



Accredited by NBA & NAAC with "A" Grade Recognised by UGC under section 2(f) & 12(B) Approved by AICTE -New Delhi Permanently Affiliated to JNTUK, SBTET Ranked as "A" Grade by Govt.o

Internship Report
On
EMBEDDED SYSTEM DEVELOPER
By
BELLAPU JAI SHANKAR SRI VASTAV
(22K65A0403)
Bachelor of Engineering
In
Electronics and Communication Engineering
AICTE-EDUSKILLS supported by MICROCHIP
TECHNOLOGY

#806, DLF Cyber City, Technology Corridor, Bhubaneswar, Odisha
751024, eduskills foundation.org, Microchip Technology-Chandler, Arizona,
USA

FROM MAY -2023
TO JULY-2023
(10 Weeks)



Certificate of Virtual Internship

This is to certify that

Jai Shankar Srivatsav

SASI Institute of Technology and Engineering

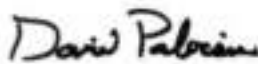
has successfully completed 10 weeks

Embedded Developer Virtual Internship

During September - November 2023

Supported By





Dave Paloian
Academic Program Manager
Microchip



Shri Buddha Chandrasekhar
Chief Coordinating Officer (CCO)
NEAT Cell, AICTE



Dr. Satya Ranjan Biswal
Chief Technology Officer (CTO)
EduSkills



Certificate ID :a1eb8d7dded3988cafeb988d34cd9d24

Student ID :STU6440d6cfe1b971681970895

Internship Report on Embedded Developer

DECLARATION

I BELLAPU JAI SHANKAR SRI VASTAV, REG No.22K65A0403 ,

student of Electronics

and communications engineering at Sasi Institute Of Technology & Engineering, Tadepalligudem. hereby declare that the summer Training Report entitled “Embedded Developer “ is an authentic record of my own work as requirements of Industrial Training during the period from 29/05/23 to 24/07/23.I obtained the knowledge of what is embedded system, what is firm ware. Through the selfless efforts of the Employee arranges to me by the Organization . A Training reports was made on the same and the suggestions are by the faculty were duly in corporate

BELLAPU JAI SHANKAR SRI VASTAV

22K65A0403

SIGNATURE OF THE INCHARGE

SIGNATURE OF THE HOD

List of Contents

Topic	Pg. No
Chapter1: Company Profile	1
1.1 : Profile	1
1.1.1 : Vision	1
1.1.2 : Mission	1
1.1.3 : Objectives	2
1.2 : Company Products and Services Offered	2
1.2.1 : Products	2
1.2.2 : Services	3
1.3 : Contact Details	4
Chapter2:Topics Learned During The Course	5
2.1 : Embedded Systems	
2.1.1 : Introduction	5
2.2 : Structure of Embedded system	6
2.2.1 : Components	6
2.2.2 : Functionality	7
2.2.3 : Block Diagram	10
2.3 : Software Tool Used	11
2.4 : Microcontrollers and Microprocessors	11
2.4.1 :Introduction and Brief Explanation	11
2.5 :Embedded System Concept	15
2.5.1 :The Goals of Embedded System	15
2.5.2 :Types of Embedded System	17
2.5.3 :Applications of Embedded System	20
Chapter3:8-bit PIC® Microcontrollers	26
3.1:PIC® MICROCONTROLLER OVERVIEW	26
3.1.1:Embed with Confidence	26

3.1.2:Advantages	26
3.2:MIGRATION STRATEGY	27
3.2.1:Scalability	27
3.3:8-BIT PIC® MICROCONTROLLER ARCHITECTURES	28
3.3.1:Basic Architecture	28
3.3.2:Explanation	29
3.4:LOW PIN COUNT AND SPACE-CONSTRAINED	30
3.4.1:Features	30
3.4.2:Family	30
3.4.3:Applications	32
3.5: MEDIUM PIN COUNT: 20, 28 and 40-PIN	33
3.5.1:Features	33
3.5.2:Working	34
3.5.3:AdvantagesandDisadvantages	34
3.6:PIC10/PIC12/PIC16 FAMILIES	36
3.6.1:Diagram	36
3.6.2:High –Performance Pic18 Family	36
3.6.3:High Performance of 8-bit Micro controllers	37
 Chapter4: MPLAB® X IDE User’s Guide	 38
4.1 :Introduction	38
4.2 : Tutorial	39
4.3 :Basic Tasks	42
4.4 :Editor and Trouble Shooting	42
4.5:MPLAB X IDE differs considerably from MPLAB IDE v8	44
 Chapter5:Syntax and Structures of C	 45
5.1 :Introduction, Comments , Variables	45
5.2 :Loops ,Functions , Arrays, Pointers	47
5.3 :Data Structures ,State machines	48
5.4 :Callbacks	50
5.5:Advanced Embedded C Tips, Tricks, and Cautions	51
 Chapter6:Internet Of Things (IOT)	
6.1 :Introduction	54
6.2 :Network Architecture	55
6.3 :Messaging Protocols	57

6.4 :Application Development	59
6.5 :Conclusion	
Chapter7:Exploring Bluetooth Low Energy from First Steps to Final App	
7.1 :Project-1: Cable Replacement	62
7.2 :Project-2: Remote Temperature Sensor	63
7.3 :Project-3:Data Logger	67
7.4 :Programming Steps	68
7.5 :Conclusion	
References	69

List Of Figures

Sl. No	Fig. No	Name of the figure	Page. No
1.	1.1	Company Logo	1
2.	2.1	Basic Diagram	5
3.	2.2	Structure Of Embedded System Components	6
4.	2.3	Von Neumann Architecture	9
5.	2.4	Block Diagram	10
6.	2.5	Mplab XIDE -logo	11
7.	2.6	Microprocessors	12
8.	2.7	Microcontrollers	14
9.	2.8	Difference versus Mp & Mc	15
10.	2.9	Types of Embedded Systems	16
11.	2.10	Applications	19
12.	3.1	Pic Micro Controllers	21
13.	3.2	Scalability of pic Micro controllers	27
14.	3.3	Performance	28
15.	3.4	Base Line Architecture	28
16.	3.5	Mid range Architecture	29
17.	3.6	Basic Architecture	29
18.	3.7	Pic10/Pic12/Pic16 families	30
19.	3.8	High Performance Pic Family	31
20.	3.9	Applications	32
21.	3.10	Families	33
22.	4.1	Pic® Mcu Data Sheet – Program Memory And Stack	36
23.	4.2	Pic® Mcu Data Sheet – Instructions (Excerpt)	38
24.	4.3	Project Wizard – Select Device	39
25.	4.4.1	Project Wizard – Select Devices	40
26.	4.4.2	Mplab XIde Desktop	41
27.	4.5	Editor Quick Performance	41
28.	4.6	Tutorial	43
29.	4.7	Basic Tasks	45
30.	4.8	Editor and Trouble Shooting	48
31.	5.1	Editor Quick Performance	51
32.	6.1	MW Architecture	57
33.	7.1	Projects	58
34.	7.5	Conclusion	68

CHAPTER1

Company Profile

It is pleasure in introducing “AICTE-EDU SKILLS” as a leading website Provide internship services industry focusing on quality standards and customer values. It is also a leading Skills and Talent Development company that is building a manpower pool for global industry requirements. This website takes the help of many international companies for providing internships of student on various domains. One of that companies is “MICRO TECHNOLOGY ”.

1.1 Profile



Fig1.1CompanyLogo

The company offers broad range of customized software applications powered by concrete technology and industry expertise. It also offers end to end embedded solutions and services. They deal with broad range of product development along with customized features ensuring at most customer satisfaction and also empower individual with knowledge, skills and competencies that assist them to escalate as integrated individuals with a sense of commitment and dedication.

1.1.1 Vision

To Empower Unskilled Individual with knowledge, skills and technical competencies in the field of Information Technology and Embedded engineering which assist them to escalate as integrated individuals contributing to company’s and Nation’s growth.

1.1.2 Mission

- Providecosteffectiveandreliablesolutionstocustomersacrossvariouslatesttechnologies.
- Offer scalable end-to-end application development and management solutions

- Provide cost effective highly scalable products for varied verticals.
- Focus on creating sustainable value growth through innovative solutions and unique partnerships.
- Create, design and deliver business solutions with high value and innovation by leveraging technology expertise and innovative business models to address long-term business objectives.
- Keep our products and services updated with the latest innovations in the respective requirement and technology.

1.1.3 Objectives

- To develop software and Embedded solutions and services focussing on quality standards and customer values.
- Offer end-to-end embedded solutions which ensure the best customer satisfaction.
- To build Skilled and Talented manpower pool for global industry requirements.
- To develop software and embedded products which are globally recognized.
- To become a global leader in Offering Scalable and cost-effective Software solutions and services across various domains like E-commerce, Banking, Finance, Health care and much more.
- To generate employment for skilled and highly talented youth of our Country INDIA.

1.2 Company Products and Services Offered

1.2.1 Products

- **MICRO CONTROLLERS (MCUs)**

Effortlessly meet the ever-changing requirements of modern electronics with our portfolio of scalable 8-bit, 16-bit and 32-bit microcontrollers (MCUs), Digital Signal Controllers (DSCs) and microprocessors (MPUs). Our flexible peripherals and functions make it easy to create differentiated applications that set you apart from your competition. You'll find it simple to get started by using our intuitive design environments and visual configuration tools, while our proven reference designs and professionally-tested software libraries lower your design risk.

- **Analog**

Manage Create complete systems while streamlining your development process and reducing your design risk. We design our mixed-signal, linear, interface and power products for seamless integration with our extensive portfolio of microcontrollers, digital signal controllers, microprocessors and FPGAs. Well-known for our thorough documentation, we provide parts that work on the bench the same way they're described on paper.

- **Amplifiers and Linear ICs**

For the simplest to the most complex designs, our extensive portfolio of amplifiers and comparators enables you to develop low-risk solutions with minimal risk of a forced redesign. These devices are also backed by our client-driven obsolescence practice of continuing to supply a product for as long as possible and while demand for the product exists.

1.2.2 Services

- **ASIC Design Services**

Micro semi offers custom integrated circuit design of Analog Mixed-Signal solutions for leading Aerospace, Avionics, Defense, Industrial and Automotive companies and is a leading ASIC manufacturer of analog embedded systems. Our custom solutions include IC chips requiring high-voltages, radiation tolerance, a focus on safety standards, and tolerance to harsh environments. Our fabless model ensures maximum flexibility in process selections allowing for optimized designs and cost-effective solutions.

- **Embedded Design and Development**

I thus expertise in Design and development of embedded products and offers solutions and services infield of Electronics.

- **FPGA and SOC Services**

FPGA & SOC design services team provides complete design services ranging from initial architecture, test bench development, feature addition to timing closure. In addition, ourDesign Services team has an in-depth expertise in developing applications.

- **Implant Training**

After completing the training. Students will be nourished and will be trained throughout with practical experience. Students will be exposed to industrial standards which boost their carrier. Students will become Acquaint to various structural partitions such as labs, workshops, assembly units, stores, and administrative unit and machinery units. They help students to understand their functions, applications and maintenance. Students will be trained from initial stage that is from collection of Project Requirements, Project Planning, Designing, implementation, testing, deployment and maintenance there by helping to understand the business model of the industry. Entire project life cycle will be demonstrated with hands on experience. Students will also be trained about management skills and teambuilding activities. They assure that by end of implant training students will Enhance communication skills and acquire technical skills, employability skills, start-up skills and will be aware of risks in industry, management skills and many other skills which are helpful to professional engagement.

- **Courses**

MICROCHIP TECHNOLOGY provides courses for students according to the interest of students keeping in mind the current technology and assist them for their further Employment. Company provides various courses such as Micro C, Embedded Systems along with live projects.

1.3 Contact Details



#Microchip Technology Inc.
2355 W. Chandler Blvd.
Chandler, AZ 85224
United States.



website at www.microchip.com
university@microchip.com.

CHAPTER2

Topics Learnt During the Course

The objective of the internship is to apply theoretical knowledge of “Embedded Systems” to create real time Sensor Devices, in order to achieve these following basic concepts were learnt:

- Micro Processors and Micro Controllers
- IOT

2.1 : Introduction

Embedded means something that is attached to another thing. An embedded system can be thought of as a computer hardware system having software embedded in it. An embedded system can be an independent system or it can be a part of a large system. An embedded system is a microcontroller or microprocessor based system which is designed to perform a specific task. For example, a fire alarm is an embedded system; it will sense only smoke. An embedded system has three components

- It has hardware.
- It has application software.
- It has Real Time Operating system (RTOS) that supervises the application software and provide mechanism to let the processor run a process as per scheduling by following a plan to control the latencies. RTOS defines the way the system works. It sets the rules during the execution of application program. A small scale embedded system may not have RTOS.

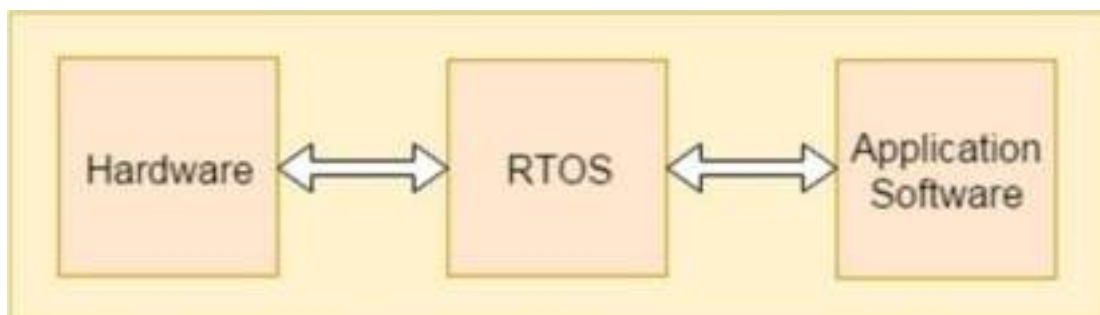


Fig2.1 Basic diagram

So we can define an embedded system as a Microcontroller based, software driven, reliable, real-time control system.

2.1 Structure Of Embedded System Components

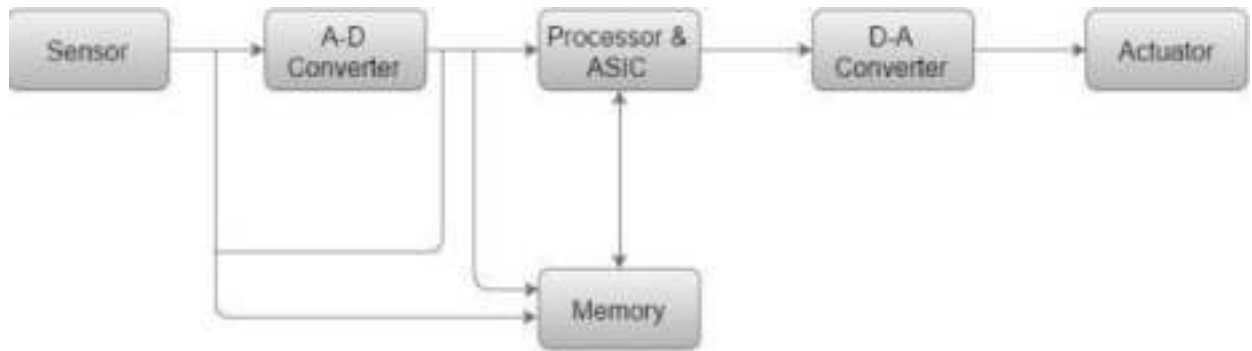


Fig 2.2: Structure of Embedded System

The following illustration shows the basic structure of an embedded system:-

- Sensor
- A-D Converter
- Processor & ASICs
- D-A Converter
- Actuator

- **Sensor** – It measures the physical quantity and converts it to an electrical signal which can be read by an observer or by any electronic instrument like an A2D converter. A sensor stores the measured quantity to the memory.
- **A-D Converter**– An analog-to-digital converter converts the analog signal sent by the sensor into a digital signal.
- **Processor & ASICs** – Processors process the data to measure the output and store it to the memory.
- **D-A Converter** – A digital-to-analog converter converts the digital data fed by the processor to analog data
- **Actuator** – An actuator compares the output given by the D-A Converter to the actual (expected) output stored in it and stores the approved output.

2.2 : Embedded Systems Functionality

Common reaction constraints identify jitter, deadlines, and throughput. The origin of these specific constraints is from the behavioral requirements of a system. However, typical execution constraints place bounds on the available processor power, speed, and failure rates of hardware. The origin of these specific constrictions is from the implementation requirements of a system.

We study execution constraints in computer engineering and reaction constraints in control theory. Obtaining control of the interaction of computation with both types of constrictions affords the ability to meet set requirements, which is the key to embedded systems design.

At its core, system design is a process of deriving (from requirements) a model from which a system generates more or less automatically. We define a model as an abstract depiction of a particular system. For example, take software design, which is the process of deriving a program for compilation. Another example is hardware design, which is the process of deriving a hardware description for synthesizing a specific circuit.

2.3.1 :The Processes of Embedded Systems

An embedded system controls various other electronic devices, which makes it a controller. As I am sure you are aware, it consists of embedded software, embedded hardware, and an environment. Overall, there are two categories of embedded systems: microcontrollers and microprocessors. We discussed the basics of the origin of the microcontroller earlier. However, the basis for the microprocessor derives from the von Neumann architecture.

The von Neumann architecture's primary elements are as follows:

- Storing instructions and data as binary digits.
- Storing instructions and data in primary storage.
- Retrieves instructions from memory serially, i.e., in order and one at a time.

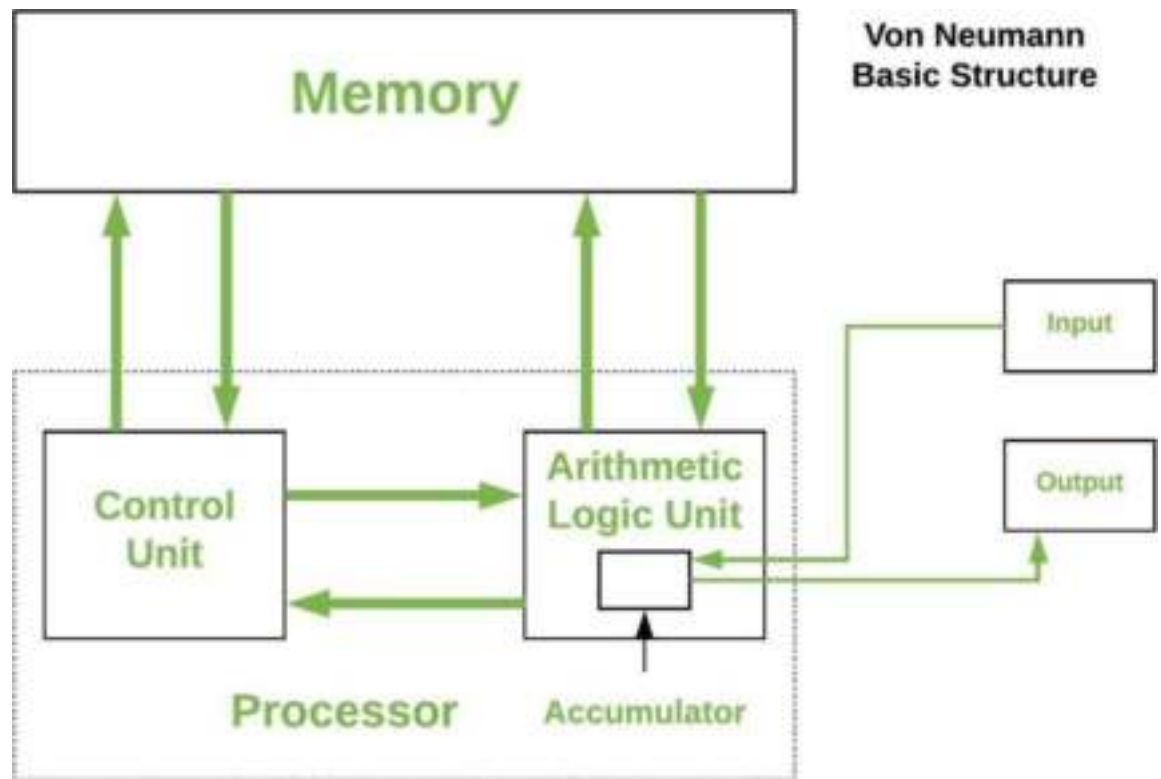


Fig 2.3: Von Neumann Architecture

2.3.2 : The Steps in the Embedded System Design Process

The various steps in the embedded system design process are as follows:

Abstraction: During this step, we abstract issues related to the system.

Software + hardware architecture: In this stage, we obtain a complete understanding of the software and hardware before initializing the design process.

Extra functional properties: During this stage, we assess the main design to gain a total understanding of the additional functions we need to implement.

System-related family of design: When you design a system, it is necessary to refer to any previous system-related designs within the same family of designs.

Modular design: One should make separate module designs so that you can utilize them later when needed.

Mapping: Here is where we conduct software mapping; for example, we map program flow and data flow into one.

User interface design: As its name implies, this correlates to the requirements of the user.

Therefore, we are considering user requirements, the function of the system, and environmental

analysis.

Refinement: At this stage, we will refine each module and every component to ensure that the software team fully understands the requirements to meet.

2.4:Block Diagram

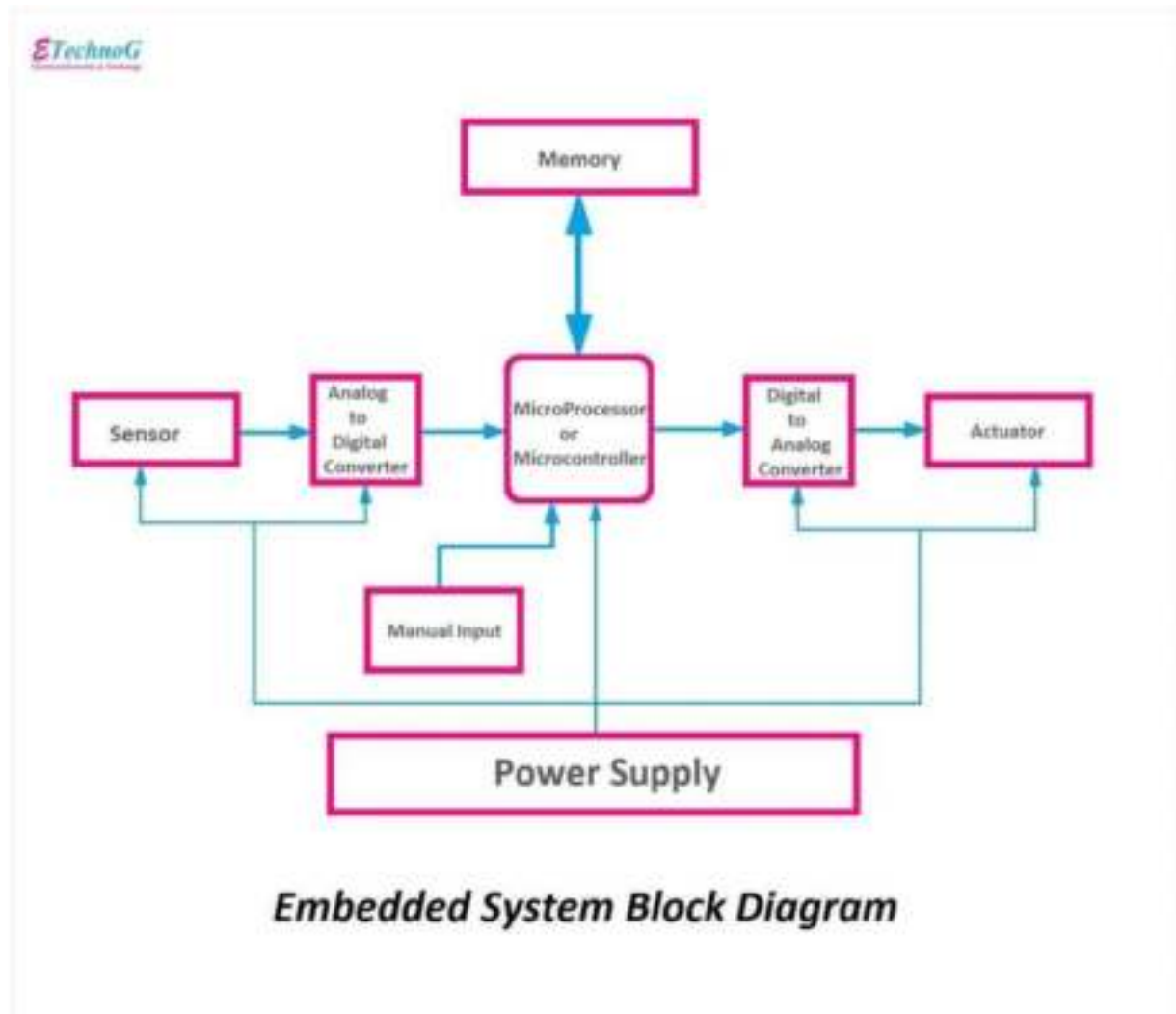


Fig 2.4: Block Diagram Of Embedded System

2.5: Software tool used



Fig 2.5:Mplab XIDE-Logo

MPLAB® X IDE is a software program that is used to develop applications for Microchip microcontrollers and digital signal controllers. (Experienced embedded-systems designers may want to skip to the next chapter.) This development tool is called an Integrated Development Environment, or IDE, because it provides a single integrated “environment” to develop code for embedded microcontrollers. This chapter describes the development of an embedded system and briefly explains how MPLAB X IDE from Microchip is used in the process. Topics discussed here include the following:

2.5.1 : An Overview of Embedded Systems

An embedded system is typically a design that uses the power of a small microcontroller like the Microchip PIC ® microcontroller (MCU) or dsPIC ® digital signal controller (DSC). These microcontrollers combine a microprocessor unit (like the CPU in a personal computer) with some additional circuits called peripherals, plus some additional circuits, on the same chip to make a small control module requiring few other external devices. This single device can then be embedded into other electronic and mechanical devices for low-cost digital control.

2.6: Micro Controllers & Micro Processors

2.6.1 : Micro Processors

Microprocessors are central processing units (CPUs) designed on a microchip, commonly referred to as a microprocessor or simply "processor." They serve as the brain of a computer system and are responsible for executing instructions and performing calculations

for various tasks and applications.

Microprocessors are integrated circuits that contain millions or billions of transistors and other electronic components. These components work together to process data, perform arithmetic and logic operations, control input/output devices, and manage the overall operation of the computer system. The key characteristics of microprocessors include:

- **Instruction Set Architecture (ISA):** The set of instructions that a microprocessor can execute. These instructions dictate how data is processed, moved, and manipulated within the CPU.
- **Clock Speed:** The speed at which a microprocessor can execute instructions, measured in cycles per second (hertz). Higher clock speeds generally lead to faster processing.
- **Cache Memory:** Small, high-speed memory located on the microprocessor chip. Cache memory is used to store frequently accessed data and instructions, which helps reduce the time required to fetch data from the main memory.
- **Cores:** Modern microprocessors often have multiple processing cores, allowing them to execute multiple tasks concurrently (multi-core processors). This enhances performance and multitasking capabilities.

Microprocessors are found in a wide range of devices, from traditional desktop and laptop computers to smartphones, tablets, gaming consoles, and various embedded systems. They have significantly evolved over the years, becoming more powerful, energy-efficient, and capable of handling complex tasks. The advancements in microprocessor technology have been instrumental in driving the development of modern computing devices and systems.

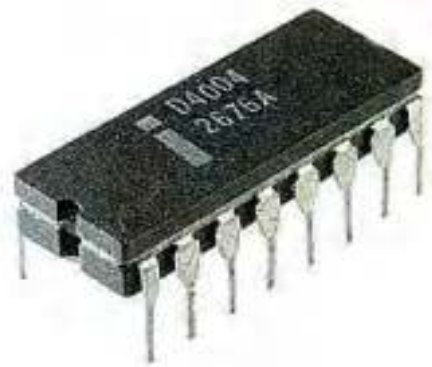


Fig 2.6: Micro Processor

2.6.2 : Micro Controllers

Microcontrollers are integrated circuits designed to function as a compact and self-contained computer system on a single chip. They are a type of microprocessor but with additional built-in components like memory, input/output peripherals, timers, and other essential elements. Unlike general-purpose microprocessors that are used in a variety of computing devices, microcontrollers are specifically intended for embedded systems and applications that require control and automation.

Key features of microcontrollers include:

- **CPU Core:** Microcontrollers have a central processing unit (CPU) core like a traditional microprocessor. However, the CPU core is often simpler and optimized for low-power and real-time processing.
- **Memory:** Microcontrollers typically include both program memory (flash or ROM) for storing the program code and data memory (RAM) for temporary data storage during program execution.
- **Input/ Output (I/O) Ports:** Microcontrollers have digital and analog I/O pins that allow them to interface with external devices and sensors. These pins can be used to read inputs from sensors and control various actuators like motors, LEDs, and displays.
- **Timers and Counters:** Built-in timers and counters help microcontrollers keep track of time, generate precise timing intervals, and measure external events.
- **Peripherals:** Microcontrollers often come with various built-in peripherals, such as serial communication ports (UART, SPI, I2C), analog-to-digital converters (ADC), pulse-width modulation (PWM) controllers, and more. These peripherals facilitate communication with other devices and enable a wide range of applications.
- **Low Power:** Microcontrollers are designed to operate with low power consumption, making them suitable for battery-operated or energy-efficient applications.

Microcontrollers are widely used in numerous applications, such as:

- Home automation and smart devices
- Industrial automation and control systems
- Automotive systems (e.g., engine control units, airbag systems)
- Robotics and drones
- Consumer electronics
- Medical devices
- Internet of Things (IOT) devices

They are preferred for applications where cost, size, power efficiency, and real-time processing capabilities are essential. Due to their versatility, microcontrollers have become a crucial component in modern electronic products and play a significant role in the advancement of technology and automation.

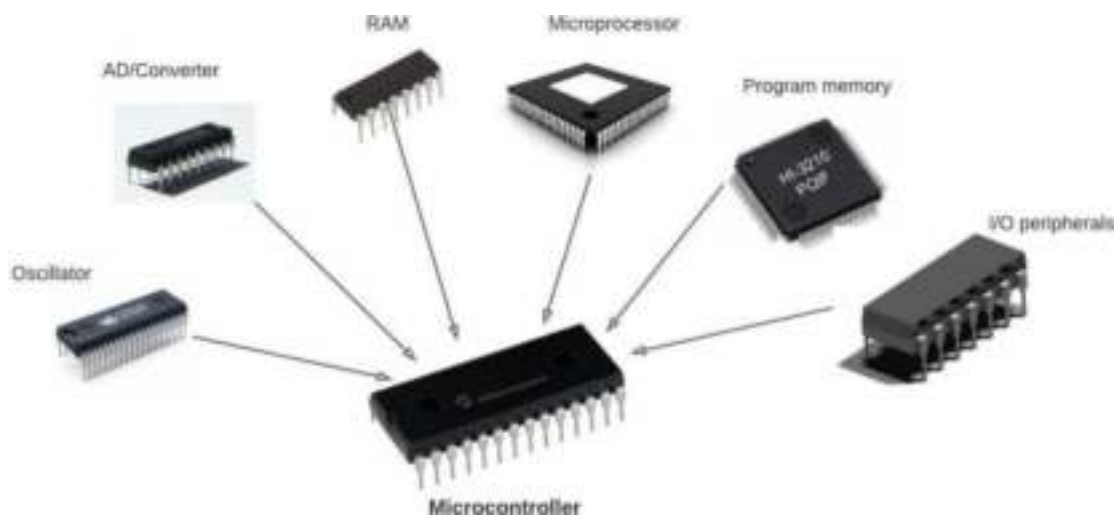


Fig2.7: Micro Controller

- **Difference between Micro Processor and Micro Controller**

Here Microprocessor and microcontroller are two fundamental components

used in many electronic devices, but they serve different purposes and have distinct characteristics

Summary	Microprocessor	Microcontroller
Applications	Advanced data processing, video, computer vision, personal computers, fast communications, multi-core computation.	Embedded devices, control systems, smartphones, consumer electronics.
Processing Power	Higher	Lower
Memory	External - Flexible	Internal – Limited Size
Power Consumption	Higher	Lower
Size	Larger	Smaller
Price	Expensive	Cheaper
I/O	Need external peripherals with I/O pins	Programmable digital and analog I/O pins

Microcontroller vs Microprocessor

Fig 2.8: Difference b/w Mp & Mc

2.7: Embedded System Goals

Embedded systems serve various goals and purposes depending on the specific application and industry. The main goals of embedded systems include:

- **Real-time operation:** Many embedded systems are designed to perform tasks in real-time, meaning they must respond to input and produce output within strict time constraints. This is crucial in applications like automotive control systems, medical devices, and industrial automation, where timely and precise responses are necessary for safety and efficiency.
- **Low power consumption:** Embedded systems are often used in devices that run on batteries or have limited power sources. One of the key goals is to minimize power consumption to prolong battery life and reduce overall energy usage.

- **Compact size:** Embedded systems are usually integrated into small devices or components. Their design aims to minimize physical size and occupy as little space as possible while still fulfilling the required functionality.
- **Reliability and robustness:** Embedded systems are commonly deployed in critical applications, where failures can have severe consequences. Ensuring high reliability and robustness is crucial to maintain system integrity and prevent malfunctions or accidents.
- **Cost-effectiveness:** Many embedded systems are used in consumer electronics and mass-produced devices. Therefore, cost-effectiveness is an essential goal, aiming to provide the desired functionality at an affordable price.
- **Integration and compatibility:** Embedded systems need to integrate seamlessly with other components and systems, both within the device they are embedded in and with external devices or networks. Compatibility with other technologies is vital to ensure proper communication and interoperability.
- **Security:** With the rise of connected embedded systems in the context of the Internet of Things (IoT), security has become a critical goal. Embedded systems must be designed with robust security measures to protect data, prevent unauthorized access, and safeguard against cyber-attacks.
- **Ease of development and maintenance:** Embedded systems development often involves a combination of hardware and software design. Simplifying the development process and ensuring ease of maintenance are crucial for efficient production and post-production support.
- **Specific functionality:** Each embedded system has a unique set of tasks and functionalities it must perform. The primary goal is to fulfill those specific requirements effectively and efficiently.
- **Adaptability and upgradability:** In some cases, embedded systems need to adapt to changing requirements or accommodate future upgrades. This goal ensures that the system can be modified or expanded without requiring a complete redesign.

Overall, the goals of embedded systems revolve around providing reliable, efficient, and cost-effective solutions for specific applications, while meeting the unique requirements of the target environment and users.

2.8: Embedded System Types

Embedded Systems are classified based on the two factors i.e.

- Performance and Functional Requirements
- Performance of Micro-controllers

Based on Performance and Functional Requirement it is divided into 4 types as follows :

1. Real-Time Embedded Systems :

A Real-Time Embedded System is strictly time specific which means these embedded systems provide output in a particular/defined time interval. These type of embedded systems provide quick response in critical situations which gives most priority to time based task performance and generation of output. That's why real time embedded systems are used in defense sector, medical and health care sector, and some other industrial applications where output in the right time is given more importance.

Further this Real-Time Embedded System is divided into two types i.e.

- **Soft Real Time Embedded Systems –**

In these types of embedded systems time/deadline is not so strictly followed. If deadline of the task is passed (means the system didn't give result in the defined time) still result or output is accepted.

- **Hard Real-Time Embedded Systems –**

In these types of embedded systems time/deadline of task is strictly followed. Task must be completed in between time frame (defined time interval) otherwise result/output may not be accepted.

Examples :

- Traffic control system
- Military usage in defense sector
- Medical usage in health sector

2. Stand Alone Embedded Systems :

Stand Alone Embedded Systems are independent systems which can work by themselves they don't depend on a host system. It takes input in digital or analog form and provides the output.

Examples :

- MP3 players
- Microwave ovens
- calculator

3. Networked Embedded Systems :

Networked Embedded Systems are connected to a network which may be wired or wireless to provide output to the attached device. They communicate with embedded web server through network.

Examples :

- Home security systems
- ATM machine
- Card swipe machine

4. Mobile Embedded Systems :

Mobile embedded systems are small and easy to use and requires less resources. They are the most preferred embedded systems. In portability point of view mobile embedded systems are also best.

Examples :

- MP3 player
- Mobile phones
- Digital Camera

Based on Performance and micro-controller it is divided into 3 types as follows:

1. Small Scale Embedded Systems :

Small Scale Embedded Systems are designed using an 8-bit or 16-bit micro-controller. They can be powered by a battery. The processor uses very less/limited resources of memory and processing speed. Mainly these systems does not act as an independent system they act as any component of

computer system but they did not compute and dedicated for a specific task.

2. Medium Scale Embedded Systems :

Medium Scale Embedded Systems are designed using an 16-bit or 32-bit micro-controller. These medium Scale Embedded Systems are faster than that of small Scale Embedded Systems.

Integration of hardware and software is complex in these systems. Java, C, C++ are the programming languages are used to develop medium scale embedded systems. Different type of software tools like compiler, debugger, simulator etc are used to develop these type of systems.

3. Sophisticated or Complex Embedded Systems :

Sophisticated or Complex Embedded Systems are designed using multiple 32 -bit or 64-bit micro-controller. These systems are developed to perform large scale complex functions. These systems have high hardware and software complexities. We use both hardware and software components to design final systems or hardware products.

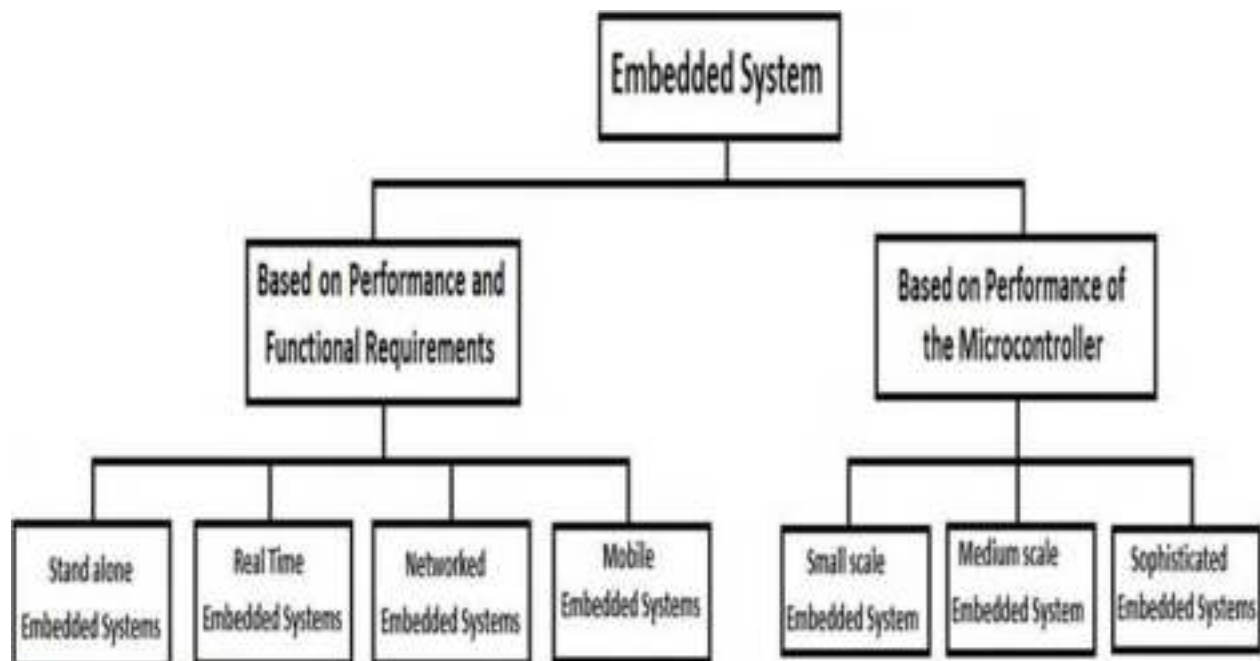


Fig 2.9: Types of Embedded Systems

2.9: Applications

Embedded systems play a crucial role in various applications across multiple industries due to their reliability, compact size, and efficiency. Some of the common applications of embedded systems include:

- 1. Consumer Electronics:** Embedded systems are found in everyday consumer electronics such as smartphones, digital cameras, smart TVs, home appliances (microwaves, washing machines, refrigerators), and gaming consoles.
- 2. Automotive Industry:** Modern vehicles heavily rely on embedded systems for engine control units (ECUs), anti-lock braking systems (ABS), airbag systems, entertainment systems, and various safety features like collision avoidance.
- 3. Industrial Automation:** Embedded systems are extensively used in industrial automation for controlling machinery, robotics, process control, data acquisition, and monitoring systems in manufacturing plants.
- 4. Healthcare:** Medical devices and equipment, such as infusion pumps, pacemakers, glucose monitors, and medical imaging systems, incorporate embedded systems to provide accurate and timely patient care.
- 5. Aerospace and Defense:** Embedded systems are utilized in avionics, communication systems, radar systems, navigation systems, and unmanned aerial vehicles (UAVs) for military and civilian applications.
- 6. Home Automation:** Smart home systems use embedded systems to control and manage various aspects of home environments, such as lighting, heating, cooling, security cameras, and smart locks.
- 7. Internet of Things (IoT):** IoT devices rely heavily on embedded systems to connect everyday objects to the internet, enabling data collection, analysis, and control remotely.
- 8. Wearable Devices:** Smart watches, fitness trackers, and other wearable gadgets leverage embedded systems to process sensor data, display information, and support various functionalities.
- 9. Communication Systems:** Embedded systems are essential components in networking equipment like routers, switches, modems, and base stations for seamless data communication.
- 10. Digital Signage and Kiosks:** Embedded systems power interactive displays and kiosks

used in public spaces, retail environments, and information centers.

11.Security Systems: Embedded systems are integral to security devices like surveillance cameras, access control systems, and alarm systems.

12. Environmental Monitoring: Embedded systems are employed in weather stations, pollution monitoring equipment, and other environmental sensors for data collection and analysis.

13.Energy Management: Smart grids and energy management systems rely on embedded systems to monitor energy consumption and optimize power distribution.

14. Entertainment and Gaming: Video game consoles, handheld gaming devices, and arcade machines use embedded systems to deliver immersive gaming experiences.

15. Agriculture: Embedded systems are utilized in precision agriculture for tasks like automated irrigation, soil monitoring, and drone-based crop surveillance.

The versatility of embedded systems continues to drive innovation across various industries, contributing to the advancement of technology and improving the quality of life.



Fig 2.9.1: Applications of Embedded Systems

2.10 : Examples

Embedded systems are specialized computing systems designed to perform specific tasks or functions within larger systems. They are often integrated into various devices, machines, and appliances. Here are some examples of embedded systems found in everyday life:

Smartphones: The modern smartphone is an example of a powerful embedded system that combines a multitude of functionalities, such as communication, computation, multimedia, and more.

Home Appliances: Many home appliances, like washing machines, refrigerators, microwave ovens, and air conditioners, contain embedded systems to control and monitor their operations.

Automotive Systems: Vehicles contain numerous embedded systems, such as engine control units (ECUs), airbag systems, entertainment systems, and navigation systems.

Smart TVs: Smart televisions incorporate embedded systems to provide smart features like internet connectivity, apps, and streaming services.

1. Digital Cameras: Digital cameras use embedded systems for image processing, storage, and user interface functions.

2. Wearable Devices: Devices like smart watches and fitness trackers utilize embedded systems to monitor health data and provide various functionalities.

3. Industrial Control Systems: Embedded systems play a crucial role in controlling and monitoring processes in industries, including manufacturing, energy, and transportation.

4. Medical Devices: Many medical devices, such as pacemakers, infusion pumps, and glucose monitors, are equipped with embedded systems to assist in patient care.

5. Home Automation Systems: Smart home devices like smart thermostats, security cameras, and lighting systems rely on embedded systems for automation and control.

6. ATM Machines: Automated Teller Machines (ATMs) use embedded systems for transaction processing and security.

7. Point-of-Sale (POS) Systems: POS systems, commonly found in retail environments, use embedded systems to handle transactions and inventory management.

2.11 : Advantages

Embedded systems offer several advantages that make them ideal for a wide range of applications and industries. Here are some key advantages of embedded systems:

- **Efficiency:** Embedded systems are designed to perform specific tasks efficiently, without the overhead of a general-purpose computer. This specialization allows them to execute

tasks quickly and with lower power consumption.

- **Size and Form Factor:** Embedded systems can be compact and lightweight, making them suitable for integration into small devices and products with limited physical space.
- **Real-time Operation:** Many embedded systems are designed for real-time operation, enabling them to respond quickly to input and produce output within specific timing constraints. This is essential for applications like automotive control, industrial automation, and medical devices.
- **Reliability:** Embedded systems are often used in critical applications where reliability is paramount. Due to their dedicated nature, they tend to have fewer points of failure compared to general-purpose systems.
- **Cost-Effectiveness:** By focusing on specific functionalities, embedded systems can be more cost-effective than developing full-scale general-purpose computers for specialized tasks.
- **Power Efficiency:** Many embedded systems are optimized for low power consumption, making them suitable for battery-powered and energy-efficient devices.
- **Tailored Functionality:** Manufacturers can customize embedded systems to meet the exact requirements of their products or applications, providing precise and tailored functionality.
- **Fast Boot-Up:** Embedded systems typically have faster boot-up times compared to general-purpose computers, making them suitable for applications that require quick start-up, such as automotive infotainment systems or consumer electronics.
- **Security:** Embedded systems can be designed with security measures specific to their application, reducing the risk of unauthorized access or data breaches.
- **Integration with Hardware:** Embedded systems are tightly integrated with the hardware they control, allowing for optimized performance and lower overhead compared to general-purpose software running on standard hardware.
- **Standalone Operation:** Many embedded systems can operate independently without constant human intervention, making them suitable for remote or unmanned applications.
- **Scalability:** Embedded systems can be designed with scalability in mind, enabling them to adapt to different performance and memory requirements as needed.
- **Long Lifecycle:** Embedded systems are often designed for a long product lifecycle, providing stability and continuity for applications that require long-term support.
- **Simplified User Interfaces:** Embedded systems can have straightforward user interfaces tailored to the specific application, making them user-friendly and easy to operate.

Overall, the advantages of embedded systems stem from their specialized design, efficiency, reliability, and ability to provide tailor-made solutions for a wide array of applications, making them indispensable in modern technology and industry.

2.12 : Disadvantages

While embedded systems offer numerous advantages, they also come with certain disadvantages that should be considered during their design and implementation. Some of the disadvantages of embedded systems include:

- **Limited Resources:** Embedded systems often have constrained resources, such as limited processing power, memory, and storage. This limitation can make it challenging to run complex algorithms or handle large datasets.
- **Lack of Flexibility:** Due to their specialized nature, embedded systems may lack the flexibility to adapt to new requirements or functionalities easily. Making significant changes may require hardware modifications or complete redesigns.
- **Development Complexity:** Developing embedded systems can be more challenging and time-consuming than traditional software development due to the need for close integration with hardware and real-time constraints.
- **Debugging and Testing:** Identifying and fixing bugs in embedded systems can be more complex, especially when dealing with hardware-software interactions and real-time behavior.
- **Maintenance and Upgrades:** Embedded systems are often designed for long lifecycles, and updating them with the latest technology or features can be difficult and costly.
- **Security Vulnerabilities:** Embedded systems, especially those connected to networks (IoT devices), can be vulnerable to security breaches if not properly designed and secured.
- **Non-standard Interfaces:** Embedded systems may use proprietary or non-standard interfaces, making it challenging to interface with other systems or replace components.
- **Energy Efficiency Challenges:** While embedded systems are generally more power-efficient than general-purpose computers, optimizing them for power consumption can still be a complex task.
- **Time-to-Market Constraints:** The design and development process for embedded systems can be time-consuming, which may be a challenge in fast-paced industries with short time-to-market requirements.
- **Cost Considerations:** Customized embedded systems may have higher development and manufacturing costs, especially for low-volume applications.

- **Lack of Standardization:** There is often a lack of standardization in embedded systems, which can lead to compatibility issues between different systems and components.
- **Reliability Risks:** While embedded systems are known for their reliability, any hardware or software failure can have significant consequences, especially in critical applications like medical devices or automotive control systems.
- **Limited User Interface:** The compact size and resource constraints of embedded systems may result in limited or less user-friendly interfaces compared to general-purpose devices.

Despite these disadvantages, embedded systems continue to be widely used and preferred in various industries due to their specialized capabilities, efficiency, and ability to perform critical tasks in real-time. Careful design and planning can help mitigate some of these challenges and ensure the successful deployment of embedded systems.

CHAPTER3

8-bit PIC® Microcontrollers

8-bit PIC® microcontrollers are a family of microcontrollers developed by Microchip Technology Inc. PIC stands for "Peripheral Interface Controller," and these microcontrollers are widely used in various embedded systems and electronic applications. They are known for their simplicity, cost-effectiveness, and versatility

3.1 :PIC® MICROCONTROLLER OVERVIEW

3.1.1 : Embed with Confidence

- Why do our customers choose us?

It's because 8-bit PIC® microcontrollers are powerful, while remaining quick and easy to design into a wide variety of applications. High functionality and reduced risk during the development cycle gives our customers the confidence to focus their time and energy on innovating their end product.

Integration:

- Looking to optimize your project?

A broad product portfolio allows Microchip to offer engineers an appropriate integration of both analog and digital peripherals, ranging from simple digital to sophisticated analog modules. These integrated peripherals minimize component count and thereby lower total system cost while increasing reliability.

3.1.2 : Advantages

- Instructions and data are on separate buses, increasing speed and overall performance
- Diverse Flash memory offerings with industry leading endurance and retention
- Self-write Flash with remote programmability
- Data EEPROM option for frequent data changes on secure, non-volatile memory
- Wide range of packages and pin counts



Fig3.1: Pic Micro controller

3.2 : MIGRATION STRATEGY

3.2.1 : Scalability

As part of an inherent strategy to offer customers a low-risk development environment, the PIC microcontroller family offers easy migration within the complete range of products. Migration between the different PIC microcontrollers enables several advantages such as future cost reductions, feature enhancements and late development changes with minimal impact to the existing hardware, software and the engineering development environment. The 8-bit PIC microcontroller family is pin compatible within a given pin count as well as code compatible between the different architectures. This offers a seamless migration path between the different PIC microcontrollers that protects investments made in software development and design tools



Fig3.2: Scalability of PIC Micro controller

3.3 :8-BIT PIC® MICROCONTROLLER ARCHITECTURES

3.3.1 :Basic Architecture

Microchip's 8-bit PIC microcontrollers fall into three product architecture categories providing a variety of options for any application requirement. The specific families include: Baseline – PIC10F and some PIC12F and PIC16F; Mid-Range – PIC12F and PIC16F; and High Performance – PIC18F with J and K-Series. All device families have low power capability, flexible Flash program memory, and separate instruction and data buses to enable single cycle execution

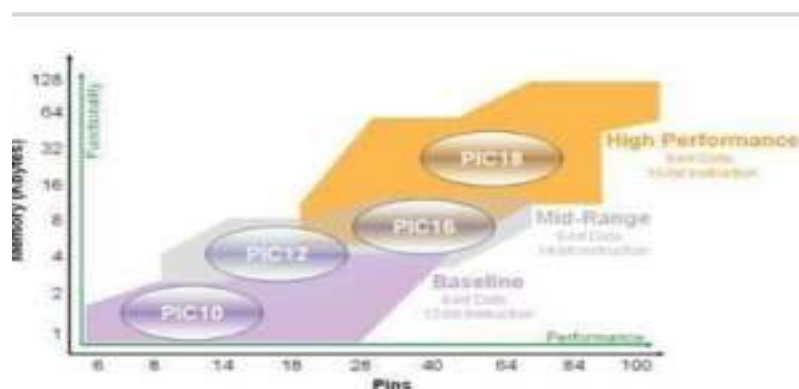


Fig 3.3 : Performance

3.3.2 :Explanation

- **BASELINE ARCHITECTURE**
 - **Smallest form factor**
 - **Most cost sensitive**
 - **6-pin to 40-pin**

Baseline devices utilize 12-bit program word architecture and enables the most cost- effective product solutions. Low operating voltages make this architecture ideal for battery operated applications. These MCUs allow engineers to use microcontrollers in applications that have historically been void of such devices. Whether it is cost or space constraints, Baseline PIC microcontrollers address these concerns by providing a pricing structure that makes them nearly disposable, with form factors that can easily be implemented into the most space constrained designs.



Fig 3.4:Base line Architecture



Fig 3.5 : Mid Range Architecture

- 5 MIPS operating performance
- Rich peripheral set
- Optimal cost-to-performance ratio



Fig 3.6: Basic Architecture

3.4 :LOW PIN COUNT AND SPACE-CONSTRAINED

3.4.1 :Features

A Complete Low Pin Count Solution Microchip offers the most complete family of low pin count Flash microcontrollers available. These devices range from easy-to-use 6-pin MCUs, which can replace discrete logic functions, to 20-pin MCUs with advanced analog and communications peripherals that can serve as the central processor in an application. Microchip continues to push the 8-bit MCU limits with the PIC10F family of 6-pin MCUs. This family offers 384-764 bytes of Flash memory and 16-24 bytes of data RAM memory with basic functionality and options for comparators, A/D converters and Flash data memory.

3.4.2 :Family

The PIC10F Family (PIC10F222/220/206/204/202/200). This family expands into territory not traditionally served by MCUs for any space-constrained application with its small form, performance and extremely low cost.

- “Electronic Glue” – Design in a PIC10F from the start to accommodate bug fixes and last-minute changes. Avoid costly, time-consuming silicon revisions or board changes.
- Logic Control – Optimize board space and cost with a PIC10F MCU for a more sophisticated solution that can take the place of passive discrete logic functions

Such as delays, smart gates, signal conditioning, simple state machines, encoders/decoders, etc.

- Intelligent Disposable Electronics – Given the small form factor and economical cost, it's ideal for emerging “disposable” applications such as drug or pregnancy testers, dialysis monitoring (blood sugar) and more.
- Waveform Generation – Replace traditional 555 timers, PWMs, remote control encoders, pulse generation, programmable frequency source, resistor programmable oscillators and more.
- Mechatronics – Traditional mechanical functions like smart switches, remote I/O s, LEDflashers and other forms of mechanical timers and switches can be replaced easily.

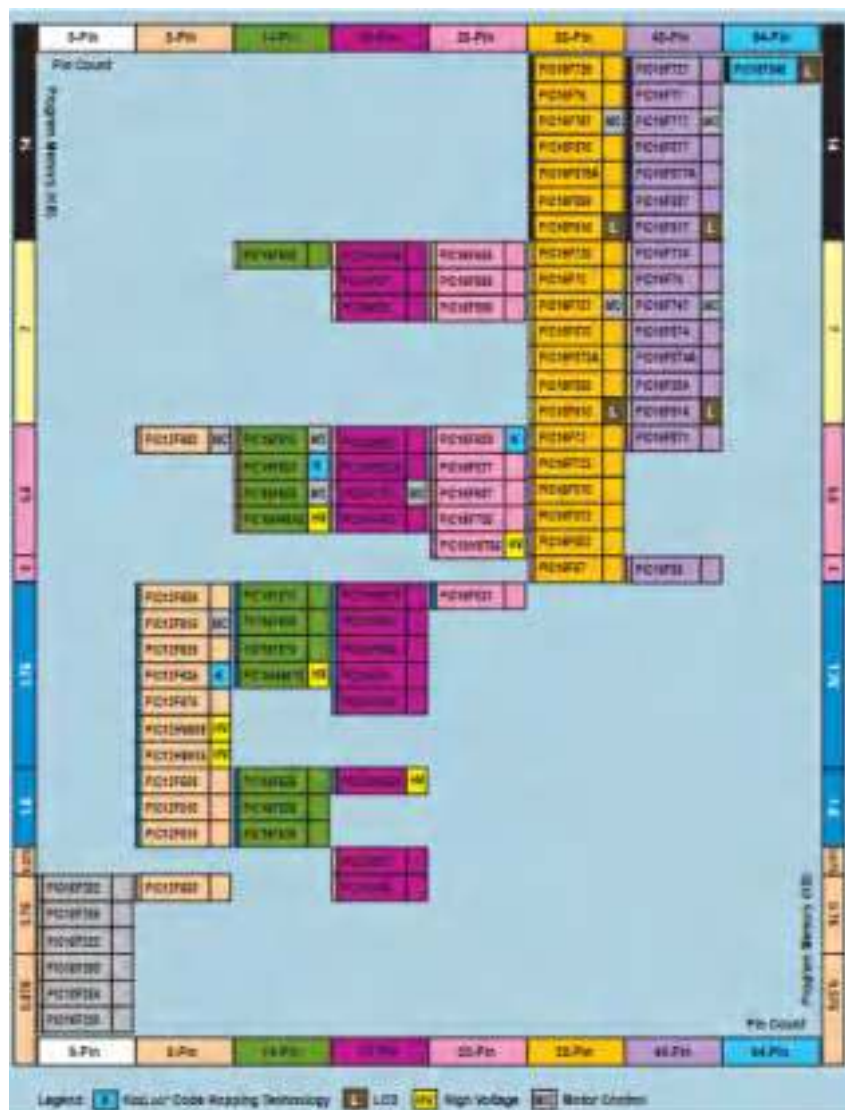


Fig 3.7: PIC10/PIC12/PIC16 FAMILIES

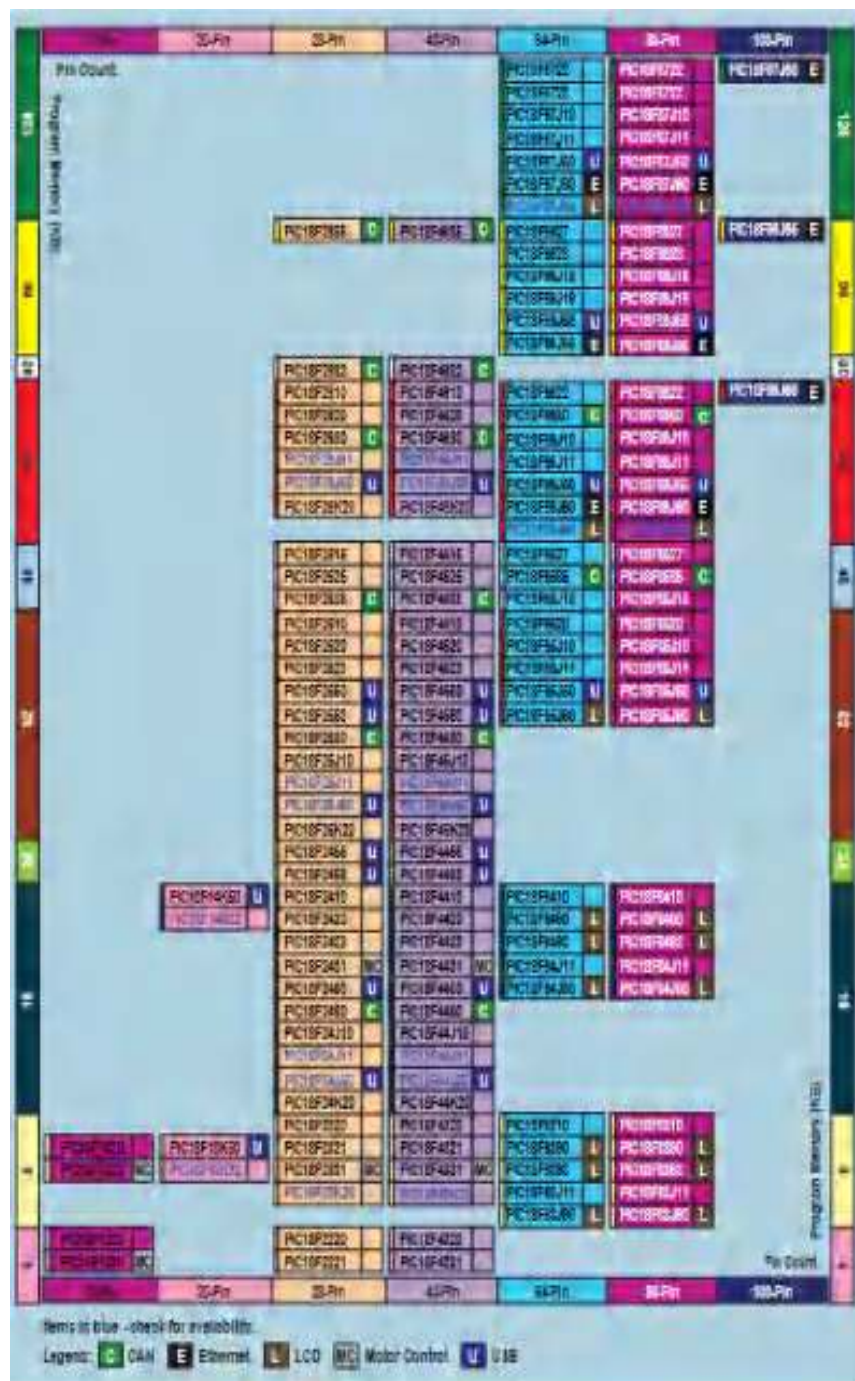


Fig 3.8: HIGH-PERFORMANCE PIC18 FAMILY

3.4.3 : Applications

PIC microcontrollers are used in a wide range of applications, such as consumer electronics, industrial automation, automotive systems, medical devices, smart home devices, and more.

APPLICATION EXAMPLES				
Consumer	Medical	Industrial	Automotive	Appliance
Set-top boxes Toys/Games Audio/Video Equipment Exercise Equipment	Mobile Telephones Diagnostic Equipment Biometrics	Clean Room Interfaces Motion Control Security/Alarms	Car Radio/GPS Occupant Sensors Climate Control Interface Windows/Locks/Doors	White Goods Thermostats Motion Control Proximity Sensing

Fig 3.9 : Applications

3.5 : MEDIUM PIN COUNT: 20, 28 and 40-PIN

3.5.1 :Features

Supervised 8-bit PIC® Microcontroller Solutions www.microchip.com/midrange Many embedded systems have evolved from being single-function “islands of control” to multi-dimensional entities that perform a number of tasks capable of communicating with other systems. Microchip’s latest 20-, 28- and 40-pin Mid-Range families aim to satisfy the demands of these complex systems

3.5.2 :Working

The terms "medium pin count" refer to the number of physical pins available on an integrated circuit package, such as microcontrollers. In this case, we are specifically referring to 8-bit PIC microcontrollers with medium pin count options. The medium pin count options for 8-bit PIC microcontrollers typically include 20, 28, and 40-pin packages. Each of these options offers different capabilities and features.

20-Pin Microcontrollers: 20-pin PIC microcontrollers are often used in applications where space is limited, and a relatively smaller number of I/O pins and peripherals are required. They are suitable for simpler projects and applications that don't demand extensive functionality.

28-Pin Microcontrollers: 28-pin PIC microcontrollers strike a balance between the compactness of 20-pin devices and the additional functionality provided by 40-pin devices.

They offer more I/O pins and peripherals than 20-pin devices, making them suitable for a broader range of applications.

40-Pin Microcontrollers: 40-pin PIC microcontrollers provide a larger number of I/O pins and more extensive peripheral integration compared to 20-pin and 28-pin devices. These microcontrollers are well-suited for more complex projects and applications that require multiple interfaces and functionalities.

The specific features and capabilities of 8-bit PIC microcontrollers in these medium pin count options can vary depending on the model and series. Some common features that can be found in these microcontrollers include timers, UARTs, SPI, I2C, analog-to-digital converters (ADCs), digital-to-analog converters (DACs), PWM channels, and more.

Designers often select a microcontroller with an appropriate pin count based on the requirements of their specific application. Smaller projects with limited requirements may opt for 20-pin microcontrollers, while more extensive projects with more peripherals and interfaces may choose 28-pin or 40-pin microcontrollers.

Microchip, the manufacturer of PIC microcontrollers, provides detailed datasheets and reference manuals for each of their microcontroller models, offering information on pin assignments, features, electrical characteristics, and more. This enables developers to make informed decisions when selecting the appropriate microcontroller for their projects.

3.5.2: Advantages & Disadvantages

Advantages of Medium Pin Count (20, 28, and 40-pin) PIC Microcontrollers:

Cost-Effectiveness: Medium pin count PIC microcontrollers strike a balance between low-cost 8-bit microcontrollers with limited pins and more expensive high-pin-count microcontrollers. They offer a cost-effective solution for a wide range of applications.

Space Efficiency: The medium pin count options provide a reasonable number of I/O pins and peripherals while still being space-efficient. This is beneficial for applications with limited board space or size constraints.

Versatility: Medium pin count microcontrollers are versatile and suitable for a wide range of applications, from simple projects with basic requirements to more complex designs with additional functionalities.

Peripheral Integration: Despite their smaller form factor, medium pin count PIC microcontrollers often feature a significant level of peripheral integration, including timers, communication interfaces, ADCs, PWM, and more. This reduces the need for external components, saving space and cost.

Ease of Use: The medium pin count options are often straightforward to work with due to their manageable number of pins and peripherals. They are suitable for hobbyists, students, and small-scale projects.

Low Power Consumption: 8-bit PIC microcontrollers, including those with medium pin counts, are generally designed with low power consumption in mind, making them suitable for battery-operated devices and energy-efficient applications.

Disadvantages of Medium Pin Count (20, 28, and 40-pin) PIC Microcontrollers:

Limited I/O Pins: Compared to higher pin count microcontrollers, medium pin count options have a limited number of I/O pins. This may be a limitation in more complex applications that require a large number of interfaced devices or peripherals.

Limited Processing Power: 8-bit microcontrollers, including those in the medium pin count range, have less processing power compared to higher-bit microcontrollers like 16-bit or 32-bit devices. This may restrict the computational capabilities of the microcontroller in certain applications.

Memory Constraints: Some medium pin count microcontrollers may have limited Flash memory and RAM, which can be a limitation for projects with extensive code or data storage requirements.

Less Peripheral Options: While medium pin count microcontrollers have a reasonable level of peripheral integration, they may not offer as many peripheral options as higher pin count microcontrollers. Developers may need to carefully select a microcontroller that matches the specific requirements of their project.

Scalability: In some cases, projects may outgrow the capabilities of medium pin count microcontrollers, necessitating the transition to higher pin count devices. This could require redesign and development efforts.

Less Specialization: Medium pin count microcontrollers may not be as specialized for certain application niches as more dedicated or application-specific microcontrollers with higher pin counts.

Despite these disadvantages, medium pin count PIC microcontrollers continue to be widely used in numerous applications due to their cost-effectiveness, space efficiency, and versatility. The selection of the appropriate microcontroller should be based on a careful evaluation of the specific

requirements of the project.

3.6 :PIC10/PIC12/PIC16 FAMILIES

3.6.1 :Diagram

PIC 8 bits Families				
	Base Line	Mid-Range	Enhanced Mid-Range	PIC18
No. of Pins	6-40	8-64	8-64	18-100
Program Memory	Up to 3 KB	Up to 14 KB	Up to 28 KB	Up to 128 KB
Data Memory	Up to 134 Bytes	Up to 368 Bytes	Up to 1.5 KB	Up to 4 KB
Instruction Length	12-bit	14-bit	14-bit	16-bit
No. of Instruction set	33	35	45	53
Speed	5 MIPS	5 MIPS	8 MIPS	Up to 16 MIPS
Features	<ul style="list-style-type: none"> • Comparator • 8-bit ADC • Data Memory • Internal Oscillator 	In addition of baseline <ul style="list-style-type: none"> • SPI • I²C • UART • PWM • 10-bit ADC • OP-Amps 	In addition of Mid-range <ul style="list-style-type: none"> • High Performance • Multiple communication peripherals 	In addition of Enhanced Mid-range <ul style="list-style-type: none"> • CAN • LIN • USB • Ethernet • 12-bit ADC
Families	PIC10, PIC12, PIC16	PIC12, PIC16	PIC12F1XXX, PIC16F1XXX	PIC18

Fig 3.10 : Families

3.6.2 :High –Performance Pic18 Family

Historically, the PIC18 family of microcontrollers has been known for its versatility, offering a wide range of options to suit various applications. Some members of the PIC18 family might be considered "high-performance" in the context of the 8-bit PIC microcontrollers.

Typically, the characteristics that contribute to higher performance in 8-bit PIC microcontrollers include:

Higher Clock Speed: Some PIC18 microcontrollers have higher clock speeds compared to their lower-end counterparts, enabling faster processing of instructions.

Larger Program Memory: High-performance PIC18 devices may come with larger program memory (Flash) capacity, allowing for more extensive and complex code storage.

Enhanced Peripherals: Certain PIC18 devices offer enhanced peripherals such as advanced PWM modules, multiple UARTs, more ADC channels, more timers, etc., which contribute to their high-performance capabilities.

Hardware Multiply and Divide Instructions: Some high-performance PIC18 devices may have hardware multiply and divide instructions, which speed up mathematical operations.

Extended Instruction Set: A few PIC18 microcontrollers have an extended instruction set that allows them to perform more complex operations efficiently.

DMA (Direct Memory Access): DMA capabilities in some high-performance PIC18 devices enable efficient data transfer between memory and peripherals without CPU intervention.

Low Latency Interrupt Handling: High-performance PIC18 microcontrollers often provide low-latency interrupt handling, which is critical for real-time applications.

Higher Precision ADCs: Some high-performance PIC18 devices feature higher resolution or precision ADCs for accurate analog measurements.

It's important to note that the term "high-performance" is relative to the 8-bit microcontroller category, and when compared to higher-bit architectures like 16-bit or 32-bit microcontrollers, the performance levels might be significantly lower. Therefore, the selection of a "high-performance" PIC18 microcontroller should be based on the specific requirements of the application and the desired performance characteristics. Always refer to the latest datasheets and official documentation from Microchip to choose the most suitable microcontroller for a particular project.

3.6.3 :High Performance of 8-bit Micro controllers

"High Performance of 8-bit Microcontrollers (40-pin)":

- Higher Clock Speed
- Larger Program Memory
- Enhanced Peripherals
- Hardware Multiply and Divide Instructions
- Extended Instruction Set
- Direct Memory Access (DMA)
- Low Latency Interrupt Handling
- Higher Precision ADCs
- Rich Ecosystem
- Wide Application Range

CHAPTER4

MPLAB® X IDE User's Guide

4.1 : Introduction

MPLAB® X IDE is a software program that is used to develop applications for Microchip microcontrollers and digital signal controllers. (Experienced embedded-systems designers may want to skip to the next chapter.)

- This development tool is called an Integrated Development Environment, or IDE, because it provides a single integrated “environment” to develop code for embedded microcontrollers.
- This chapter describes the development of an embedded system and briefly explains how MPLAB X IDE from Microchip is used in the process.

Topics discussed here include the following:

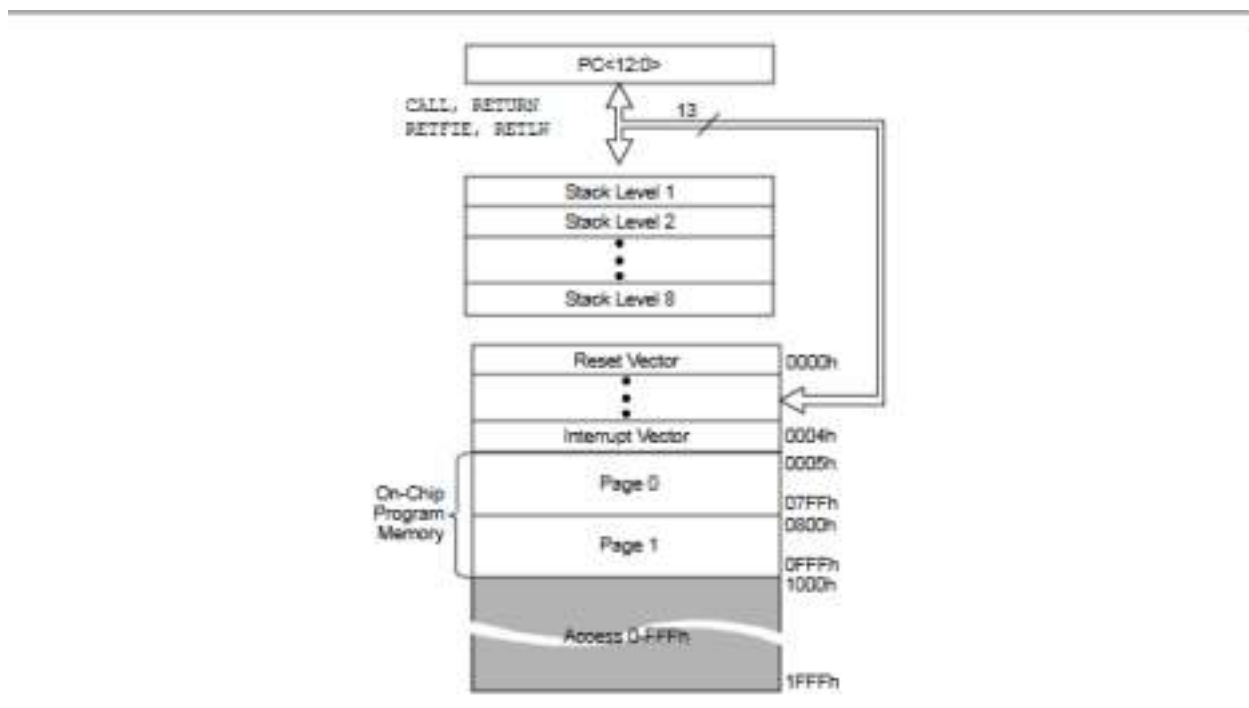


Fig 4.1: PIC® MCU DATA SHEET – PROGRAM MEMORY AND STACK

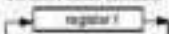
RRNCF	Rotate Right f (no carry)								
Syntax	[label] RRNCF f, [d], n								
Operands	0 ≤ f ≤ 255 d ∈ {0, 1} n ∈ {0, 1}								
Operation	(f ← f) → dest ← f >> n (f ← 0) → dest ← f >> n								
Status Affected	N, Z								
Encoding	<table><tr><td>0100</td><td>0000</td><td>1111</td><td>1111</td></tr></table>	0100	0000	1111	1111				
0100	0000	1111	1111						
Description	<p>The contents of register f are rotated one bit to the right. If n is 0, the result is placed in W. If n is 1, the result is placed back in register f (default). If n is 0, the Access Bank will be selected, overriding the BSR value. If n is 1, then the bank will be selected as per the BSR value (default).</p> <div></div>								
Words	1								
Cycles	1								
Q Cycle Activity	<table><tr><th>Q1</th><th>Q2</th><th>Q3</th><th>Q4</th></tr><tr><td>Decode</td><td>Read register f</td><td>Process Data</td><td>Write to destination</td></tr></table>	Q1	Q2	Q3	Q4	Decode	Read register f	Process Data	Write to destination
Q1	Q2	Q3	Q4						
Decode	Read register f	Process Data	Write to destination						
Example 1	RRNCF REG, 1, 0								
Before instruction	REG = 1111 1111								
After instruction	REG = 1111 1111								
Example 2	RRNCF REG, 0, 1								
Before instruction	W = 1111 1111								
After instruction	W = 1111 1111								

Fig 4.2: PIC® MCU DATA SHEET – INSTRUCTIONS (EXCERPT)

4.2 : Tutorial

This tutorial provides a guided example for working with an MPLAB X IDE project.

- Setting Up the Hardware and Software
 - Tutorial Equipment
 - Installation and Set Up
- Creating and Setting Up a Project
 - Create a New Project
 - View Changes to the Desktop – File and Navigation Panes
 - View or Make Changes to Project Properties
 - Set Up or Change Debugger or Programmer Options
 - Set Up or Change for Language Tool Options
 - Set Language Tool Locations
 - Add a New File to the Project
 - View Changes to the Desktop – Editor Pane
 - Configuration Bits
 - View Changes to the Desktop – Task Pane

- Running and Debugging Code
 - Build a Project
 - Run Code
 - Debug Run Code
 - Control Program Execution with Breakpoints
 - Step Through Code
 - Watch Symbol Values Change
 - View Device Memory (including Configuration Bits)
 - Program a Device

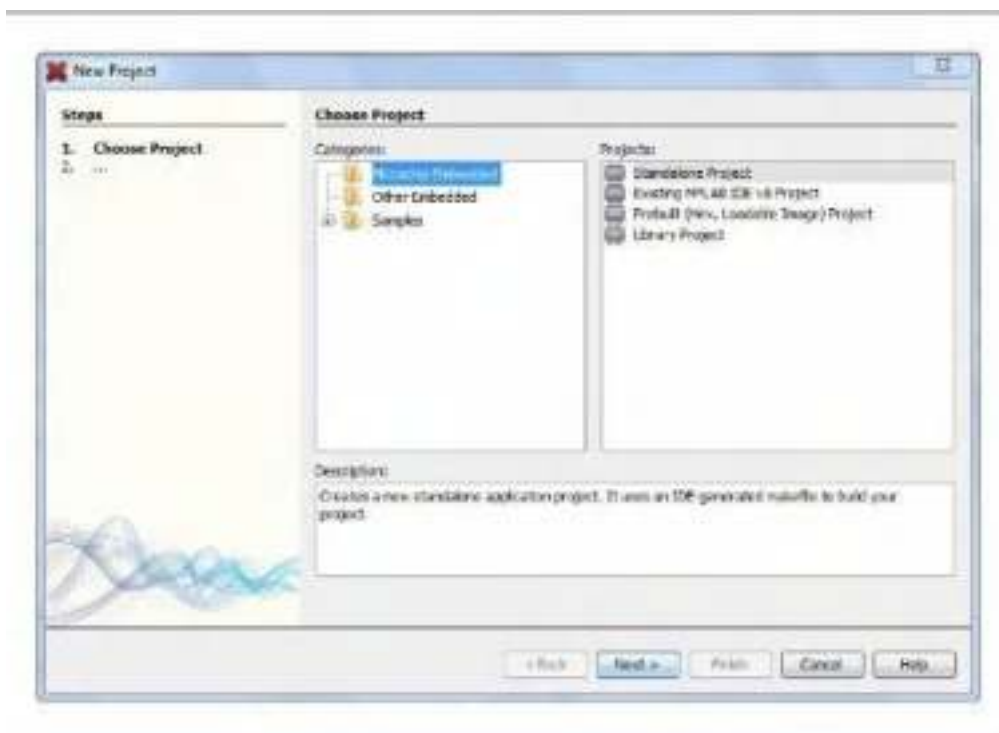


Fig 4.3 :PROJECT WIZARD – CHOOSE PROJECT



Fig 4.4: PROJECT WIZARD – SELECT DEVICE



Fig 4.4: MPLAB X IDE DESKTOP

4.3 : Basic Tasks

- **Preliminaries**

The following steps show how to work with projects in MPLAB X IDE

1. Before You Begin, install MPLAB X IDE, set up any hardware tools (install USB drivers and properly connect to your target) and install language tools for your selected device. Then launch MPLAB X IDE to begin

- **Create and Build a Project**

1. The New Project wizard. Then View Changes .In Desktop Panes.
2. View or Make Changes to Project Properties in the Project Properties dialog. Also Set Up or Change Debugger/Programmer Tool Options and Set Up or Change Language Tool Options in the same dialog.
3. Set Language Tool Locations and Set Other Tool Options in the Tools Options dialog.
4. Create a New File to add to your project or Add Existing Files to a Project. Enter or edit your application code to the File window.
5. Discover other features for Editor Usage.
6. Add and Set Up Library and Object Files.
7. Set File and Folder Properties to keep or exclude individual files or entire folders from the build.
8. Set Build Properties for your project. Here you may select the project configuration type, pre- and post-build steps, using the checksum as the user ID, and loading an alternative hex file on build.
9. Build a project

- **Execute Code**

- **Debug Code**

- **Program a Device**

4.4 :Editor & Trouble Shooting

MPLAB X IDE is built upon the Net Beans platform, which provides a built-in editor to create new code or change existing code.

C compiler information regarding the editor is available under the Net Beans help topic C/C++/Fortran Development>Working with C/C++/Fortran Projects>Editing Source Files in C/C++/Fortran Projects>About Editing C and C++ Files. MPLAB X IDE Editor

features are discussed below.

- Source Editor General Tasks: Quick Reference
- Editor Usage
- Editor Options
- Editor Red Bangs
- Code Folding

To perform this task	Follow these steps
Open a file that is not available in the Projects window or the Files window.	<ol style="list-style-type: none"> 1. Choose File>Open File. 2. In the file chooser, navigate to the file. Then click Open.
Maximize the Source Editor.	Do one of the following: <ul style="list-style-type: none"> • Double-click a file's tab in the Source Editor. • Make sure that the Source Editor window has focus and then press Shift-Escape. • Choose Window>Configure Window>Maximize.
Revert a maximized Source Editor to its previous size.	Do one of the following: <ul style="list-style-type: none"> • Double-click a file's tab in the Source Editor. • Press Shift-Escape. • Choose Window>Configure Window>Restore.
Display line numbers.	Choose View>Show Line Numbers .
View two files simultaneously.	<ol style="list-style-type: none"> 1. Open two or more files. 2. Click the tab of one of the files and drag it to the side of the window where you want the file to be placed. When a red preview box appears to show you where the window will be placed, release the mouse button to drop the window. <p>The window can be split horizontally or vertically, depending on where you drag the tab.</p>
Split the view of a single file.	<ol style="list-style-type: none"> 1. Right-click the document's tab in the Source Editor and choose Clone Document. 2. Click the tab of the cloned document and drag it to the part of the window where you want the copy to be placed.
Format code automatically.	Right-click in the Source Editor and choose Format. If any text is selected, only that text will be formatted. If no text is selected, then the whole file is formatted.

Fig 4.5 :EDITOR QUICK REFERENCE

4.4.1 :Troubleshooting

This section is designed to help you troubleshoot any problems, errors or issues you encounter while using MPLAB X IDE. If none of this information helps you, see “Support” for ways to contact Microchip Technology.

- USB Driver Installation Issues
- Cross-Platform (Operating System) Issues
- Windows Operation System Issues
- NetBeans Platform Issues
- MPLAB X IDE Issues

- Errors
- Forums

4.5 : MPLAB X IDE differs considerably from MPLAB IDE v8

Because MPLAB X IDE is based on the Net Beans platform, many features will be different from MPLAB IDE v8. Please see the following for more detailed information:

- Chapter 2. “Before You Begin”
- Chapter 3. “Tutorial”
- Chapter 5. “Additional Tasks”

A short list of major feature differences is presented below:

1. MPLAB X IDE is based on the open-source, cross-platform Net Beans platform. Third parties can easily add functionality as plug-ins. MPLAB X IDE components that are specific to Microchip products are still proprietary. MPLAB IDE v8 is proprietary and Windows operating system based. Third parties can add to v8 with design information from the MPLAB development group.

2. MPLAB X IDE is project-based (no workspaces). In MPLAB X IDE, you must create a project to develop your application. Creating a project involves selecting a device, as well as selecting and setting up language tools, debug tools, programming tools and other project specifics. This ensures all items needed for successfully developing an application are present. Multiple project grouping is handled by multiple configurations.

MPLAB IDE v8 is device-based. Although it is always highly recommended that you use a project in v8 to create your application, it is not required. Workspaces are used to contain some set up information, including multi-project grouping. As projects have changed significantly, reading “Tutorial” is recommended.

3. MPLAB X IDE allows multiple tool selection Example 1: Connect several MPLAB ICD 3 debuggers into several computer USB ports. Then access the Project properties to easily switch between the debuggers, which are identified by their serial numbers (SN).

Example 2: Connect one MPLAB ICD 3 debugger into a computer USB port and one MPLAB PM3 programmer into another USB port. Then access the Project properties to easily switch between the tools. MPLAB IDE v8 does not allow multiple tool selection.

CHAPTER5

Syntax and Structures of C

5.1 : Introduction, Comments ,Variables

5.1.1 : Introduction

C is a powerful and widely used programming language known for its efficiency, portability, and versatility. Developed in the early 1970s by Dennis Ritchie at Bell Labs, C has since become one of the most popular programming languages in the world. It serves as the foundation for many other programming languages and has been instrumental in the development of operating systems, embedded systems, software applications, and more.

5.1.2 : Comments

Comments in C are used to add explanatory notes or remarks within the source code. These comments are ignored by the compiler during the compilation process and do not affect the behavior of the program. They serve as documentation for programmers, making the code easier to understand and maintain. C supports two types of comments:

Single-line comments: These comments start with two forward slashes“//” and continue until the end of the line. Anything after „//“ on the same line is considered a comment.

EX:-// This is a single-line comment in C

```
int x = 10; // Another comment after a statement
```

Multi-line comments: Multi-line comments, also known as block comments, start with /* and end with */. These comments can span multiple lines.

EX:-/* This is a

multi-line comment

in C */

```
int y = 20;
```

5.1.3 : Variables

In C, variables are used to store and manipulate data. They represent memory locations where values are stored. Before using a variable, it must be declared, specifying its data type and optionally an initial value. C supports various data types, including integers, floating-point

numbers, characters, and user-defined types. Here are some key points about variables in C:

Declaration: Variables are declared by specifying the data type followed by the variable name.

```
Ex:-int age;    // Integer variable
float pi = 3.14; // Floating-point variable with an initial value
char initial;  // Character variable
```

Initialization: Variables can be initialized with an initial value at the time of declaration.

```
Ex:-int count = 0;    // Integer variable initialized to 0
char grade = 'A';    // Character variable initialized to 'A'
```

Scope: The scope of a variable determines where it can be accessed within the code. Variables can have global scope (accessible throughout the entire program) or local scope (accessible only within a specific block or function).

Lifetime: The lifetime of a variable is the duration for which it exists in memory. Local variables have a limited lifetime, while global variables exist throughout the program's execution.

Constants: Constants are special variables whose values cannot be changed after initialization. In C, constants are typically defined using the „const“ keyword.

```
EX:-constant MAX_SIZE = 100;
```

Data Types: C supports various data types, including int, float, char, double, and user-defined types using struct.

```
Ex:-struct Person {
char name[20];
int age;
};
```

Type Modifiers: C allows the use of type modifiers like unsigned, signed, short, and long to modify the data type's range and representation.

```
Ex:-unsigned intpositiveNum = 100;
longlonglargeNum = 1234567890LL;
```

5.2 : Loops ,Functions , Arrays, Pointers

5.2.1 : Loops in C:

- **for loop:** A loop that repeatedly executes a block of code for a fixed number of times, based on a loop control variable.
- **while loop:** A loop that repeatedly executes a block of code as long as a specified condition is true.
- **do-while loop:** A loop that executes a block of code first and then checks the condition. It ensures that the block is executed at least once.

5.2.2 : Functions in C:

Functions in C are blocks of code that perform a specific task and can be called from other parts of the program.

- Functions have a return type, a name, and optional parameters (arguments).
- They are defined using a function prototype and implemented using a function body.

5.2.3 : Arrays in C:

Arrays in C are collections of elements of the same data type, stored in contiguous memory locations.

- Arrays are declared with a specific size, and individual elements are accessed using an index.
- The first element of an array has an index of 0, the second element has an index of 1, and so on.

5.2.4 : Pointers in C:

Pointers are variables that store memory addresses.

They allow direct access to memory locations, enabling more efficient memory manipulation.

Pointers are extensively used for dynamic memory allocation and passing data by reference to functions.

➤ Pointer Arithmetic:

Pointer arithmetic allows moving the pointer to the next or previous memory location based on the data type size.

`ptr++`: Moves the pointer to the next element (increases the address).

`ptr--`: Moves the pointer to the previous element (decreases the address).

➤ Null Pointer:

A null pointer is a pointer that does not point to any memory location. It is represented by the value `NULL`. It is used to indicate that the pointer does not currently point to valid data.

➤ **Void Pointer (Generic Pointer):**

A void pointer is a special type of pointer that can hold the address of any data type.

It is used in scenarios where the data type of the pointed-to data is not known or can change dynamically.

➤ **Pointer to Functions:**

Pointers can also store the address of functions.

Function pointers allow calling different functions based on the pointer's value, enabling runtime function selection.

➤ **Pointer and Arrays Relationship:**

Arrays and pointers are closely related in C.

The name of an array can be used as a pointer to the first element of the array.

EX:

```
intarr[5] = {1, 2, 3, 4, 5};
```

```
int *ptr = arr; // ptr points to the first element of the array (arr[0])
```

Loops, functions, arrays, and pointers are fundamental concepts in C programming. They provide powerful tools for controlling program flow, organizing data, and efficiently manipulating memory, enabling developers to create complex and efficient programs.

5.3: Data Structures & Sate Machines

5.3.1 : Data Structures in C:

Data structures in C allow programmers to organize and store data in a way that facilitates efficient operations and data manipulation. Some commonly used data structures in C include:

Arrays: A collection of elements of the same data type, accessed using an index. Arrays provide a simple way to store and retrieve data.

Structures: Structures allow grouping multiple variables of different data types into a single unit. They enable creating custom data types to represent complex entities.

EX:-struct Person {

```
char name[50];
```

```
int age;
```

```
float height;
```

```
};
```

Linked Lists: A dynamic data structure where elements (nodes) are connected through pointers. Linked lists can be easily resized and are useful for dynamic data management.

Stacks: A data structure that follows the Last-In-First-Out (LIFO) principle. Elements are inserted and removed from the same end (top) of the stack.

Queues: A data structure that follows the First-In-First-Out (FIFO) principle. Elements are inserted at the rear and removed from the front of the queue.

Trees: Hierarchical data structures with a root node and child nodes. Trees are used in various applications like binary search trees and decision trees.

Graphs: A collection of nodes (vertices) connected by edges. Graphs are used to model complex relationships between data elements.

Data structures are essential for efficient data organization and manipulation in various algorithms and programs.

5.3.2 : State Machines in C:

A state machine is a model that represents the behavior of a system or application as a set of states and transitions between these states. It is particularly useful for designing systems with complex behavior or sequential logic. In C, state machines can be implemented using conditional statements, loops, and switches to represent the different states and transitions.

Finite State Machine (FSM):

A finite state machine has a finite number of states and can transition from one state to another based on inputs or events. It can be implemented using a switch-case or if-else statements to handle different states and their transitions.

```
EX:-typedef enum {  
    STATE_IDLE,  
    STATE_RUNNING,  
    STATE_PAUSED,  
    STATE_STOPPED  
} State;  
  
State currentState = STATE_IDLE;  
  
while (true) {  
    switch (currentState) {  
    case STATE_IDLE:  
        // Handle behavior for idle state  
        break;
```

```
case STATE_RUNNING:
    // Handle behavior for running state
break;
case STATE_PAUSED:
    // Handle behavior for paused state
break;
case STATE_STOPPED:
    // Handle behavior for stopped state
break;
}

// Check for state transitions and update currentState
}
```

State machines are particularly useful in applications like embedded systems, protocol parsing, and user interfaces, where the system's behavior depends on different events and conditions. In conclusion, data structures and state machines are crucial concepts in C programming for efficient data organization and modeling complex behaviors. They provide the tools and techniques to design robust and organized algorithms and systems.

5. 4: Callbacks

- In C, callbacks are a powerful programming technique that involves passing a function as an argument to another function. The receiving function can then call the passed function (callback function) at the appropriate time or in response to specific events. Callbacks enable a form of dynamic behavior, allowing the behavior of a program to be customized or extended without modifying the original function's code.

Callback Function Syntax:

A callback function is a regular C function with a specific signature (return type and parameter list) that can be used as an argument to another function.

EX:-

```
return_type (*callback_function_name)(parameters);
```




Fig 5.1: Callbacks in C

5.5 :Advanced Embedded C Tips, Tricks, and Cautions

Advanced Embedded C programming requires careful consideration of performance, memory usage, and real-time constraints. Here are some tips, tricks, and cautions to help you write efficient and reliable embedded C code:

5.1 : Tips and Tricks:

Optimize Memory Usage: In embedded systems, memory is often limited. Minimize the use of global variables and dynamic memory allocation. Use bitfields and packed structures to optimize memory usage.

Avoid Floating-Point Operations: Floating-point operations are resource-intensive. Whenever possible, use fixed-point arithmetic or lookup tables to perform calculations.

Use Volatile Qualifier: Use the volatile qualifier for variables accessed in an interrupt service routine (ISR) or shared between the main program and ISRs to ensure proper handling of shared memory.

Use Compiler Optimization: Enable compiler optimizations to produce more efficient code. But be cautious; aggressive optimizations may cause unexpected behavior in certain situations.

Choose the Right Data Types: Select appropriate data types based on the range and precision required for variables. Use `uintX_t` and `intX_t` types for fixed-width integers.

Watch for Endianness: Be aware of the target system's endianness when dealing with multi-byte data and external interfaces.

Avoid Function Calls in Critical Sections: Minimize the use of function calls in critical sections or ISRs to reduce the risk of stack overflow and maintain real-time responsiveness.

Use Inline Functions: For small, frequently used functions, use inline functions to reduce function call overhead.

Use Hardware Peripheral Libraries: Leverage manufacturer-provided hardware peripheral libraries to access hardware features efficiently.

Optimize Loops: Optimize loops by reducing loop iterations, unrolling loops, and minimizing loop overhead.

5.5.2: Cautions:

Avoid Blocking Operations: Avoid long blocking operations that may cause the system to miss critical events or introduce unacceptable delays.

Beware of Race Conditions: Be cautious when accessing shared resources between ISRs and the main program to avoid race conditions and data corruption.

Stack Overflow: Keep track of stack usage, especially in recursive or deeply nested functions, to avoid stack overflow.

Interrupt Priority: Be aware of interrupt priorities, and avoid long-running ISRs that may delay high-priority interrupts.

Watch for Undefined Behavior: Be careful to avoid undefined behavior in C, such as accessing uninitialized variables or dereferencing NULL pointers.

Check Compiler Compatibility: Ensure that the C code is compatible with the specific compiler and version used for the target platform.

Beware of Side Effects: Avoid complex expressions with side effects that may lead to unexpected behavior.

Memory Alignment: Ensure that data structures are correctly aligned, especially when dealing with DMA or hardware interfaces that require aligned memory access.

Watch for Endian Conversion: When dealing with external interfaces or communication protocols, take care of endian conversion as needed.

Code Reviews and Testing: Conduct thorough code reviews and extensive testing to catch potential bugs and issues before deployment.

Embedded C programming requires meticulous attention to detail, and adhering to these tips, tricks, and cautions will help you develop efficient, reliable, and maintainable code for embedded systems. Always refer to the target hardware's datasheets and reference manuals and follow best practices to ensure the success of your embedded projects.

CHAPTER6

Internet of Things (IoT)

6.1 : Introduction

The Internet of Things (IoT) is a transformative technology that connects everyday physical objects and devices to the internet, enabling them to collect, exchange, and act on data. These objects, often referred to as "things," can range from simple sensors and actuators to complex machines and appliances. At the heart of the IoT revolution lies embedded systems, which form the foundation of this interconnected ecosystem.

Embedded systems are specialized computing devices designed to perform specific tasks or functions. They are often integrated into larger systems to provide intelligence and control. These systems consist of hardware components, such as microcontrollers or microprocessors, and software that runs on them. Embedded systems are widely used in various applications, including consumer electronics, automotive, healthcare, industrial automation, and more.

The convergence of IoT and embedded systems has given rise to a plethora of innovative applications and possibilities. Here's how IoT and embedded systems work together:

- **Sensing and Data Collection:** Embedded systems are equipped with sensors that can detect physical parameters such as temperature, humidity, pressure, motion, and more. These sensors continuously collect data from the surrounding environment and the devices they are integrated into.
- **Connectivity:** IoT enables embedded systems to communicate and share data over the internet or local networks. This connectivity can be achieved through various wireless technologies like Wi-Fi, Bluetooth, Zigbee, cellular networks, or wired connections like Ethernet.
- **Data Processing and Analysis:** Embedded systems have limited computational capabilities compared to traditional computers. However, they can process and analyze data locally to derive meaningful insights and make real-time decisions. In some cases, they may offload more complex processing to cloud servers.

- **Remote Monitoring and Control:** The integration of IoT and embedded systems allows remote monitoring and control of devices and processes. Users can access data, receive alerts, and remotely manage connected devices through smartphones, tablets, or computers.
- **Automation and Smart Actions:** IoT-enabled embedded systems can be programmed to respond intelligently to certain conditions or triggers. For example, a smart thermostat can adjust the temperature based on user preferences or ambient conditions.
- **Security and Privacy:** With the increased interconnectivity, security becomes a crucial aspect of IoT with embedded systems. Measures must be implemented to protect data and devices from unauthorized access and malicious attacks.
- **Scalability and Interoperability:** As the number of connected devices grows, the IoT infrastructure must be scalable and capable of integrating with diverse devices from various manufacturers. Standards and protocols play a vital role in ensuring interoperability.

IoT with embedded systems is driving significant advancements in various sectors, including smart homes, smart cities, agriculture, healthcare, transportation, and industrial automation. However, it also presents new challenges, such as ensuring data privacy, managing device updates, and addressing potential security vulnerabilities. As technology continues to evolve, the combination of IoT and embedded systems is expected to revolutionize how we interact with the physical world and transform industries for the better.

6.2 : Network Architecture

The network architecture of the Internet of Things (IoT) is a crucial aspect that defines how IoT devices and systems communicate, process data, and interact with each other. The architecture must be designed to handle the unique challenges posed by the massive scale, diverse devices, and varied applications of IoT. There are several layers in the IoT network architecture, each playing a specific role in the overall functionality. Here is a simplified overview of the typical IoT network architecture:

Perception Layer (Sensing Layer):

The bottom layer of the IoT architecture is the perception layer, where data is collected from various sensors and devices. These sensors can detect physical parameters such as temperature,

humidity, motion, light, pressure, etc.

The devices in this layer could be standalone sensors or **embedded systems equipped with** sensors. The collected data is raw and unprocessed at this stage.

Network Layer (Connectivity Layer):

The network layer is responsible for connecting the IoT devices and sensors to the internet or a local network. Various communication protocols are used in this layer, such as Wi-Fi, Bluetooth, Zigbee, Z-Wave, LoRaWAN, cellular networks (3G, 4G, 5G), Ethernet, etc.

The choice of communication protocol depends on factors like data transfer rate, range, power consumption, and the specific requirements of the IoT application.

Processing Layer (Middleware Layer):

The processing layer sits between the perception layer and the application layer. It performs data processing, filtering, and aggregation to extract valuable information from the raw sensor data.

This layer may include edge computing, where data processing occurs locally on the edge devices (e.g., gateways or edge servers) to reduce latency and bandwidth usage.

It also handles device management, security, and data encryption.

Application Layer:

The application layer is where the processed data is used to derive insights and make decisions.

This layer consists of various applications and services that provide specific functionalities for end-users or other systems.

Applications can range from simple user interfaces on mobile devices to sophisticated cloud-based services that analyze and visualize data for business intelligence.

The applications can be consumer-focused (e.g., smart home applications) or enterprise-focused (e.g., industrial automation, predictive maintenance, asset tracking).

Business Layer (Enterprise Layer):

The topmost layer is the business layer, which deals with business-specific processes, analytics, and decision-making.

In enterprise IoT deployments, this layer may integrate with existing enterprise systems (like ERP or CRM) to utilize IoT data for improving operational efficiency and business outcomes.

Cloud and Data Center:

While not always a distinct layer, cloud computing and data centers play a crucial role in IoT architectures. They provide the scalability and computational resources needed to handle large volumes of data, perform complex analytics, and store data for long periods.

It's essential to note that IoT network architectures can vary significantly based on the specific use case, scale, and deployment requirements. Moreover, edge computing has become increasingly important in IoT to address the challenges of latency, bandwidth, and data privacy. The distributed nature of IoT architectures, combining edge and cloud resources, enables efficient data processing and decision-making closer to the data sources, leading to more responsive and resilient IoT systems.

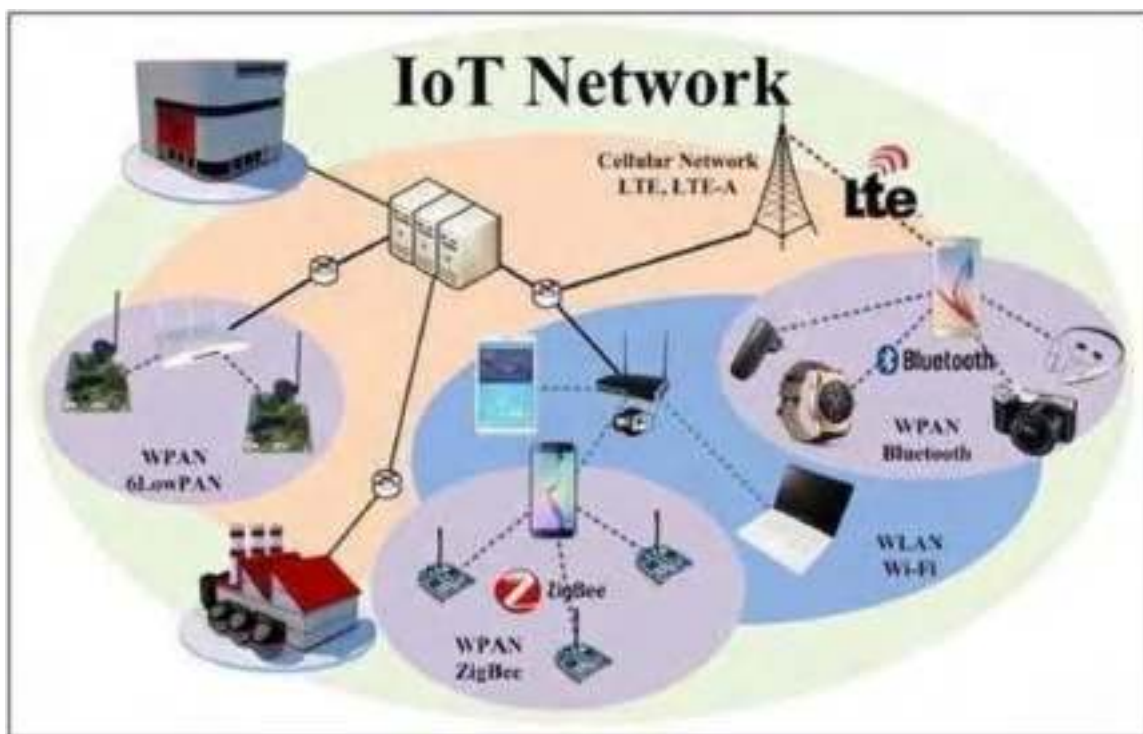


Fig 6.1: Network Architecture

6.3 :Messaging Protocols

Messaging protocols play a crucial role in enabling communication between devices and systems in various networked environments, including the Internet of Things (IoT). These protocols define the rules and conventions for exchanging data and messages, ensuring that information is transmitted reliably and efficiently. Here are some commonly used messaging protocols:

MQTT (Message Queuing Telemetry Transport):

MQTT is a lightweight and efficient messaging protocol designed for constrained environments, making it well-suited for IoT devices with limited resources (e.g., low-power microcontrollers). It follows a publish-subscribe model, where clients publish messages to a topic, and other clients subscribe to those topics to receive the messages they are interested in.

MQTT is known for its low overhead, making it ideal for scenarios with intermittent or low-bandwidth connections.

CoAP (Constrained Application Protocol):

CoAP is another lightweight protocol designed for IoT and constrained environments.

It is built on top of the User Datagram Protocol (UDP) and enables efficient communication between resource-constrained devices.

CoAP supports request-response interactions and provides features like resource discovery, caching, and observing (allowing clients to receive notifications of resource changes).

AMQP (Advanced Message Queuing Protocol):

AMQP is a robust and feature-rich messaging protocol that provides reliable message queuing and delivery.

It supports both point-to-point and publish-subscribe communication patterns.

AMQP is commonly used in enterprise messaging systems and IoT applications that require guaranteed delivery and advanced message routing capabilities.

HTTP (Hypertext Transfer Protocol):

While not specifically designed for IoT, HTTP is widely used in IoT applications, particularly for devices with more computing power and memory.

HTTP is the foundation of the World Wide Web and uses a request-response model, making it suitable for accessing RESTful APIs and web services.

It is commonly used for device management, data retrieval, and control in IoT applications.

DDS (Data Distribution Service):

DDS is a communication protocol designed for real-time, high-performance, and scalable data distribution.

It is commonly used in industrial IoT and mission-critical applications where timely and reliable data exchange is essential.

DDS supports a publish-subscribe communication model and provides features like Quality of Service (QoS) settings for fine-grained control over message delivery.

WebSockets:

WebSockets are a protocol that enables full-duplex, bidirectional communication between a client and a server over a single TCP connection.

WebSockets are commonly used in web applications and IoT scenarios where real-time data exchange and continuous communication are required.

These are just a few examples of messaging protocols commonly used in the context of IoT. The choice of protocol depends on factors such as device capabilities, network conditions, reliability requirements, and the specific use case of the IoT application. It's also worth noting that some IoT platforms and ecosystems may use proprietary messaging protocols or adapt existing ones to better suit their requirements.

6.4: Application Development

Application development in IoT involves creating software solutions that leverage the capabilities of interconnected devices to solve specific problems or provide valuable services. These applications can range from simple data monitoring and control systems to sophisticated solutions using advanced analytics and machine learning. Here's a general overview of the steps involved in IoT application development:

Identify Use Case and Requirements:

Start by identifying the use case or problem you want to address with your IoT application.

Understand the requirements, user needs, and the scope of the project.

Define the types of devices to be connected, the data to be collected, and the actions the application should perform.

Choose the Right Hardware:

Select appropriate IoT hardware devices (sensors, actuators, microcontrollers, etc.) that fit the use case and fulfill the requirements. Consider factors like connectivity options, power consumption, processing capabilities, and form factor.

Select Communication Protocol:

Choose the messaging protocol that best suits your IoT application. Consider factors like data volume, latency requirements, and device constraints. MQTT, CoAP, HTTP, or other protocols may be selected based on the specific use case.

Data Collection and Connectivity:

Set up the communication infrastructure to connect the IoT devices to the cloud or a local gateway. Implement data collection mechanisms to retrieve information from the connected devices and sensors.

Cloud Platform Integration:

Choose an IoT cloud platform that meets your application's requirements for data storage, processing, and management. Integrate your IoT application with the cloud platform, enabling seamless data transfer and synchronization.

Data Processing and Analysis:

Process the collected data to extract meaningful insights and perform necessary analytics. Apply algorithms or machine learning models to derive valuable information from the data.

User Interface Development:

Design and develop a user interface (UI) for your IoT application. The UI may include web-based dashboards, mobile apps, or other interfaces for users to interact with the IoT system.

Security Implementation:

Implement security measures to protect data and devices from unauthorized access, data breaches, and other cyber security threats. Use encryption, authentication, and access control mechanisms to secure data and communications.

Device Management:

Implement device management functionalities to monitor and control IoT devices remotely. Over-the-air (OTA) updates and configuration management are essential for maintaining and managing IoT devices efficiently.

Testing and Deployment:

Thoroughly test your IoT application to ensure its reliability, scalability, and performance under various conditions. Deploy the application on the chosen platform or infrastructure, making it accessible to end-users.

Continuous Monitoring and Improvement:

Monitor the performance of your IoT application in real-world scenarios and gather user feedback. Continuously improve the application by iterating on the design, adding new features, and addressing any issues that arise.

IoT application development requires a multidisciplinary approach, involving skills in hardware engineering, software development, data analytics, and user experience design. Collaboration between domain experts, developers, and data scientists is essential to create successful IoT applications that deliver tangible value to users and organizations.

6.5 : Conclusion

In conclusion, the Internet of Things (IoT) is a transformative technology that has revolutionized the way we interact with the physical world. It connects everyday objects and devices to the internet, enabling them to collect, share, and act on data. IoT applications have permeated various industries, offering innovative solutions to real-world challenges and enhancing efficiency, productivity, and convenience.

Embedded systems form the backbone of IoT, providing the intelligence and control necessary to enable seamless communication between devices. These systems, often equipped with sensors and actuators, collect data from the environment and interact with other devices through various communication protocols.

To facilitate communication between devices and systems, messaging protocols play a crucial role in ensuring reliable and efficient data exchange. MQTT, CoAP, AMQP, HTTP, DDS, and WebSockets are some of the commonly used protocols, each tailored to specific use cases and device constraints.

In IoT application development, a systematic approach is essential. Identifying the use case, selecting the right hardware and communication protocols, integrating with cloud platforms, implementing data processing, and ensuring security are key steps in the development process. Additionally, user interface development, device management, testing, deployment, and continuous improvement are critical aspects to deliver successful IoT applications that provide tangible benefits to users.

As the IoT landscape continues to evolve, we can expect even more advanced applications and solutions to emerge, addressing complex challenges and improving the way we live and work. However, with the immense potential of IoT comes the responsibility to prioritize security and privacy, ensuring that devices and systems are safeguarded against potential threats and vulnerabilities.

In conclusion, IoT with embedded systems and effective messaging protocols has opened up a world of possibilities, and the continued advancement of this technology holds the promise of a more connected, efficient, and intelligent future. By leveraging the power of IoT responsibly and ethically, we can create a world where devices seamlessly collaborate to enhance our lives, industries, and society as a whole.

CHAPTER7

Exploring Bluetooth Low Energy from First Steps to Final App

7.1 : Project-1: Cable Replacement

Project Title: Cable Replacement Project in Embedded Systems

Project Overview:

The Cable Replacement project aims to replace traditional wired connections with wireless solutions using embedded systems. The goal is to enhance flexibility, mobility, and reduce the maintenance and installation costs associated with cabling in various applications. This project focuses on implementing a reliable and secure wireless communication system to replace cables in a specific use case.

Project Objectives:

- Design and develop an embedded system for wireless communication.
- Achieve reliable data transmission with low latency and minimal packet loss.
- Implement security measures to protect data during wireless transmission.
- Optimize power consumption for efficient battery operation.
- Perform rigorous testing to ensure system performance and reliability.

Use Case Scenario:

For instance, consider an industrial automation scenario where sensors, actuators, and control devices are currently connected through long cables. Implementing a wireless solution would make the system more flexible, allowing seamless reconfiguration and easy scalability. It would also reduce the risk of cable damage and save time and effort in cable management.

System Architecture:

- Select appropriate wireless communication protocol (e.g., Bluetooth, Zigbee, or Wi-Fi) based on the specific use case requirements.
- Design the embedded system to include microcontrollers, wireless transceivers, power management units, and any necessary sensors/actuators.
- Develop communication protocols and data formats for seamless data exchange.

Power Optimization:

Implement power-saving techniques, such as low-power modes, sleep/wake cycles, and duty cycling, to prolong battery life. Optimize data transmission intervals to strike a balance between

energy efficiency and real-time responsiveness.

Security Measures:

Implement encryption and authentication mechanisms to ensure data privacy and prevent unauthorized access. Incorporate secure boot processes to safeguard against firmware tampering.

Testing and Validation:

Conduct extensive testing to validate the reliability and performance of the wireless communication system. Test the system under various scenarios, including range testing, interference testing, and robustness against packet loss.

Documentation and User Manual:

Create comprehensive documentation detailing the system design, hardware schematics, software flow, and communication protocols.

Develop a user manual to guide users on system setup, configuration, and troubleshooting.

Deployment and Demonstration:

Deploy the cable replacement system in the target use case environment. Demonstrate the successful replacement of traditional cables with the wireless solution, showcasing its benefits and performance improvements.

Future Scope:

Explore opportunities for further enhancements, such as integrating edge computing for local data processing. Consider scalability options for deploying the cable replacement solution in other relevant applications.

The Cable Replacement Project in Embedded Systems presents an exciting opportunity to transform conventional wired setups into more versatile, efficient, and cost-effective wireless systems. By successfully implementing this project, the team can demonstrate the potential of embedded systems and wireless communication to revolutionize various industries and improve their operational efficiency.

7.2 : Project-2: Remote Temperature Sensor

Project Title: Remote Temperature Sensor

Project Overview:

The Remote Temperature Sensor project aims to develop an embedded system that can accurately measure and transmit temperature data wirelessly from remote locations. The project focuses on creating a cost-effective and energy-efficient solution that can be deployed in various applications, such as environmental monitoring, agriculture, and HVAC systems.

Project Objectives:

Design and build a compact and low-power temperature sensor node with sensing capabilities. Implement a wireless communication module for data transmission to a central monitoring system or a cloud platform.

- Ensure high accuracy and reliability of temperature measurements.
- Optimize power consumption for prolonged battery life and efficient operation.
- Develop a user-friendly interface for system configuration and data visualization.

System Architecture:

- Utilize a microcontroller with analog-to-digital converter (ADC) capabilities to interface with the temperature sensor.
- Integrate a wireless communication module (e.g., Bluetooth Low Energy, LoRaWAN, or Zigbee) for data transmission.
- Implement a power management system to control the sensor node's power modes.

Temperature Sensing and Calibration:

Select an appropriate temperature sensor (e.g., thermistor, thermocouple, or digital temperature sensor) with the required accuracy and temperature range.

Calibrate the sensor for accurate temperature measurements and compensate for any calibration drift.

Wireless Data Transmission:

Implement a communication protocol to transmit temperature data wirelessly to a central gateway or receiver. Choose a suitable communication range and frequency based on the deployment environment.

Power Optimization:

Implement sleep/wake cycles and low-power modes to conserve energy during idle periods. Optimize the data transmission interval to balance power consumption and data frequency.

Data Visualization and User Interface:

Develop a user-friendly interface for configuring the sensor node and visualizing real-time temperature data. Provide options for setting temperature thresholds and alarms.

Security Measures:

Implement encryption and authentication to secure data transmission and prevent unauthorized access. Incorporate secure boot mechanisms to protect the system from potential attacks.

Testing and Validation:

Conduct rigorous testing to ensure the accuracy and reliability of temperature measurements. Verify the wireless communication performance and range in real-world scenarios.

Documentation and User Manual:

Create detailed documentation of the system design, hardware connections, and software implementation. Develop a user manual to guide users on setting up and configuring the remote temperature sensor.

Deployment and Demonstration:

Deploy the remote temperature sensor nodes in relevant environments to collect temperature data. Demonstrate the functionality and benefits of the system to potential users or stakeholders.

Future Scope:

Explore possibilities for expanding the system with additional sensors for multi-parameter monitoring.

Integrate cloud connectivity for seamless data storage, analytics, and remote access.

The Remote Temperature Sensor project offers an excellent opportunity to showcase the capabilities of embedded systems in remote monitoring applications. By successfully implementing this project, the team can provide valuable insights into temperature variations in various environments, contributing to better decision-making and resource optimization in different industries and sectors.

7.3 : Project-3:Data Logger

Project Title: Data Logger

Project Overview:

The Data Logger project aims to develop an embedded system that can log and store data from various sensors and inputs over an extended period. The system will provide a reliable and efficient way to collect, store, and manage data for analysis and decision-making in diverse applications such as environmental monitoring, industrial automation, and scientific research.

Project Objectives:

Design and build a data logging system capable of interfacing with multiple sensors and data sources. Implement data storage capabilities to efficiently store collected data for later retrieval.

- Develop a user-friendly interface for configuration and data visualization.
- Optimize power consumption to ensure prolonged operation and minimal battery usage.
- Provide data security measures to protect sensitive information.

System Architecture:

Utilize a microcontroller with sufficient memory and processing capabilities to handle data collection and storage tasks. Integrate analog and digital interfaces to connect with a variety of sensors and input devices. Implement data buses or communication protocols to handle data exchange efficiently.

Data Collection and Storage:

Interface with various sensors (e.g., temperature, humidity, pressure, etc.) and data sources to collect information. Implement data buffering and compression techniques to optimize storage usage. Utilize non-volatile memory (e.g., EEPROM or SD card) to ensure data persistence in case of power loss.

User Interface and Configuration:

Develop a user-friendly interface to configure the data logging parameters, such as sampling intervals, sensor selection, and data storage settings.

Provide options for data visualization and export for easy data analysis.

Power Optimization:

Implement sleep/wake cycles and low-power modes to minimize power consumption during idle periods. Optimize data collection intervals to balance power usage and data frequency.

Data Security and Integrity:

Implement data encryption to protect sensitive information from unauthorized access.

Employ checksums or error-checking mechanisms to ensure data integrity during storage and retrieval.

Data Retrieval and Export:

Develop methods to retrieve logged data from the system, either through a local interface or remotely via wireless communication. Provide options for exporting data to external devices or cloud platforms for further analysis.

Testing and Validation:

Conduct rigorous testing to verify the accuracy and reliability of data collection and storage.

Validate the system's performance and power efficiency under various operating conditions.

Documentation and User Manual:

Create comprehensive documentation outlining the system architecture, hardware connections, and software implementation. Develop a user manual to guide users on setting up and operating the data logger effectively.

Deployment and Demonstration:

Deploy the data logging system in relevant applications to collect and store data for analysis.

Demonstrate the capabilities and benefits of the data logger to potential users or stakeholders.

Future Scope:

Explore opportunities for integrating additional sensors and data sources to create a more comprehensive data logging solution. Consider options for real-time data transmission and cloud integration for instant data access and analysis.

The Data Logger project represents a valuable tool for efficiently collecting and managing data in diverse applications. By successfully implementing this project, the team can contribute to better data-driven decision-making, research, and monitoring in various industries, promoting efficiency and informed choices.

7.4 :Programming Steps

- Project Planning
- Hardware Setup and Connections
- IDE and Compiler Setup
- Initialize and Configure Peripherals
- Sensor Data Acquisition
- Data Processing and Logic
- Communication and Data Transmission
- User Interface
- Power Management
- Error Handling and Debugging
- Testing and Validation
- Documentation
- Deployment and Integration
- Maintenance and Updates
- Final Testing and Validation
- Documentation and User Manual
- Project Demonstration and Delivery

7.5 : Conclusion

In conclusion, exploring Bluetooth Low Energy (BLE) from first steps to developing a final app offers a fascinating journey into the world of wireless communication. Throughout the process, we have learned the fundamentals of BLE, its advantages, and its significance in modern applications.

Starting with the basics, we familiarized ourselves with BLE's low-power characteristics, making it ideal for various battery-operated devices. We delved into the concept of central and peripheral devices, understanding how they interact and exchange data efficiently.

During our exploration, we acquired essential knowledge about the Generic Attribute Profile (GATT) and the role of services and characteristics in organizing data transfer. The understanding of advertising and scanning processes enabled us to discover nearby BLE devices and initiate connections seamlessly.

As we progressed, we learned how to implement BLE communication in mobile applications, both as central and peripheral devices. We successfully developed an app that communicated with other BLE devices, enabling us to exchange data, trigger actions, and create interactive experiences.

Furthermore, we grasped the importance of security in BLE communication and learned how to implement encryption and authentication measures to protect data integrity and user privacy.

The journey from first steps to the final app in BLE development has empowered us to create innovative and practical solutions for diverse industries, including healthcare, fitness, home automation, and IoT.

In conclusion, our exploration of Bluetooth Low Energy has opened up a world of possibilities, fostering the development of energy-efficient, wireless communication solutions that enhance the user experience and improve our daily lives. As we continue to advance in the field of BLE development, we contribute to the ever-evolving landscape of technology, enabling seamless connectivity and smart interactions in the digital age.

REFERENCES

- [1] www.microchip.com
- [2] <https://mu.microchip.com/8-bit-microcontrollers-architecture-of-the-pic16>
- [3] <https://mu.microchip.com/intro-to-the-mplab-x-ide>
- [4] <https://mu.microchip.com/getting-started-with-pic16f1xxx-mcus-using-mcc-and-state-machines>
- [5] <https://mu.microchip.com/syntax-and-structure-of-c>