SYMBIOSIS INTERNATIONAL (DEEMED UNIVERSITY)

॥वसुधैव कुटुम्बकम्॥

Microcontrollers and Embedded Systems

Unit V: **Embedded Systems**

Symbiosis Institute of Technology, Nagpur

Symbiosis Institute of Technology, Nagpur Campus (SIT-N)

Embedded Systems:

- Introduction to Embedded System,
- Characteristics,
- Architecture of real-time Embedded Systems,
- Embedded Operating System,
- RTOS,
- Types of Real-Time Tasks,
- Features of RTOS.

Introduction to Embedded System

An **Embedded System** is a **microprocessor- or microcontroller-based system** that is **designed to perform a specific task** or function within a larger mechanical or electrical system.

Unlike general-purpose computers (like PCs), which are designed to handle a wide range of tasks, **embedded systems are task-specific** and often operate under **real-time constraints**.

Key Characteristics of Embedded Systems

1. **Dedicated Functionality**: Designed for a specific task or application.
2. **Real-Time Operation**: Often required to respond within a specified time (soft or hard real-time systems).
3. **Resource Constraints**: Limited memory, processing power, and energy.
4. **Reliability and Stability**: Must operate continuously and reliably.
5. **Embedded within a Larger System**: Not standalone—works as a component of a larger device.
6. **Often Interacts with Hardware**: Uses sensors and actuators for input/output.

Basic Structure of an Embedded System

An embedded system typically consists of the following components:

1.Hardware:

1. **Microcontroller or Microprocessor** (e.g., ARM, PIC, AVR)
2. **Memory** (RAM, ROM, Flash)
3. **Input Devices** (Sensors, switches)
4. **Output Devices** (LEDs, LCDs, actuators)
5. **Communication Interfaces** (UART, SPI, I2C, CAN)

2.Software:

1. Embedded software/firmware written in C/C++ or Assembly.
2. Often uses a **Real-Time Operating System (RTOS)** or **bare-metal programming**

Types of Embedded Systems

1.Based on Performance & Functional Requirements:

1. **Real-time Embedded Systems** (e.g., anti-lock braking systems)
2. **Stand-alone Embedded Systems** (e.g., washing machines)
3. **Networked Embedded Systems** (e.g., smart home IoT devices)
4. **Mobile Embedded Systems** (e.g., mobile phones)

2.Based on Complexity:

1. **Small-Scale** (8-bit microcontrollers)
2. **Medium-Scale** (16/32-bit controllers, RTOS support)
3. **Sophisticated Systems** (multiple processors, complex software)

Applications of Embedded Systems

Embedded systems are everywhere. Here are some common applications:

- **Consumer Electronics**: Smart TVs, microwave ovens, cameras
- **Automobiles**: Engine control units, airbag systems, GPS
- **Industrial Automation**: PLCs, robotics, CNC machines
- **Medical Devices**: Pacemakers, MRI machines, infusion pumps
- **Telecommunications**: Routers, modems, base stations
- **Home Automation**: Smart thermostats, security systems
- **Military and Aerospace**: Drones, missile guidance systems

Advantages of Embedded Systems

- Compact size
- Low power consumption
- Cost-effective for mass production
- High reliability and efficiency
- Customizable and optimized for specific tasks

Challenges in Embedded Systems

- Real-time constraints
- Debugging and testing difficulty
- Limited resources (memory, processing power)
- Security and safety concerns
- Hardware-software co-design complexity

Common Tools & Languages

- **Languages**: C, C++, Assembly, Python (for prototyping), Rust (emerging)
- **IDEs**: Keil µVision, MPLAB X, Arduino IDE, STM32CubeIDE
- **Simulators & Debuggers**: Proteus, Multisim, JTAG, GDB
- **RTOS examples**: FreeRTOS, VxWorks, Micrium µC/OS, ThreadX

Future Trends in Embedded Systems

- **AI on Edge**: Machine learning models running on embedded hardware (TinyML)
- **IoT Integration**: Connectivity and control through the internet
- **Energy Efficiency**: Low-power designs with sleep modes
- **Security**: Hardware and software-based encryption
- **Autonomous Systems**: Drones, robotics, self-driving cars

Architecture of Real-Time Embedded Systems

A **Real-Time Embedded System (RTES)** is an embedded system that must respond to inputs or events **within a strict timing deadline**. The **architecture** of such systems is designed to ensure **predictability, reliability, and performance** under real-time constraints.

1. General Overview of RTES Architecture

Real-Time Embedded Systems consist of two main domains:

- **Hardware architecture**
- **Software architecture**

These work together to ensure that tasks meet their **real-time deadlines** (hard, firm, or soft).

2. Hardware Architecture

The hardware is responsible for processing inputs, controlling outputs, and executing real-time tasks. It consists of:

a. Processor Core

- **Microcontroller (MCU)** or **Microprocessor (MPU)**
  - Examples: ARM Cortex-M, RISC-V, AVR, PIC
- Determines computing capability.
- May have built-in timers, interrupt controllers, ADCs, etc.

b. Memory

- **ROM/Flash**: Stores the firmware or program code.
- **RAM**: Stores runtime data, stack, buffers, etc.
- **EEPROM** (optional): For non-volatile data storage.

c. Timers and Counters

- Crucial for real-time task scheduling, delays, and measuring time intervals.

d. I/O Ports

- To interface with external devices (sensors, actuators).
- Examples: GPIO, UART, SPI, I2C.

e. Interrupt Controller

- Manages asynchronous event handling.
- Prioritizes interrupt requests based on urgency.

f. Communication Interfaces

- Wired: CAN, USB, Ethernet
- Wireless: Bluetooth, Wi-Fi, Zigbee (for networked RTES)

g. Watchdog Timer

- Ensures the system is reset if it hangs or becomes unresponsive.

3. Software Architecture

Software handles real-time scheduling, task execution, and resource management. It includes:

- **a. Real-Time Operating System (RTOS)**

An RTOS is central to real-time systems. It handles:

- **Task scheduling**
- **Interrupt handling**
- **Inter-process communication (IPC)**
- **Resource management**

Examples: **FreeRTOS**, **VxWorks**, **RTEMS**, **Micrium**, **QNX**

- **b. Task Management**

- Tasks are divided into **periodic** (e.g., sensor reading every 10ms) and **aperiodic** (e.g., responding to a button press).
- Each task has:
  - **Priority**
  - **Deadline**
  - **Execution time (WCET - Worst Case Execution Time)**

- **c. Scheduler**

- Responsible for selecting which task runs at any moment.
- Types of scheduling:
  - **Preemptive Priority Scheduling**
  - **Round-Robin**
  - **Rate Monotonic Scheduling (RMS)**
  - **Earliest Deadline First (EDF)**

- **d. Inter-Process Communication (IPC)**

Mechanisms for tasks to share data:

•Semaphores

•Queues

•Mailboxes

•Message passing

- **e. Device Drivers**

•Interface between hardware and the OS or firmware.

•Abstract hardware details from upper layers.

- **f. Middleware (Optional)**

•Communication stacks, file systems, protocol handlers, etc.

- **g. Application Layer**

•The logic that fulfills the system's real-time function.

•For example: Controlling a motor based on sensor input.

4. Real-Time Constraints

RTES must ensure timing guarantees. There are 3 types of real-time systems:

Type	Description
Hard Real-Time	Missing a deadline = system failure (e.g., pacemakers, airbags)
Firm Real-Time	Occasional deadline miss = degraded performance (e.g., video streaming)
Soft Real-Time	Deadline misses tolerated, but affect quality (e.g., multimedia players)

Typical Flow in a Real-Time Embedded System

1.Sensor detects a change → signal sent to MCU.

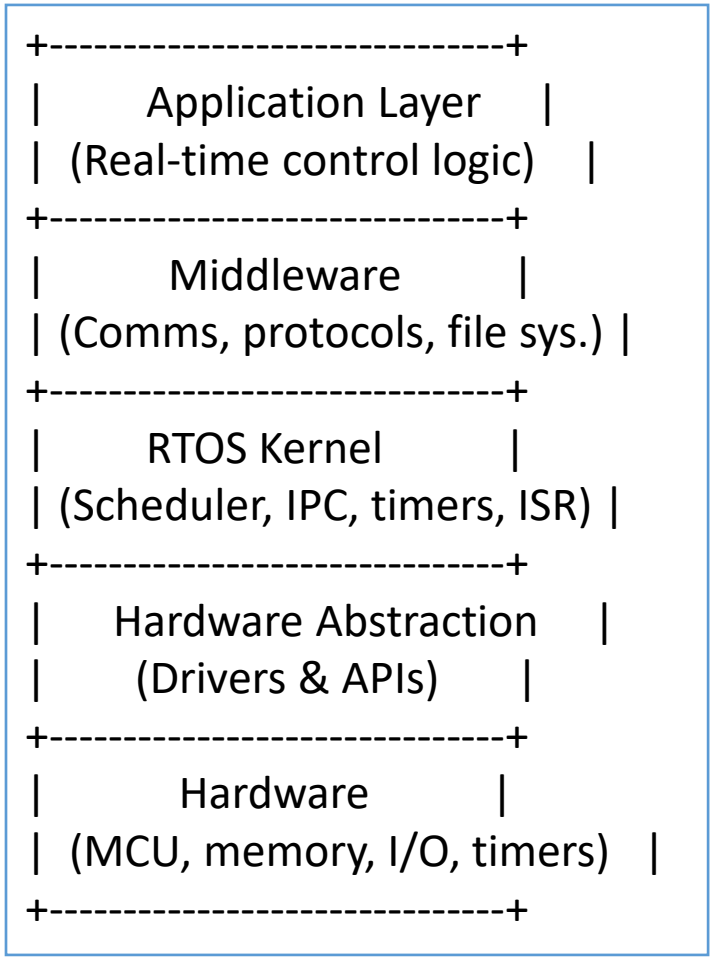
2.Interrupt generated → triggers **ISR (Interrupt Service Routine)**.

3.ISR may **wake up** a real-time task or schedule it via the RTOS.

4.Task executes based on **priority & timing**.

5.Output sent to actuator/display, response logged or transmitted.

6. Layered View of RTES Architecture



7. Design Considerations

- **Determinism**: System behavior must be predictable.
- **Latency**: Response time should be minimal.
- **Throughput**: Efficient handling of high-frequency inputs/outputs.
- **Fault tolerance**: Safe recovery from errors.
- **Power Efficiency**: Important for battery-powered RTES.
- **Scalability & Modularity**: Easy to update or scale.

Example: RTES in a Smart Traffic Light System

- Sensors detect vehicle presence.
- RTOS schedules:
  - Timer task to control lights.
  - Sensor handling task.
  - Emergency override task.
- Interrupts from sensors adjust light durations in real-time.
- Networked communication updates traffic server.

An **Embedded Operating System (EOS)** is a specialized OS **designed to manage hardware and software resources** in an embedded system. Unlike general-purpose OS (like Windows or Linux), it is **highly optimized**, **resource-constrained**, and tailored for specific **real-time** or **non-real-time** tasks.

Key Responsibilities:

- Task management
- Memory management
- Device driver integration
- Communication between tasks (IPC)
- Power management
- Input/output control

Types of Embedded Operating Systems:

Type	Description	Example
RTOS	Real-Time Operating System with deterministic behavior	FreeRTOS, VxWorks
Bare-metal	No OS, simple loop or interrupt-driven logic	Arduino
Embedded Linux	Linux kernel customized for embedded use	Raspbian, OpenWRT
RT-Linux	Real-time patches to Linux for real-time tasks	PREEMPT-RT

2. Real-Time Operating System (RTOS)

What is an RTOS?

A **Real-Time Operating System (RTOS)** is a specialized EOS that ensures **deterministic and predictable task execution**. It is used in applications where **timing is critical**, such as medical devices, automotive systems, aerospace, etc.

Core Functions of RTOS:

- **Multitasking**: Handles multiple concurrent tasks.
- **Task Scheduling**: Ensures high-priority tasks get CPU time.
- **Real-Time Clock (RTC)**: Maintains precise timing.
- **Interrupt Management**: Efficient response to hardware signals.
- **Inter-Process Communication (IPC)**: Manages data sharing among tasks.

RTOS Examples:

- **FreeRTOS** (open source, widely used in IoT)
- **VxWorks** (Wind River, used in NASA Mars rovers)
- **QNX** (automotive and industrial systems)
- **Micrium µC/OS**, **ThreadX**, **RTEMS**, **Zephyr**

3. Types of Real-Time Tasks

RTOS handles different **types of tasks** based on timing constraints and behavior:

- **A. Based on Timing:**
- **B. Based on Criticality:**
- **C. Based on Priority:**

Task Type	Description	Examples
Periodic	Occurs at regular intervals	Sensor polling every 10ms
Aperiodic	Occurs at unpredictable times	Button press, network packet
Sporadic	Occurs irregularly but with a minimum inter-arrival time	Emergency alert signal

Task Type	Description
Hard Real-Time Task	Must meet deadlines 100%, failure leads to system crash (e.g., pacemaker)
Firm Real-Time Task	Occasional deadline misses allowed, but output is discarded (e.g., online trading systems)
Soft Real-Time Task	Missed deadlines degrade performance but are tolerable (e.g., video streaming)

Task Type	Description
High-priority task	Gets immediate CPU attention, can preempt lower-priority tasks
Low-priority task	Runs when no high-priority tasks are ready

4. Features of RTOS

A good RTOS must provide specific features to meet the demands of real-time systems:

A. Deterministic Behavior (Predictability)

•Guarantee that high-priority tasks get CPU time **within fixed deadlines**.

B. Multitasking with Priority Scheduling

•Supports multiple tasks, scheduled by:

- **Fixed-priority preemptive scheduling**
- **Round-robin**
- **Rate Monotonic Scheduling (RMS)**
- **Earliest Deadline First (EDF)**

C. Fast Context Switching

•Switching between tasks must be quick and efficient.

D. Minimal Latency

•**Interrupt latency** and **task switching latency** must be minimal.

E. Real-Time Clock and Timers

•Timers used for periodic tasks, timeouts, and delays.

F. Inter-Process Communication (IPC)

•Mechanisms like:

- **Semaphores**
- **Message queues**
- **Event flags**
- **Mailboxes**

G. Synchronization & Resource Management

•Avoid race conditions using:

- Mutexes
- Priority inheritance
- Critical sections

H. Scalability and Portability

•Should work with different hardware and be scalable from small MCUs to large SoCs.

I. Modularity

•Designed in modules (task scheduler, memory, drivers) for easier maintenance and upgrades.

J. Power Management

•Support for sleep modes, idle states (especially important for IoT and battery-powered systems).

Comparison: RTOS vs General-Purpose OS

Feature	RTOS	General OS (e.g., Linux, Windows)
Task Response Time	Deterministic, real-time	Non-deterministic
Multitasking	Yes, with priority scheduling	Yes, general scheduling
Memory Usage	Very low	High
Boot Time	Fast (milliseconds)	Slow (seconds)
Power Consumption	Optimized	Not optimized
Use Cases	Medical, automotive, IoT	Desktops, laptops, servers

Real-Life Example: RTOS in Automotive Systems

In a modern car:

- **Airbag system**: Hard real-time task
- **Infotainment system**: Soft real-time task
- **Engine control**: Periodic real-time task
- **ABS**: Aperiodic, time-critical task

All these run **concurrently**, managed by an **automotive RTOS** like **QNX** or **AUTOSAR RTOS**.