SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A++' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

19ECT312 – EMBEDDED SYSTEM DESIGN

III YEAR/ VI SEMESTER

1

UNIT 3 –PROGRAMMING CONCEPTS AND EMBEDDED PROGRAMMING IN C++

TOPIC 3 - Embedded Programming in C++

Introduction to Embedded Systems

Definition: Embedded systems are specialized computing systems designed to perform specific functions within a larger system or device\

Characteristics:

- Constrained resources (memory, processing power)
- Real-time operation
- Interaction with physical environment (sensors, actuators)

Importance and applications:

- Automotive (engine control units, infotainment systems)
- Consumer electronics (smartphones, wearable devices)
- Industrial automation (PLCs, robotics)

Challenges:

- Optimization for resource-constrained environments
- Real-time performance requirements
- Hardware-software co-design considerations

Overview of C++ for Embedded Systems

Why C++?

Abstraction: Provides high-level constructs without sacrificing performance.
Modularity: Supports object-oriented programming, facilitating code reuse and maintenance.
Efficiency: Allows fine-grained control over memory and hardware resources
.

Key features:

Classes and objects
Inheritance and polymorphism
Templates and generic programming
Standard Template Library (STL)

Basic Concepts in Embedded C++

Data types and memory representation:
- Fundamental types (int, float, char)
- Fixed-width integer types (stdint.h)
- Understanding memory layout (stack, heap, data, text segments)

Pointers and memory management:
- Pointer arithmetic
- Dynamic memory allocation (new/delete, malloc/free)

Control flow constructs:
- if-else statements
- Loops (for, while, do-while)

Functions:
- Modular programming
- Passing arguments by value vs. reference

Low-Level Programming in C++

Memory-mapped I/O:

Accessing hardware peripherals directly through memory addresses
Using volatile keyword to prevent compiler optimizations

Bit manipulation techniques:

Setting, clearing, and toggling bits
Bitwise operators (&, |, ^, <<, >>)

Accessing hardware peripherals:

Register definitions and bitfields
Using hardware abstraction layers (HALs)

Interrupt Handling in Embedded C++

Understanding interrupts:

Introduction to interrupt-driven programming
Interrupt vectors and priority levels

Writing interrupt service routines (ISRs) in C++:

Marking ISRs with appropriate attributes (e.g., **attribute**((interrupt)))
Handling interrupt context and latency

Techniques for managing interrupts:

Interrupt nesting and prioritization
Deferred interrupt handling

Embedded C++ and Object-Oriented Programming

Encapsulation, inheritance, and polymorphism:

- Designing classes to represent hardware components (e.g., sensors, actuators)
- Inheritance hierarchies for peripheral drivers
- Polymorphic behavior for abstracting hardware interfaces

Using classes and objects:

- Instantiation and initialization
- Access specifiers (public, private, protected)
- Member functions and data members

Memory Management in Embedded C++

Static vs. dynamic memory allocation:

Stack vs. heap memory
Stack usage for local variables and function calls
Heap allocation for dynamic data structures (e.g., linked lists, trees)

Memory footprint optimization techniques:

Minimizing global variables
Static analysis tools for memory usage profiling
Custom memory allocators for resource-constrained systems

Real-Time Operating Systems (RTOS) with C++

Introduction to RTOS for embedded systems:

- Task scheduling and prioritization
- Inter-task communication and synchronization

Using C++ features with RTOS APIs:

- Thread creation and management
- Synchronization primitives (semaphores, mutexes)

Task scheduling and synchronization in RTOS-based embedded applications:

- Priority inversion and priority inheritance
- Deadlock avoidance and detection