**A VIRTUAL INTERNSHIP REPORT ON**

**EMBEDDED SYSTEM DEVELOPER**

**Submitted in partial fulfillment of requirements for the award of the degree of**

# BACHELOR OF TECHNOLOGY IN ELECTRONICS AND COMMUNICATION ENGINEERING

## Submitted by

**KURUVABATHINI SHASHIKANTH 20091A04H3**

## Under the Guidance of Mentor

**Dr.N.Ramanjaneyulu M.Tech.,Ph.D.,MIETE**

## Associate Professor, Dept. of ECE

****

ESTD -1995

# DEPARTMENT OF

**ELECTRONICS AND COMMUNICATION ENGINEERING**

### RAJEEV GANDHI MEMORIAL COLLEGE OF ENGINEERING & TECHNOLOGY (AUTONOMOUS)

Affiliated to JNTUA, Anantapuramu, Accredited by NBA (Tier-1), New Delhi NAAC of UGC with A+ Grade, New Delhi

Nandyal – 518501, Andhra Pradesh

**ACADEMIC YEAR: 2023-2024**

DEPARTMENT OF

ELECTRONICS AND COMMUNICATION ENGINEERING



ESTD – 1995

**CERTIFICATE**

This is to certify that the report on “Embedded System Developer Virtual Internship” is a bonafide work of KURUVABATHINI SHASHIKANTH, 20091A04H3, IV B.Tech.,I- Semester student in the **Department of Electronics and Communication Engineering**, **Rajeev Gandhi Memorial College of Engineering and Technology (Autonomous)**, Nandyal, Affiliated to JNTUA, Anantapuramu, during the academic year **2023-2024**, in fulfillment of the requirement for the award of the degree of **Bachelor of Technology** of this university.

## Signature of the Mentor Signature of the Coordinator

**Dr. N. Ramanjaneyulu Mr.J.Leela Mahendra Kumar**

**M.Tech ., Ph.D.,MIETE M.Tech., (Ph.D)**

**Associate Professor, Dept.of ECE Asst. Professor, Dept. Of ECE**

## HOD OF ECE

**Dr. Kethepalli Mallikarjuna**

## Committee Members: 1.

**2.**

## 3.

**Date of Presentation:**

DEPARTMENT OF

ELECTRONICS AND COMMUNICATION ENGINEERING



ESTD – 1995

**Internship on Embedded System Developer**

Name of the Student : KURUVABATHINI SHASHIKANTH

Regd. Number : 20091A04H3

Department : ECE

Program : B.Tech

Regulation : R20

Year & Semester : IV B. Tech – I Semester

Duration : 19/05/2023 – 25/07/2023

Domain : Embedded Systems

Supported by : Microchip

Type of Internship : Free

Signature of the Student Signature of the HOD

# ACKNOWLEDGEMENT

I take this opportunity to thank one and all who have helped me in completing this internship successful. I would like to thank respected Chairman **Dr.M.Santhiramudu Garu** and Managing Director **Sri. M. Sivaram Garu** for providing facilities in the college.

My sincere thanks to my mentor **Dr. N. Ramanjaneyulu,** in Department of Electronics and Communication Engineering for his/her simulating guidance and assistance from the beginning of the internship.

My sincere thanks to my coordinator **Mr.J.Leela Mahendra Kumar**, Department of ECE for his continuous support for successful completion of internship.

I am very much thankful to **Dr. Kethepalli Mallikarjuna, Head of the Department of Electronics and Communication Engineering** for his help and encouragement in completing this internship.

My sincere thanks to our beloved principal **Dr. T. Jaya Chandra Prasad,** for providing the best faculty and facilities throughout this academic year.

Finally, I am thankful to my entire faculty and lab technicians for their good wishes and constructive criticism, which lead to the successful completion of the internship.

### BY

**KURUVABATHINI SHASHIKANTH**

20091A04H3

# VISION OF THE INSTITUTION

* To develop this rural based engineering college into an institute of technical education with global standards
* To become an institute of excellence which contributes to the needs of society
* To inculcate value- b a s e d education with noble goal of **“Education for peaceand progress”**

# MISSION OF INSTITUTION

To build a world class undergraduate program with all required infrastructurethat provides strong theoretical knowledge supplemented by the state of art skills

* To establish postgraduate programs in basic and cutting edge technologies
* To create conductive ambiance to induce and nurture research
* To turn young graduates to success oriented entrepreneurs
* To develop linkage with industries to have strong industry institute interaction
* To offer demand driven courses to meet the needs of the industry and society
* To inculcate human values and ethos into the education system for an all- round development of students

# RGMCET QUALITY POLICY

* To improve the teaching and learning
* To evaluate the performance of students at regular intervals and take necessary steps for betterment
* To establish and develop centers of excellence for research and consultancy
* To prepare students to face the competition in the market globally and realize the responsibilities as true citizen to serve the nation and uplift the country’s pride

# VISION OF ECE DEPARTMENT:

* To become a center of excellence in academics and research in Electronics and communication engineering to meet the present and future needs of society with human face

# MISSION OF ECE DEPARTMENT:

* To educate the students in latest technologies to achieve best standards in theoretical and practical aspects.
* To have a strong collaboration with electronics industry.
* To develop indigenous and appropriate technologies at low cost to help the rural people.

Internship Certificate



# CONTENTS

## S. No. Content Page No

1. [Executive Summary 1](#_bookmark0)
   1. Embedded System Developer virtual internship 1
2. [Introduction 3](#_bookmark1)
   1. Introduction to Embedded System Developer 3
   2. [Benefits of Embedded Systems 5](#_bookmark2)
   3. [Limitations of Embedded Systems 6](#_bookmark3)
   4. [Introduction to Embedded Systems 7](#_bookmark4)
   5. Embedded System Framework 9
   6. [Understanding MPLAB X IDE 10](#_bookmark5)
3. [Overview of the Organization 12](#_bookmark6)

### Internship

* 1. Embedded System Developer virtual internship

### Activity log and weekly report

* + 1. Activity log for first week 13
    2. Weekly report for first week 14
    3. Activity log for second week 16
    4. Weekly report for second week 17
    5. Activity log for third week 20
    6. Weekly report for third week 21
    7. Activity log for fourth week 23
    8. Weekly report for fourth week 24
    9. Activity log for fifth week 26
    10. Weekly report for fifth week 27
    11. Activity log for sixth week 29
    12. Weekly report for sixth week 30
    13. Activity log for seventh week 33
    14. Weekly report for seventh week 34
    15. Activity log for eighth week 37
    16. Weekly report for eighth week 38
    17. Activity log for nineth week 40
    18. Weekly report for nineth week 41
    19. Activity log for tenth week 43
    20. Weekly report for tenth week 44

### Outcomes Description 48

* 1. Technical skills acquired as an Embedded System Developer

|  |  |  |
| --- | --- | --- |
| 6. | **Conclusion** | 52 |
|  | **References** | 53 |

# EXECUTIVE SUMMARY

In today's rapidly evolving technological landscape, embedded systems play a pivotal role in powering a wide range of devices, from smartphones and IoT devices to automotive systems and medical equipment. As an embedded system developer, my expertise and experience are focused on designing, developing, and optimizing these critical systems to meet the demands of modern applications.

## Learning Objectives:

* To Learn the PIC16F1x microcontroller family, enabling them to proficiently utilize it in their own applications by covering its architecture, memory map, instruction set, interrupt structure, and special features.
* To know how to use the MPLAB® X IDE to create, debug, and test a basic "blink an LED" program on a PIC MCU Nano development board.
* To learn embedded C language firmware for microcontrollers, covering key C language constructs, decision-making, loops, functions, arrays, multi-file projects, and data pointers within a non-hardware framework, facilitating a focus on C language proficiency and application across ANSI C compilers.
* To learn advanced C programming language concepts and techniques for creating structured, portable code with a focus on data structures, pointers, function pointers, and hands-on labs.
* To learn students in advanced C programming concepts related to linked lists, including their use with and without dynamic memory allocation, to enhance flexibility in embedded projects.
* To learn the effective use of callbacks in embedded programming, enhancing code modularity and efficiency through function pointers in both polled and interrupt-driven environments.
* To advance and refine the skills of experienced C programmers by addressing common mistakes, optimizing code, and introducing techniques for more efficient and readable projects.

## Outcomes Achieved:

* Proficiency in PIC16F1x microcontroller usage.
* Competence with MPLAB® X IDE for basic program development.
* Strong foundation in embedded C language firmware.
* Advanced C programming skills with a focus on data structures and function pointers.
* Understanding of linked list implementation and flexibility in embedded projects.
* Effective use of callbacks for code modularity and efficiency.
* Identification and resolution of common programming mistakes.
* Code optimization techniques for improved efficiency and readability.

1. **INTRODUCTION**
   1. **Introduction to Embedded Systems:**

Embedded system developers play a pivotal role in the world of technology, contributing to the creation of devices and systems that impact our daily lives. These professionals specialize in designing and programming the 'brains' of electronic devices, ensuring they function efficiently and effectively. Whether it's the microcontroller in your smartphone, the control unit in your car, or the firmware in a medical device, embedded system developers are the architects behind the scenes.

Their expertise lies at the intersection of hardware and software, as they must understand the intricacies of microprocessors, microcontrollers, sensors, and other electronic components. They use programming languages like C and C++ to write code that controls the behaviour of these devices, optimizing performance, power consumption, and responsiveness.

Embedded system developers need to think creatively to solve complex problems, as they often work in resource-constrained environments, where memory and processing power are limited. They also consider real-time constraints, ensuring that systems respond rapidly and reliably to external stimuli.

The applications of embedded systems are virtually limitless, from consumer electronics to industrial automation, healthcare devices to automotive systems, and even space exploration. As technology continues to advance, the role of embedded system developers becomes increasingly critical in driving innovation and pushing the boundaries of what is possible in the digital world.

### Role and Responsibilities:

Embedded system developers are responsible for a wide range of tasks, including hardware-software integration, firmware development, debugging, and testing. They collaborate with hardware engineers, software engineers, and other stakeholders to bring embedded systems to life.

### Hardware and Software Expertise:

They possess a deep understanding of both hardware and software components, allowing them to bridge the gap between the physical world of electronics and the logical world of programming.

### Programming Languages:

Embedded system developers commonly use programming languages like C, C++, and sometimes assembly language to write efficient and compact code that runs on microcontrollers and microprocessors.

### Real-time Systems:

Many embedded systems require real-time processing capabilities, and developers must design and implement systems that meet strict timing constraints.

### Resource Optimization:

Working within resource-constrained environments, they optimize code for memory and processing power to ensure efficient operation.

### Cross-disciplinary Collaboration:

They collaborate with various teams, including hardware engineers, to ensure that the software and hardware components of embedded systems work seamlessly together.

### Industry Applications:

Embedded system developers are found in diverse industries such as automotive, aerospace, healthcare, consumer electronics, and industrial automation, each with its unique challenges and requirements.

### Rapid Technological Advancements:

The field is dynamic, with ongoing advancements in hardware, software, and connectivity driving innovation and demanding continuous learning and adaptation.

### Impact on Daily Life:

Embedded systems are integral to modern life, from smartphones and smart home devices to medical equipment and automotive safety systems.

### Career Opportunities:

The demand for skilled embedded system developers continues to grow, offering a wealth of career opportunities in a variety of industries.

Embedded system developers are the architects of the digital world, shaping the technology that surrounds us and driving progress in countless domains. Their expertise in blending hardware and software makes them indispensable in an increasingly interconnected and technologically driven society.

## Benefits of Embedded Systems:

Certainly, let's outline the benefits of becoming an embedded system developer:

### High Demand and Job Security:

Embedded system developers are in high demand across various industries, ensuring excellent job security and a wide range of career opportunities.

### Diverse Applications:

Embedded systems are used in a diverse range of applications, from automotive and healthcare to IoT devices and aerospace, providing opportunities to work on exciting and impactful projects.

### Innovation and Creativity:

The field allows for creativity and innovation as developers design solutions for complex real-world problems with limited resources.

### Competitive Salaries:

Embedded system developers often command competitive salaries due to their specialized skills and the critical role they play in product development.

* + - **Cross-disciplinary Knowledge:**

Developers gain cross-disciplinary knowledge in both hardware and software, making them versatile and valuable team members.

### Real-time Systems Expertise:

Developing real-time systems fosters expertise in ensuring timely responses and reliability, skills that are transferrable to various industries.

### Contribution to Technology Advancements:

Embedded system developers contribute to technological advancements, shaping the future of connected devices and smart technologies.

### Problem Solving and Debugging Skills:

The field hones problem-solving and debugging skills, as developers must troubleshoot hardware and software issues in complex systems.

### Constant Learning:

The rapid evolution of technology ensures that embedded system developers are continually learning and adapting to new tools and methodologies.

## Limitations of Embedded Systems:

Certainly, here's a structured explanation of the limitations of being an embedded system developer:

### Hardware Constraints:

Embedded system developers often work with limited hardware resources, including restricted memory and processing power. These constraints can significantly impact the design and functionality of embedded systems.

### Real-time Demands:

Developing real-time systems requires a deep understanding of timing constraints. Meeting these demands can be challenging and crucial for systems that must respond promptly to external events.

### Complex Integration:

Integrating various hardware and software components in embedded systems can be complex. Developers must handle different interfaces, protocols, and components, which can lead to integration challenges.

### Limited Debugging Tools:

Debugging embedded systems can be challenging due to limited debugging tools and remote deployment scenarios. Identifying and fixing issues in such environments can be time-consuming.

### Legacy Systems:

Maintaining and upgrading legacy embedded systems can be daunting. In some cases, original documentation and developers may not be available, making it difficult to understand and modify older systems.

### Heterogeneous Hardware:

Embedded system developers often work with a variety of hardware platforms and microcontrollers. Adapting to different hardware configurations can be demanding, requiring continuous learning and adjustment.

### Security Concerns:

Security is a paramount concern in embedded systems. Developers must implement robust security measures to protect against vulnerabilities, adding complexity to their work.

### Lack of Standardization:

Embedded systems development lacks universal standards, leading to variations in tools, libraries, and platforms. This can complicate cross-platform development and interoperability.

### Long Development Cycles:

Embedded system development, especially in safety-critical industries, can involve extended development cycles. Thorough testing and validation processes contribute to project timelines.

### Niche Specialization:

Specializing in embedded systems may limit career options outside this field. Skills and knowledge may not always transfer easily to other software development roles.

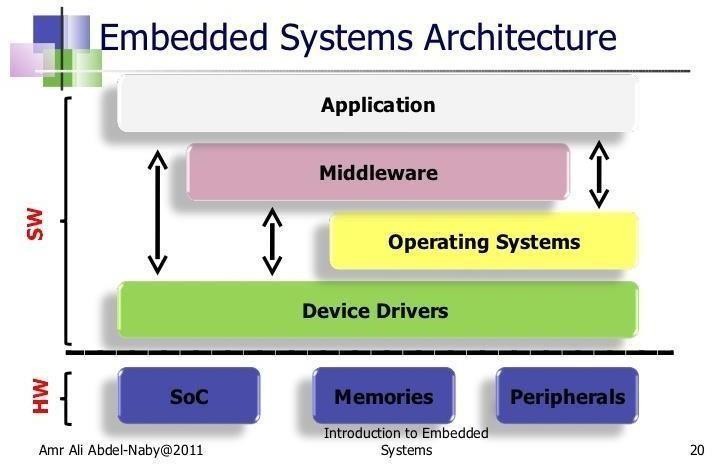
## Introduction to Embedded Systems:

Choosing the right platform for embedded systems development depends on various factors, including your project requirements, familiarity with specific tools and hardware, and budget. Here are some popular platforms and tools used for embedded systems development:

* + - **Arduino:** Arduino is a beginner-friendly platform for embedded systems development. It offers a wide range of boards with various microcontrollers, making it easy to get started with hardware projects. The Arduino IDE provides a simple development environment.
    - **Raspberry Pi:** While Raspberry Pi is primarily a single-board computer, it can also be used for embedded systems development. It's more powerful than typical microcontrollers and is suitable for projects that require more computational capabilities and can run Linux-based operating systems.
    - **Microcontrollers:** Depending on your project's requirements, you can choose microcontrollers from various manufacturers like Atmel (now Microchip Technology), STMicroelectronics, Texas Instruments, and others. These microcontrollers often require knowledge of C/C++ programming and are commonly programmed using development platforms like Atmel Studio or PlatformIO.
    - **RTOS (Real-Time Operating Systems):** If your project needs real-time capabilities, you can use an RTOS like FreeRTOS, RT-Thread, or Zephyr. These provide task scheduling and other real-time features for more complex embedded systems.
    - **Custom Hardware:** For specialized projects, you may need to design custom PCBs and select microcontrollers based on your requirements. Tools like Altium Designer and KiCad can be used for PCB design.
    - **Development Tools and IDEs:** Depending on the chosen platform, you'll need development tools and Integrated Development Environments (IDEs). Examples include Arduino IDE, PlatformIO, MPLAB X IDE (for Microchip PIC and AVR), and STM32CubeIDE (for STM32 microcontrollers).
    - **Programming Languages:** C and C++ are the most common programming languages for embedded systems. You may also use languages like Python on platforms like Raspberry Pi.
    - **Debugging and Testing:** Tools like JTAG debuggers, oscilloscopes, logic analyzers, and multimeters are essential for debugging and testing embedded systems.
    - **Communication Protocols:** Depending on your project, you may need to work with various communication protocols like UART, SPI, I2C, Ethernet, or wireless protocols like Bluetooth, Wi-Fi, and LoRa.
    - **Power and Energy Considerations**: Be mindful of power consumption and energy efficiency, especially for battery-powered devices. Tools for power profiling and measurement are crucial.
    - **Security**: If your embedded system needs security features, consider platforms that support hardware security modules and encryption algorithms.
    - **Community and Documentation**: The availability of community support and documentation can significantly impact your development experience. Platforms with active communities often have extensive resources available online.

Ultimately, the choice of platform for embedded systems development will depend on your project's specific requirements, your expertise, and your budget. It's a good idea to start with a platform you're comfortable with if you're new to embedded systems and gradually explore more advanced options as you gain experience.

## Embedded Systems Framework:

****

**Figure 1: Functional Attributes In Architecture Of Any Embedded System**

Certainly, here's a more detailed explanation of some popular embedded system frameworks and libraries: As shown in figure 1.

Arduino is a widely-used framework and platform for embedded systems development, especially favored by beginners. It offers an integrated development environment (IDE) and a range of microcontroller boards, simplifying hardware interaction through a vast library of pre- written code for various sensors and actuators. While it's user-friendly, it may not be the best choice for resource-intensive or real-time applications.

PlatformIO is an open-source ecosystem designed to streamline embedded development. It supports numerous development platforms, including Arduino, ESP8266, ESP32, and more. PlatformIO integrates seamlessly with popular code editors like Visual Studio Code, providing a unified development experience. It's a versatile choice for various projects and supports a broader range of microcontrollers compared to the Arduino IDE.

CMSIS (Cortex Microcontroller Software Interface Standard) by ARM provides a standardized set of APIs for ARM Cortex-M processors. It offers hardware abstraction layers (HALs) and peripheral driver libraries, simplifying software development for ARM-based microcontrollers. It's particularly valuable when working with ARM Cortex-M devices and ensures portability across different microcontroller manufacturers.

mbed OS is an open-source operating system designed for IoT and embedded systems, offering a rich set of libraries and APIs for ARM-based microcontrollers. It emphasizes connectivity and security, making it suitable for IoT applications. Its focus on ease of use and abstraction of hardware complexities can help accelerate development.

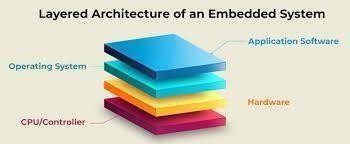
## Understanding MPLAB X IDE:

MPLAB X IDE is an integrated development environment (IDE) created by Microchip Technology for the development of embedded systems based on Microchip's microcontrollers, including PIC (Peripheral Interface Controller) and AVR (Alf and Vegard's RISC processor). It's a powerful tool designed to streamline the development process and make it easier for engineers and developers to create firmware for Microchip microcontrollers.as shown in fig 2.

Here's a breakdown of the key components and features of MPLAB X IDE:

* + - **Code Editor:** MPLAB X provides a code editor with features like syntax highlighting,code completion, and code navigation, which enhance the productivity of developers while writing code.
    - **Project Management:** You can organize your code into projects, making it easier to manage multiple source files, libraries, and configurations for different microcontroller projects.
    - **Compiler and Debugger Integration:** MPLAB X supports various compilers, including the Microchip XC compilers. It integrates with the compilers seamlessly, allowing you to build and debug your code from within the IDE. It also supports debugging through various hardware debuggers.
    - **Peripheral Libraries:** Microchip provides a set of libraries and peripheral drivers that simplify the interaction with the microcontroller's hardware peripherals, such as GPIO pins, timers, UARTs, and more. These libraries are typically used to abstract hardware details and accelerate development.
    - **Project Configuration:** MPLAB X allows you to configure project settings, including microcontroller selection, clock configuration, and optimization settings. It also supports different programming tools and debuggers.
    - **Device Support:** It supports a wide range of Microchip microcontroller families, including PIC, AVR, dsPIC, and PIC32, making it versatile for various applications.
    - **Simulator:** MPLAB X includes a simulator that allows you to test your code without needing physical hardware. This is useful for initial development and debugging.
    - **Debugging Tools:** The IDE offers a comprehensive set of debugging tools, including breakpoints, watch windows, variable inspection, and real-time execution tracing, which help diagnose and fix issues in your code.
    - **Plugins and Extensions:** MPLAB X can be extended with plugins and extensions, which can add extra functionality and tool support to the IDE.
    - **Version Control Integration:** It supports integration with version control systems like Git, allowing you to manage your source code and collaborate with team members efficiently.
    - **Community and Documentation:** There is an active user community, forums, and extensive documentation available, which can be helpful for troubleshooting and learning.

MPLAB X IDE is a valuable tool for developing firmware for Microchip microcontrollers, providing an integrated environment that simplifies the development process and enhances productivity. It's commonly used by embedded systems engineers and hobbyists working on projects based on Microchip's PIC and AVR microcontrollers.



**Figure 2 : Architecture of Embedded Systems**

# OVERVIEW OF THE ORGANISATION

Microchip Technology Inc. is a prominent organization in the embedded systems industry, specializing in semiconductor manufacturing. They are well-known for providing microcontrollers, microprocessors, and integrated circuits essential for embedded systems development. Their product portfolio includes popular families like PIC (Peripheral Interface Controller) and AVR microcontrollers, extensively used by embedded systems engineers and hobbyists. Microchip offers a comprehensive suite of development tools, such as integrated development environments (IDEs), compilers, debuggers, and programming tools, with MPLAB X IDE as a notable example. Additionally, they manufacture a wide range of peripheral ICs, sensors, communication modules, and power management solutions crucial for embedded system design. With support for real-time operating systems (RTOS), a robust developer community, and educational resources, Microchip plays a pivotal role in empowering developers to create diverse embedded applications across industries, including automotive, industrial, consumer electronics, and healthcare, making them a trusted and influential organization in the field of embedded systems as shown in the figure 3.



**Figure 3 : Microchip Technology**

# ACTIVITY LOG AND WEEKLY REPORT

## Activity log for first week

### WEEK-1:

|  |  |  |
| --- | --- | --- |
| **Day** | **Brief description of the daily activity** | **Learning Outcome** |
| Day–1 | Course on 8 Bit Microcontrollers | Learnt about PIC16 Microcontroller Architecture |
| Day–2 | Studied about Program Memory | Learnt about Program Memory |
| Day–3 | Studied about Data Memory Organization | Learnt about Data Memory Organization |
| Day–4 | Studied about Instruction Set | Learnt about Instruction Set |
| Day–5 | Studied about Interrupt and Hardware Stack | Learnt about Interrupt Vector and Hardware Stack |
| Day–6 | Studied about Special Features | Learnt about Special Features |

**Table 4.1: Activity log for the first week**

## Weekly report for first week:

**WEEK–1 (From** Date: 08-05-2023 **to** Date: 13-05-2022**)**

## Objective of the Activity Done:

To know the basics of PIC 16 Microcontroller Architecture**. Detailed Report:**

The PIC16 microcontroller, developed by Microchip Technology, is a widely-used component in embedded systems. It is part of the PIC (Peripheral Interface Controller) family and finds applications in various domains. At its core, the PIC16 features a Central Processing Unit (CPU) responsible for executing instructions stored in its memory. It follows a Harvard architecture, meaning it has distinct program and data memory spaces. The microcontroller is equipped with registers, including special function registers (SFRs) that oversee various aspects of its operation as shown in figure 4**.**

### Program Memory:

Program Memory, often referred to as Flash Memory, is where the microcontroller stores the instructions that make up the program.

In the case of PIC16 microcontrollers, this memory is non-volatile, which means that the program remains intact even when the power is turned off as shown in figure 5.

### Data Memory Organization:

Data Memory, also known as RAM (Random Access Memory), is used for temporary storage of data and variables that the program uses during execution.

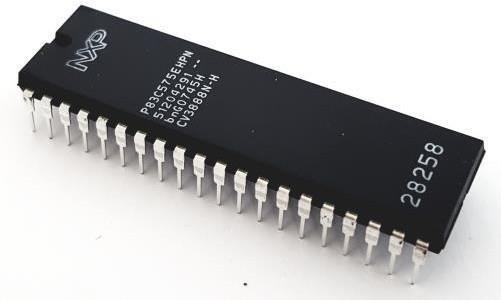
Unlike Program Memory, Data Memory is volatile, meaning it loses its contents when the power is turned off.

### Instruction Set:

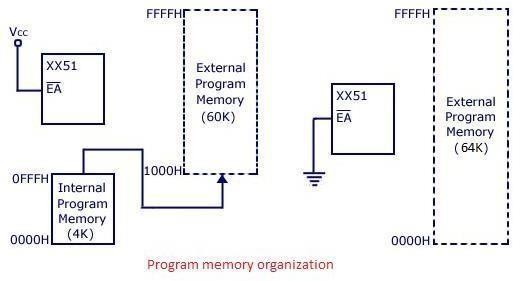
This is a collection of commands that a microcontroller can execute. Each command corresponds to a specific operation, like arithmetic, logical, or control flow tasks.

### Interrupts and Hardware Stack:

Interrupts allow the microcontroller to respond to immediate events, temporarily pausing the current task to execute a specific subroutine. The hardware stack is a special memory area that manages return addresses during these interruptions.



### Figure 4 : 8 Bit Micro-Controller

****

**Figure 5 : Data Memory Organization of 8 Bit-Controller**

## Activity log for the second week

### WEEK-2:

|  |  |  |
| --- | --- | --- |
| **Day** | **Brief description of the daily activity** | **Learning Outcome** |
| Day–1 | Studied about Basics of the MPLAB X IDE | Learnt about Basics of theMPLAB X IDE |
| Day–2 | Studied about Create and Edit Projects inMPLAB X | Learnt about DevelopingProjects |
| Day–3 | Studied about Debugging Tools and Methods | Learnt about Debugging Tools and Methods |
| Day–4 | Studied about Debugging Tools and Methods | Learnt about Watchdog Timer |
| Day–5 | Studied about Task and State Machines | Learnt about Task and State Machines |
| Day–6 | Studied about Task and State Machines | Learnt about Task and State Machines |

**Table 4.2 Activity log for the second week**

## Weekly report for second week:

**(From** Date 15-05-2023 **to** Date 20-05-2023)

## Objective of the Activity Done:

To develop a simple program.

## Detailed Report:

### Program for blink LED

#include <xc.h>

#define \_XTAL\_FREQ 8000000 // Define your microcontroller's oscillator frequency void main(void) {

TRISAbits.TRISA0 = 0; // Set pin RA0 as output while(1) {

PORTAbits.RA0 = 1; // Turn on the LED

delay\_ms(500); // Delay for 500 milliseconds PORTAbits.RA0 = 0; // Turn off the LED

delay\_ms(500); // Delay for 500 milliseconds

}

}

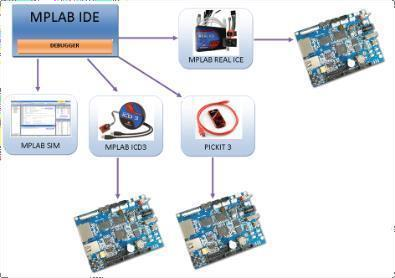
Creating and editing projects in MPLAB X is a crucial process when working with Microchip microcontrollers. To create a new project, you start by launching the MPLAB X IDE on your computer and selecting "New Project" from the "File" menu. Here, you choose the project type and the microcontroller you're working with. Next, you select the compiler toolchain, provide a name for your project, and specify the directory for its storage. Then, you configure project settings like clock frequency and memory organization. Once the project is set up, you can add source files (C, C++, or assembly) to it. These files can be either newly created or imported from existing sources. Additionally, you have the option to customize build settings and compiler options as per your requirements as shown in figure 6.

Debugging in MPLAB X involves using tools like the In-Circuit Debugger (ICD), programmers, and the Simulator to identify and resolve issues in embedded systems development. Methods include setting breakpoints, single-stepping through code, using watch windows, examining the call stack, inspecting variables, and viewing memory. Direct access to peripheral registers and, in some cases, logic analyzers may also be employed. Proficiency in these tools and methods is crucial for effective debugging in MPLAB X as shown in figure 7.

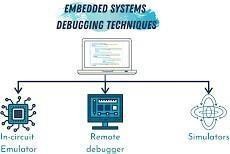
### Tasks:

In embedded systems, a task refers to a unit of work or a piece of code that performs a specific function. Tasks are often used in multitasking environments where multiple tasks can run concurrently. Each task can represent a different aspect of the system's functionality.

A state machine is a modelling technique used to represent the behaviour of a system. It defines a set of states that a system can be in and the transitions between those states based onevents or conditions. State machines are particularly useful for modelling complex systems with numerous possible states and transitions. They help in visualizing and understanding system behaviour and are often implemented in code to control the flow of a program.



### Figure 6 : Block Diagram of MPLAB X IDE



**Figure 7 : Debugging Tools of Embedded Systems**

### Activity log for the third week

**WEEK-3:**

|  |  |  |
| --- | --- | --- |
| **Day** | **Brief description of the daily activity** | **Learning Outcome** |
| Day–1 | Studied about Introduction to Embedded C. | Learnt about Syntax and Structure of C. |
| Day–2 | Studied about Comments and Variables. | Learnt about Comments and Variables. |
| Day–3 | Studied about Literals, Header File and Constants. | Learnt about Types of Literals, Header File and Constants. |
| Day–4 | Studied about Symbolic Constants. | Learnt about Symbolic Constants and Type Conversions. |
| Day–5 | Studied about Operators. | Learnt about Operators. |
| Day–6 | Studied about Operators. | Learnt about Precedence and Associativity. |

### Table 4.3: Activity log for the third week

* + 1. **Weekly report for third week WEEK–3 (From** Dt 22-05-2023 **to** Dt 27-05-2023) **Objective of the Activity Done:**

To know the syntax and structure of C.

## Detailed Report:

Understanding the syntax and structure of the C programming language is fundamental for writing efficient and error-free code. Here's a brief summary of the key aspects you've learned.

## Syntax and Structure of C:

### Basic Structure:

C programs are organized into functions. The program execution starts from the main function.

Functions consist of a header and a body, where the header includes the function's return type,name, and parameters.

### Comments:

Comments in C are used to add explanatory notes within the code. They can be either single Line comments (using //) or multi-line comments (between /\* and \*/).

### Data Types:

C has several basic data types like int, char, float, and double for representing different kinds of values. You've learned how to declare variables using these data types.

### Variables and Constants:

Variables are names given to memory locations that hold data values. Constants, on the other hand, are values that don't change during program execution.

### Operators:

C provides a wide range of operators for performing operations on variables and values. These include arithmetic operators (+, -, \*, /), relational operators (<, >, ==, !=, etc.), logical operators (&&, ||, !), and more.

### Control Structures:

You've likely learned about control structures like if-else statements for conditional execution,for and while loops for iteration, and switch statements for multi-way branching.

### Functions:

Functions allow you to break down a program into smaller, manageable pieces. They can take

parameters as inputs, perform operations, and return a value.

### Arrays:

Arrays are collections of elements of the same type that are stored in contiguous memory

locations.They allow you to work with sets of data efficiently.

### Pointers:

Pointers are variables that store memory addresses. They allow you to manipulate memory directly

and are a powerful feature of C.

### Structures:

Structures enable you to group together variables of different types under a single name.

This allows

you to create more complex data types.

### File I/O:

You may have learned about reading from and writing to files in C, using functions like fopen,

fprintf, fscanf, fclose, etc.

### Error Handling:

Handling errors and exceptions is an important aspect of programming. You may have learned about

using if statements and other constructs for error-checking.

Mastering the syntax and structure of C forms a strong foundation for further development.

## Activity log for the fourth week Week 4

|  |  |  |
| --- | --- | --- |
| **Day** | **Brief description of the daily activity** | **Learning Outcome** |
| Day–1 | Studied about Decision Statements. | Learnt about If and Switch Statements. |
| Day–2 | Studied about Loops. | Learnt about for, while, do-while loops. |
| Day–3 | Studied about Functions. | Learnt about Function parameters and Declarations. |
| Day–4 | Studied about Types of Variables. | Learnt about Types of Variables and their Scope. |
| Day–5 | Studied about Arrays. | Learnt about Arrays declaration and Types. |
| Day–6 | Studied about Strings. | Learnt about Strings Declaration and Usage. |

**Table 4.4 Activity log for the fourth week**

## Weekly report for four week

**WEEK–4 (From Dt** 29-05-2023 **to** Dt 03-06-2023)

## Objective of the Activity Done:

To know about the examples and structure of C syntax.

## Detailed Report:

### Decision Statements:

Decision statements, like if, else if, and else, allow you to execute different blocks of code based on certain conditions. These statements are crucial for implementing branching logic in your programs.

if (condition) {

// Code to execute if condition is true

} else {

// Code to execute if condition is false

}

### Loops:

Loops, such as for, while, and do-while, enable you to repeat a block of code multiple times. They're used for tasks that require repetitive execution, like iterating through arrays or processing data.

for (initialization; condition; increment/decrement) {

// Code to execute in loop

}

while (condition) {

// Code to execute as long as condition is true

}

do {

// Code to execute at least once, then repeat as long as condition is true

} while (condition);

### Functions:

Functions are blocks of code that perform a specific task. They allow you to modularize your code, making it more organized, easier to understand, and reusable. Functions can take parameters as inputs and may return a value.

return\_type function\_name(parameter\_type parameter\_name) {

// Function body

return value; // If return type is not 'void'

}

### Types of Variables:

In C, you've learned about basic data types like integers (int), characters (char), floating-point numbers (float and double), and more. These variables store different types of values. data\_type variable\_name; // Declaration

data\_type variable\_name = value; // Declaration with initialization

### Arrays:

Arrays are collections of elements of the same data type stored in contiguous memory locations. They allow you to work with sets of data efficiently, using a single variable name. data\_type array\_name[size]; // Declaration

array\_name[index] = value; // Assigning value to an element

### Strings:

Strings in C are essentially arrays of characters, terminated by a null character ('\0'). They are used to represent text. You've likely learned about functions like strlen, strcpy, strcat, etc., for string manipulation.

char greeting[] = "Hello"; // String declaration and initialization char name[20]; // Declaring a character array for a string

### Activity log for the fifth week

**Week 5**

|  |  |  |
| --- | --- | --- |
| **Day** | **Brief description of the daily activity** | **Learning Outcome** |
| Day–1 | Studied about Pointers. | Learnt about Pointers. |
| Day–2 | Studied about Pointers and Functions. | Learnt about Pointers and Functions. |
| Day–3 | Studied about Pointers and Strings. | Learnt about Pointers and String. |
| Day–4 | Studied about Arithmetic Operations on Pointers. | Learnt about pre and post Increments of Pointers. |
| Day–5 | Studied about Pointers and Arrays. | Learnt about Pointers and Arrays. |
| Day–6 | Studied about Advanced C Concepts. | Learnt about Advanced C Concepts. |

**Table 4.5 Activity log for the fifth week**

## Weekly report for fifth week

**WEEK–5(From** Dt 05-06-2023 **to** Dt 10-06-2023)

## Objective of the Activity Done:

To know about Pointers.

## Detailed Report:

### Pointers:

Pointers are variables that store memory addresses. They allow you to indirectly access and manipulate data in memory.

Example:

int num = 10;

int \*ptr = &num; // Pointer declaration and initialization with the address of 'num'

### Pointers and Functions:

Pointers can be used to pass data between functions efficiently, which can be particularly useful when working with large data structures.

Example:

void changeValue(int \*ptr) {

\*ptr = 20; // Dereferencing the pointer to change the value of the variable it points to

}

### Pointers and Strings:

Pointers are frequently used with strings. They can point to the first character of a string and traverse it.

Example:

char \*message = "Hello";

### Pre and Post Increments of Pointers:

Pre and post increments of pointers are operations that are used to change the memory address a pointer is pointing to.

Example:

int arr[] = {10, 20, 30, 40};

int \*ptr = arr;

int val1 = \*(++ptr); int val2 = \*(ptr )

### Pointers and Arrays:

Arrays and pointers have a close relationship. An array name itself is a pointer to the first element of the array.

Example:

int arr[] = {10, 20, 30, 40};

int \*ptr = arr; // 'ptr' points to the first element of 'arr'

* + 1. **Activity log for the sixth week**

## WEEK-6:

|  |  |  |
| --- | --- | --- |
| **Day** | **Brief description of the daily activity** | **Learning Outcome** |
| Day–1 | Studied about Structures | Learnt about Structure Definition, Declaration. |
| Day–2 | Studied about Structures | Learnt about Usage and About typedef Keyword |
| Day–3 | Studied about difference between Unions and Structures | Learnt about differences between Structures and Unions with examples |
| Day–4 | Studied about Pointers to Unions and Structures | Learnt about Union and Structure Pointers |
| Day–5 | Studied about Arrays of Pointers | Learnt about Initialisation and Use, Dereferencing |
| Day–6 | Studied about Function Pointers | Learnt about mechanism of Function Pointers |

**Table 4.6: Activity log for the sixth week**

## Weekly report for six week

**Week 6** (From Date 12-06-2023 to Date 17-06-2023)

## Objective of the Activity Done:

To know about Structures and Unions

## Detailed Report:

### Structure Definition and Declaration:

A structure is a user-defined data type in C that allows you to group together variables under a single name.

Example: struct Person {

char name[50]; int age;

};

struct Person john;

After defining a structure, you can create instances of it by declaring variables.

### Usage and typedef Keyword:

Once you've defined a structure, you can use it to create variables just like any other data type.

Example:

struct Person john; john.age = 30;

strcpy(john.name, "John Doe");

The typedef keyword allows you to give a type a new name, making it more readable. typedef struct Person Person;

Person jane;

### Differences between Structures and Unions:

* + **Structures:**

A structure allows you to group different data types under a single name. Each member has its own memory space.

All members can hold different values simultaneously.

* + Example:

struct Rectangle {

int length; int width;

};

* + **Unions:** Aunion is similar to a structure, but all members share the same memory space. Only one member can hold a value at a time.
  + Example:

union Number {

int i; float f;

};

### Union and Structure Pointers:

Pointers to unions and structures allow you to work with dynamic data and pass large data structures efficiently.

Example:

struct Person \*ptrPerson; union Number \*ptrNumber;

### Initialization, Use, and Dereferencing:

You can initialize structures and unions at the time of declaration or later using assignment. struct Person john = {"John Doe", 30};

To access members, you use the . operator for structures and unions. printf("Name: %s, Age: %d\n", john.name, john.age);

For pointers, you use the -> operator.

ptr Person = &john;

printf("Name: %s, Age: %d\n", ptrPerson->name, ptrPerson->age);

### Function Pointers:

Function pointers allow you to store and call functions indirectly. They are used in advanced scenarios like callbacks and dynamic function dispatch.

Example:

int add(int a, int b) { return a + b;

}

int (\*ptrAdd)(int, int) = add; int result = ptrAdd(10, 20);

In this example, ptrAdd points to the add function and can be used to call add indirectly. These advanced concepts are powerful tools in C programming, especially when dealing with complex data structures and dynamic behaviours.

* + 1. **Activity log for the seventh week**

|  |  |  |
| --- | --- | --- |
| **Day** | **Brief description of the daily activity** | **Learning Outcome** |
| Day–1 | Studied about Enumeration | Learnt about Creation and Syntax of Enum |
| Day–2 | Studied about Enumeration | Learnt about Enum use in switch/case statements |
| Day–3 | Studied about State Machines | Learnt about implementation of State Machines |
| Day–4 | Studied about LinkedList Overview | Learnt about types of LinkedList |
| Day–5 | Studied about advantages of LinkedList | Learnt about implementation of LinkedList |
| Day–6 | Studied about LinkedList with Dynamic Memory Allocation | Learnt about implementation using Dynamic Memory Allocation |

## WEEK-7:

**Table 4.7 : Activity log for the seventh week**

## Weekly report for seven week

**WEEK –7 (From** Date 19-06-2023 **to** Date 24-06-2023**)**

**Objective of the Activity Done:** To know about Enumerators, State Machines and LinkedList.

## Detailed Report:

### Creation and Syntax of Enum:

An enum, short for enumeration, is a user-defined data type in C that allows you to define a list of named values. It's particularly useful when you have a set of related constants.

Example:

enum Days {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};

### Enum Use in switch/case Statements:

Enums are often used with switch statements to improve code readability. This is especially helpful when you have a set of options or states.

Example:

enum Colors {Red, Green, Blue}; enum Colors color = Green; switch (color) {

case Red:

printf("The color is Red."); break;

case Green:

printf("The color is Green."); break;

case Blue:

printf("The color is Blue."); break;

default:

printf("Unknown color.");

}

### Implementation of State Machines:

State machines are models of computation used to represent the behavior of a system. They're crucial in embedded systems and control systems.

Example:

enum States {Idle, Running, Stopped}; enum States currentState = Idle; switch (currentState) {

case Idle:

// Perform actions for Idle state break;

case Running:

// Perform actions for Running state break;

case Stopped:

// Perform actions for Stopped state break;

default:

// Handle unexpected state

}

### Types of Linked Lists:

There are primarily three types of linked lists:

### Singly Linked List:

 Each node points to the next node in the sequence.

 Requires less memory compared to doubly linked lists due to having only one link per node.

 Easier to implement and manage because of its linear structure.

### Doubly Linked List:

 Each node points to both the next and previous nodes in the sequence.  Allows efficient traversal in both forward and backward directions.

 Makes it easier to insert or delete elements at any position in the list.

### Circular Linked List:

 The last node points back to the first node, forming a loop.

 Useful for scenarios where data processing needs to be continuous, such as managing tasks in a round-robin scheduler.

 Simplifies looping through a list since there's no real end.

### Implementation of Linked Lists:

A linked list is a collection of nodes where each node contains a value and a reference (or link) To the next node in the list.

Example:

struct Node { int data;

struct Node\* next;

};

### Implementation Using Dynamic Memory Allocation:

Dynamic memory allocation allows you to create and manage data structures whose size can change during runtime.

Example:

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

## Activity log for the eighth week

## 

### 

|  |  |  |
| --- | --- | --- |
| **Day** | **Brief description of the daily activity** | **Learning Outcome** |
| Day–1 | Studied about LinkedList without Dynamic  Memory Allocation | Learnt about implementation without Dynamic Memory Allocation |
| Day–2 | Studied about Introduction to  **.**  Callbacks | Learnt about Callbacks |
| Day–3 | Studied about Callback  Components | Learnt about Callback Components and polled with examples |
| Day–4 | Studied about Callbacks with  Interrupt Assist | Learnt about Callback Function and Registration |
| Day–5 | Studied about Callback Execution  from Interrupts | Learnt about Execution of Callback Interrupts |
| Day | Studied about Callbacks with  Libraries and Object Files | Learnt about Callback Libraries and Object Files |

**Table 4.7 : Activity log for the eight week**

## 4.8.2 Weekly report for eight week

**WEEK –8** (From Date 26-06-2023 to Date 01-07-2023)

## Objective of the Activity Done:

To know about Callbacks.

## Detailed Report:

### Implementation without Dynamic Memory Allocation:

In embedded systems programming, dynamic memory allocation (using functions like malloc in C/C++) is often discouraged or even prohibited due to the unpredictability of memory usage and potential fragmentation issues. Instead, programmers typically use statically allocated memory (e.g., arrays or structures) to manage memory efficiently.

### Callbacks:

Callbacks are functions or function pointers that are passed as arguments to other functions. They are used to define a piece of code that should be executed at a later point in time or in response toa specific event. Callbacks are commonly used in event-driven programming and are essential in scenarios where asynchronous or event-driven behaviour is required.

### Callback Components and Polling:

Callback components are modules or functions that can be registered to be called when a specific event occurs. Polling involves periodically checking the state of something, such as As input sensor, to determine if an event has occurred. Callbacks can be used in combination with polling to handle events as they occur.

### Callback Functions and Registration:

Callback functions are the functions that are called when an event occurs. Registration involves associating a callback function with a specific event or condition. This is typically done using function pointers or some kind of callback registration mechanism provided by the programming environment.

### Execution of Callback Interrupts:

In embedded systems, interrupts are used to handle events that require immediate attention. Callback functions can be executed as a response to these interrupts. When an interrupt occurs, the processor stops its current execution and jumps to the interrupt service routine (ISR), which may call one or more callback functions.

### Callback Libraries and Object Files:

Callback libraries are collections of pre-defined callback functions or components that can be used in software development to handle common tasks or events. Object files are compiled code files that can be linked together to create executable programs. Callback libraries may be distributed as object files for easy integration into projects.

These concepts are crucial in embedded systems programming, real-time systems, and event- driven applications, where efficient and predictable handling of events and memory management are essential. Depending on the specific platform and programming language you’re working with the implementation and usage of these concepts may vary, but the core principles remain the same.

## Activity log for the nineth week

|  |  |  |
| --- | --- | --- |
| **Day** | **Brief description of the daily activity** | **Learning Outcome** |
| Day–1 | Studied about Advanced Embedded C Tips and Tricks. | Learn about Goes To Functionality. |
| Day–2 | Studied about Advanced Embedded C Tips and Tricks. | Learn about = and == Mistakes. |
| Day–3 | Studied about Advanced Embedded C Tips and Tricks. | Learn about Testing for Header File Inclusion. |
| Day–4 | Studied about Programming Cautions. | Learn about Implicit Promotion Risks. |
| Day–5 | Studied about Programming Cautions. | Learnt about mixing variable types |
| Day–6 | Studied about Programming Cautions. | Learnt about Initializing and Clearing Structures and Macro Cautions. |

**Table 4.9: Activity log for the nineth week**

## Weekly report for ninth week

**WEEK –9** (From Date 03-07-2023 to Date 08-07-2023)

### Objective of the Activity Done:

To know about Programming Tips ,Tricks and Cautions.

## Detailed Report:

### 1. Goes To Functionality:

"Goes To" functionality is not a standard term in programming. It's possible that you might be

referring to a specific functionality or feature in a programming context that uses the phrase "goes to." To provide more information, please clarify the context or provide additional details.

### 2. = and == Mistakes:

In programming, = and == are two different operators used for different purposes:

= is the assignment operator, used to assign a value to a variable. For example, x = 5; assigns the value 5 to the variable x.

== is the equality operator, used to compare two values for equality. For example, if (x == 5) checks

if the value of x is equal to 5.Mistakes involving these operators often occur when:

Using = instead of == in conditional statements, which can lead to unintended assignments.

Using == for assignment, which is not valid and can lead to compiler errors or unexpected behaviour.

### Testing for Header File Inclusion:

In C and C++ programming, you can test for the inclusion of header files using preprocessor

directives like #ifndef, #define, and #endif. This is commonly used to prevent header files from

being included multiple times, which can cause compilation errors due to duplicate declarations.

This technique is often referred to as an "include guard." For example: #ifndef MY\_HEADER\_H

#define MY\_HEADER\_H

// Header file contents go here #endif

### Implicit Promotion Risks:

In C and C++, implicit type promotion can lead to unexpected behaviour. When you mix different data types in expressions, the compiler may promote one type to another in order to perform the operation. Risks can arise if you're not aware of these promotions. For example, mixing signed and unsigned types in arithmetic operations can lead to unexpected results.

### Mixing Variable Types:

Mixing variable types refers to using variables of different data types together in expressions or assignments. It's important to understand the implicit type conversions and promotions that may occur when mixing types, as this can affect the correctness of your code and lead to unexpected results.

### Initializing and Clearing Structures and Macro Cautions:

When working with structures in C and C++, it's important to initialize them properly to avoid undefined behaviour. You can initialize a structure using curly braces, and you should also clear (reset to initial values) structures when needed. Macros are often used for this purpose. However, care should be taken when using macros to ensure they behave as expected and don't lead to unexpected side effects or errors in your code.

## Activity log for the tenth week

|  |  |  |
| --- | --- | --- |
| **Day** | **Brief description of the daily activity** | **Learning Outcome** |
| Day–1 | Studied about Foundations of IOT | Learnt about Network Architecture |
| Day–2 | Studied about Foundations of IOT | Learnt about Network Architecture |
| Day–3 | Studied about Network Connectivity | Learnt about Routers,Switches |
| Day–4 | Studied about Network Connectivity | Learnt about Transport Layers Protocols |
| Day–5 | Studied about IOT Ecosystem | Learnt about Cloud Models and Trade-offs |
| Day–6 | Studied about IOT Security | Learnt about Security Needs in Embedded Applications |

**Table 4. 10: Activity log for the tenth week**

## Weekly report for tenth week

**WEEK –10** (From Date 10-07-2023 to Date 15-07-2023)

**Objective of the Activity Done:** To know about Network Architecture.

## Detailed Report:

### Network Architecture:

Network architecture refers to the design and organization of computer networks. It includes the arrangement and interconnection of various network components and devices, such as routers, switches, servers, and clients, to facilitate data communication. Network architecture can be centralized or distributed, depending on the requirements of the network.

### Routers and Switches:

Routers are network devices that connect different networks and make decisions about how to route data packets between them. They operate at the network layer (Layer 3) of the OSI model. Switches are devices that operate at the data link layer (Layer 2) of the OSI model. They are used to connect devices within a local area network (LAN) and efficiently forward data packets based on MAC addresses.

### Transport Layer Protocols:

The transport layer in the OSI model is responsible for end-to-end communication between devices. Common transport layer protocols include:

TCP (Transmission Control Protocol): Provides reliable, connection-oriented communication with error checking and flow control.

UDP (User Datagram Protocol): Offers connectionless, lightweight communication without the reliability features of TCP.

### Cloud Models and Trade-offs:

Cloud computing offers various service models, including Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Each model has its trade-offs: IaaS provides infrastructure resources like virtual machines. Users have more control but must manage the underlying infrastructure.

PaaS abstracts infrastructure management and focuses on application development. It's less flexible but simplifies deployment.

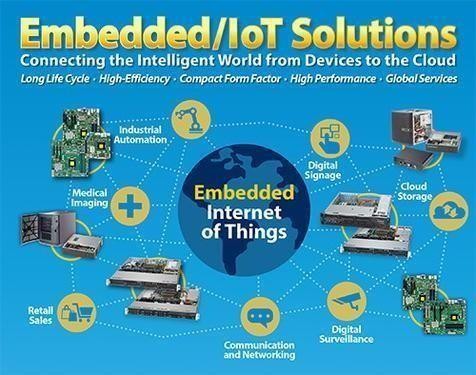
SaaS offers complete applications accessible over the internet. It's the least flexible but requires minimal management.

### Security Needs in Embedded Applications:

Security is crucial in embedded applications to protect against threats like unauthorized access, data breaches, and system vulnerabilities. Security measures in embedded systems may include:

* + **Access control**: Restricting access to authorized users.
  + **Encryption**: Protecting data from eavesdropping by encrypting it.
  + **Authentication**: Verifying the identity of users or devices.
  + **Secure boot**: Ensuring that only trusted code is executed during system startup.
  + **Firmware updates**: Securely updating device firmware to patch vulnerabilities.
  + **Monitoring and logging**: Tracking system events for security auditing.

These concepts are fundamental in the fields of networking, cloud computing, and embedded systems, each with its own set of principles and best practices. If you have specific questions or need more information on any of these topics, please feel free to ask as shown in figure 8.



**Figure 8 : Embedded Systems IOT**

## Summary:

* + **Software Development:** They write, optimize, and debug code that runs on microcontrollers, microprocessors, or other specialized hardware platforms. This code controls the behavior of the embedded system.
  + **Hardware Integration**: They work closely with hardware engineers to ensure that the software interacts seamlessly with the underlying hardware components, such as sensors, actuators, and communication interfaces.
  + **RTOS (Real-Time Operating System) Familiarity:** They may work with real-time operating systems or develop applications that require precise timing and responsiveness, such as in automotive control systems or medical devices.
  + **Peripheral Device Drivers:** They create drivers to facilitate communication between the embedded system and external devices like sensors, displays, and communication interfaces.
  + **Low-Level Programming Languages:** They are proficient in languages like C, C++, and assembly language, which are commonly used for embedded systems development.
  + **Testing and Debugging:** They rigorously test their code to ensure it meets specifications and troubleshoot any issues that arise during development.
  + **Power Management:** They optimize code for power consumption, a crucial consideration in battery-powered and energy-efficient devices.
  + **Security Considerations**: They implement security measures to protect embedded systems from unauthorized access or malicious attacks, which is especially important in applications like IoT devices.
  + **Documentation and Collaboration**: They maintain comprehensive documentation of their code and work closely with cross-functional teams, including hardware engineers, QA testers, and product managers.
  + **Firmware Update and Maintenance:** They may be responsible for creating mechanisms for updating the firmware on embedded devices, ensuring that they can be patched or upgraded over time.
  + **Regulatory Compliance**: In industries like healthcare and automotive, they ensure that the embedded systems comply with industry-specific standards and regulations.
  + **Problem-Solving Skills**: They need to be adept at identifying and addressing complex issues, especially when dealing with real-time constraints or hardware-software interactions.
  + **Continuous Learning**: Given the rapid pace of technological advancement, embedded systems developers need to stay updated with the latest tools, technologies, and best practices in the field.

In summary, an Embedded Systems Developer is a highly skilled professional responsible for designing, developing, and maintaining the software that enables specialized computing devices to perform specific tasks or functions. They play a crucial role in a wide range of industries, contributing to the development of products ranging from consumer electronics to critical medical devices.

# OUTCOMES DESCRIPTION

## Technical Skills as an Embedded System Developer

### Proficiency in PIC16F1x Microcontroller Usage:

This skill indicates a deep understanding and experience in working with PIC16F1x series microcontrollers. It involves familiarity with their architecture, features, and the ability to effectively utilize them in embedded systems projects.

### Competence with MPLAB® X IDE for Basic Program Development:

MPLAB® X IDE is an integrated development environment used for programming and debugging PIC microcontrollers. Proficiency in using this IDE implies the ability to efficiently write, debug, and manage code for PIC microcontrollers using MPLAB® X.

### Strong Foundation in Embedded C Language Firmware:

This denotes a solid grasp of the C programming language in the context of embedded systems. It includes knowledge of how to write firmware (software that runs directly on the hardware) using C, which is essential for developing applications on microcontrollers.

### Advanced C Programming Skills with a Focus on Data Structures and Function Pointers:

This indicates a high level of expertise in C programming. It involves not only writing code but also utilizing advanced programming concepts like data structures (e.g., arrays, linked lists) and function pointers. These skills are crucial for creating efficient and organized embedded software.

### Understanding of Linked List Implementation and Flexibility in Embedded Projects:

Linked lists are a fundamental data structure. Understanding their implementation in an embedded context demonstrates the ability to manage data dynamically, which is essential in scenarios where the size of data is not fixed.

### Effective Use of Callbacks for Code Modularity and Efficiency:

Callback functions are used for event handling in software. Being able to implement and utilize callbacks demonstrates a sophisticated approach to code design, enabling modularity and efficiency in embedded systems.

### Identification and Resolution of Common Programming Mistakes:

This skill involves the ability to recognize and rectify typical errors and issues that may arise during the development process. It indicates a high level of troubleshooting and debugging proficiency.

### Code Optimization Techniques for Improved Efficiency and Readability:

This skill involves the ability to refine code for better performance (efficiency) and readability. Optimized code is crucial in embedded systems, where resources may be limited, and readability is essential for maintainability.

These skills collectively demonstrate a high level of expertise in embedded systems development, showcasing proficiency in both hardware and software aspects of microcontroller-based systems. An individual with these skills would be well-equipped to tackle complex embedded projects with efficiency and precision.

## Sample Code for Embedded System Developer:

### Scenario: Digital Thermometer with LCD Display

* + - **Prerequisites:**

Basic knowledge of C programming.

A microcontroller board (e.g., Arduino or Raspberry Pi). A digital temperature sensor (e.g., DHT11 or DS18B20). An LCD display (e.g., 16x2 character LCD).

Breadboard, wires, and basic electronic components.

### Tasks:

Project Setup:

Set up your microcontroller development environment (e.g., Arduino IDE).

### Sensor Integration:

Connect the digital temperature sensor to the microcontroller following the datasheet and pinout diagrams.

### LCD Integration:

Connect the LCD display to the microcontroller and configure it for use.

### Programming:

Write a C program to read temperature data from the sensor and display it on the LCD.

### Temperature Conversion:

Implement code to convert the sensor's output into degrees Celsius or Fahrenheit.

### Display Formatting:

Format and display the temperature data along with a descriptive message (e.g., "Temperature: 25°C").

### Testing and Debugging:

Test the system to ensure accurate temperature readings and correct LCD display functionality.

Debug any issues that arise during testing.

### Documentation:

Document your project, including the circuit diagram, code explanations, and usage instructions.

### Optimization (Optional):

Optimize the code for better efficiency and readability.

Consider adding features like humidity measurement if using a sensor that supports it.

### Deployment:

Use the digital thermometer with LCD display for monitoring room temperature.

### Topic : Microcontroller LED Blinking (Using Arduino)

void setup() { pinMode(13, OUTPUT);

}

void loop() { digitalWrite(13, HIGH); delay(1000); digitalWrite(13, LOW); delay(1000);

}

**Explanation:** This code demonstrates LED blinking using the Arduino platform. In the setup function, we configure pin 13 as an output. The loop function then toggles the LED on and off with one-second intervals.

### Topic: C Programming - Structures

#include <stdio.h> struct Student {

char name[50]; int age;

};

int main() {

struct Student student1; strcpy(student1.name, "John"); student1.age = 20;

printf("Name: %s\n", student1.name); printf("Age: %d\n", student1.age); return 0;

}

**Explanation:** This C program demonstrates the use of structures. We define a structure Student with two members: name and age. We create an instance of the Student structure, populate its members, and then print the student's name and age.

### Topic : C Programming - Callback Function

#include <stdio.h>

void greet(void (\*callback)(void)) { printf("Hello! ");

callback();

}

void sayWorld() { printf("World!\n");

}

int main() { greet(sayWorld); return 0;

}

**Explanation:** This C program illustrates callback functions. The greet function accepts a callback function as an argument and calls it. In the main function, we pass the sayWorld function as a callback to greet, which then prints "Hello!" and invokes the callback, resulting in the output "Hello! World!" when executed.

# CONCLUSION

My Embedded System Developer internship has been truly transformative. I've gained invaluable insights into microcontroller architecture, particularly the PIC 16 microcontroller, enabling me to design and program embedded systems effectively. The intricacies of these microcontrollers, which once seemed like magic, have become a comprehensible language, empowering me to create efficient and functional embedded systems.

My grasp of C programming has significantly deepened, encompassing essential syntax, advanced concepts like pointers, structures, and callbacks. Practical experience has demonstrated their power in creating functional embedded applications, reinforcing the idea that mastering these fundamentals is pivotal to success in the field.

Exploring network architecture has opened new horizons, emphasizing the importance of interconnected systems in today's world, paving the way for IoT and connected devices. Understanding how embedded systems seamlessly communicate and integrate with networks has broadened my perspective, igniting my curiosity about the boundless possibilities in this interconnected landscape.

This internship has underscored that success in this field requires more than technical skills— it demands attention to detail, problem-solving prowess, and adherence to best practices. It's a testament to the intricate web of dependencies in complex projects, where precision and creativity must harmonize to create innovative solutions.

In conclusion, this internship has expanded my technical knowledge, fueled my passion for embedded systems, and equipped me with the skills and mindset needed to excel in this dynamic field. I'm now better equipped to tackle real-world challenges and contribute to innovative embedded solutions. I'm immensely grateful for the mentorship received and eagerlyanticipate a fulfilling career as an Embedded System Developer.

## References

1. <https://eduskillsfoundation.org/index.php/microchip-academy/>
2. <https://mu.microchip.com/page/embedded-system-design/>
3. <https://mu.microchip.com/8-bit-microcontrollers-architecture-of-the-pic16/>
4. <https://mu.microchip.com/intro-to-the-mplab-x-ide/>
5. https://mu.microchip.com/getting-started-with-pic16f1xxx-mcus-using-mcc-and-state- machines
6. <https://mu.microchip.com/syntax-and-structure-of-c>
7. <https://mu.microchip.com/advanced-c-programming>
8. <https://mu.microchip.com/c-programming-linked-list-data-structures>
9. <https://mu.microchip.com/c-programming-callbacks>
10. <https://mu.microchip.com/advanced-embedded-c-tips-tricks-and-cautions>
11. <https://mu.microchip.com/design-considerations-for-your-first-iot-project/586342>
12. [https://mu.microchip.com/exploring-bluetooth-low-energy-ble-for-simple-](https://mu.microchip.com/exploring-bluetooth-low-energy-ble-for-simple- applications/611848) [applications/611848](https://mu.microchip.com/exploring-bluetooth-low-energy-ble-for-simple- applications/611848)
13. <https://mu.microchip.com/creating-a-sensor-node-for-azure-iot-central>
14. [https://mu.microchip.com/motor-control-workshop-using-dspic-digital-signal-controllers-](https://mu.microchip.com/motor-control-workshop-using-dspic-digital-signal-controllers-dsc-dual-core-devices) [dsc-dual-core-devices](https://mu.microchip.com/motor-control-workshop-using-dspic-digital-signal-controllers-dsc-dual-core-devices)