Embedded Computing

microcontroller or microprocessor → perform dedicated tasks

part of a larger system

Embedded System Design process

design, develop, test, and deploy an embedded system

real-time, cost, and resoure constraints

Stages

1. Requirement Analysis → 2. Specification → 3. HW/SW Design → 4. Integration → 5. Testing → 6. Validation → 7. Deployment

Example: **Temperature monitoring system**

Uses sensor (LM35) + ADC + 8051
Reads temperature → compares with threshold
Controls output (fan/heater, relay)
Displays result on LCD
Timer → periodic sampling
Interrupts → real-time response
Continuous monitoring, fast and accurate operation

Process (in embedded systems)

independent program; own memory + resources; managed by OS

TinyOS - open-source; lightweight; wireless sensor networks (WSNs); resource-constr-ained systems

eg : sensor nodes

IPC - allows processes or tasks to exchange data and synchronize their activities

IPC - Inter-Process Communication

Program Design Models

Top-Down → Break system into smaller modules

Bottom-Up → Build system from low-level components

Modular Design → Independent functional blocks

State Machine Model → States + transitions → predictable behavior

Super Loop Model → Simple infinite loop (less responsive)

Interrupt-Driven Model → Handles time-c-ritical tasks

RTOS Model → Multitasking with scheduling

Memory Interfacing

Connects external memory (RAM/ROM) to MCU (using address, data, and control signals)

Ensures correct data transfer + selection + address selection

decoding circuit for proper memory selection

Process

MCU sends address of memory/I/O device during execution

Decoding circuit selects correct memory chip

Control signals used for read/write operations

Interfacing matches memory requirements with MCU signals

I/O Device Interfacing

Connects external devices (keyboard, sensor, display)

Uses: Latches; Buffers

Impact on Performance:

Proper interfacing → faster data transfer
Poor interfacing → delays, inefficiency

I/O Device Interfacing (cont)

Example: **Smart home monitoring system**

Inputs: Sensors (temperature, light), switches (connected via I/O port)

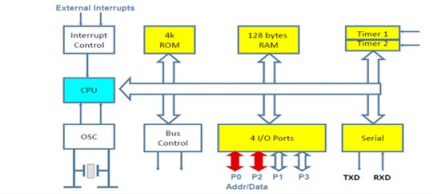
Uses ADC to convert analog sensor data

Outputs: Relay (fan/light), LCD, buzzer

Reads inputs → processes → controls outputs

timers for real-time; continuous monitor

fig. 1



RTOS

manage multiple tasks

time-critical operations are executed (deadline)

task - basic unit of execution; function/ activity; control of RTOS scheduler; runs concurrently

priority-based scheduling - each task assigned priority; CPU allocated to highest-priority task

role of scheduler

1. Task Selection
2. CPU Allocation
3. Preemption Control
4. Task State Management
5. Deterministic Execution
6. Context Switching Trigger

Multi-tasking - execute multiple tasks concurrently ; sharing CPU

Features

Deterministic Timing (predictable response times for tasks)
Priority-Based Scheduling (highest-priority ready task first)

Process & Task

In RTOS

Example: **Smart Industrial Monitoring and Control System**

Tasks:	Sensor reading Data processing Control Display Communication Alarm
Task Priority:	High → Alarm, Control Medium → Sensor Low → Display, Communication
Process: Group of related tasks	Monitoring → Sensor + Processing Control → Control + Alarm UI → Display + Communication

Uses priority-based preemptive scheduling

Tasks communicate using queues/shared memory

Synchronization via semaphores/mutex

Interrupts trigger high-priority tasks

Context switching enables multitasking

Context Switching

saves state of current task; restores state of another task; share CPU

Context includes: PC, registers, stack pointer

RTOS uses to schedule tasks on priority

Supports preemption (high-priority tasks interrupt low-priority) in RTOS

Context Switching (cont)

Improves CPU utilization

RTOS requires fast and predictable

Priority based Scheduling

priorities based on importance

Highest-priority task first

preemptive scheduling (Handles asynchronous events)

deterministic behavior (predictable execution)

low response time for critical tasks

Improves CPU utilization

used in RTOS-based systems

Issue: Priority inversion

Solution: Priority inheritance protocol