<b>Clock)</b>			devices.
<b>Signal Ground (FG)</b>	-	1	Frame ground.
<b>NC (No Connection)</b>	9, 10, 11, 18, 23, 24, 25	9, 10, 11, 18, 23, 24, 25	Not connected (reserved).

### Universal Serial Bus (USB)

- **Definition:** High-speed, wired serial bus for data communication, introduced in **1995** by Intel, Microsoft, IBM, and others.
- **Topology:** Star topology with a **USB host** supporting up to **127 devices**.
- **Connectors:**
  - **Type A:** Upstream (host).
  - **Type B:** Downstream (slave).
  - Includes **Mini** and **Micro USB** for small devices.
- **Data Transmission:**
  - Packet-based, **host-initiated** with standards like **OHCI** and **UHCI**.
  - Differential signals for noise immunity; supports up to **5m cable length**.
- **Power Supply:** Provides **5V, 500mA** to connected devices.
- **Device Identification:** Uses **Product ID (PID)** and **Vendor ID (VID)** for driver identification.
- **Data Transfer Types:**
  - **Control:** Device configuration.
  - **Bulk:** Block data (e.g., printers).
  - **Isochronous:** Real-time streaming (e.g., audio).
  - **Interrupt:** Small data (e.g., mouse, keyboard).
- **Data Rates:**
  - **USB 1.0:** Low (1.5 Mbps), Full (12 Mbps).
  - **USB 2.0:** High (480 Mbps).
  - **USB 3.0:** Super Speed (4.8 Gbps).

### USB Connector Pin Details

Pin	Signal Name	Description
1	VBUS	Power Supply (+5V).
2	D-	Data Line (Negative).
3	D+	Data Line (Positive).
4	GND	Ground.

This pin configuration is applicable for **Type A**, **Type B**, **Mini USB**, and **Micro USB** connectors.

### IEEE 1394 (FireWire)

- **Definition:** High-speed, **isochronous serial communication bus** developed by **Apple in 1985**, standardized by **IEEE in 1995**.
- **Other Names:** FireWire (Apple), i.LINK (Sony), Lynx (TI).
- **Topology:** **Peer-to-peer, point-to-multipoint** (supports up to **63 devices** in a tree structure).
- **Cable Length:** Up to **15 feet**.
- **Data Rates:** **400 Mbps to 3.2 Gbps**.

- **Differential Data Transfer:** Uses **twisted-pair** cables for **better noise immunity**.
- **Connectors:**
  - **4-pin:** Data only.
  - **6-pin (Alpha):** Data + **Power (24-30V)**.
  - **9-pin (Beta):** Higher speeds, **Power (9-12V for battery devices)**.
- **Use Cases:** Digital cameras, camcorders, scanners, **direct device-to-device communication** (e.g., scanner to printer).
- **Comparison with USB:** Faster than **USB 2.0**, **no host required**, but **costlier**.

## Wireless Communication Interfaces

### **Infrared (IrDA):**

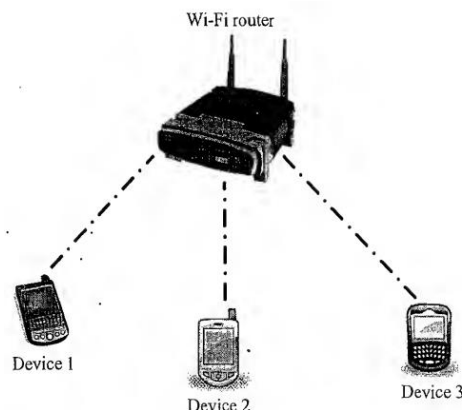
- **Half-duplex, line-of-sight wireless communication.**
- **Range: 10 cm – 1 m**, extendable with higher power.
- **Data Rates: 9.6 kbps – 16 Mbps** (SIR, MIR, FIR, VFIR, UFIR).
- Uses **IR LED for transmission** and **photodiode for reception**.
- **Used in:** TV remotes, file transfer, mobile phones (before Bluetooth).

### **Bluetooth (BT):**

- **Short-range, low-power wireless communication (2.4 GHz RF band).**
- **Range: ~30 feet**, **Data Rate: Up to 1 Mbps**.
- **Supports: Point-to-point and point-to-multipoint (Piconet, max 7 slaves).**
- **Profiles:** GAP (connection setup), SPP (serial data), FTP (file transfer), HID (keyboards/mice).
- **Popular in:** Phones, headsets, speakers, wearables.

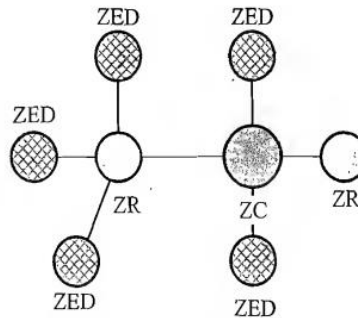
### **Wi-Fi (IEEE 802.11):**

- **Wireless networking communication, IP-based.**
- **Range: 100-300 feet**, **Data Rates: 1 Mbps – 150 Mbps**.
- Requires **Wi-Fi Router** for managing connections.
- **Security Protocols:** WEP, WPA, WPA2.
- **Used in:** Internet access, smart devices, IoT.



### ZigBee (IEEE 802.15.4):

- **Low-power, low-data-rate WPAN (Wireless Personal Area Network).**
- **Range: Up to 100 meters, Data Rate: 20 – 250 Kbps.**
- **Network Roles:**
  - **Coordinator (ZC):** Initiates and manages the network.
  - **Router (ZR):** Passes data between devices.
  - **End Device (ZED):** Communicates but doesn't route.
- **Used in:** Home automation, smart meters, IoT, industrial monitoring.



Zigbee Network Model

### GPRS (General Packet Radio Service):

- **Packet-switched mobile data communication over GSM networks.**
- **Max Data Rate: 171.2 kbps.**
- **Uses:** IP, PPP, X.25 protocols.
- **Replaced by:** EDGE, HSDPA, LTE for higher speeds.
- **Used in:** Mobile internet, GPS trackers, M2M communication.



Infrared (IrDA) Communication



Bluetooth (BT) Communication



Wi-Fi Communication



ZigBee Communication



GPRS Communication

### Embedded Firmware

- **Definition:** Embedded firmware consists of **program instructions** and **configuration settings** stored in an embedded system's memory.
- **Development Methods:**
  1. **High-Level Languages (C/C++)** – Uses **IDEs** like Keil for coding, compiling, debugging, and simulation.
  2. **Assembly Language** – Uses **processor-specific instructions** but is **harder to write and debug**.
- **HEX File Creation:** Converts code into machine-readable **binary format** using **cross-compilers** or **vendor utilities**.
- **Recommended Approach:** **High-level languages** are preferred due to **ease of coding, portability, faster debugging, and team collaboration**.
- **Control Algorithm Approaches:**
  1. **Super Loop (Infinite Loop)** – Runs continuously like `while(1){}`.
  2. **Task Scheduler (RTOS/GPOS)** – Splits functions into tasks managed by an operating system.



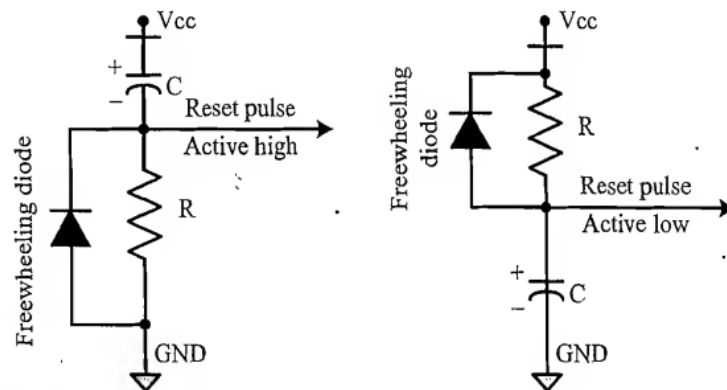
## 1.10 Other system components, PCB and passive components

### Other System Components – Key Points

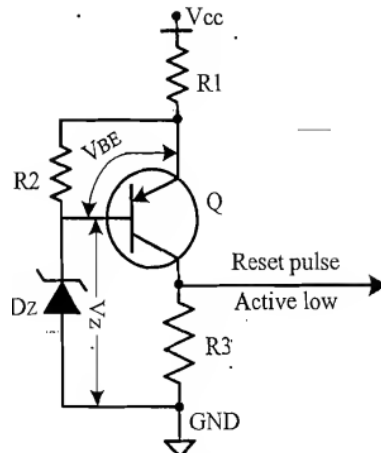
- **Definition:** Essential **circuits/ICs** required for proper **functioning** of an embedded system.
- **Examples:**
  - **Watchdog Timer** – Prevents system hang-ups.
  - **Reset IC/Circuit** – Ensures proper system startup.
  - **Brown-out Protection** – Prevents malfunction during power drops.
- **Integration:** Some **controllers/SoCs** have these built-in; others require **external components**.
- **Additional Components:** **Level translators, specific function ICs**, and interface circuits as needed.

### Essential System Components – Key Points

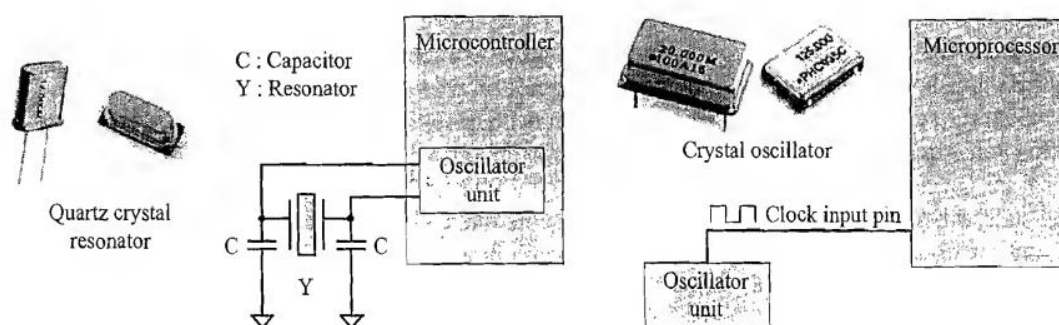
- **Reset Circuit:** Ensures proper startup by resetting the processor to a known state at power-on. Uses **passive RC circuits or Reset ICs**.



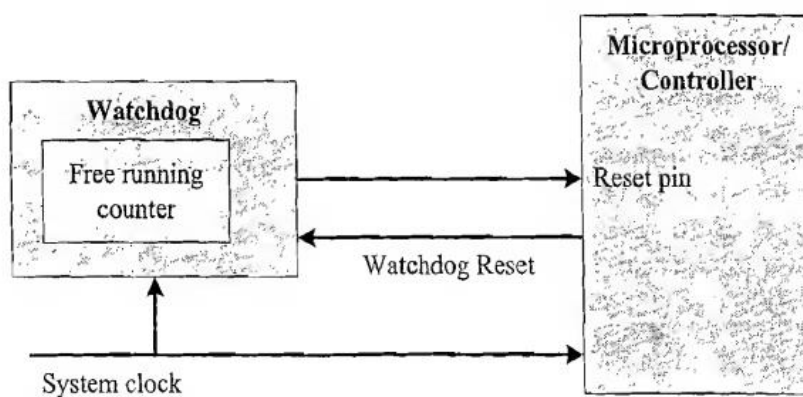
- **Brown-out Protection:** Prevents malfunction when voltage drops below a threshold. Implemented using **Zener diodes, transistors, or supervisor ICs**.



- **Oscillator Unit:** Generates clock signals for processor operation. Uses **quartz crystals, ceramic resonators, or external oscillator chips**.



- **Real-Time Clock (RTC):** Maintains system time, date, and alarms. Essential for **OS-based systems** for time synchronization.
- **Watchdog Timer:** Monitors firmware execution and resets the system if it **hangs or malfunctions**. Built-in in most processors or available as **external ICs**.



### PCB and Passive Components – Key Points

- **Printed Circuit Board (PCB):** The **foundation** of an embedded system, designed based on schematics and used for **component mounting and firmware testing**.
- **Passive Components:** Essential **supporting elements** like **resistors, capacitors, diodes**, etc., ensuring stable operation.
  - **Example:** Regulator IC with **filter capacitors** for a stable, ripple-free power supply.

---

### Outcomes

---

At the end of the module, students will be able to:

CO-1: Explain characteristics of Embedded System design [L2]

### TEXT BOOKS:

Shibu K V, “Introduction to Embedded Systems”, Second Edition, McGraw Hill Education

### Reference Books/ Link

NPTL Lectures: <https://nptel.ac.in/courses/108102045>

Embedded Systems, IIT Delhi, Prof. Santanu Chaudhary