Certainly! Embedded systems play a crucial role in our daily lives, although they often go unnoticed as they are integrated into various devices and machinery. These systems are specialized computing systems dedicated to specific tasks within larger systems. A "Pay After Placement" course in embedded systems is an opportunity for individuals to acquire skills in this field with the promise of payment only after securing a job. Let's delve into the benefits and challenges of pursuing such a course:

Introduction to Embedded Systems

Embedded systems are the backbone of modern technology, found in devices ranging from smartphones and smart appliances to automotive control systems and industrial machinery. These systems are designed to perform specific functions, often in real-time, making them integral to the functionality of the devices or systems they are a part of.

Benefits of a Pay After Placement Course in Embedded Systems

1. High Demand for Skills: The demand for professionals with expertise in embedded systems is consistently high. Completing a course in this field can open up numerous job opportunities across industries, including automotive, healthcare, consumer electronics, and more.

2. Industry-Relevant Learning: A Pay After Placement course is structured to provide practical and industry-relevant knowledge. This ensures that graduates are well-prepared to meet the demands of the job market.

3. Hands-on Experience: Embedded systems involve hands-on work with hardware and software integration. Such courses typically include practical labs and projects, allowing students to gain valuable hands-on experience, a crucial aspect in the field of embedded systems.

4. Diverse Applications: Embedded systems are used in various applications, offering professionals the flexibility to work in different industries. This diversity allows for a dynamic and interesting career path.

5. Competitive Salaries: Due to the specialized nature of embedded systems, professionals in this field often command competitive salaries. The pay-after-placement model aligns the success of the training program with the success of the graduates in securing well-paying jobs.

Challenges of a Pay After Placement Course in Embedded Systems

1. Intensive Learning Curve: Embedded systems require a deep understanding of both hardware and software components. The learning curve can be steep, and individuals without a background in electrical engineering or computer science may find it challenging initially.

2. Rapid Technological Changes: The field of embedded systems evolves rapidly, with constant advancements in technology. Staying updated with the latest tools and techniques is crucial, and professionals need to engage in continuous learning to remain competitive.

3. Complexity of Real-time Systems: Many embedded systems operate in real-time environments, where timing constraints are critical. Designing and implementing real-time systems can be complex and require a high level of precision.

4. Hardware Limitations: Embedded systems often have resource constraints, such as limited memory and processing power. Optimizing code and designing efficient algorithms to fit within these limitations can be challenging.

5. Varied Industry Requirements: While the diversity of applications is a benefit, it also means that professionals may need to adapt to different industry requirements. This adaptability can be challenging for some individuals.

EMBEDDED SYSTEMS ONLINE TRAINING SYLLABUS

Module:1 - Programming & 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.
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.
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).
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.
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.
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.

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.
File Handling: This section would cover the concepts of file handling, including how toopen, read, write, and close files.
Advanced Topics: This section would cover some advanced topics in C programming, such as dynamic memory allocation, preprocessor directives,Macros and libraries.
Software development methodologies
Programming best practices

Module:2 - 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.
Data types and variables: Understanding the basic data types used in embedded systems and how to work with variables in embedded C
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
Functions: Defining and calling functions in Embedded C to organize code and increase reusability.
Pointers and memory management: Understanding how pointers work in embedded systems and how to manage memory effectively.
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
Hardware control: Understanding and implementing how to control different hardware peripherals such as LEDs, LCD displays, and sensors using Embedded C
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.
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.
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.
Interfacing with Peripherals: This section would cover the concepts of interfacing with different peripherals, including sensors, devices, actuators, and communication interfaces.

Module:3 - C++ Programing

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:
Introduction to C++: Overview of the C++ programming language, its history, and its features.
Fundamentals: Understanding basic concepts such as variables, data types, operators, control structures, and functions.
Object-oriented programming: Learning the basics of OOP concepts such as classes, objects, inheritance, polymorphism, and encapsulation.
Functions: This module covers the concept of functions in C++, including function declaration, definition, and call.
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.
File Input/Output: This module covers the concepts of reading from and writing to files in C++.

Module:4 - 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

Module:5 - Linux Programing

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

Module:6 - 8051 Microcontroller

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

Module:7 - PIC Microcontrollers

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

Visit: Embeddedbox.com | **Call**: 8847725945 | **Mail**: Embeddedbox.official@gmail.com

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.

Module:8 - ARM7 Microcontrollers

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.

Module:9 - STM32 Microcontrollers

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
STM32 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.

Module:10 - 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 greentooth
Interfacing of Wi-Fi
Interfacing Ethernet
Mobile Wi-Fi and greentooth Applications
CAN Protocol & its practical implementation
IOT complete Module with practicals.

Module:11 - 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, greentooth, 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.

Module:12 - Communication Protocol

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.

Module:13 - Serial UART/ USART

UART (Universal async="true"hronous Receiver/Transmitter) is a type of serial communication that allows devices to communicate with each other using async="true"hronous serial communication. USART (Universal Synchronous and async="true"hronous Receiver/Transmitter) is a type of serial communication that allows devices to communicate with each other using both synchronous and async="true"hronous serial communication. The points includes
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 signaling and voltage levels, MAX 232 & its 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

Module:14 - I2C 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

Module:15 - SPI Protocol

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.

Module:16 - 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

Module:17 - 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