TECHNOSCRIPTS EMBEDDED

ADVANCE CAREER TRACK IN EMBEDDED SYSTEMS

This embedded systems advanced course is a Career Oriented Technical course that builds on the knowledge and skills acquired in embedded systems. It is designed to provide students with a deeper understanding of the principles and techniques used to design, develop, and implement complex embedded systems.

The embedded systems advanced course is designed for students who want to expand their knowledge and skills in embedded systems development. It's intended for students who want to pursue a career in embedded systems, or for those who are working in the field and want to improve their skills and advance in their careers.

The course contents cover all the required topics, starting from basics like C language, Embedded C, 8 basic microcontrollers to more advanced topics such as application drivers, advanced microcontroller programming, communication protocols, Advance Controllers, Hardware Interfacing techniques, wireless technology with a focus on practical applications and hands-on experience.
It starts with a basic introduction to hardware and software, then goes over in-depth training on each module.

The course will also include lab sessions and hands-on projects that allow students to apply the concepts they have learned in a practical setting. Additionally, the course will be taught by industry experts who can provide insight into real-world embedded systems development and current industry trends.

Duration:

4 months

Mode of training:

We provide classroom training, online classes or hybrid model.
You can opt for Training in the classroom or you can decide to go through the training online.

Training Syllabus:

The Advanced Embedded Systems Course is structured in a systematic, step-by-step approach. Each module starts with basic concepts and proceeds to deep-dive on a relevant topic. An advanced embedded system course will transform you from a beginner to an expert. This course contains modules that are designed with a sequence in order to get you started on the right path. Each module starts with fundamentals, then deep-dives into detailed topics that will guide you all the way to a real-world project, deep knowledge that is applicable across many different domains of embedded.

INTRODUCTION TO PROGRAMMING AND C LANGUAGE:

C is a popular choice for programming embedded systems, which are small, specialized computer systems that are integrated into other devices or products. This section would cover the basics of programming, including variables, data types, operators, and control structures, pointers, file handling, preprocessor directives.

1. **Input and Output:** This section would cover the different ways to input and output data in C, including standard input and output functions such as printf and scanf.
2. **Control Structures:** This section would cover the different control structures in C, including conditional statements (if, if-else, and switch) and looping structures (while, for, and do-while).
3. **Arrays and Strings**: This section would cover the concepts of arrays and strings, including how to declare and initialize them, how to work with them, and how to manipulate strings using string library functions.
4. **Functions:** This section would cover the concepts of functions, including how to define and call functions, how to pass parameters and return values, and how to work with recursion.
5. **Pointers:** This section would cover the concepts of pointers, including how to declare and initialize pointers, how to work with pointers, and how to pass pointers as function arguments.
6. **Structures and Unions:** This section would cover the concepts of structures and unions, including how to define and use them and how to pass structures and unions as function arguments.
7. **File Handling:** This section would cover the concepts of file handling, including how to open, read, write, and close files.
8. **Advanced Topics:** This section would cover some advanced topics in C programming, such as dynamic memory allocation, preprocessor directives, Macros and libraries.
9. **Software development methodologies:** Programming best practices

EMBEDDED C PROGRAMMING

Embedded C is a version of the C programming language that is specifically designed for use in embedded systems, which are small, specialized computer systems that are integrated into other devices or products. Embedded C is a subset of C, and it includes additional features and libraries that are specific to embedded systems, such as support for low-level hardware access and real-time constraints.

1. **Data types and variables:** Understanding the basic data types used in embedded systems and how to work with variables in embedded C
2. **Control structures:** Using control structures such as if-else statements, loops, and switch-case statements to control the flow of a program as per requirements of embedded application
3. **Functions:** Defining and calling functions in Embedded C to organize code and increase reusability.
4. Pointers and memory management: Understanding how pointers work in embedded systems and how to manage memory effectively.

5. **Interrupts and timing:** Learning how to use interrupts to handle external events, and how to use timer functions to manage timing and scheduling using embedded C
6. **Hardware control:** Understanding and implementing how to control different hardware peripherals such as LEDs, LCD displays, and sensors using Embedded C
7. **Microcontroller Architecture:** This section would cover the architecture of microcontrollers, including the memory organization, and the internal and external peripherals, hardware and software components, and their applications. Embedded developers must be aware of architecture to write efficient programs using embedded C.
8. **Programming the Microcontroller:** This section would cover the basics of programming microcontrollers using C language, including the use of registers, bit manipulation and the creation of interrupt service routines.
9. **Embedded C Programming:** This section would cover the concepts of embedded C programming, including the use of pointers, structures and unions, memory management, and the use of device-specific libraries.
10. **Interfacing with Peripherals:** This section would cover the concepts of interfacing with different peripherals, including sensors, devices, actuators, and communication interfaces.

C++ PROGRAMMING

C++ is commonly used in embedded systems due to its ability to handle low-level memory management and its support for object-oriented programming. C++ is also a good choice for embedded systems because it can be used to write both high-level and low-level code.

A basic training syllabus for C++ may include the following topics:

1. **Introduction to C++:** Overview of the C++ programming language, its history, and its features.
2. **Fundamentals:** Understanding basic concepts such as variables, data types, operators, control structures, and functions.
3. **Object-oriented programming:** Learning the basics of OOP concepts such as classes, objects, inheritance, polymorphism, and encapsulation.
4. **Functions:** This module covers the concept of functions in C++, including function declaration, definition, and call.
5. **Arrays and strings:** Understanding how to work with arrays and strings in C++. Pointers and memory management: Understanding how pointers work in C++, and how to manage memory effectively.
6. **File Input/Output:** This module covers the concepts of reading from and writing to files in C++.

INTRODUCTION TO EMBEDDED

- What is embedded System
- Embedded Design development life cycle
- Embedded System Programming
- Embedded Systems Design Issues Electronics Designing Concepts
- Trends in Embedded Systems

- Challenges and Design Issues in Embedded Systems
- Memory (RAM, ROM, EPROM, EEPROM, FLASH)
- Host & Target Development environment Cross Compilers, Debuggers
- Programming Techniques used in Embedded
- Introduction to Embedded Development tools
- Assemblers, Compilers, Linkers, Loaders,
- Embedded In-Circuit Emulators and JTAG
- Tools, Build Tools for Embedded Systems
- Debugging and troubleshooting techniques
- Project development and integration

LINUX PROGRAMMING

Linux is a popular operating system that is widely used in embedded systems due to its many benefits such as its open-source nature, high reliability, flexibility, and portability.

- Introduction to Linux operating systems and its history
- Linux command line and shell scripting
- File management and permissions
- Text editors and basic programming tools
- Concepts used in Linux
- Accessing the command line (terminal and desktop)
- Accessing and using manual pages
- Working with the command line and the shell
- Piping and redirection
- Linux OS Fundamentals
- Different Linux commands like cp, mv mount
- Introduction to VI editor. VI editor settings
- Creating script
- Shell variables conditions (if else)
- Shell control structures
- Shell programs to read command line parameters
- Linux lab for shell programming
- Process creation & Process termination
- Threads, programming on threads
- Inter process communication
- Different IPC mechanism like shared memory semaphores, message queues
- Process synchronization mechanism, mutex
- Linux system calls for signals

PIC MICROCONTROLLER

PIC microcontroller (Peripheral Interface Controller) is a family of microcontrollers manufactured by Microchip Technology that are widely used in embedded systems. They are known for their small size, low power consumption, and wide range of peripherals and interfaces.

PIC microcontrollers can be found in a variety of different industries, including automotive and industrial control systems, medical devices, consumer electronics, robotics and automation, energy management systems and communication systems.

- Introduction to PIC Family of microcontrollers
- Introduction to the PIC18F4520 Microcontroller: This may include an overview of the device's features and specifications, such as its architecture, memory, and peripheral interfaces.
- Overview of Architecture of 18F4520
- Processor Core and Functional Block Diagram
- Description of memory organization
- Overview of ALL SFR's and their basic functionality
- Developing, Building, and Debugging ALP's
- Using MPLAB software
- Programming in C: This may cover the basics of programming in C for the PIC18F4520, using the C18 compiler and MPLAB IDE
- Peripherals and Hardware Interfacing: This may cover how to use the PIC18F4520's on-chip peripherals, such as GPIO, UART, ADC, and PWM.
- Timers and counters: This may cover how to use the PIC18F4520's timers and counters to measure time and generate pulse-width modulated signals.
- Debugging and troubleshooting techniques
- Project implementation on PIC18F4580

Overall outcome of the course would be understanding the architecture of PIC 18F4520, students should be able to program the device using assembly and C, use and interface the device's peripherals, handle interrupts, use timers and counters, use external memory and develop projects using PIC 18F4520.

ARM7 MICROCONTROLLER

ARM7 microcontroller is the most widely used processor in embedded systems. This microprocessor family uses the ARM7 CPU core and has a wide range of peripheral options, making it an ideal choice for applications requiring high-performance and low power consumption, superior real-time performance.

Introduction to ARM Architecture: This may include an overview of the device's features and specifications, such as its 32-bit ARM core, on-chip memory, and peripheral interfaces.

Overview of ARM & Processor Core: This may cover the ARM Cortex-M3 architecture, including the instruction set, exception handling, and memory management.

- Data Path and Instruction Decoding
- Comparison of ARM Series (ARM 10, ARM 11, Cortex)
- Conditional Execution, ARM Development Environment
- Assembler and Compilers, Software Interrupts

- Introduction to ARM family of processors
- **Keil uVision IDE:** This may cover how to use the Keil uVision integrated development environment (IDE) to develop, debug, and program the LPC2148.
- **Programming in C:** This may cover the basics of programming in C for the LPC2148, including how to use the device's peripheral interfaces and libraries.
- ARM Bus Architecture, System Peripherals, Pin Connect Block
- Timer/Counter with Interrupt
- UART programming (polling/interrupt)
- Hardware Debugging Tools
- **Peripherals and Interfacing:** This may cover how to use the LPC2148's on-chip peripherals, such as GPIO, UART, ADC, and PWM, RTC
- **Interrupt Handling:** This may cover how to use the LPC2148's interrupt controller to handle interrupt requests.
- **Real-time Clock:** This may cover how to use the LPC2148's on-chip real-time clock (RTC) to keep time and schedule events.
- **Project-based learning:** Many LPC2148 courses will include hands-on project work to give students the opportunity to apply what they have learned.

Overall outcome of the course would be understanding the architecture of LPC 2148, students should be able to program the device using C and Keil IDE, use and interface the device's peripherals, handle interrupts, use RTC and develop projects using LPC 2148.

STM32 ARM CORTEX

STM32 is a family of microcontrollers manufactured by STMicroelectronics that are based on ARM Cortex-M cores. The ARM Cortex-M is a 32-bit processor core designed for use in embedded systems. STM32 microcontrollers come in a variety of different series, each with its own set of features and capabilities.

STM32 microcontroller's core is designed for real-time applications, with a low interrupt latency and high performance, making it suitable for applications that require fast response time and high processing power.

STM32 microcontrollers are well supported by STMicroelectronics, and a wide range of software and development tools are available to help developers get started with the platform. This includes the STM32CubeMX software, which is used to configure the microcontroller's peripherals and generate code, as well as the STM32CubeIDE, an integrated development environment (IDE) for programming and debugging STM32-based applications.

- Introduction to STM32 microcontrollers and their architecture
- ST Microcontrollers and the STM32 platform
- Key Features and uses of STM32
- Understand The Internals OF STM32 Microcontroller Hardware
- Interface Various Peripherals Inside OF STM32 Microcontrollers
- Use of software and tool chains compiler, debugger and ICSP
- Programming languages for STM32 microcontrollers

- Input/output Programing with STM32
- Communication protocols implementation on STM32 such as UART, I2C, and SPI
- STM Debugging and troubleshooting techniques
- STM32 peripherals such as timers, ADC, UART
- Sensor Interfacing: Analog and Digital sensors ADC with PWM
- STM32CubeMX and STM32CubeIDE usage
- Project development and integration using STM32 Nucleo or Discovery boards.

HARDWARE INTERFACING

- Interfacing of LEDs
- Interfacing of Switches
- Interfacing of Relays
- Interfacing of LCD
- Interfacing of 7 Segment Display
- Interfacing of ADC
- Interfacing of Stepper Motors
- Interfacing of DC Motors
- Interfacing of IR Sensors
- Interfacing of Ultrasonic Sensors
- Interfacing of MEMS Sensors
- Interfacing of RF Modules
- Interfacing of Real Time Clock
- Serial Communication
- Interfacing of Camera
- Interfacing Using I2C Protocol
- Interfacing Using SPI Protocol
- PWM Techniques
- Interfacing of Bluetooth
- Interfacing of Wi-Fi
- Interfacing Ethernet
- Mobile WIFI and Bluetooth Applications
- CAN Protocol & its practical implementation

IOT COMPLETE MODULE WITH PRACTICAL INTERNET OF THINGS

Internet of Things (IoT) devices are becoming increasingly popular in embedded systems. These devices are used to connect and control a wide range of devices and systems, Embedded systems in IoT devices typically use Python or other programming languages to control the device's hardware and communicate with other devices. For example, an IoT-enabled thermostat would use a microcontroller and Python code to control the temperature and communicate with a mobile app or remote server.

- Introduction to IoT and its applications in various industries
- Understanding the IoT architecture and its components such as sensors, devices, gateways, and cloud platforms
- IoT software and programming: This module covers the different software and programming languages used in IoT, such as C & embedded C.
- Programming and communication protocols for IoT devices such as MQTT, CoAP, and HTTP
- Networking and connectivity options for IoT devices such as WIFI, Bluetooth, Internet.
- Communicating with server & Data uploading on server
- Industry trends and future developments: This module covers the latest trends and future developments in the field of IoT.
- Building and deploying IoT projects and applications such as smart home systems, industrial automation & connected systems.

COMMUNICATIONS PROTOCOLS

Communication protocols play a crucial role in embedded systems, as they enable communication between different devices and systems.

The choice of communication protocol depends on the specific requirements of the embedded system, such as the distance between devices, the amount of data to be transmitted, and the power consumption of the devices.

SERIAL UART/USART

UART (Universal Asynchronous Receiver/Transmitter) is a type of serial communication that allows devices to communicate with each other using asynchronous serial communication.

USART (Universal Synchronous and Asynchronous Receiver/Transmitter) is a type of serial communication that allows devices to communicate with each other using both synchronous and asynchronous serial communication.

This point includes the following topics:

- Introduction to UART/USART Understanding the UART/USART protocol architecture
- UART/USART communication parameters such as baud rate, data bits, stop bits, and parity
- UART/USART signalling and voltage levels, MAX 232 & it uses
- Implementing the UART/USART protocol on microcontrollers
- Using UART/USART in embedded devices
- Communication modes and its configuration
- Transmitting & receiving data using UART/USART
- Implementation of UART/USART is Serial & wireless Interfacing with devices

12C PROTOCOL

I2C (Inter-Integrated Circuit) is a communication protocol that allows multiple devices to communicate with each other using a two-wire interface.

I2C Protocol Training include the following topics:

- Introduction to I2C protocol and its history
- Understanding the I2C protocol architecture and its components such as master, slave, and clock/data lines
- I2C protocol message format and its fields such as address, data, and control bits
- I2C bus arbitration and priority mechanism
- I2C bus error handling and detection
- Implementing the I2C protocol on microcontrollers and microprocessors
- Using I2C in embedded systems and electronic devices
- Advanced topics such as I2C over long distance and multi-master communication

SPI PROTOCOLS

SPI (Serial Peripheral Interface) is a communication protocol that allows multiple devices to communicate with each other using a synchronous serial interface.
SPI is widely used in embedded systems, such as sensor interfaces, communication between microcontrollers, memory, and data storage devices.

A SPI Protocol Training include the following topics

- Introduction to the SPI protocol and its applications.
- Understanding the SPI protocol mechanism and its components such as master, slave, and clock/data lines.
- SPI protocol message format and its fields such as address, data, and control bits
- SPI bus arbitration and priority mechanism
- Multi-slave communication in SPI
- Implementing the SPI protocol on microcontrollers and microprocessors.
- Using the SPI protocol in embedded systems and electronic devices.

MQTT PROTOCOL

These protocols are commonly used for IoT, MQTT is a publish-subscribe protocol.
MQTT is a lightweight publish-subscribe protocol that is particularly well-suited for IoT and machine-to-machine (M2M) communication, where small code footprint and low bandwidth are critical.
MQTT is widely used in IoT systems, such as sensor networks, industrial automation, and home automation systems.

An MQTT (Message Queuing Telemetry Transport) syllabus would include the following topics:

- Introduction to MQTT and its history
- Understanding the MQTT protocol architecture and its components such as clients, brokers, and topics
- MQTT message format and its fields such as message type, QoS and topic
- MQTT Connect, Publish, Subscribe and Disconnect process
- MQTT Quality of Service levels
- Implementing the MQTT protocol
- Using MQTT in IoT Applications

CAN PROTOCOL

Can protocol be a communications protocol used in the automotive industry for in-vehicle networking. It is based on the Controller Area Network (CAN) bus standard, and is used to connect various electronic control units (ECUs) in a vehicle, such as the engine control unit, transmission control unit, and body control module. It is widely used in modern vehicles to control various systems and functions, such as engine and transmission control, electronic stability control, and advanced driver assistance systems.

CAN Training Syllabus include:

- Introduction to the CAN protocol and its history
- Understanding the CAN protocol architecture and its components such as frames, identifiers, and error handling
- Types of CAN Frames: Data Frame, Remote Frame, Error Frame, overload frame
- CAN protocol message format and its fields such as ID, DLC, Data and CRC
- Bit timing and synchronization in CAN
- CAN bus arbitration and priority mechanism
- CAN bus error handling and detection, types of errors
- Implementing the CAN protocol on microcontroller like STM32
- Programing & testing communication between nodes using CAN Protocol