Embedded Software Engineering

embed CORE

Module 1: Orientation

"This module will get the participants acquainted with Linux systems and the usage of Linux commands, command prompt, text editors, etc…"

15+ hrs of live interactive sessions which include a conceptual framework and hands-on research examples

Technical Requirements

Desktop/Laptop with the below configuration

RAM: 2GB or more if running on a physical host/ 4GB or more if running as a guest (Using VM Virtual Box)

SSD/HHD: 30 GB of free space

Operating System: Ubuntu 18.04 LTS with Internet Access

Course Curriculum

- What is Linux
- Different types of Linux
- 0Installing Linux and setting up the environment
- Introduction to the command line
- Linux Commands
- Text editor
- Tool Chain
- GNU Compiler Collection
- Compile and Build Process
- Load and Execute Process

Module 2: Advanced C for Embedded programming and Systems Programming

“Advanced C programming will help in understanding the intrinsic nature of programming language features, gain capabilities to use language-specific functionalities, features to express logic in ‘C’. These fundamentals would help in learning any programming language faster and easier”

40+ hrs of live interactive sessions which include a conceptual framework and hands-on research examples

Technical Requirements

Desktop/Laptop with the below configuration

RAM: 2GB or more if running on a physical host/ 4GB or more if running as a guest (Using VM Virtual Box)

SSD/HHD: 30 GB of free space

Operating System: Ubuntu 18.04 LTS with Internet Access

Course Curriculum

Course Introduction

- Task and Categories of Task
- Introduction to Programming Language
- Program and Building Blocks of Program

Data Types and Constants

- Single byte
- Multibyte
- Integer, real, character, string constants

Data representation in memory and sizes

- Writing and retrieving the data in memory
- Symbolic References and Variable Creation
- Intricacy of Symbol names and Byte Order
- Endianness and Order of Storing
- ASSCI Character Set and Uni-Character set
- Size of *char, int, float, and double*
- ILP & LP

Program Content

Modifiers

- Signed
- Unsigned
- Short
- Long

Generic Structure of C Programming Language

- Structure of C Program
- Simple C source code Implementation
- Compiling C source using GCC
- Compile time, Build time, and Runtime.
- Generate ANSI C standard output

Operators and Operands

- Operands and Expression and Types of Expressions
- Unary, Binary, and Trinary Operators
- Arithmetic Operators
- ***Sizeof*** Operator
- Relational Operators and Logical Operators
- Short Circuit Evaluation
- Compound Assignment and Arithmetic Assignment Operator
- ‘,’ Operator and Conditional Operator
- ***long long datatype***
- Single Statement and Compound Statement
- Bitwise Operators
- Precedence and Associativity
- Incremental and Decremental Operators
- Assignment Operators
- Bitwise Operators
- Conditional Operators

Program Content

Statements

- Single Statement
- Compound Statement
- Parent Block
- Child Block
- Type Casting

Input/output Functions

- Formatted
- Unformatted
- Printf
- Scanf
- Conversion Characters

Controlling Instruction Execution

- Decision Control
- Switch Case
- Iterative or Loops
- Unconditional Control Structures

Functions

- Introduction to Functions
- Advantages
- Defining a Function
- Using a Function
- Calling Function or Caller Function
- Caller Function
- Called Function
- Program Image
- Calling and Called Functions Communication
- Parameters, actual parameters, rules of parameters
- Formal arguments
- Called to Calling function Communication
- Top to bottom
- Bottom to top
- Use of Void
- Recursion
- Stack Pointer or Stack Frame
- Storage Classes and Its Properties

Program Contents

Pointers

- Relation between **'&'**, **'*'** and Composition of Pointer
- Why is pointer Associated with type
- Dereference operator, Null Pointer and Wild Pointer
- Memory Violation, Illegal Memory Access, and Segmentation Fault
- Pointer Operations and C Specifiers
- Call by Value and Call by Reference
- Void Pointer and dereferencing a Void Pointer
- Pointer to a Pointer

Arrays

- Intro to Derived Datatypes
- Defining an array and its memory organization
- Retrieving the value using arrays
- Why array index starts with zero
- Pointer and Arrays Relation
- Arrays Limitations
- Pointer to an Array and Array of Pointers
- Double Dimension and Triple Dimension Arrays
- Passing an Array as an Argument to a Function
- Returning an Array from a Function

Function Pointers

- Introduction to Function Pointers
- Defining a Function Pointer
- Operations on Function Pointers
- Returning a Pointer from a Function
- Passing a Function as an Argument
- Introduction to Call Back and Use Cases of Call Back
- Returning a Function from Another Function
- Introduction to Layered Approach

Strings

- Introduction to Strings
- Defining a String and Internal Memory Representation
- Pointer to a String and Character Array
- Passing a String to a Function
- Returning a String from a Function
- Understanding Internal of string Inbuilt Functions

Program Content

Dynamic Memory Allocations

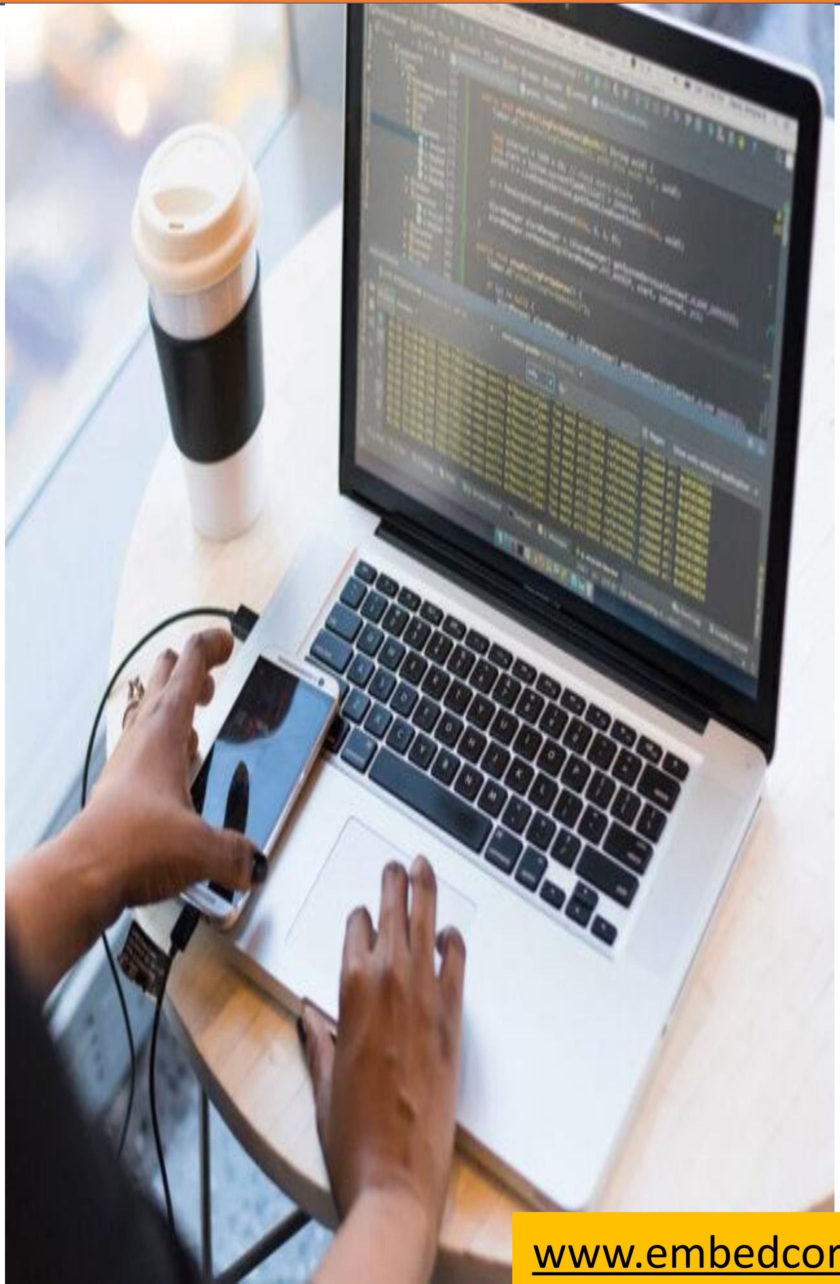
- Introduction to Dynamic Memory Allocation
- ***malloc(),calloc(),realloc(),bzero() and free***
- Dangling Pointer
- DMA to 2-Dimensional array
- Character Array VS string
- ***Strdup(), ftoken(), strtoken(), memcpy and memmove()***

Typedef

- Alias Name to Primitive Datatypes
- Scope of typedef
- Alias Names to Extended Datatypes
- Use cases of typedef

Enumeration

- Introduction to Enumeration
- Defining enums
- Scope and size of enum
- Use cases



Structures

- Introduction to Structures
- Defining a Structure and Internal Memory representation
- Structure object and size
- Various Methods of Initializing a Structure
- Accessing Structure Members
- Data Alignment, Aligned Vs Unaligned Data
- Structure Padding and Internals
- Pointer to a Structure and Accessing Elements Using Structure Pointer
- Passing a Structure to a Function and Returning a Structure from a Function
- Array of Structures
- Operations on Structures
- Nested Structures
- Structure Bit Fields and Internals of Bit Fields

Program Content

Unions

- Introduction to Unions
- Unions Vs Structures
- Use cases of Unions

Pre-Processor Directives

- Introduction to Pre-Processor
- Object Like Macro VS Function Like Macro
- Issues with Function Like Macros
- Conditional Compilation
- Inline Functions
- Use Cases

C Qualifier

- Volatile Keyword
- Const and Volatile
- Asynchro’s access of Volatile data and its side effects

Variable argument Length Functions and Used Cases

File I/O and Use Cases

Command Line Arguments and Use Cases

Module 3: Algorithmic Thinking

"Algorithmic thinking is the backdrop for programmers in opening the mind for computational thinking, problem-solving, implementing the efficient algorithm, this module is designed to acquire mastery of Data structures and Algorithmic patterns which aid to crack coding Interviews of top-notch companies effortlessly "

40+ hrs of live interactive sessions which include a conceptual framework and hands-on research examples

Technical Requirements

Desktop/Laptop with the below configuration

RAM: 2GB or more if running on a physical host/ 4GB or more if running as a guest (Using VM Virtual Box)

SSD/HHD: 30 GB of free space

Operating System: Ubuntu 18.04 LTS with Internet Access

Hardware: Laptop/Desktop with Ubuntu 18.04

Course Curriculum

Linear Storage Patterns

- Sequential
- Linked List

Non-Linear Storage

- Trees
- Hash

Hybrid Storage

Data Structure Patterns

- 1D-Array, List
- Stack
- Queue
- Set, MultiSet
- Map/Dictionary, MultiMap
- Sorted Set, Sorted MultiSet
- Sorted Map, Sorted MultiMap
- Heap/Priority Queue

Program Content

Algorithmic Patterns

- Adhoc Thinking
- Binary Search Thinking
- Divide and Prune Thinking
- Recursive / Divide and Conquer Thinking
- Dynamic Programming Thinking

Problem Domains

- Sorted Array Problems
- Rotated Sorted Array Problems
- Linked List Problems
- Binary Tree Problems
- BST Problems
- Balanced BST Problems
- Sorting Problems
- Selection Problems
- Random Generator and Shuffling
- 1-D Array and List Applications
- Stack Applications
- Queue Applications
- Set Applications
- Map Applications
- Sorted Set Applications
- Sorted Map Applications
- Priority Queue Applications

Algorithmic Analysis

- Time Complexity
- Space Complexity
- Notations Big-o (O), Theta (Θ), omega (Ω) and small-o(o)
- Aspects Best Case, Average case, Worst Case and Amortized Case

Module 4: BareMetal Programming and Protocols Development

"BareMetal programming will enable you in understanding schematics, and the right methods of programming MCUs by looking at the data sheets, reference manuals, step by step approach to implement protocols like I2C, SPI, UART, CAN, MODBUS, TCP/IP using GSM/GPRS, FTP using GSM/GPRS, CAN Transceivers, Zigbee, etc in generic for any microcontroller"

35+ hrs of live interactive sessions which include a conceptual framework and hands-on research examples

Technical Requirements

Desktop/Laptop with the below configuration

RAM: 2GB or more

SSD/HHD: 30 GB of free space

Operating System: Windows 10 or above with Internet Access

Hardware: Discovery Board and EmbedCore Cape board

embed CORE

Course Curriculum

- Embedded C Introduction
- Embedded C vs General C Programming
- Introduction to Controllers
- Controllers VS Processor
- Setting up the Environment
- Interpreting Datasheet
- Details of Development Board
- GPIO Programming and driver implementation
- Interrupts
- ADC Programming
- LCD Programming
- PWM Programming
- DMA Programming
- PLL Programming
- RTC and RCC Programming
- Timers Programming
- System tick timer
- USART driver development
- I2C driver development
- SPI driver development
- CAN driver development
- GSM/GPRS Module, Zigbee Module, and Modbus

embed CORE

Module 5: Controller-Based Project

Every Participant will work on a Controller Based Project

Module 6: System Programming with Linux

"System programming with Linux gives Participants deep insights into the operating system concepts, Linux system architecture, and low-level interfaces required to build system-level applications on Linux. This module is carefully designed with practical exercises that provide participants with the knowledge required to write complex systems, networks, and multithreaded applications"

45+ hrs of live interactive sessions which include a conceptual framework and hands-on research examples

Technical Requirements

Desktop/Laptop with the below configuration

RAM: 2GB or more if running on a physical host/ 4GB or more if running as a guest (Using VM Virtual Box)

SSD/HHD: 30 GB of free space

Operating System: Ubuntu 18.04 LTS or above with Internet Access

Hardware: Laptop/Desktop with Ubuntu 18.04

Course Curriculum

Fundamental Concepts

- Library Fundamentals
- Static Linker and Dynamic Linker
- Creation and use of Static and dynamic libraries
- Constructors and destructors
- ELF and Program Execution
- *ldconfig*
- Dynamically Loaded Libraries
- GOT, PLT, and Lazy Binding
- ABI, SYSV

Memory Management

- Virtual addressing (Virtual Memory)
- No VM VS Virtual Memory
- Benefits of using VM
- Process Memory Layout
- Segments or Mappings
- Stack allocation and deallocation
- Dynamic memory allocation and deallocation
- Demand paging, Locking Memory, and Memory Protection
- Linux Common Memory Issues and Test cases GDB, Valgrind, and Sanitizer tools

File I/O

- Over View
- *Open(), create(), read(), write(), and close ()*
- File Descriptor and Open file relation
- Accessing file attributes
- Modifying File Offset
- File mode
- Changing file attributes

Advanced File I/O

- Scatter – Gather I/O
- MT app file I/O
- File I/O Via Memory Mapping
- DIO and AIO
- Multiplexing or Async blocking I/O or Alternative I/O Models
- Linux I/O architecture
- Race Hazards and File Locking

Program Content

Processes management

- Converting a Program into a Process
- Process IDs
- Memory Layout of a Process and Process Address Space
- Kernel Process Address Space
- Process Scheduler
- Process Creation and Termination Calls
- Executing Programs

Signals

- Overview of Signals
- Signal Disposition
- Useful Signal-Related Functions
- Signal Types and Categories
- Generation of Signal and Delivery
- Signal Handler and Designing Async Signal Handler
- Process Communication Using Signals

Timers

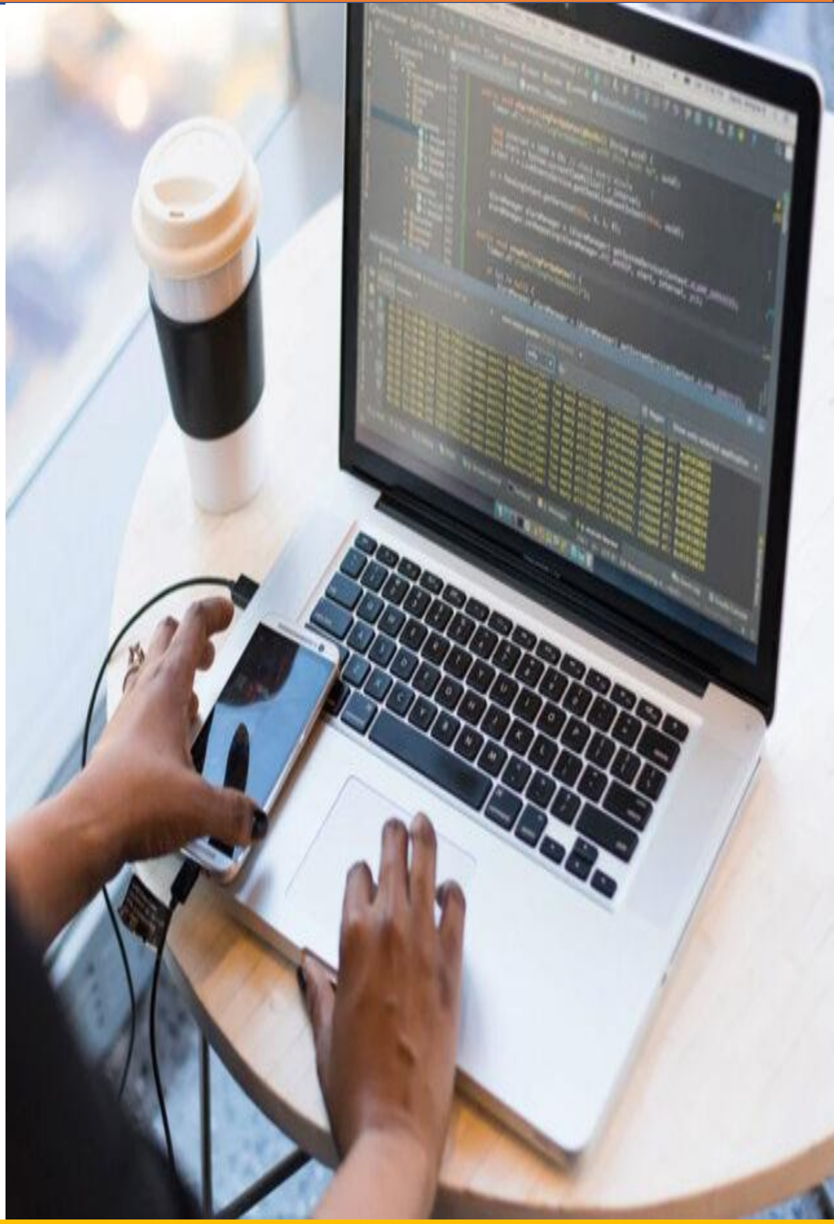
Threads

- Concurrency and Need of Concurrency
- Pthreads Library
- Thread Creation and Termination
- Thread ID's
- Joining a Thread
- Detaching a Thread and Thread Attributes
- Signals and Threads
- Threads and Process Control

Synchronization

- Shared Resources and Critical Section
- Atomic Operations
- Mutex and Insights of Mutex
- Locking, Unlocking and Deadlock
- Mutex vs Spinlocks
- Reader-Writer Locks and Producer Consumer
- Condition Variables

Program Content



IPC

- Pipes
- FIFO
- POSIX Semaphores
- POSIX Shared Memory
- Unix Domain Sockets

Internet Domain Sockets

- Internet Domain Sockets
- Data-representation Issues
- Loopback and Wildcard Addresses
- Host Addresses and Port Numbers
- Host and service conversion
- Internet Domain Sockets Example
- Additional Sockets System Calls

Module 7: Embedded Linux

18+ hrs of live interactive sessions which include a conceptual framework and hands-on research examples

Technical Requirements

Desktop/Laptop with the below configuration

RAM: 2GB or more if running on a physical host/ 4GB or more if running as a guest (Using VM Virtual Box)

SSD/HHD: 30 GB of free space

Operating System: Ubuntu 18.04 LTS or above with Internet

Hardware: BeagleBone Black and EmbedCore Cape board

Course Curriculum

- Introduction to Embedded Linux
- Bootloader
- Build process
- Required packages
- Build root
- Tool chain, kernel and rootfs configuration
- Porting images to target board

Module 8: Linux Based Project

Every Participant will work on a Linux Based Project