Embedded Application Development
NEP- Semester - IV

Year	II	CourseCode:CSCS209 Course Title: Embedded Application Development	Credits	4
Sem.	IV		Hours	75
			Category	C
Course Prerequisites, if Any	• Microprocessor & Microcontrollers introduction • Assembly Language Programming • Operating System and Computer Organization Concepts			
Internal Assessment Marks:25	EndSemesterMarks:75	Duration of ESA (Theory): 03 hrs. Duration of ESA(Practical):03hrs.		
Course Outcomes	• Understand the basics of Embedded Systems • Appreciate the application domains of Embedded Systems • Gain proficiency in programming embedded systems • Exploreinterfacingtechniquesforsensors,actuators,andotherperipheraldevices commonly used in embedded applications • Developskillsindesigning,implementing,anddebuggingembeddedsoftware			

Unit No.	Course Content	Hours
	Theory Component	
Unit-I	**Introduction** Embedded Systems and General-purpose Computer Systems – History – Classifications – Applications – Purpose of Embedded Systems – Characteristics and Quality Attributes	9
Unit-II	**Embedded Systems** Application specific – washing machine – domain specific – automotive Embedded Hardware: Memory – I/O – Interrupt – Processors – External peripherals Peripherals: Control and Status Registers– Device Driver– Timer Driver– Watchdog Timers	9
Unit-III	**Microcontrollers** Microcontrollers and Embedded processors –Overview of8051 family.8051 hardware – I/O pins – Ports – Circuits – External Memory Programming: Data Types– I/O Programming–Logicoperations–Data conversion Programs	9
Unit-IV	**DesigningEmbeddedSystemwith8051Microcontroller** Factorstobeconsideredinselectingacontroller–8051Microcontroller–Designingwith8051 Programming: Structure of embedded program – infinite loop – compiling, linking & debugging	9

Unit-V	**Real Time Operating System(RTOS)** Operating system basics–Types of OS–Real-Time Characteristics–Selection Process of an RTOS Design and Development: Embedded system development Environment – IDE – types of file generated, disassembler – de-compiler – simulator – emulatoranddebugging,embeddedproductdevelopmentlife-cycle,trends in embedded industry	9
Practical Component		
Exercises	1. Configuretimercontrolregistersof8051anddevelopa program to generate given time delay 2. Port I/O:Useoneofthefourportsof8051forO/P 3. interfaced to eight LED's. Simulate binary counter (8 bit) on LED's 4. Serial I/O: Configure 8051 serial port for asynchronous serial communication with serial port of PCexchange text messages	30
	to PC and display on PC screen. Signify end of message by carriage return 5. Interface 8051 with D/A converter and generate square wave of given frequency on oscilloscope 6. Interface the microcontroller with external devices (e.g., sensors, displays, or other microcontrollers) using serial communication. Implement simple data exchange protocols and verify communication 7. Generate PWM signals to control the brightness of LEDs or the speed of a motor. Experiment with different duty cycles and frequencies 8. Write programs to store and retrieve data from non-volatile memory (e.g., EEPROM or Flash). Implement dynamic memory allocation techniques using RAM	
Recommended Learning Resources		
Print Resources	1. Shibu KV, "Introduction to Embedded Systems "Second Edition, Tata McGraw Hill, 2017. 2. Rajkamal, "Embedded Systems-Architecture, Programming and Design", ThirdEdition,McGrawHillEducation,2008.	
SyllabusDesign:Dr.S.K.V.Jayakumar,Professor,PUDoCS		

Unit-1 Introduction

Embedded Systems and General-purpose Computer Systems – History – Classifications – Applications – Purpose of Embedded Systems – Characteristics and Quality Attributes.

What is Embedded System?

An Electronic / Electromechanical system which is designed to perform a specific function and is a combination of both hardware and firmware (Software)

E.g. Electronic Toys, Mobile Handsets, Washing Machines, Air Conditioners, Automotive Control Units, Set Top Box, DVD Player etc…

Embedded Systems are:

- Unique in character and behavior
- With specialized hardware and software

General-Purpose Computing System vs. Embedded System

General-Purpose Computing System	Embedded System
A system which is a combination of generic hardware and General Purpose Operating System for executing a variety of applications	A system which is a combination of special purpose hardware and embedded OS for executing a specific set of applications
Contains a General-Purpose Operating System (GPOS).	It may or may not contain an operating system for functioning.
Applications in a General-Purpose Computing System are alterable (programmable) by the user. It is possible for the end user to re-install the Operating System and add or remove user applications.	The firmware of an Embedded System is pre-programmed and is non-alterable by the end-user
Performance is the key deciding factor in selecting a General-Purpose Computing System. The rule followed is: "Faster is Better."	Application specific requirements (like performance, power requirements, memory usage etc) are the key deciding factors
Less or not at all tailored toward reduced operating power requirements, but they offer options for different levels of power management.	Highly tailored to take advantage of the power-saving modes supported by hardware and the Operating System.
Response requirements are not time critical	Execution behavior in Embedded Systems is deterministic for certain types of systems, such as Hard Real-Time systems.
Execution behavior if the systems need not be deterministic	Execution behavior is deterministic for certain type of embedded systems like „Hard Real Time" systems

History of Embedded Systems

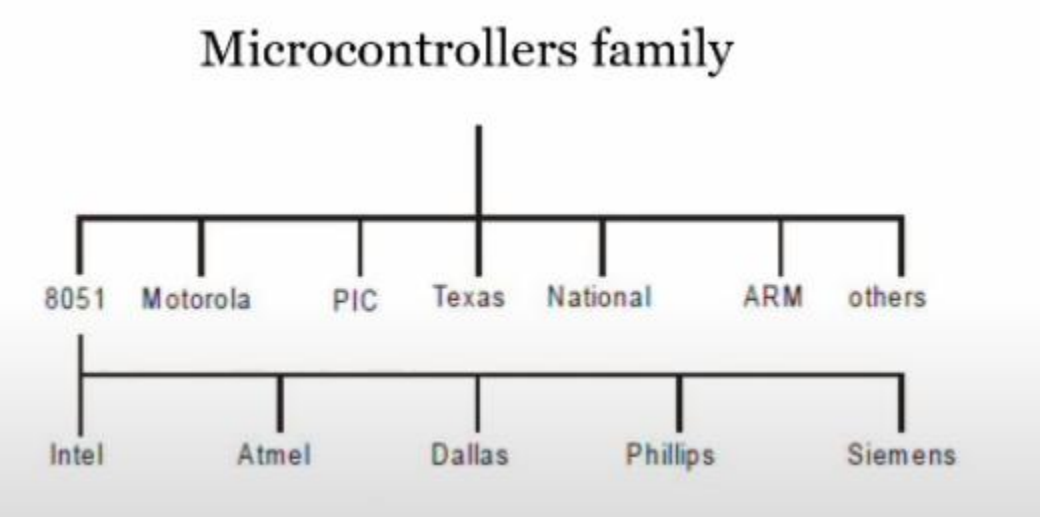
By the late 1960s and early 1970s, the price of integrated circuits dropped and usage surged. The first microcontroller was developed by **Texas Instruments** in 1971. The TMS1000 series, which became commercially available in 1974, contained a 4-bit processor, read-only memory and random-access memory.

Intel Microcontrollers 8048 was introduced in 1976 and was the first of Intel's Micro Controller. It was used as the processor in the PC Keyboard of IBM.

In 1978 Motorola Controllers was introduced as 6801, and 6811 Micro controller introduced at 1985 (8-bit CPU, 8K ROM, 256 Bytes RAM, 512Bytes EEPROM)

PIC(Programmable Interface Controller) is a family of micro controllers made by Microchip Technology. The original PIC 1650 was developed by General Instruments.

Atmel introduced its first Flash Microcontroller AT89C51 in 1993, based on the 8051 core. Low power, high performance CMOS 8-bit MC with 4Kbytes of flash programmable and erasable read only memory.



Classification based on Complexity & Performance

Small Scale: The embedded systems built around low performance and low cost 8 or 16 bit microprocessors/ microcontrollers. It is suitable for simple applications and where performance is not time critical. It may or may not contain OS.

Medium Scale: Embedded Systems built around medium performance, low cost 16 or32 bit microprocessors / microcontrollers or DSP (Digital Signal Processing). These are slightly complex in hardware and firmware. It may contain GPOS (General-Purpose Operating System) /RTOS (Real-Time Operating System, example **FreeRTOS**).

Large Scale/Complex: Embedded Systems built around high performance 32 or 64 bit RISC processors/controllers, RSoC or multi-core processors and PLD. It requires complex hardware and software. These system may contain multiple processors/controllers and co-units/hardware accelerators for offloading the processing requirements from the main processor. It contains RTOS for scheduling, prioritization and management.

3. Embedded Systems-Classification Based on deterministic behavior: It is applicable for Real Time systems. The application/task execution behavior for an embedded system can be either deterministic or non-deterministic
These are classified into two types

a. Soft Real time Systems: Missing a deadline may not be critical and can be tolerated to a certain degree

b. Hard Real time systems: Missing a program/task execution time dead line can have catastrophic consequences (financial, human loss of life, etc.)

4. Embedded Systems-Classification Based on Triggering: These are classified into two types

a) Event Triggered: Activities within the system(e.g., taskrun-times) are dynamic and depend upon occurrence of different events .

b) Time triggered: Activities within the system follow a statically computed schedule (i.e., they are allocated time slots during which they can take place) and thus by nature are predictable.

Major Application Areas of Embedded Systems

1. **Consumer Electronics**: Camcorders, Cameras, etc.

   Embedded systems are used in devices like **camcorders, digital cameras, and gaming consoles** to enhance performance and optimize power consumption

2. **Household Appliances**: Television, DVD players, Washing machine, Fridge, Microwave Oven, etc.

   Appliances like **TVs, DVD players, washing machines, refrigerators, and microwave ovens** use embedded systems for automation, user control, and energy efficiency.

3. **Home Automation and Security Systems**: Air conditioners, Sprinklers, Intruder detection alarms, Closed Circuit Television (CCTV) Cameras, Fire alarms, etc.

   Embedded systems enable **smart home technologies**, such as **air conditioners, sprinklers, CCTV cameras, fire alarms, and intrusion detection alarms**, for enhanced safety and convenience.

4. **Automotive Industry**: Anti-lock Braking Systems (ABS), Engine Control, Ignition Systems, Automatic Navigation Systems, etc.

   Modern vehicles rely on embedded systems for Anti-lock Braking Systems (ABS), Engine Control Units (ECUs), Ignition Systems, and GPS-based Navigation Systems to improve safety and performance.

5. **Telecom**: Cellular Telephones, Telephone switches, Handset Multimedia Applications, etc.

   Devices like **mobile phones, telephone switches, and multimedia handsets** use embedded systems to ensure efficient communication and signal processing.

6. **Computer Peripherals**: Embedded systems control printers, scanners, and fax machines, allowing them to perform automated tasks with high accuracy.

7. **Computer Networking Systems**:

   Network equipment like **routers, switches, hubs, and firewalls** use embedded systems for managing data traffic and ensuring secure communication.

8. **Healthcare**:

   Medical equipment such as **ECG and EEG machines, MRI scanners, and infusion pumps** use embedded systems for real-time monitoring and diagnosis.

9. **Measurement & Instrumentation**:

   Devices like **digital multimeters, digital storage oscilloscopes (CROs), and logic analyzers** use embedded systems for precision measurement and data analysis.

10. **Banking & Retail**:

    ATMs, currency counters, and Point-of-Sale (POS) systems use embedded technology for secure and fast financial transactions.

11. **Card Readers**:

    Barcode scanners, smart card readers, and handheld payment devices use embedded systems for authentication and secure data processing.

Each Embedded Systems is designed to serve the purpose of anyone or a combination of the following tasks.

- Data Collection/Storage/Representation
- Data Communication
- Data(Signal)Processing
- Monitoring
- Control
- Application Specific User Interface

Purpose of Embedded Systems

1. Data Collection/Storage/Representation:-

- Performs acquisition of data from the external world.
- The collected data can be either analog or digital
- Data collection is usually done for storage, analysis, manipulation and transmission
- The collected data may be stored directly in the system or may be transmitted to some other systems or it may be processed by the system or it may be deleted instantly after giving a meaningful representation

2. Data Communication:-

Embedded Data communication systems are deployed in applications ranging from complex satellite communication systems to simple home networking systems

Embedded Data communication systems are dedicated for data communication

The data communication can happen through a wired interface (like Ethernet, RS-232C/USB/IEEE1394 etc) or wireless interface (like Wi-Fi, GSM,/GPRS, Bluetooth, ZigBee etc) Network hubs, Routers, switches, Modem set care
Typical examples for dedicated data transmission embedded systems

3. Data (Signal) Processing

Embedded systems with Signal processing functionalities are employed in applications demanding signal processing like Speech coding, synthesis, audio video codec, transmission applications etc
Computational intensive systems

Employs Digital Signal Processors(DSPs)

4. Monitoring

- Embedded systems in this category are specifically designed for **monitoring purposes**.

- They are used to **determine the state of certain variables** using input sensors.
- They **cannot impose control** over the variables.
- A typical example is the **Electrocardiogram (ECG) machine**, which monitors a patient's heartbeat.
- The **sensors used in ECG** are different **electrodes connected to the patient's body**.
- Measuring instruments like **Digital CRO, Digital Multimeter, and Logic Analyzer**, used in control and instrumentation applications, are also examples of embedded systems for monitoring purposes.

---

5. Control

- Embedded systems with control functionalities are used to **impose control over variables** based on changes in input variables.
- These systems contain **both sensors and actuators**.
- **Sensors** are connected to the **input port** to detect changes in the environmental or measured variable.
- **Actuators** are connected to the **output port** and controlled based on input changes to adjust the controlled variable to a specified range.
- A common example is an **air conditioner**, which controls room temperature.
- The air conditioner contains:
    - A **temperature-sensing element (sensor)**, such as a **thermistor**.
    - A **handheld unit** to set the desired temperature.
    - An **air compressor unit**, which acts as the actuator.
    - The **compressor is controlled** based on the difference between the **current room temperature** and the **user-set desired temperature**.

Application-Specific User Interface

- Embedded systems designed for a **specific application**.
- Contains an **Application-Specific User Interface** (rather than a general standard UI) such as **keyboards, display units, etc.**
- Aimed at a **specific target group of users**.
- Examples include:
    - **Mobile handsets**
    - **Control units in industrial applications**

Characteristics of Embedded Systems

Embedded systems possess certain **specific characteristics**, and these are **unique** to each embedded system:

1. **Application and Domain-Specific**
2. **Reactive and Real-Time**
3. **Operates in Harsh Environments**
4. **Distributed**
5. **Small Size and Weight**
6. **Power Concerns**
7. **Single-Functioned**
8. **Complex Functionality**
9. **Tightly-Constrained**
10. **Safety-Critical**

---

1. Application and Domain-Specific

- Each **embedded system** is designed to perform a **specific function** and cannot be used for any other purpose.
- **Example:**
    - The **embedded control unit of a microwave oven** cannot be replaced with an **air conditioner's control unit** because both are specifically designed to perform different tasks.

2. Reactive and Real-Time

- **Embedded systems (E.S)** constantly interact with the real world through **sensors** and **user-defined input devices** connected to the input port.
- Any **changes in the real world** are detected by sensors/input devices in **real-time**, and the control algorithm reacts accordingly.
- E.S **produce output changes** in response to input changes, making them **reactive systems**.
- **Real-time system operation** means the system's response time must be **deterministic** (i.e., it must respond in a known amount of time).
- **Example:**
    - **Mission-critical systems** like **flight control systems** and **Anti-lock Braking Systems (ABS)** are real-time systems.

---

3. Operates in Harsh Environments

- The **design** of an embedded system should consider the **operating conditions** of its deployment area.
- **Example:**

- If the system is deployed in a **high-temperature zone**, all components should be **high-temperature grade**.
- Systems in **high-shock environments** should have **shock absorption mechanisms**.

4. Distributed

- **Embedded systems may be part of a larger system** and function as interconnected units.
- **Multiple embedded systems** together form a **single large control unit**.
- **Example:**
  - An **automatic vending machine** consists of a **card reader, vending unit, and other independent embedded units**, all working together for vending operations.

5. Small Size and Weight

- **Product aesthetics** (size, weight, shape, style) are important in product selection.
- **Compact devices** are more **convenient** to handle than bulky products.

6. Power Concerns

- **Power management** is an essential factor in embedded system design.
- The system should be designed to **minimize heat dissipation** for better efficiency.

7. Single-Functioned

- Embedded systems are **dedicated to performing a single function** efficiently.

8. Complex Functionality

- Some embedded systems **run sophisticated or multiple algorithms** to perform complex tasks.

9. Tightly-Constrained

- Embedded systems must meet **strict constraints** such as **low cost, low power consumption, small size, and fast processing**.

10. Safety-Critical

- **Safety-critical embedded systems** must **not endanger human life or the environment**.
- **Example:**
  - **Medical devices**, **automotive safety systems**, and **nuclear plant control systems**.

Quality Attributes of Embedded Systems

2.1. Performance

- The system should execute tasks **quickly and efficiently**.
- Example: A **real-time video encoder** in a **CCTV system** must process frames with minimal delay.

2.2. Real-Time Responsiveness

- The system must meet **strict timing constraints** for **critical applications**.
- Example: **Anti-lock Braking System (ABS)** must respond **within milliseconds**.

2.3. Reliability & Fault Tolerance

- Must work **without failure** for extended periods, often in **harsh environments**.
- Example: A **satellite control system** should function for **years without rebooting**.

2.4. Power Efficiency

- Optimized for **low power consumption**, especially in **battery-powered devices**.
- Example: **IoT sensors** should last **years on a small battery**.

2.5. Security

- Embedded systems need **protection against cyber threats and unauthorized access**.
- Example: **Smart home security systems** use **encryption** to prevent hacking.

2.6. Maintainability & Upgradability

- Should support **firmware updates** for bug fixes and new features.
- Example: **Smartphones receive OTA (Over-The-Air) firmware updates**.

2.7. Scalability

- Ability to **expand functionality** without redesigning the entire system.
- Example: A **modular embedded system** can add support for **new sensors**.

2.8. Cost-Effectiveness

- Must balance **performance and cost** for mass production.
- Example: **Consumer electronics (e.g., smart TVs)** use cost-optimized embedded chips.

Unit - II

Application specific – washing machine – domain specific – automotive Embedded Hardware: Memory – I/O – Interrupt – Processors – External Peripherals: Control and Status Registers– Device Driver– Timer Driver– Watchdog Timers

APPLICATION-SPECIFIC EMBEDDED SYSTEM: WASHING MACHINE

A **washing machine** is an example of an **application-specific embedded system**, meaning it is **designed for a specific function**—washing clothes. It cannot be reprogrammed for other tasks like an air conditioner or microwave oven.

How Embedded Systems Work in a Washing Machine

Key Components of an Embedded System in a Washing Machine

An embedded system in a washing machine consists of the following components:

- **Microcontroller**: Acts as the brain of the washing machine, controlling all operations.
- **Sensors**: Used to monitor various parameters (e.g., water level, temperature, load balance).
- **Actuators**: Components like motors and valves that execute operations.
- **Memory (ROM & RAM)**: Stores the washing programs and temporary data.
- **Display & User Interface**: Enables users to select modes and receive status updates.
- **Communication Interfaces**: Some modern machines support IoT (Wi-Fi/Bluetooth) for remote control.

Working of an Embedded System in a Washing Machine

The embedded system ensures efficient washing by controlling different stages:

Step 1: User Input & Program Selection

- The user selects the desired wash cycle (e.g., delicate, normal, heavy).
- The microcontroller reads the input and loads the corresponding washing program from memory.

Step 2: Water Intake Control

- The water level sensor detects water levels.
- The microcontroller signals the inlet valve to open and fill the drum to the required level.

Step 3: Temperature Regulation

- If the program requires warm water, the temperature sensor monitors the water.
- The heating element activates if needed and stops once the set temperature is reached.

Step 4: Motor Control for Washing

- The microcontroller controls the motor, adjusting speed and direction for efficient washing.
- Load sensors ensure even distribution of clothes to prevent imbalance.

Step 5: Rinsing Process

- Once washing is complete, the system drains dirty water.
- Fresh water is added, and the drum rotates to rinse clothes.

Step 6: Spinning & Drying

- The drum spins at high speed to remove excess water.
- Sensors monitor vibration and balance, adjusting speed if necessary.

Step 7: Completion & User Notification

- The embedded system signals the end of the cycle.
- The display shows the remaining time, and a buzzer or LED indicates completion.

Why It's an Application-Specific Embedded System?

- **Designed only for washing clothes** (not programmable for other tasks).
- **Contains pre-defined wash programs** controlled by an **embedded microcontroller**.
- **Uses specific sensors and actuators** to automate washing.
- **Ensures safety & efficiency** with programmed operations.

Thus, the washing machine is a perfect example of an **application-specific embedded system** that automates the washing process with **minimal user intervention**.

Automotive Embedded Hardware: Overview

Automotive embedded systems rely on specific hardware components to ensure real-time, efficient, and reliable functioning of vehicle systems. These systems involve critical components such as **memory**, **I/O devices**, **interrupts**, **processors**, and **external peripherals** to perform a range of tasks from controlling engine functions to enhancing user interface experiences.

---

1. Memory

- Automotive embedded systems use various **memory types** to store **data, firmware, and operating systems**:
    - **Flash memory** is used to store **firmware** or **program code** because it is non-volatile.
    - **RAM (Random Access Memory)** is used for temporary data storage while processing information.
    - **EEPROM** is used to store critical, non-volatile data, such as calibration settings.

❖ The memory management function of an RTOS kernel is slightly different compared to the General Purpose Operating Systems

❖ The memory allocation time increases depending on the size of the block of memory needs to be allocated and the state of the allocated memory block (initialized memory block consumes more allocation time than un- initialized memory block)

❖ Since predict table timing and deterministic behavior are the primary focus for an RTOS, RTOS achieves this by compromising the effectiveness of memory allocation

❖ RTOS generally uses ‘block’ based memory allocation technique, instead of the usual dynamic memory allocation techniques used by the GPOS.

❖ RTOS kernel uses blocks of fixed size of dynamic memory and the block is allocated for a task on a need basis. The blocks are stored in a ‘Free buffer Queue’.

❖ Most of the RTOS kernels allow tasks to access any of the memory blocks without any memory protection to achieve predictable timing and avoid the timing overheads

❖ RTOS kernels assume that the whole design is proven correct and protection is unnecessary. Some commercial RTOS kernels allow memory protection as optional and the kernel enters a fail-safe mode when an illegal memory access occurs

❖ The memory management function of an RTOS kernel is slightly different compared to the General Purpose Operating Systems

❖ A few RTOS kernels implement Virtual Memory concept for memory allocation if the system supports secondary memory storage (like HDD and FLASH memory).

❖ In the ‘block’ based memory allocation, a block of fixed memory is always allocated for tasks on need basis and it is taken as a unit. Hence, there will not be any memory fragmentation issues.

❖ The memory allocation can be implemented as constant functions and thereby it consumes fixed amount of time for memory allocation. This leaves the deterministic behavior of the RTOS kernel untouched.

---

2. I/O (Input / Output)

- I/O interfaces are used to exchange data between **sensors, controllers**, and **actuators**. They enable communication between the embedded system and its environment.
- For example, **inputs** could include data from **sensors** like speedometers, temperature sensors, and pressure sensors, while **outputs** could control **motors**, **lights**, or **displays**.

---

3. Interrupts

- **Interrupts** are signals that inform the processor that an event requiring immediate attention has occurred.
    - **Hardware interrupts** are triggered by external devices (e.g., sensor data arriving).
    - **Software interrupts** are triggered by software routines for managing tasks.
- **Interrupt handling** is critical for systems like **airbags, ABS (Anti-lock Braking System)**, where real-time response is needed to ensure safety.

Interrupt Handling

Interrupts inform the processor that an external device or an associated task requires immediate attention of the CPU.

Interrupts can be either Synchronous or Asynchronous.

Interrupts which occurs in sync with the currently executing task is known as Synchronous interrupts. Usually the software interrupts fall under the Synchronous Interrupt category. Divide by zero, memory segmentation error etc are examples of Synchronous interrupts.

For synchronous interrupts, the interrupt handler runs in the same context of the interrupting task.

Asynchronous interrupts are interrupts, which occurs at any point of execution of any task, and are not in sync with the currently executing task.

The interrupts generated by external devices (by asserting the Interrupt line of the processor/controller to which the interrupt line of the device is connected) connected to the processor/controller, timer overflow interrupts, serial data reception/ transmission interrupts etc are examples for asynchronous interrupts.

For asynchronous interrupts, the interrupt handler is usually written as separate task (Depends on OS Kernel implementation) and it runs in a different context. Hence, a context switch happens while handling the asynchronous interrupts.

Priority levels can be assigned to the interrupts and each interrupts can be enabled or disabled individually.

Most of the RTOS kernel implements 'Nested Interrupts' architecture. Interrupt nesting allows the pre-emption (interruption) of an Interrupt Service Routine (ISR), servicing an interrupt, by a higher priority interrupt.

---

4. Processors

- Automotive embedded systems typically use **microcontrollers (MCUs)** or **microprocessors (MPUs)** to execute tasks.
    - **MCUs** are preferred for simpler, low-power tasks such as controlling the engine, lights, or sensors.
    - **MPUs** are more powerful and are used in systems requiring high processing power, such as navigation or infotainment systems.
- **Dual-core processors** are sometimes used for separation of critical tasks (e.g., braking systems) from non-critical tasks (e.g., entertainment systems).

5. External Peripherals

- **External peripherals** include devices that interface with the microcontroller to support communication and additional functionalities.
    - **Sensors**: Measure various vehicle parameters (e.g., speed, fuel level, exhaust temperature).
    - **Actuators**: Control mechanical elements like the throttle or brakes.
    - **Displays and User Interfaces**: Show vehicle status or provide feedback to the driver.
    - **Communication devices**: Can include **CAN (Controller Area Network)** bus systems for vehicle-to-vehicle or vehicle-to-sensor communication.

6. Control and Status Registers

- **Control Registers** manage the operation of hardware components (e.g., turning on or off a device, enabling a timer, or initiating communication).
- **Status Registers** store information about the current state of devices (e.g., whether an I/O device is ready to send or receive data).

7. Device Driver

- A **device driver** is software that allows the embedded system to interact with external hardware peripherals.
    - It **translates** requests from the software (e.g., sensors, actuators) into instructions that hardware can understand.
    - Drivers ensure **efficient and reliable** communication between the system's hardware and software layers.

8. Timer Driver

- **Timer drivers** manage **timers** that provide accurate time-related operations in embedded systems.
- **Timers** are used in automotive applications to manage **delays** between tasks (e.g., engine control cycles), and **periodic events** (e.g., checking sensor data at regular intervals).

9. Watchdog Timers

- **Watchdog timers** monitor the health of embedded systems.
    - They are used to detect **malfunctions** by resetting the system if it fails to operate correctly within a defined time period.

- For example, if the engine control system becomes unresponsive, the **watchdog timer** will reset it, ensuring the system returns to a known state.

In an **Embedded System**, external peripherals are hardware components connected to the microcontroller or microprocessor to enhance its functionality. These peripherals require **Control and Status Registers (CSR)** for interaction and efficient communication.

1. Control and Status Registers (CSR)

Control and status registers are memory-mapped registers that enable communication between the processor and external peripherals. They help in configuring devices, monitoring their status, and controlling operations.

Types of Registers:

- **Control Registers**: Used to configure the operation of peripherals (e.g., enabling/disabling interrupts).
- **Status Registers**: Indicate the current state of the peripheral (e.g., flag for data ready).
- **Data Registers**: Store data to be transmitted or received.

2. Device Driver

A **device driver** is software that acts as an interface between the operating system (or firmware) and the external peripheral hardware. It provides an abstraction layer for communication and manages hardware operations.

Functions of Device Drivers in Embedded Systems:

- Initialization of hardware components
- Configuring and managing registers
- Handling interrupts and events
- Enabling communication protocols (I2C, SPI, UART)

3. Timer Driver

A **Timer Driver** is responsible for managing hardware timers in an embedded system. Timers are essential for time-based operations such as generating delays, scheduling tasks, and measuring time intervals.

Types of Timers:

- **General-purpose timers**: Used for event counting and time delays.
- **PWM Timers**: Used for Pulse Width Modulation in motor control and signal generation.

Timer Driver Functions:

- Start/Stop timer
- Set time duration
- Generate interrupts when the timer expires

4. Watchdog Timers (WDT)

A **Watchdog Timer (WDT)** is a special hardware timer designed to reset the system if it encounters a malfunction or unresponsive state. It prevents system crashes due to software bugs or external disturbances.

Working Principle:

- The system must reset the watchdog timer periodically.
- If the system fails to reset it within a predefined time, the watchdog resets the system.

Applications of Watchdog Timers:

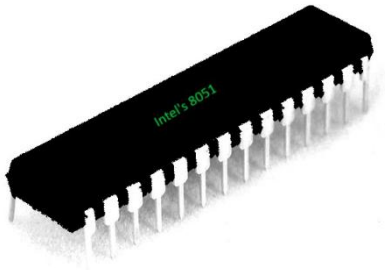
- Preventing system hang-ups
- Ensuring system reliability in critical applications (e.g., automotive, industrial automation)

Unit III - Microcontrollers

Microcontrollers and Embedded processors –Overview of 8051 family. 8051 hardware – I/O pins – Ports – Circuits – External Memory Programming: Data Types– I/O Programming–Logic operations–Data conversion Programs

Overview of the 8051 Microcontroller Family

The **8051 microcontroller** is an 8-bit microcontroller developed by **Intel in 1981** for embedded system applications. It belongs to the **MCS-51 family** and is widely used due to its simple architecture, reliability, and ease of programming. Many semiconductor manufacturers, such as **Atmel, NXP, Silicon Labs, and STMicroelectronics**, have developed derivatives of the 8051 with enhancements like additional memory, peripherals, and power-saving modes.



---

1. Features of the 8051 Microcontroller

- **8-bit CPU** (Processes 8-bit data at a time)
- **4KB ROM (Programmable Memory)**
- **128 bytes RAM (Internal Data Memory)**
- **32 I/O pins** (Organized in **4 ports: P0, P1, P2, P3**)
- **Two 16-bit timers/counters** (Timer 0 and Timer 1)
- **5 Interrupt sources** (Two external, two timer, one serial)
- **Full Duplex UART for Serial Communication**
- **Supports Boolean Processor** (Bit-addressable RAM and Registers)

8051 Family Variants

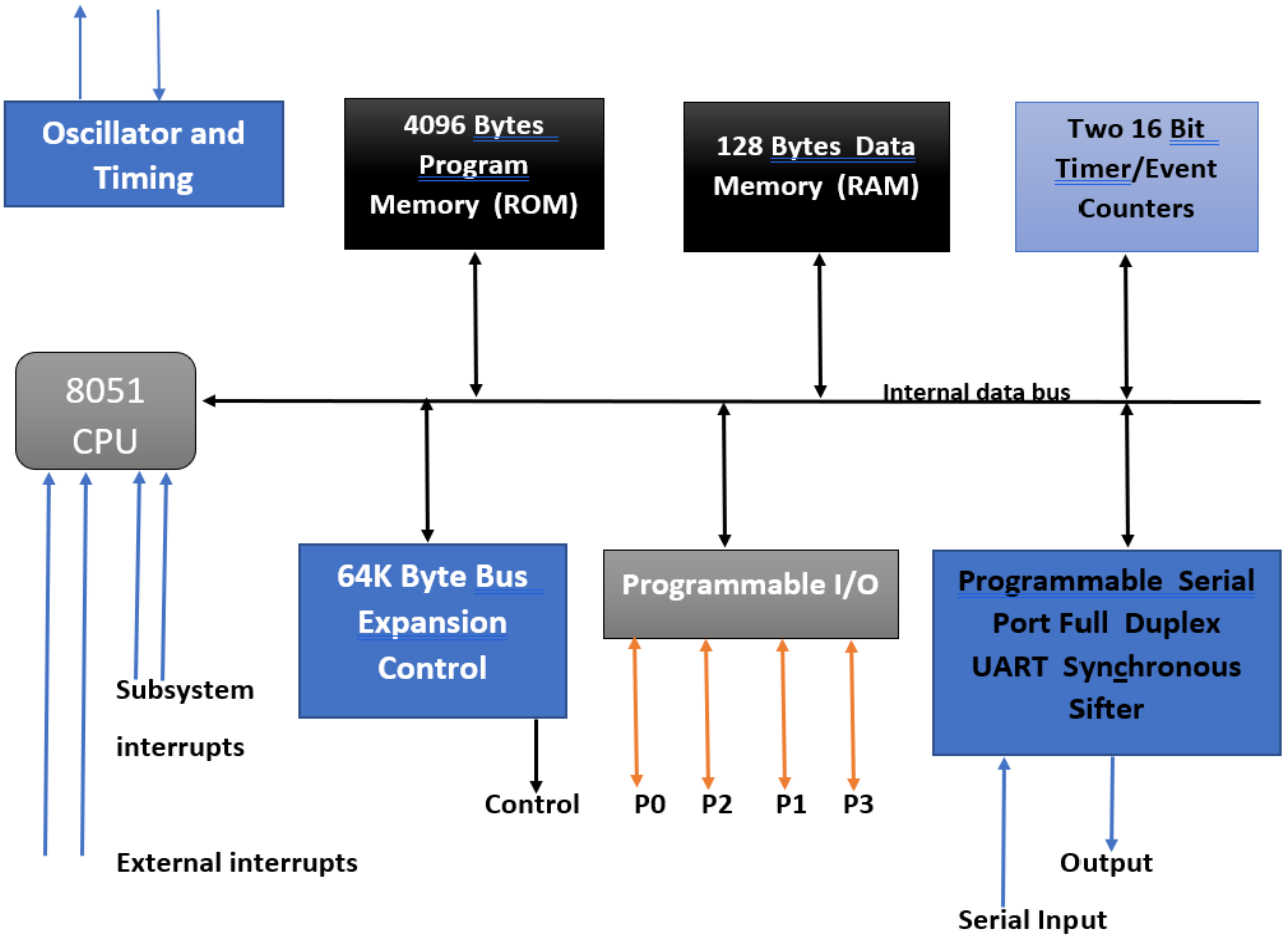
Microcontroller	Manufacturer	Features
Intel 8051	Intel	Basic 8051 microcontroller with 4KB ROM, 128B RAM
AT89C51	Atmel	Flash memory instead of ROM
AT89S51	Atmel	In-System Programmable (ISP)
P89V51RD2	NXP	More RAM and faster operation
DS89C420	Dallas Semiconductor	High-speed variant with additional features

---

2. 8051 Hardware Architecture

The 8051 microcontroller consists of several hardware components:

Block Diagram of 8051




```
+-------------------------------+
|         8-bit CPU             |
|-------------------------------|
|         Program Memory (ROM)  |
|-------------------------------|
|         Data Memory (RAM)     |
|-------------------------------|
|         I/O Ports (P0 - P3)   |
|-------------------------------|
|         Timers (T0, T1)       |
|-------------------------------|
|         Serial Communication  |
|-------------------------------|
|         Interrupt Controller  |
+-------------------------------+
```

Key Components

1. **CPU (Central Processing Unit)**
   - Executes instructions stored in ROM
   - Manages communication between different components
2. **Memory Organization**
   - **ROM (Read-Only Memory)**: Stores the program permanently (4KB in 8051)
   - **RAM (Random Access Memory)**: Used for temporary data storage (128 bytes)
   - **Special Function Registers (SFRs)**: Control hardware components
3. **I/O Ports (Input/Output Pins)**
   - **32 I/O Pins**, grouped into **4 Ports (P0, P1, P2, P3)**
4. **Timers and Counters**
   - Two **16-bit timers** (Timer 0 & Timer 1)
   - Used for time delays, pulse width modulation (PWM), and counting events
5. **Serial Communication (UART)**
   - Supports **full-duplex** communication using **SBUF register**
   - Uses **TXD (Transmit) and RXD (Receive) pins**
6. **Interrupts**
   - **External Interrupts (INT0, INT1)**
   - **Timer Interrupts (TF0, TF1)**
   - **Serial Communication Interrupt**

3. I/O Pins and Ports in 8051

Overview of I/O Ports

The 8051 has **4 bi-directional I/O ports** (**P0, P1, P2, and P3**), each consisting of 8 pins.

Port	Pins	Function
Port 0 (P0)	P0.0 - P0.7	Open-drain, multiplexed with address/data bus
Port 1 (P1)	P1.0 - P1.7	General-purpose I/O
Port 2 (P2)	P2.0 - P2.7	General I/O, higher address bus
Port 3 (P3)	P3.0 - P3.7	Special function I/O

Special Functions of Port 3

Pin	Function
P3.0	RXD (Serial Data Receive)
P3.1	TXD (Serial Data Transmit)
P3.2	INT0 (External Interrupt 0)
P3.3	INT1 (External Interrupt 1)
P3.4	T0 (Timer 0 Input)
P3.5	T1 (Timer 1 Input)
P3.6	WR (External Memory Write)
P3.7	RD (External Memory Read)

---

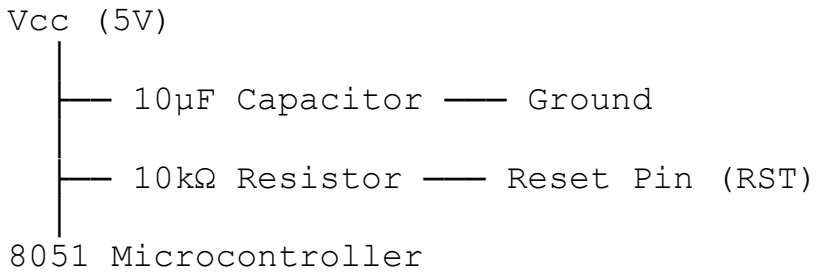
4. Circuits in 8051 Microcontroller

Basic 8051 Circuit Diagram

To operate the **8051 microcontroller**, the following components are needed:

- **Power Supply (5V DC)**
- **Clock Oscillator (11.0592 MHz Crystal)**
- **Reset Circuit (Capacitor & Resistor)**
- **Pull-up Resistors (for Port 0)**

Basic Power & Reset Circuit



---

5. External Memory in 8051

Why External Memory?

The **basic 8051** has only **4KB ROM** and **128B RAM**, which may not be enough for complex applications. External memory is used for:

- **Program Memory (ROM / Flash Memory)**
- **Data Memory (SRAM / EEPROM)**

External ROM (Program Memory)

- Connected using **PSEN (Program Store Enable)**
- Typically **64KB external ROM** can be interfaced

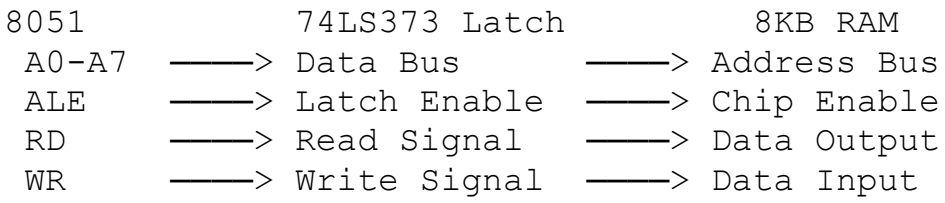
External RAM (Data Memory)

- Connected via **RD (Read) and WR (Write) pins**
- Can extend RAM up to **64KB**

Interfacing External RAM with 8051

Pin	Function
PSEN	Enables external ROM
ALE	Address Latch Enable
RD	Read Signal
WR	Write Signal

Example: Interfacing External 8KB RAM



8051 Microcontroller Programming

The **8051 microcontroller** is programmed using **Assembly Language** or **Embedded C**. This section covers:

1. **Data Types in 8051**
2. **I/O Programming**
3. **Logic Operations**
4. **Data Conversion Programs**

---

1. Data Types in 8051

The **8051 microcontroller** is an **8-bit processor**, meaning it processes **8-bit data at a time**. The commonly used data types are:

1.1 Integer Data Types

Type	Size	Range
`unsigned char`	8-bit	0 to 255
`signed char`	8-bit	-128 to 127
`unsigned int`	16-bit	0 to 65,535
`signed int`	16-bit	-32,768 to 32,767

1.2 Boolean Data Type

- **Bit-addressable data**: The 8051 has **16-bit addressable locations (bit variables)**.
- Example:
  ```
  bit flag = 1;  // 1-bit variable stored in bit-addressable memory
  ```

1.3 Special Data Type (SFRs)

Special Function Registers (SFRs) control **timers, ports, serial communication**, etc.
Example:

```
c
CopyEdit
sfr P1 = 0x90;  // Define Port 1
sfr TMOD = 0x89; // Timer Mode Register
```

---

2. I/O Programming in 8051

8051 has 32 I/O pins, divided into **four 8-bit ports (P0, P1, P2, P3)**. Each pin can function as **input or output**.

2.1 Writing to a Port (Output)

```
#include <reg51.h>
void main() {
    P1 = 0xFF;  // Set all P1 pins HIGH
    while(1);
}
```

 Explanation: `P1 = 0xFF;` sets all bits of **Port 1** to **1 (HIGH, 5V)**.

2.2 Reading from a Port (Input)

```
#include <reg51.h>
void main() {
    unsigned char x;
    x = P2;  // Read data from Port 2
    while(1);
}
```

 Explanation: `x = P2;` reads **8-bit data** from Port 2.

2.3 LED Blinking Example

```
#include <reg51.h>
sbit LED = P1^0; // Define LED at P1.0

void delay() {
    int i;
    for(i=0; i<30000; i++);  // Software delay
}

void main() {
    while(1) {
        LED = 0;  // LED ON
        delay();
        LED = 1;  // LED OFF
        delay();
    }
}
```

---

3. Logic Operations in 8051

3.1 Logical AND, OR, XOR

Operation	Symbol	Example
AND	&	P1 = P1 & 0xF0;
OR	`	`
XOR	^	P3 = P3 ^ 0xAA;

Example: **Masking Upper 4 Bits**

```
P1 = P1 & 0xF0; // Clears lower 4 bits of P1
```

3.2 Bitwise NOT (~)

```
P1 = ~P1;  // Complement all bits of Port 1
```

---

4. Data Conversion Programs in 8051

Data conversion programs convert **binary, BCD, ASCII, and decimal numbers**.

4.1 Binary to BCD Conversion

```
#include <reg51.h>
unsigned char binaryToBCD(unsigned char bin) {
```

```
    unsigned char high, low;
    high = bin / 10;  // Extract tens place
    low = bin % 10;   // Extract ones place
    return (high << 4) | low;  // Combine as BCD
}

void main() {
    unsigned char result;
    result = binaryToBCD(27);  // Convert 27 to BCD
(0x27)
    while(1);
}
```

4.2 BCD to ASCII Conversion

```
unsigned char BCDToASCII(unsigned char bcd) {
    return (bcd & 0x0F) + 0x30;  // Convert lower 4-bits
to ASCII
}
```

4.3 ASCII to Decimal Conversion

```
unsigned char ASCIIToDecimal(unsigned char ascii) {
    return ascii - 0x30;  // Convert ASCII '0'-'9' to
Decimal 0-9
}
```

4.4 Hexadecimal to ASCII Conversion

```
unsigned char hexToASCII(unsigned char hex) {
    if(hex <= 9) return hex + '0';  // Convert 0-9
    else return hex + 'A' - 10;  // Convert A-F
}
```

Unit IV
Designing Embedded System with 8051 Microcontroller

Factors to be considered in selecting a controller–8051 Microcontroller–

Designing with 8051 Programming: Structure of embedded program – infinite loop – compiling, linking & debugging

---

Factors to be Considered in Selecting a Controller

When choosing a microcontroller for an embedded application, several key factors must be evaluated:

1. **Performance Requirements** – Based on speed (clock frequency), instruction execution time, and real-time responsiveness.
2. **Memory Requirements** – Consider the size of **Flash (ROM)** for program storage and **RAM** for data storage.
3. **Power Consumption** – Important in battery-operated or portable devices; low-power modes and sleep states are beneficial.
4. **Number of I/O Pins** – Should match the number of input/output devices the application uses.
5. **Peripheral Features** – Required features such as ADC, timers, PWM, UART, SPI, I2C, etc.
6. **Cost and Availability** – A cost-effective microcontroller that's readily available is always preferred.
7. **Development Tools** – Availability of compilers, debuggers, and simulation tools eases development.
8. **Community and Vendor Support** – Access to documentation, sample code, and technical support helps in quick development.

The **8051 microcontroller** is popular because it meets many of these criteria for small to medium embedded applications.

8051 Microcontroller

The **8051** is an 8-bit microcontroller originally developed by Intel. It includes:

- **4 KB ROM**, **128 bytes of RAM**
- **32 I/O lines** divided over 4 ports (P0 to P3)
- **Two 16-bit timers/counters**
- **One full-duplex serial port**
- **5 interrupt sources**
- **Boolean processor for bit-level operations**
  Its simplicity, availability, and robust instruction set make it ideal for control-based

applications such as home automation, industrial machinery, and embedded appliances.

Designing with 8051 Programming

Designing an embedded system using the 8051 involves writing software (firmware) that controls hardware directly. This includes:

Structure of an Embedded Program

An embedded C program for 8051 typically has:

- **Header includes** for hardware registers
- **Main() function** where the logic is written
- **Initialization code** for ports, timers, or peripherals
- **An infinite loop (`while(1)` or `for(;;)`)** that continuously runs, since embedded systems operate perpetually

```
#include <reg51.h>   // Header for 8051

void main() {
  P1 = 0x00;       // Set Port 1 as output
  while(1) {
     P1 = 0xFF;   // Turn ON all port pins
  }
}
```

Compiling, Linking, and Debugging

- **Compiling**: Converts source code (.c) to object code (.obj).
- **Linking**: Combines object files and libraries into a final executable (hex/bin file).
- **Debugging**: Using simulators or on-chip debuggers to check logic errors, memory access, and variable values.

Tools like **Keil µVision** or **MPLAB** provide an integrated environment to write, compile, simulate, and debug embedded C programs for the 8051.

Unit V

Real Time Operating System(RTOS)

Operating system basics–Types of OS–Real-Time Characteristics– Selection Process of an RTOS, Design and Development: Embedded system development Environment – IDE – types of file generated, disassembler – de-compiler – simulator – emulator and debugging, embedded product development life-cycle,trends in embedded industry

---

Operating System Basics

An **Operating System (OS)** is the foundational software that manages a computer's hardware and software resources. It acts as an interface between the hardware and the user/application. Its core functions include:

- **Process Management**: Handles creation, execution, and termination of processes.
- **Memory Management**: Allocates and deallocates memory space as needed by different programs.
- **File Management**: Manages data storage, file naming, directories, and access permissions.
- **Device Management**: Controls and communicates with peripheral devices (e.g., keyboard, printer).
- **Security & Access Control**: Protects data and restricts unauthorized access

Types of Operating Systems

1. **Batch Operating System**
   - Jobs are grouped and executed without user interaction.
   - Common in mainframes.
2. **Time-Sharing Operating System**
   - Allows multiple users to share system resources simultaneously.
   - Example: UNIX.
3. **Distributed Operating System**
   - Manages a group of networked computers and makes them appear as a single system.
4. **Network Operating System**
   - Provides networking functions such as file sharing and remote access.
5. **Mobile Operating System**
   - Used in smartphones/tablets.
   - Examples: Android, iOS.
6. **Embedded Operating System**
   - Lightweight OS designed for embedded systems with limited resources.
   - Examples: FreeRTOS, VxWorks, µC/OS-II.
7. **Real-Time Operating System (RTOS)**
   - Designed for applications with strict timing requirements.

Real-Time Operating System (RTOS)

A **Real-Time Operating System** must ensure consistent timing and meet deadlines. Characteristics include:

- **Determinism**: Predictable task execution timing.
- **Low Latency**: Fast response to external or internal events.
- **Multitasking Support**: Manages multiple processes efficiently.
- **Preemptive Scheduling**: Higher-priority tasks can interrupt lower-priority ones.
- **Minimal Jitter**: Low variation in task execution timing.
- **Task Prioritization**: Critical tasks get executed first.
- **Real-Time Clocks & Timers**: Supports accurate scheduling and time measurement.

A **Real-Time Operating System (RTOS)** is designed to handle tasks within strict time constraints. It ensures that high-priority tasks execute **predictably and on time**, which is critical in embedded applications like medical equipment, robotics, or automotive control. Unlike general-purpose OS (e.g., Windows), an RTOS focuses on **determinism**, reliability, and minimal latency.

Selection Process of an RTOS

To choose the right RTOS, consider:

1. **Application Requirements**
    - Is hard real-time or soft real-time needed?
2. **Hardware Compatibility**
    - Does the RTOS support the target microcontroller?
3. **Footprint**
    - Small memory usage is ideal for limited hardware.
4. **Task Management**
    - Number of tasks, inter-process communication (IPC), semaphores.
5. **Tool Support**
    - Debuggers, IDEs, profilers, and documentation.
6. **Licensing & Cost**
    - Free (e.g., FreeRTOS) vs commercial (e.g., QNX, VxWorks).
7. **Community & Vendor Support**
    - Helps resolve issues and speeds up development.

Design and Development: Embedded System Development Environment

An **Embedded System Development Environment** is a combination of hardware and software tools used to write, test, and deploy embedded applications.

IDE (Integrated Development Environment)

An IDE integrates:

- Code editor
- Compiler
- Debugger

- Simulator/Emulator support

IDE (Integrated Development Environment)

An IDE integrates:

- **Text Editor**: Write and edit source code.
- **Compiler**: Converts C/ASM code to machine code.
- **Debugger**: Monitor code execution.
- **Simulator/Emulator Interface**: Test code without hardware.

Examples: **Keil µVision**, **STM32CubeIDE**, **Arduino IDE**, **Eclipse + GCC**.

Types of Files Generated

During embedded development, the following files are produced:

File Type	Extension	Description
Source Code	`.c`, `.asm`, `.h`	Human-readable code
Object File	`.obj`	Machine code without linking
Executable File	`.hex`, `.bin`	File uploaded to microcontroller
Map File	`.map`	Shows memory allocation and usage
List File	`.lst`	Assembly listing with addresses

Disassembler and De-compiler

- **Disassembler** converts machine code back into **assembly code**. Useful for debugging or reverse engineering.
- **De-compiler** tries to convert machine code into a **high-level language** like C. It is harder and less accurate but useful for analysis when source code is lost.

Simulator and Emulator and Debugging

- **Simulator** mimics the microcontroller behavior in software without real hardware. Ideal for early-stage testing.
- **Emulator** is hardware that replicates the target system. Offers **real-time debugging** with hardware-in-loop.
- **Debugging tools** include breakpoints, step-through execution, watch variables, and memory inspection to identify logic errors and runtime issues.

Embedded Product Development Life-Cycle

1. **Requirement Analysis**
2. **System Specification**
3. **Hardware/Software Partitioning**
4. **Hardware Design**
5. **Software Design**
6. **Implementation & Testing**
7. **Integration**
8. **System Testing**
9. **Deployment**
10. **Maintenance/Updates**

This lifecycle ensures structured development from concept to product launch.

Trends in Embedded Industry

- **AI and Machine Learning at Edge** – Embedded AI in devices like cameras and wearables.
- **IoT Integration** – Embedded devices are increasingly connected via the Internet.
- **Low Power Design** – Battery-efficient designs for wearables and sensors.
- **RISC-V Architecture** – Open-source ISA gaining popularity.
- **Security** – Embedded systems now include features like encryption, secure boot, and hardware TPM.
- **RTOS adoption** – Use of FreeRTOS, Zephyr, and commercial RTOS is growing in even small-scale applications.