ED600: Certified Embedded Software Engineer – Syllabus

Module -1 Embedded C and ARM Cortex Microcontroller

Objectives:

To set the required background in embedded system concepts, Embedded ‘C’ language such as Memory management, Pointers, Data structures and architecture of the ARM Cortex processor for highly deterministic real-time applications.

Outcomes:

After successful completion of the module, the students will be able to:

- Develop embedded application using Embedded C Programming
- Choose right ARM Cortex controller with Embedded C Programming for various Applications

Duration: 140 Hours

Module topics:

‘C’ and Embedded-C

- ✓ Introduction to ‘C’ programming
- ✓ Storage Classes
- ✓ Data Types
- ✓ Controlling program flow
- ✓ Arrays
- ✓ Functions
- ✓ Memory Management
- ✓ Pointers
- ✓ Arrays and Pointers
- ✓ Pointer to Functions and advanced topics on Pointers
- ✓ Structures and Unions
- ✓ Data Structures
- ✓ Linked List
- ✓ Stacks, Queues
- ✓ Conditional Compilation
- ✓ Pre-processor directives
- ✓ File operations
- ✓ Bitwise operations
- ✓ Typecasting

Embedded Concepts

- ✓ Introduction to embedded systems

- ✓ Application Areas
- ✓ Categories of embedded systems
- ✓ Overview of embedded system architecture
- ✓ Specialties of embedded systems
- ✓ Recent trends in embedded systems
- ✓ Architecture of embedded systems
- ✓ Hardware architecture
- ✓ Software architecture
- ✓ Application Software
- ✓ Communication Software
- ✓ Development and debugging Tools

Introduction to ARM Cortex

- ✓ Architecture Introduction to 32-bit Processors
- ✓ The ARM Architecture
- ✓ Overview of ARM
- ✓ Overview of Cortex Architecture
- ✓ Cortex M4 Register Set and Modes
- ✓ Cortex M4 Processor Core
- ✓ Data Path and Instruction Decoding
- ✓ ARM Cortex M4 Development Environment
- ✓ Assembler and Compiler
- ✓ Linkers and Debuggers
- ✓ ARM-Thumb & Thumb2 instructions
- ✓ Mixing ARM & Thumb Instructions
- ✓ Memory hierarchy
- ✓ Memory Mapping
- ✓ Cache

Cortex M4 Microcontrollers & Peripherals

- ✓ Cortex M4 based controller architecture
- ✓ Memory mapping, Cortex M4 Peripherals – RCC
- ✓ GPIO
- ✓ Timer, System timer
- ✓ UARTs, LCD,ADC & PWM
- ✓ Cortex M4 interrupt handling – NVIC
- ✓ Application development with Cortex M4 controllers using standard peripheral libraries

Module -2 Embedded Linux

Objective of the Course:

To Skilling the students in Configure, Deploying and Debugging the Linux OS onto a Target Board to build a complete Embedded Product using Linux Kernel.

Outcome of the Course:

After successful completion of this module, Students will be able to:

1. Configure Linux environment for ARM based Target Boards.
2. Configure Tool-Chain for ARM Platforms.
3. Demonstrate Linux Booting Process and to configure Linux Kernels on ARM based Embedded Boards.
4. Develop ARM based Embedded Applications with Linux OS.

Duration: 70 Hours

Module topics:

1. Introduction:

- ✓ Basic Operating System Concepts
- ✓ History& Benefits of Linux
- ✓ Fundamentals of Embedded Linux OS
- ✓ Comparison of Embedded OS
- ✓ Embedded OS Tools and IDE
- ✓ Embedded Linux Applications and Products.

2. Architecture of Embedded Linux:

- ✓ What is Kernel?
- ✓ Task of kernels
- ✓ Types of kernels
- ✓ Kernel Architecture Overview
  - ➢ User Space
  - ➢ Kernel Space
- ✓ Kernel Functional Overview
  - ➢ File System
  - ➢ Process Management
  - ➢ Address Spaces and Privilege Levels

- ➢ Memory Management
- ➢ System Calls
- ➢ Inter Process Communication (IPC) – Pipes, FIFo & Shared Memory
- ➢ Device Drivers
- ➢ Network

3. Commands in Linux:

- ✓ Log In Linux system and Log in Remote Linux Systems- Getting Help
- ✓ Accessing & Working with the Command Line and Shell
- ✓ System Access, Entering Commands
- ✓ Boot Methods-Creating User Accounts &Managing Users
- ✓ Creating Groups & Managing Groups
- ✓ Directory Management
- ✓ File Permissions and Ownership
- ✓ vi Text Editor

4. **Configuring the Linux Environment:**

- ✓ Linux environment
- ✓ Types of Hosts
- ✓ Types of Host/Target Development Setups
- ✓ Types of Host/Target Debug Setups
- ✓ Embedded Environment Tools
- ✓ GNU Tool-chain Cross Compilers

5. **Tool-chain: Configuration and Cross-Compilation:**

- ✓ What is a tool-chain?
- ✓ Native vs. cross-compilation
- ✓ Toolchain Components
- ✓ Toolchain choices
- ✓ Using build root to build the toolchain
- ✓ Configuration options
- ✓ Adding path variables to startup scripts (.bashrc)
- ✓ The CROSS_COMPILE variable
- ✓ Validating the cross-compiler

6. Linux Bootloader & U-Boot:

- ✓ Boot-loader Phases
- ✓ U-boot – Embedded boot loader
- ✓ What does u-boot do?
- ✓ Navigating the u-boot sources
- ✓ Configuring and Cross-compiling u-boot

- ✓ Installing u-boot on the target
- ✓ Understanding u-boot commands
- ✓ Changing environment variables to setup kernel booting
- ✓ Transferring files to the target using tftp

7. **Embedded Linux Kernel:**

- ✓ Kernel Features
- ✓ Kernel Subsystems
  - Memory Manager
  - Scheduler
  - Embedded Storage
  - I/O Subsystem
  - Network Subsystem
- ✓ Navigating the kernel sources
- ✓ Kernel Configuration
- ✓ Kernel Compilation
- ✓ Booting the kernel using u-boot
- ✓ Module compilation and Installation to RootFS
- ✓ Procedure for adding a new driver to the kernel
- ✓ Applying patches

8. Building Root File System:

- ✓ Introduction to File system
- ✓ Linux directory structure
  - Organization and Important directories
  - /dev file system
- ✓ What next after kernel booting
  - init and startup scripts
- ✓ Downloading & Compiling RootFS
- ✓ RootFS in Flash/SD Card Storage

9. Porting OS in ARM Board:

- ✓ Kernel Compilation
- ✓ Booting the kernel using u-boot
- ✓ Porting Linux in ARM Board

10. Embedded Linux Application Programming

- ✓ Application Developments using Input Devices
- ✓ Application Developments using Output Devices
- ✓ Application Developments using Peripherals

Module -3 Embedded RTOS

Objectives:

To demystifying RTOS concept practically using Free RTOS and STM32 MCUs by

1. Understanding of RTOS concepts
2. Use cases for tasks, semaphores, queues, event flags and timers
3. Better insights of RTOS internal design and implementation
4. Design concepts needed to build an embedded system using RTOS
5. Applying taught concepts using one of the famous commercial open source RTOS.

Outcome of the Course:

After successful completion of this module, Students will be able to:

- List Step by step method to run RTOS on STM32 MCUs
- Demonstrate RTOS Scheduler with memory Management.
- Choose Right ways of Synchronizing between a task and an interrupt using semaphores.
- apply mutual exclusion between Tasks using Mutex services and semaphores
- Understand complete ARM Cortex M and FreeRTOS Priority model and its configuration related information's.

Duration: 70 Hours

Module topics:

- ✓ RTOS Introduction
  - Setting Up the Environment-Downloading and Installing RTOS
- ✓ Creating RTOS based project for STM32 MCUs
- ✓ RTOS Task Creation
- ✓ Exercise: RTOS Hello World App and Testing on hardware
- ✓ RTOS app debugging using SEGGER System View Tools
- ✓ IDLE Task and Timer Svc Task of RTOS
- ✓ RTOS Scheduler
- ✓ Context switching
- ✓ RTOS Task Notification
- ✓ Overview of RTOS Memory manage, STACK and Synchronization services
- ✓ RTOS Kernel Coding Style
- ✓ RTOS Task Deletion
- ✓ ARM Cortex M Interrupt Priority and RTOS Task Priority
- ✓ Interrupt safe APIs and Task yielding
- ✓ RTOS Task States
- ✓ RTOS : Delay APIs and its Significance
- ✓ RTOS Hook Functions
- ✓ RTOS Scheduling Policies

- ✓ RTOS Queue Management
- ✓ Semaphore for Synchronization, mutual exclusion and Interrupt Management
- ✓ Mutual exclusion

Module -4 Internet of Things (IoT)

Objectives:

To equip the students with the information required in deploying and Delivering an IoT Technologies suitable for Smart Industry.

Outcomes:

After successful completion of the module, the students will be able to:

- Implement an IoT application using Development Boards
- Develop problem solving capability using python scripts
- Choose right Data Analytic/ Machine learning tool for various IoT application
- Implement Various ML algorithms using Python.

Duration: 210 Hours

Module topics:

- ✓ IoT Concepts
  - Introduction to IoT, WoT and M2M
  - Basics of Internet & Review of TCP/IP
  - IoT Layering concepts
  - Introduction to Wireless Sensor Networks
  - Routing Protocols in WSN
  - Wireless PAN
  - Different PAN standards - Bluetooth & Zigbee, GSM, Wifi
  - IoT Development Boards
  - Data logging
- ✓ IoT Data Analytics
  - Python Programming
    - An Introduction to Python
    - Beginning Python Basics
    - Python Program Flow
    - Functions& Modules
    - Exceptions Handling
    - File Handling
    - Classes in Python
  - Data Science and Analytics
    - An Introduction to Data Science and Analytics
    - Data Analysis Using NumPy,

- Data Analysis Using Pandas
- Data Visualization – Pandas, Matplotlib, Seaborne, Plotly and Cufflinks

- Statistical Learning
  - Descriptive & Inferential Statistics,
  - Probability Concept: Marginal, Joint & Conditional Probability, Bayes Theorem
  - Probability Distributions,
  - Entropy &Information Gain,
  - Regression & Correlation,
  - Confusion Matrix, Bias & Variance
- Machine Learning
  - Introduction to Machine Learning
  - Linear Regression
  - Logistic Regression
  - K-Means Clustering
  - Decision Tree
  - Random Forest
  - K-Nearest Neighbors
  - Support Vector Machine
  - Naive Bayes

Module -5 Embedded Protocols & Device Drivers

Objectives:

To equip the students with the information required in embedded protocols and to implement the device drivers in the Linux kernel.

Outcomes:

After successful completion of the module, the students will be able to:

- Demonstrate Different embedded protocols like SPI, I2C, USB and CAN.
- Choose right protocol for the different embedded applications.
- Build driver program for various devices in Linux kernel.

Duration: 105 Hours

Module topics:

Embedded Concepts:

- ✓ Embedded Protocols
- ✓ Overview of Embedded TTY, I2C protocols, SPI, CAN Processor Bus, USB
- ✓ Overview of Linux Device drivers
- ✓ Linux Drivers overview, Review of Kernel 'Embedded C' Programming, Device driver developing Environment, the First driver.

- ✓ The Character driver: Name vs Number, Registration & the Cleanups, Kernel Data Structures & File Operations, Linux Device Model & Bus Architectures, Analog & Digital I/Os
- ✓ Low-level Accesses: Memory Access, Hardware Access.
- ✓ USB Drivers: Device & Driver Layout, USB Core, Driver & Device Registration,
- ✓ USB & its Functionalities.
- ✓ Interrupts: Interrupts & IRQs, Soft IRQs, and Exceptions.
- ✓ Block Drivers
- ✓ File System Modules: Virtual File System, The Five Operation Sets, Interaction with the Block Device
- ✓ Network Drivers

Module -6 Seminar and Case Study

Duration: 35 Hours

Module -7 Project Work

Duration: 210 Hours